The Pennsylvania State University

The J. Jeffrey and Ann Marie Fox Graduate School

Automated Requirements Quality Measurement (ARQM): A Tool for Requirements Quality Analysis

A Praxis in

Engineering

by

Dylan Porter

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Engineering

May 2026

The praxis of Dylan Porter was reviewed and approved by the following:

Joanna F. DeFranco, Ph.D.
Associate Professor of Software Engineering
Associate Director, Doctor of Engineering in Engineering Program
Professional Doctoral Culminating Experience Advisor
Chair of Committee

Everton Guimaraes, Ph.D.
Associate Professor of Software Engineering
Committee Member

Satish Srinivasan, Ph.D.
Associate Professor of Information Science
Committee Member

Sven Bilén, Ph.D.
Professor of Engineering Design, Electrical Engineering, and Aerospace Engineering
Director, Doctor of Engineering in Engineering Program

ABSTRACT

As systems grow more complex the need for clear, testable, and unambiguous software requirements increases, thus requirements engineering has become critical to build quality products with strong user acceptance. As a result of added complexity, requirement documentation has also become more complicated. Existing tools have tried to mitigate the tedious analytical processes associated with requirements engineering by automating various parts of the requirements engineering process. Of the existing tools, many are either unavailable, utilize outdated methods, or require extensive user input.

The Automated Requirement Quality Measurement (ARQM) tool mitigates several limitations of existing solutions, while utilizing modern AI-based methods for both requirement identification and quality analysis. As a modern tool, ARQM eliminates the necessity of human input, by (1) identifying requirements in unstructured documents, and (2) analyzing the associated quality of all requirements based on select quality attribute criteria. Through the utilization of the definitions provided by the ISO 29148:2018 standard, ARQM provides analytical insights for ambiguity, singularity, verifiability, and feasibility requirement violations.

As an added foundation of the ARQM tool, the output includes a PDF artifact that shows the requirements with violations and an associated explanation for improvement. Distinct from other tools, ARQM focuses on a practical and easy experience for requirement processing. The goal of the tool is to minimize human requirement evaluation, while also improving readability and intuitiveness of requirement violations.

The ARQM AI models were trained on publicly available human-annotated datasets for requirement identification and quality analysis. It was determined that both lightweight and advanced models generalized well, allowing for full automation of the ARQM tool. The ARQM tool was validated by common metrics such as accuracy, precision, and recall among the various

classes during training. These metrics were utilized across both human-annotated datasets and the PURE dataset to help provide insight into the ARQM tool's ability to both identify and analyze requirements. Additionally, the analysis of the ARQM tool includes the agreement among human annotators and the ability to learn annotator patterns for both identification and analysis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Glossary

Abstractive Summarization: A method to summarize text without reusing existing references, words, or phrases.

Accuracy: A way to describe how often a model correctly predicted a class during inference. Accuracy is a common metric to analyze model effectiveness.

Artifacts: Any document or data that could potentially contain requirement information. Examples include recordings, requirement documents, or standards.

Child Requirements: Sub requirements that are more granular than their associated parent. A child requirement typically helps specify or outline the function of the parent requirement.

Classification: The categories associated with a model inference task, such as determining requirement category or type of quality violation.

Cosine Similarity: A way to determine the similarity of vectors. Helpful in determining semantic relationships among text.

Datasets: A structured form of data utilized to help train statistical and AI-based models.

Domain Knowledge: Refers to the specific knowledge relevant to a given requirement or requirement document.

Embeddings: A way to vectorize text and encapsulate semantical properties. Embeddings can help add information to improve model inference results.

Ensemble: Typically relates to the utilization of multiple models to help improve the stability of inference. Examples include Random Forest, where model results either depend on an averaging or majority vote method.

Explanatory Text: Text that does not necessarily pertain to requirements. Typically, explanatory text helps further define requirements or associated context. In isolation, explanatory text does not constitute requirement text.

Extraction: A method to extract components from within text or requirements. For instance, automated extraction may identify actions, agents, and relationships for requirement text.

Extractive Summarization: Alternatively to abstractive summarization, extractive summarization identifies the most important text to help summarize large text.

F1-Score: A way to culminate other statistics to help better understand model performance. F1-Score is a measurement of model performance.

Features: Concepts that may help during the model inference process. Features are often correlated and tightly related to information gain, allowing an improved model training process.

Few-Shot Learning: A deep learning method that performs well when there is limited data. This method can often better generalize when presented with representative pairs of data.

Fine-Tuning: Taking an already trained model and introducing it to a new task. During training, the model is fine-tuned to help perform inference. Typically, fine-tuning limits the dataset requirements by utilizing existing model knowledge.

Functional Requirements: Requirements that are directly linked to a user's wants or needs. Typically, functional requirements are more specific and formal in representation.

Heuristics: General rules-of-thumb that hold true in most cases. Heuristics are represented in requirements automation through pre-defined lists and syntactical representations.

IEEE 29148:2018 Standard: A standard defining the ideal requirement process, ranging from requirement elicitation to requirement management. Additionally, the standard specifies quality attributes that help improve requirement quality.

Indexing: A method that represents the hierarchy and relationships of requirements for requirement documents. Helps to understand the various requirement groups and requirement dependencies.

Kappa Agreement: A way to measure user agreement. Typically utilized for labeling tasks. A higher Kappa agreement indicates a positive agreement among reviewers.

Lemmatization: The reduction of words into their base form. Lemmatization is typically used to help limit the feature space during model training.

Lexical Analysis: A type of analysis that looks at individual words and morphemes to determine requirement classification and potential quality attribute violations. Associated heuristics would typically consist of pre-defined word or phrase lists.

Lists: A collection of heuristics that often indicate requirement violation. Lists helps with quick inference and mapping of explanations for model inference.

Meta Model: A model that synthesizes the complexity of other models into a simpler model. Functionally, the meta model is trained on output from a more complicated model to learn patterns. Meta models are important to achieve XAI and faster inference.

Modeling: A way to synthesize requirement documents into a formal representation. Models can help provide a deeper analysis of requirement quality and often work with both heuristics and human input.

Natural Language: The type of language typically present within requirement documents. Natural language is typically unformal, leaving opportunities for ambiguous language and poor requirement understanding.

Non-Functional Requirements: Typically pertain to derived or quality requirements. Network speed, task complexity, and user satisfaction are often associated with this category of requirements. Typically, non-functional requirements are less formal and more difficult to validate or measure.

Ontology: A structured representation of requirements. An ontology typically contains the individual requirement units, the associated hierarchy, and the relationships between them.

Overfitting: When a model learns the training data but fails to learn the general patterns for unseen data, limiting the viability of the model.

Parent Requirements: A requirement entity that typically encapsulates multiple child requirements. Additionally, parent requirements typically contain low granular specifications for features.

Pragmatic Analysis: A type of analysis that determines the likely reading of text, instead of the strict semantical representation. Pragmatic analysis is crucial for determining whether a quality violation is useful during the automation process.

Precision: A metric used to assess the associated performance of models. Precision helps to better understand the performance among different classes instead of the broad accuracy.

Prompt Engineering: A method to help hone model results through the use of natural language. This method typically contains task definitions, examples, and context specification.

Quality Analysis: A method to analyze the associated quality of requirements. Typically, quality analysis utilizes a specific quality model and heuristics to help measure quality. These quality models can analyze requirements on multiple different aspects, allowing for a dynamic analysis of requirement documentation.

Quality Attribute: A quality attribute pertains to a specific type of quality standard for a requirement. Quality attributes such as ambiguity and feasibility analyze requirements based on different definitions and expectations.

Quality Indicator: A quality indicator or typically heuristics that map to an associated quality attribute. Indicators generally help provide guidance for potential violations across a quality model.

Quality Model: A quality model incorporates best practices for requirements. Typically, these models include multiple quality attributes that help to specify ideal requirements.

Question Answering: An AI-based method that uses natural language in the form of questions and answers to extract out critical information. This task can help with entity extraction and relationship generation.

Recall: A metric that helps analyze model performance across all classification possibilities. Recall helps to show the proportion of entities that were uncovered for each classification.

Requirement Document: A document that specifies a general system or product. Consists of explanatory text, context, and associated requirements. Requirement documents are the primary artifacts that typically contain specified requirements.

Requirement Document Metadata: A data structure that encapsulates the structure and relationships within a requirement document. The metadata would include requirement groups, requirements, hierarchical representations, context, and document summarization. Typically, this data is used as input for quality analysis of requirements.

Requirement group: An entity that can include multiple requirements and requirement groups.

Requirement Hierarchy: A relationship that includes both requirements and requirement groups. Typically, entities are nested at various levels to help specify varying degrees of granularity.

Requirement Identification: An automated process meant to extract all requirement text from a requirement artifact.

Requirement Quality: A way to measure how good or bad a requirement is. Typically, quality models provide guidance for determining requirement quality.

Requirements: A way to represent a user want or need with various details and constraints.

Requirements Elicitation: A method that helps derive important and desired requirements from users. Requirement elicitation can also include research to help derive non-specified requirements.

Requirements Engineering: A process to help translate a desire into a practical system implementation.

Requirements Management: A process that keeps track of requirement evolution and importance. This process can help with budgeting and prediction processes pivotal for adequate product delivery.

Requirements Traceability: Typically refers to the tracing of requirements to documentation or reasoning provided by the user. For instance, requirements can trace to external documentation and user conversations for support. Additionally, requirement traceability can reference test cases to help verify implementation.

Requirements Validation: A process to validate the importance, relevance, and practicality of associated requirements. Validation can serve as a way to strengthen requirement specifications based on user input.

Rule-Based: A method that utilizes techniques such as heuristics, regex, or programming logic to accomplish a task.

Semantical Analysis: A way to uncover the actual meaning of words and phrases within a sentence. This type of analysis helps to uncover ambiguity and quality violations at a higher level of analysis.

Semi-Supervised Learning: A method to help label unlabeled data from models or methods trained on a small, labeled dataset. Additionally, features derived from unsupervised methods could help label data.

Smells: A general determination of quality adherence through the use of features or indicators. Typically, requirement smells are aggregations of various statistics or indicators, providing broad generalizations of requirement quality.

Stemming: A method that removes the stem of certain words to help mitigate the feature complexity when training models.

Supervised Learning: Training models with labeled datasets to help accomplish a task. Datasets typically need broad representation for better generalization during inference.

Syntactical Analysis: A higher-order analysis from lexical analysis. Allows for the generation of heuristics for words, phrases, and grammatical patterns to help determine quality violations.

Unsupervised learning: A method to derive features or information about data without a labeled dataset.

Vectorization: A method that translates text to a numerical vector. Methods can include Count Vectorization, Embeddings, or TF-IDF. These representations are necessary for model training.

List of Acronyms

AI: Artificial Intelligence

ANN: Artificial Neural Network

API: Application Programming Interface

ARQM: Automated Requirement Quality Measurement

BERT: Bidirectional Encoder Representations from Transformers

CNL: Controlled Natural Language

CNN: Convolutional Neural Network

IEEE: Institute of Electrical and Electronics Engineers

LDA: Latent Dirichlet Allocation

LLM: Large Language Model

LSTM: Long Short-Term Memory

ML: Machine Learning

NER: Named Entity Recognition

NLP: Natural Language Processing

OCR: Optical Character Recognition

PCA: Principal Component Analysis

POS: Part-of-Speech

PURE: Requirement Identification Dataset

QA: Question / Answering

RAG: Retrieval-Augmented Generation

RE: Requirements Engineering

RNN: Recurrent Neural Network

SHAP: Shapley Additive Explanations

SMOTE: Synthetic Minority Over-sampling Technique

SQuAD: Stanford Question Answering Dataset

SRL: Semantic Role Labeling

SVM: Support Vector Machine

TF-IDF: Term Frequency-Inverse Document Frequency

UI: User Interface

UML: Unified Modeling Language

XAI: Explanatory Artificial Intelligence

ACKNOWLEDGMENTS

During the development of this research, there were numerous individuals that helped make the ARQM tool possible. First, I want to thank Dr. Phillip Laplante. As my original advisor, he provided immense help and guidance for the research direction and implementation of the practical ARQM tool. His help directly contributed to the advancement of this project. I also want to thank Dr. Joanna DeFranco. As my second advisor, she was pivotal in providing directions on both publishing and the finalization of the ARQM project. I also want to recognize Dr. Everton Guimaraes for helping gather datasets utilized during the research and Dr. Satish Srinivasan for providing valuable direction and feedback on the research scope and process.

For the research, I want to thank Mike Dudas, Gerald Lee Wagner, Zaid Adam, Tara Murta, and Joshua Taylor Bradley for their contributions. These individuals helped provide labeled datasets for both requirement identification and quality analysis. Ultimately, their labeled datasets helped to improve the ARQM application and further the research applicability. Their contributions were immense and strongly appreciated.

Lastly, I would like to acknowledge my partner, Emily Kyle, for the strong support she gave me during my research. She helped direct me towards an alternative direction of research, with discussions and ideas pertaining to Explanatory Artificial Intelligence. With her input, the research focus shifted to alternative methods for explanation generation among quality violations. This shift was critical for the practical implementation of the ARQM application.

Chapter 1

Introduction

In modern software projects, requirements engineering is becoming increasingly complex. Problems in requirements engineering activities (i.e., elicitation, analysis, specification, validation, management etc.) can create costly project delays and lower quality upon delivery, leading to customer dissatisfaction. Requirements engineering becomes even more challenging with the increase in product complexity and user expectations. With the increase in complexity, analyzing associated requirement documents becomes a challenge and risk to most project initiatives (Laplante & Kassab, 2022).

As an added complexity to the requirements engineering process, requirements typically are represented and conveyed in multiple different ways. For instance, requirements are normally provided in the form of structured, semi-structured, or unstructured documents. Often, requirement specifications tend to follow a specified format for documenting requirements and their supporting information. Other documents, however, may not adhere to the same structured rules that requirement specifications tend to follow. Because of the complexity and differing structures of requirement documents, identifying and validating requirements across different documents becomes difficult, prompting the creation of different implementations, methods, and tools (Zhao et al., 2021).

Due to the importance of requirement analysis and the increasing complexity of systems, creating automated systems can help assist the requirements engineering process by analyzing documents, identifying requirements, and analyzing requirement document quality against a set of pre-defined quality attributes and standards (Carlson & Laplante, 2014; "ISO/IEC/IEEE

International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Ultimately, identifying and evaluating requirement quality helps with the automation of many of the requirements engineering activities. The focus is primarily to formulate practical methodologies and software that can both identify requirements and measure the overall quality of requirements. When considering the importance of quality automation processes, identifying requirements is pivotal to the automation process, as requirement identification can eliminate the need for manual requirement annotation and labeling. Additionally, requirement identification can often serve as a precursor to many other requirement automation tasks. With an adequate requirement identification process, identifying critical requirement text becomes possible, allowing for the implementation of a system that better incorporates user needs (Zhao et al., 2021).

Although requirement identification is an important part of tool automation, the task is relatively difficult because of the unstructured nature of requirement documents. Often, requirement identification is not possible with the use of pre-defined syntactical patterns or phrases. As a result, context becomes critical to requirement identification, especially when requirement documents do not follow expected requirement formats. As an additional challenge, single requirements can often span multiple sentences, making typical parsing more difficult. Even with these challenges associated with requirement identification, many critical processes, including quality analysis, require the accurate representation and inclusion of document requirements (Ahmad et al., 2020; Zhao et al., 2023).

As the typical next step of the quality automation process, requirement quality analysis is another process that can help simplify the requirements engineering process. Often, requirements are written in natural language, leaving potential interpretability or practicality issues depending on how the requirement is defined. To address potential requirement quality issues, the IEEE

29148 standard provides a foundation for requirement quality in the form of quality attributes. As the standard mentions, quality attributes measure different aspects of the requirement to determine the requirement's overall quality. As an example, quality attributes such as ambiguity, conformance, and feasibility all contribute to the overall quality of requirements. A requirement with high ambiguity would ultimately hurt the interpretability of requirement text, while an unfeasible requirement likely does not represent user needs. Through the use of a quality analysis process, the overall requirement document quality is improved through well-defined requirements, improving the accuracy and usefulness of requirement documents ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Overall, the goal of automation is to help both identify requirements within relevant documentation while also measuring requirement quality, ultimately helping to facilitate a better product quality and outcome. Existing tools have tried to assist with the requirements engineering process, but most of the tools are either unsupported or utilize technology that cannot adequately measure requirement quality. Although many of these tools are powerful, the focus of automation is to mitigate manual work related to the requirements engineering process; many tools accomplish the simplification of manual work, but fail to accurately identify and analyze complicated requirement documents, creating an opportunity for practical tool improvement (Carlson & Laplante, 2014).

Requirement Identification

Requirement identification is important for automating requirements engineering. Ultimately, identifying all the requirement-related text from within a given document helps systems check the quality of requirements, while also formulating an understanding of requirement document structure. Although requirement identification may be simple for structured documents with consistent syntactical patterns, identifying requirements for different types of documents and formats is difficult. Even with structured documents, requirements often do not follow the same syntax across different domains. Additionally, requirements come in many different forms, including user stories, use cases, models, and natural language. Because of the diverse number of formats and mediums that can contain requirements, there are a wide range of different syntaxes that can encapsulate requirements. Although requirement identification is difficult due to requirement document complexity, the process is necessary to adequately analyze requirement quality (Laplante & Kassab, 2022).

Requirement Identification through Rule-Based Methods

There are many ways to achieve requirement identification, including rule-based and heuristic methodologies. Rule-based methods primarily focus on hard-coded logical checks to identify requirements. These checks could utilize regular expressions, textual statistics, or other pre-defined patterns that look further into the text's syntax or document position. These general rules help classify most instances of requirements with a relatively high degree of accuracy. Rule-based methods can also utilize heuristics to help decide if a sentence may belong to a requirement. For instance, textual statistics and the occurrence of certain syntactical structures could help to identify the occurrence of a requirement; these methods are useful because they are less rigid than exact string matches. These common formats associated with requirements means that rule-based methods can often identify requirement text by simply observing and identifying common patterns ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Jafari et al., 2021; Luttmer et al., 2023).

Although rule-based methods are an effective way to identify requirements, formulating adequate rules and heuristics to identify requirements is difficult due to the different types of syntactical structures and documents. Further, requirements often span multiple sentences, meaning that rule-based methods may have a difficult time consolidating multiple sentences into a single requirement. Because rule-based methods often do not consider context, the methods do not adequately translate to different types of requirements either. For instance, although functional requirements tend to follow a specific format, non-functional requirements do not. Additionally, other documentation that contains unstructured language presents additional challenges to rule-based methods. Instead of manually defining general rules to identify requirements, other methods that can better encapsulate complicated patterns across different types of text can become more useful in the requirement identification process (Zhao et al., 2021, 2023).

Requirement identification through Artificial Intelligence

Because rule-based methods are unsuitable for requirement identification, Artificial Intelligence can serve as an adequate substitute to help classify text. By utilizing AI to learn complicated patterns through training data, improving inference generalizability and applicability to diverse requirement documents is possible. Ultimately, AI can learn complicated representations of text, allowing for an understanding of requirements beyond simple heuristics. From these learned pattern representations, AI can help with learning complicated, general patterns to help better identify requirements. Additionally, there are a variety of models, learning types, and configurations that can help achieve better results across a dataset of differing requirement formats (Zhao et al., 2021).

When utilizing AI, there are multiple different implementations that can help improve the results of inference for requirement identification. For instance, there are supervised, semi-supervised, and unsupervised learning methods that provide broad inference capabilities when datasets are limited. For supervised models, a labeled dataset is necessary to train and adjust the model to achieve accurate results. In this instance, the structure, preprocessing, and balance of the data is critical for achieving high model accuracy. Having a large amount of data that spans a diverse range of examples is critical to improving the model's generalization across unseen data. When the dataset does not adequately represent unseen data, the model may overfit, losing the ability to accurately classify unseen examples. As a result, supervised models often require large, diverse datasets to perform well for requirement identification (Wang & Chen, 2023).

In the case of unsupervised training models, a labeled dataset becomes less critical. Although unsupervised models are not always as accurate or reliable as supervised models, they are an effective option for uncovering features representative of the specific inference classes. For instance, unsupervised models can use similarity grouping of textual features to determine group membership. For text, certain methods such as topic modeling, embedding similarity, and dimensionality reduction are useful tools to achieve classification while uncovering pivotal features for inference. Topic modeling, for instance, can help with identifying the requirement and non-requirement text within documents by treating them as separate topics; from the group derivations, analytical processes can help uncover features representative of those classes. Unsupervised methods are also better when certain class representations are uneven, removing the need to artificially balance data classes. Lastly, unsupervised methods can help with identifying outliers and performing better general classification across unseen data when performed with certain methods like dimensionality reduction. Overall, unsupervised learning is a critical step for requirement identification and quality analysis due to dataset limitations and data complexity (Kubat, 2021).

Ultimately, both unsupervised and supervised learning methods can complement each other in the form of semi-supervised learning. Performing data exploration with unsupervised learning can help uncover features that may represent data classes; ultimately, the uncovered features are then utilized as input to train the supervised model or create labeled datasets. Unsupervised learning can also help reduce dataset noise, while amplifying features that can better identify classes for a requirement identification task. The learned features can serve as input to a supervised learning algorithm to formulate the relationships between the uncovered features and the dataset. Additionally, grouping methods can help create a labeled dataset by associating each group with a classification, increasing the labeled examples available for training. In summary, the combination of both supervised and unsupervised learning can help overcome the barriers of limited datasets, while maintaining a high level of accuracy for classification tasks (Kubat, 2021, 2021; Sallam et al., 2020; Shanthamallu & Spanias, 2022).

For most requirement automation tasks, datasets are often minimal or entirely unavailable. Although there are some public datasets available, they often are not labeled for requirements identification training. As a result, manually labelling data, utilizing unsupervised and semi-supervised methods, or utilizing data preprocessing and hyperparameter for supervised learning methods becomes necessary for better data generalization . Additionally, utilizing transfer learning can also help mitigate the amount of labeled examples necessary to perform accurate requirements identification. As an added benefit, transfer learning allows for the infusion of knowledge obtained from trained LLMs for a classification task like requirement identification; techniques like prompt engineering can help consolidate the output and achieve higher generalization without a representative dataset (Kubat, 2021; Verma & Kass, 2008; Wang & Chen, 2023).

Another consideration of requirement identification is the difficulty with multi-sentence requirements along with differing syntax or document types. These considerations complicate the issues of limited datasets; although, the proposed methods help to mitigate some of the limitations of scarce data. Some of the previously discussed methods can help improve model generalizability for diverse representations of requirements, but accurate requirement consolidation and analysis is still difficult. Ultimately, the requirements identification process should encapsulate the entire requirement unit, which may span multiple sentences. These challenges are enhanced by the limited availability of datasets but can be addressed with appropriate models and techniques to enhance generalization and consolidation of related requirement text (Sallam et al., 2020; Zhao et al., 2023).

As an added consideration, the difference between requirement and non-requirement-related text is not necessarily clear. Often, text can help further clarify existing requirements but is not ultimately requirement-related text in isolation. As a result, combining and condensing requirement text into a single unit requires further evaluation and consideration of requirement definitions and representations, especially when considering requirement identification serves as the input for many important automation processes. For instance, identifying the entirety of requirements while limiting unnecessary or unrelated explanatory text is important to many subsequent automation processes such as quality analysis. Defining the holistic unit of a requirement involves complex analysis, sentence consolidation, and strong model inference capabilities ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Zhao et al., 2023).

Lastly, requirement position considerations matter during the requirement identification process. Requirements often do not exist alone; requirements tend to exist in a hierarchical structure in which they often belong to groups present with a requirement document. Ultimately, groupings of requirements tend to specify functionality at a different level of specificity, further complicating the requirement identification and analysis process. When identifying requirements, maintaining the position of the requirement in relation to other requirements is important for additional automation processing, such as quality analysis. For instance, ambiguity may be a permissible violation for parent requirements, where child requirements should typically contain more granular specification. Additionally, the position of requirements can help with other quality attribute evaluations, including checking for completeness, consistency, and feasibility through the requirement document ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Artificial Intelligence can generally help with the requirements identification process. Although there are limitations, new methods can help extract out and classify metadata related to requirements documents. Additionally, models can learn to generalize across unseen examples of requirements, allowing for generic classification beyond rule-based heuristics. Further, AI models can utilize tacit knowledge via transfer learning to help better interpret semantics and pragmatics of language, allowing for context-specific analysis. Because datasets are limited for requirements classification, unsupervised and semi-supervised learning can help with the process of generating synthetic data to help improve training results. In conjunction with Natural Language Processing, Artificial Intelligence can likely improve the accuracy of requirements identification tasks through either well-defined and generic heuristics or task-specific models.

Requirement Quality

The natural next step after requirements identification is requirements quality analysis in the automation process. The goal is to analyze the requirement text, evaluate requirement quality against defined criteria then provide feedback on any defects. By analyzing the quality of the requirements in a requirements document, the system may avoid costly mistakes and misunderstandings further along the project's development and implementation. Therefore, analyzing requirements for quality attribute defects is important for understanding the potential of impacting system performance, delivery, and acceptance from the user. There are many different types of requirement quality attributes that apply to the requirement analysis process. For instance, the IEEE 29148:2018 standard provides an outline of the quality attributes that requirements text should typically follow. Some of these quality attributes are easily measured, while others require more extensive methods for analysis ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Measuring Requirement Quality through Rule-Based Methods

There are many ways to measure requirement quality through rule-based methods. As an example, many older tool implementations have utilized these methods to provide a general overview of document quality. Although these methods do not always yield accurate results, the utilization of rule-based methods is mostly useful for lexical and syntactical analysis. For instance, there are expected syntactical patterns that apply to the ideal requirement format. The standard suggests that requirements should be constrained by certain syntactical patterns; if requirements deviate from these patterns, there exists a requirement quality defect. Additionally, the standard mentions that there are various phrases and words that indicate additional defects for requirements. Most of these issues are measurable through rule-based methods because of the simplistic and generalized problems they tend to create (Carlson & Laplante, 2014; Davis et al., 1993; "ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

From a lexical and syntactical analytical perspective, rule-based methods can measure requirement quality through pre-defined rules and phrases that are often attributed to quality violations. Words such as 'best' or 'most' and phrases such as 'user friendly' and 'almost always,' are quality violations that are practical to detect with rule-based algorithms. These pre-defined terms provided by the standard show how certain linguistic words and phrases can create problems among requirements. Ideally, automation software can look at common words and phrases that may indicate quality errors to help improve requirement document quality (Carlson & Laplante, 2014; Femmer et al., 2017; "ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Kummler et al., 2018) .

As an added consideration, syntactical patterns can help provide useful ways to generally look at requirements through rule-based methods. Primarily, these patterns can represent both syntactical and semantical issues that extend beyond looking for common words or phrases. With Natural Language Processing (NLP), utilizing common libraries that label text with POS associations is also possible. This type of analysis allows for the detection of more generalizable patterns among requirements, especially for syntactical and semantical analysis. Importantly, these methods do not require extensive model training and tend to help better address simplistic errors that may arise when analyzing requirement quality. As an alternative, the rule-based methods can serve as an effective way to engage in Feature Engineering for different models that require complicated and diverse training data (Chantree et al., 2006; Nugues, 2024; Sallam et al., 2020).

Although rule-based methods may not necessarily work for the general case of requirement quality analysis, defining certain textual statistics and features is possible to help facilitate better results during analysis. Primarily, the detection of certain words, phrases, and patterns can help train models for effective classification. Additionally, textual statistics, such as the occurrence of certain words, semantic similarity of words to other ambiguous terms, and other textual metrics could potentially help with analyzing requirement quality.. Methods of data analysis can also help further uncover rules that could generally indicate quality violations, allowing for the expansion of rule-based analysis across disparate topics and documents (Y. Li et al., 2018; Sonbol et al., 2022).

Rule-based methods provide a way to analyze associated requirement data, derive generic patterns that define quality defects, and apply those general patterns to most requirements. Although the analysis through rule-based methods tends to be shallow and less accurate than AI-based methods, rule-based methods serve as a good starting point for analyzing most requirement quality defects on a lexical and syntactical basis. Unfortunately, rule-based methods are only applicable for certain quality attributes and levels of analysis. Different, more complex, quality attributes require a more holistic analysis of context that goes beyond what rule-based methods can likely achieve (Carlson & Laplante, 2014; "ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Measuring Requirement Quality through Artificial Intelligence

Although rule-based methods are useful for basic requirement quality analysis, Artificial Intelligence provides a way to further generalize among data for quality attribute measurement, while also allowing for the analysis of complicated quality attributes. Across lexical, syntactical, and semantic representations of text, there are many different variations of quality attribute violations that rule-based methods cannot measure. Although some rule-based methods can provide generic patterns matching across different textual examples, Artificial Intelligence can help learn and facilitate the patterns beyond the words and surface-level structure. Instead of manual observation of patterns and user-specified lists of words and phrases that tend to indicate quality violations, Non-Supervised, Semi-Supervised, and Supervised methods can help discover features of text that may help classify requirement quality violations more broadly. Additionally, quality attribute violations typically require context analysis, especially for semantical analysis for requirements. AI-based models can consider context while performing classification, leading to a more effective method of defect detection (Shanthamallu & Spanias, 2022; Zhao et al., 2021).

Because quality attribute violation datasets are scarce, Unsupervised Learning is a powerful approach to discover features for quality attribute violations. As a form of feature engineering, Unsupervised Learning can help discover and apply generic sentence features that indicate quality violations for the model training process. These uncovered features could help train models to better generalize and classify requirement quality attribute violations. Because the quality attribute classifications are different during automation, discovering unique features for all classifications is important for a holistic quality analysis. Additionally, extracting features that can indicate quality violations on a lexical, syntactical, and semantical basis is important, especially when considering that multiple similar quality attribute violations can occur at the same time within a sentence. Another consideration is the mapping and outline of quality attribute violations to explanatory text. Although Unsupervised methods provide a way to uncover features, they do not necessarily provide the positions, references, or explanations of the quality attribute violations (Zhao et al., 2021).

In general, AI-based models can perform better with classification when compared to text generation tasks. Although models can accurately classify quality attribute violations through derived features for Unsupervised Learning, the models do not necessarily generate a reason for the violation or a reference to the problematic words or phrases within the text. As a result, the problem of quality attribute measurement goes beyond simple classification tasks and, therefore, requires different models and methods to provide clarification for quality violations. As a potential solution, Sequence-to-Sequence transformers allow for the transformation of text to a textual response; the model type can also combine classification and sequence generation tasks, mitigating the need for multiple models. Additionally, these models can consider its tacit knowledge and context before generating a response, providing a higher order of analysis when compared to rule-based implementations. In the case of transformers, fine-tuning, few shot learning, and prompt engineering can help provide better explanations and context for the analysis of quality attribute violations. These methods can help the model better understand and predict results for quality attribute violations, while also further limiting the need for extensive training data for achieving better results (Wang & Chen, 2023).

As an example of model augmentation, prompt engineering can also help refine the output while providing important context to the model based on requirement document supporting information. In a simple prompt engineering scenario, a few examples of instances and results are provided to the model; often, the prompt contains instructions, sample input and output, and a specific format to return the results in. The instructions could also include the need to detect quality attribute violations on a lexical, syntactical, and semantical basis, while providing explanations for the associated violations. The explanations could manifest in a format that would either highlight the contributory text or provide an abstract suggestion for improvement (Korzyński et al., 2023).

Practical Research for Requirement Identification and Quality Analysis

With the wide variety of different implementations for both requirement identification and quality analysis, determining ideal ways to accomplish both tasks through a practical tool is useful to help with automated requirement analysis. Ultimately, the goal of the practical tool is to accomplish both requirement identification and quality analysis through the use of different methods, such as rule-based and AI-based implementations. This research ultimately compares the methods and based on the results, implements an application that utilizes the best automated implementations to provide a practical tool for requirement analysis. Ultimately, the tool's main focus is to identify and analyze requirements based on the pre-defined ISO 29148:2018 standard. For both identification and analysis, the focus of the research is to also look at ideal methods for identification and analysis by comparing the results to human-annotated datasets.

Practical Research: Requirement Identification

With requirement identification, the key issue is identifying requirement-related text from a structured, requirements document generally written in natural language. To identify requirements, documents are processed into smaller units, such as paragraphs and sentences. These smaller units are what typically encapsulate requirements. Therefore, the identification process becomes a classification problem amongst sentences or similar units, allowing for the identification and classification of requirements.. Intuitively, identifying requirements instead of non-requirement-related text is important, as failure to identify requirements could lead to significant analytical issues. Additionally, suggestions to improve the quality of non-requirements would serve to increase the brevity and clarity of the document. Therefore, the recall of positive cases for requirements should be a primary consideration. Another focus of requirement identification automation is the generalization of other requirement documents. The ability for the identification process to identify requirements throughout different documents is an important metric; considering that requirement documents tend to utilize similar language for related products, utilizing requirement identification for documents that focus on different domains and projects is critical for understanding the accuracy and effectiveness of the tool.

Practical Research: Requirement Quality

Compared to requirement identification, requirement quality analysis is a more difficult task due to factors such as different classification types, explanation generation, and the need for context to perform complicated and diverse analysis. Additionally, requirement quality analysis goes beyond basic classification; the analysis includes lexical, syntactical, semantical, and pragmatic levels of automation that further complicate the ability to detect defects. Primarily, the focus of quality analysis within a practical tool is to show a proof-of-concept for basic quality attributes while providing future direction for further research and improvement.

As an added consideration of requirement quality analysis, data and context limitations provide a barrier for model inference. However, modern methods of Natural Language Processing and Artificial Language provide ways to mitigate the limited availability of datasets. Primarily, overcoming the limitations of data and context could allow for the strong utilization of AI-based technology for general quality analysis. The analysis therefore looks at ways to overcome these various limitations, while utilizing modern AI-based methods for analysis.

Lastly, an additional barrier to quality analysis includes the visualization and presentation of quality defects to the user. To accomplish visualization, the analysis needs to visualize the reasoning behind model inferences; the various methodologies that exist to analyze existing models can help with visualizing contributions of features to results, resulting in an intuitive interaction between the system and user.

Outline

This praxis provides the background information to understand that problem domains of software requirement identification and requirement quality analysis. Additionally, the praxis outlines the implementation and results for a practical tool that focuses on document ingestion, requirement identification, and quality analysis, while also analyzing the results and conclusions of the tool validation and its performance.

Chapter 2 Background – Concepts and Standards. The second chapter focuses on establishing the concepts needed for understanding requirement identification and analysis. Primarily, the chapter discusses the IEEE 29148:2018 standard terms for identifying and measuring the quality of requirements, while discussing different levels of quality analysis, such as syntactical and lexical analysis. The chapter also focuses on the definition of requirement document structure and how associated analytical techniques can achieve automation. Lastly, the chapter considers the challenges of requirement hierarchy and the implications for quality analysis.

Chapter 3 Literature Review: Requirements Identification and Quality Analysis. This chapter focuses on past and present literature for requirement identification and analysis. The chapter discusses the problem definitions, challenges, and how existing systems have addressed limitations to formulate practical tools for requirement automation. Additionally, the chapter looks at implementations ranging from rule-based methods to AI-based implementations. Lastly, the chapter analyzes the current research and tools available for quality analysis, while proposing a practical solution based on the research gap.

Chapter 4 Automated Requirements Quality Measurement (ARQM): A New Tool for Requirements Detection and Analysis. This chapter presents the ARQM tool, including the scope and features of the application. The summary includes the history of development, problems, and evolution of ARQM's implementation. Additionally, the chapter discusses the architecture of the tool, the methods and concepts utilized for requirement identification and analysis, and the associated features of the ARQM system.

Chapter 5 ARQM Validation and Evaluation. This chapter details the performance of the ARQM system for both requirement identification and analysis. Additionally, the chapter highlights the most important features that contributed to the accuracy of the model results. The chapter also discusses the methodology, including the generation of datasets, metrics, models, and results. As part of the analysis, the chapter looks at Accuracy, Precision, Recall, and F1-Score for the general classification tasks among requirement identification and quality analysis tasks.

Chapter 6 Results and Discussion. This chapter analyzes the overall results for requirement identification and quality analysis while providing an explanation for the results. Primarily, the chapter discusses the ideal models, methods, and implementations that performed well among the various datasets. Additionally, the chapter discusses the common features that helped increase performance. Ultimately, the analysis focuses on the generalizability of the model inference in relation to the human-annotated dataset, focusing on the practical agreement of the tool to requirement participants. Lastly, the analysis discusses the associated limitations of the methodology and external factors that could impact the practical functionality of the system.

Chapter 7 Conclusion. The final chapter includes a brief analysis of the novel aspects of the research, while also discussing the problems, threats to validity, and overall result implications of the ARQM system. Additionally, the chapter discusses the role of modern methods for requirement identification and quality analysis, showing how theoretical concepts can translate to the practical implementation of a requirement automation tool. Lastly, the chapter outlines how the ARQM system generally performed among both identification and analysis tasks, the generalizability of the tool to different documents and text formats, and the effectiveness of the tool in identifying issues in requirements.

Chapter 2

Background – Concepts and Standards

Defining What Constitutes a Requirement

For the tasks of requirement identification and quality analysis, analyzing the various concepts, standards, and terminologies is pivotal to understanding both the current state of existing applications and the potential for improvement and future direction. At the base of these activities are requirements, the most fundamental unit in a software specification. The activities of both identification and analysis require a strict definition of requirements, their associated syntactic templates, and quality standards that the requirements must meet. Primarily, defining what constitutes a requirement is difficult; however, doing so is pivotal for most major automation-based requirement activities. This chapter explores the definitions and concepts necessary to both identify and analyze requirements within their respective definitions and criteria outlined within the ISO 29148:2018 standard. The purpose is to define the need for requirement-based activities, while also discussing the concepts necessary to achieve adequate automation ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Zhao et al., 2021)

As a fundamental unit, requirements are a way to express the needs, constraints, and conditions that a system should satisfy in preponderance to the users utilizing the system in development. In essence, they are bound by strict syntactic and conceptual criteria; ultimately, the delineation of what constitutes a requirement is not necessarily clear. Requirements express a need, but certain types of text that may appear to be requirement-related can be supporting or non-requirement-related text. The ISO 29148:2018 standard specifies typical syntactical templates that are most often used across differing requirement documents. These syntactical templates consist of either:

- [Condition] [Subject] [Action] [Object] [Constraint of Action]
- [Subject] [Action] [Constraint of Action]

These typical syntactic templates are presented through various examples within the standard. Table 1 explains the different aspects of the typical requirement syntax ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Table 1 Criteria Definitions for Requirement Syntactic Structures

Criteria	**Explanation**
Condition	Used to limit the applicability of a requirement based on a given criteria or stipulation.
Subject	The entity of the requirement performing the action.
Action	What the requirement is achieving when the condition is satisfied.
Object	The entities interacting with the subject to trigger an action.
Constraint of Action	Constraints such as time or delay that relates to the performed action.

Often, requirements have a common syntax that helps define them from non-requirement text. Often, requirement documents contain unstructured text, making the task of requirement identification difficult, even when a document utilizes common requirement syntaxes. As an added consideration, requirements can come in a variety of different mediums and representations. As a consequence, the structure of requirements is often different from the expected syntax, requiring advanced methods or intuition for general identification of requirements. A systematic study provides common documents utilized in automated requirement extraction; they specify that requirement specifications, user-generated text, general natural language documents, and use cases where the most common data sources for requirements; additionally, they specify that other sources including legal documents, user stories, domain documents, and interviews also contained pivotal requirement information as well. The diversity of requirement representation types is immense, leading to multiple different requirement formats and representations throughout documents (Zhao et al., 2021).

Because of the diverse requirement representation across documents and natural language, the automation task of requirements engineering becomes challenging. Additional considerations of requirements identification should reflect differing formats, sources, and domains of requirements throughout the training process, or limit the scope of automation support. As a consequence of the diverse representation of requirements, following specific templates or heuristics for requirement identification is difficult. As a way to combat the multiple requirement representations, various methods exist to help automate different types of requirement identifications. However, these methods require extensive processing, feature engineering, and dataset preparation to work effectively, limiting the impact of modern tools for broad generalization (Zhao et al., 2021).

Standard for Requirements Engineering: IEEE 29148:2018

Because of the complexity associated with requirements engineering, the IEEE 29148 standard was created to help outline industry accepted standards for all requirements engineering processes. Primarily, the goal of the standard is to help facilitate a better implementation of all requirement activities, along with the improvement of the end deliverables associated with those activities. The standard provides a definition of concepts, common requirements processes, and general guidelines for requirements activities. Importantly, the standard also explains how to properly formulate requirements while eliminating common quality issues when creating requirement documents. Because of the quality templates and rules the standard provides, the templates can serve as helpful guidance for automation systems through the measurable criteria for requirement documents ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Requirements Engineering Activities

As the standard mentions, requirements engineering is comprised of many different types of activities that span outside of the formal documentation process. For instance, the standard talks about the primary activities involved within the requirements engineering process that are both user-centric and maintenance-focused. Table 2 defines these various different types of requirements engineering activities in depth based on the standards definitions of typical requirements activities.

Table 2 Primary Requirements Engineering Activities

Activity	**Description**
Requirements Elicitation	Comprised of performing research and interfacing with the user to derive requirements. This activity utilizes prototypes, designs, and information gathering to find stakeholder needs.

Requirement Analysis	The process to create and adjust requirements met by the system to meet the needs of the stakeholders. This process converts needs, wants, and desires into a tangible outline of requirements.
Requirements Documentation	This process takes the existing requirements, organizes them, and generates well-structured documents containing requirements and other, relevant supporting information. Primarily, this process generates formal artifacts that help encapsulate all generated requirements.
Requirements Verification	This process verifies that the formal requirements are reasonable, lack quality errors, and are not contradictory. Verification allows for the analysis of requirements based on certain quality attributes to help mitigate requirement misunderstandings between the developers and users of the application.
Requirements Management	This process maintains and tracks the lifecycle and evolution of requirements. Ultimately, the activity is meant to address changing stakeholder requirements during the delivery and maintenance of the system. The activity also looks at prioritization of requirements to help align development priorities.
Requirements Traceability	The maintenance of requirement hierarchy and linkage to relevant supporting text. Also used to track design and tests associated with requirements.
Requirements Validation	Confirms that the requirements meet the needs of the stakeholder. This activity is

	pivotal to delivering the correct product with the appropriate functionality. This activity is normally accomplished through processes such as reviews, walkthroughs, and prototyping.

All of the requirements engineering activities are pivotal to the success of a system. As the standard suggests, these activities can have a drastic impact on the overall system quality. These activities also serve to help better facilitate the understanding between both the developers of the product and the product stakeholders. Ultimately, each of these activities take in a variety of input, correspondence, and user interactions to help successfully enhance the system quality. For instance, requirement elicitation requires heavy involvement with users and a close understanding of the domain before formulating the actual requirements. Typically, the goal of these processes is to create deliverables such as a requirements document. For these deliverables, the standard specifies how to enhance the overall quality of the documents to improve the outcome along with user satisfaction for the product. Primarily, the standard specifies what makes up ideal requirements definitions at the conclusion of these requirements engineering activities to enhance document quality ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Different Types of Quality Analysis

To analyze requirement quality, many methods exist to help simplify and consolidate the automation process. Of these methods, rules and heuristics allow for the analysis of requirements from a surface-level perspective; other methods that require more complicated analysis and contextual understanding help to provide better analytical insight into the semantics and pragmatics of requirement text. Primarily, each of these methods aim to measure quality attribute violations at different levels of automated analysis; while most methods are useful, modern techniques tend to utilize higher-order analysis of requirements, mitigating the need for general rules and heuristics for automated systems. Still, the different types of grammatical analysis are useful and widely implemented across different automation systems depending on the specific focus of the system.

Lexical Analysis

Of the initial grammatical analysis types discussed, lexical analysis is a basic, surface-level technique that tends to look at individual units, such as words, within a requirement. To properly analyze requirements based on a given quality criterion, analyzing individual words and their associated meaning is crucial to help determine quality violations. Using heuristics, pre-defined lists, morpheme analysis, or other NLP methods commonly used for text-based tasks, lexical analysis can provide a useful interpretation of individual word contribution to quality violations. Table 3 analyzes the different methods that are available to accomplish lexical analysis ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Nugues, 2024; Zhao et al., 2021).

Table 3 Example Lexical Analysis Components

Lexical Analysis Component	**Explanation**
Word list	Utilization of pre-defined words that often contribute to quality violations. Examples include 'all,' 'every,' 'nothing,' and 'better'. The word list would typically include words

	that often cause quality violations for requirements.
Heuristics	Rules-of-thumb that typically apply to quality violations. For instance, common words, parts of speech associated with words, and word placement could all create heuristics for quality violations.
Morpheme Analysis	Certain parts of words could enhance the potential of a quality violation. Could also apply to the formulation of heuristics to further generalize potential quality violations.
Part-of-Speech (POS) tagging	An NLP-based task to assign grammatical roles to a given word. By labeling POS tags for each word, lexical analysis can further analyze the context of the word to determine if there was a quality violation. This method can also add to existing heuristics to determine quality attribute violations.
Lemmatization / Stemming	Reduction of a word to a simpler form, either to its root form or a simplified version of the word. Helpful in limiting the number of rules needed in rule-based and heuristics-based methods for quality analysis.
Named Entity Recognition	Helps with classifying categories of proper nouns. Useful in determining word impact in relation to the text. Can help formulate and simplify better heuristics for quality violations.
Word Embedding Vectorization	A way to represent words as dense vectors, typically indicating relationships among words. The vector includes important information to better understand the word's

	context within a text. Helps with uncovering deep meaning of words beyond simple analysis.

Although lexical analysis is a lower order of analysis, modern NLP methods can create a better, more holistic approach for analyzing words. Even though traditional methods of quality analysis are not always accurate, the utilization of common NLP methods can help both derive better heuristics and apply more general rules to words based on their associated grammatical information. The increased complexity of lexical analysis makes it practical to better identify the quality contributions of words to associated quality violations. As an added way to encapsulate accurate analysis of quality violations, modern NLP methods allow for the storing of context, allowing algorithms to further refining the lexical heuristics and lists associated with potential violations. Further, the ability to create dense vector representations of words helps to further understand the contribution of words to quality violations (Nugues, 2024; Plag et al., 2007).

Syntactical Analysis

At a higher level, syntactical analysis allows for a more complex and comprehensive analysis of requirement documents. Syntax-based analysis ultimately deals with analyzing the structure of sentences and the relationships between the individual words and components within a sentence; functionally, syntactical analysis helps to understand the rules and structure of text. This type of method makes an automated analysis of requirement structure possible, including an analysis of conformance to ideal requirement syntactical templates. Ultimately, deriving heuristics for quality violations based on syntax is possible; such heuristics would potentially allow for a thorough understanding of requirements quality beyond lexical analysis. By utilizing syntactical analysis in conjunction with modern NLP-based methods, the analysis of requirements can include quality metrics beyond static lists. For instance, heuristics can include grammatical patterns regardless of the underlying sentence content. As a result of more general heuristics, these syntactical rules would be more generalized across different sets of data, eliminating the need for custom defined lists of words and phrases utilized in past automation systems.

Table 4 discusses the different methods available for accomplishing basic syntactical analysis within automation systems ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Nugues, 2024; Zhao et al., 2021).

Table 4 Example Syntactical Analysis Components

Syntactical Analysis Component	**Explanation**
Phrase list	A pre-defined list of phrases that often indicate quality violations, such as 'as applicable,' 'higher quality,' 'as a minimum,' and 'but not limited to'.
Heuristics	Pre-defined rules that apply to syntactical structures and the units within text that often indicate quality violations. Heuristics are rules-of-thumb that are applicable to sentence structure, occurrence and position of words, parts-of-speech, and Named Entity Recognition (NER) tasks.
Part-of-Speech tagging	Helps to determine the classification of grammar assigned to each component within a text, allowing for the application of general rules and heuristics to determine quality violations.
Sentence features	Statistics about the text that can help determine quality violations, such as occurrence of words, occurrence of certain

	parts of speech, word relationships, sentence length, and punctuation used. Can be used for both heuristic analysis and model training or feature extraction.
Named Entity Recognition	Helps with the classification of proper nouns, which can enhance the utility of heuristics that check for quality violations such as missing agent, action, or other qualifiers within requirements. Named Entity Recognition is used to help better identify a word's impact in relation to the entire text.

Syntactical analysis is a promising method for requirement quality analysis. Past methods have utilized pre-defined lists of phrases, but modern NLP-based methods allow for more generalizable heuristics through NLP-based techniques such as Named Entity Recognition, Part-of-Speech tagging, and other NLP methods. This type of analysis is especially useful if the heuristics represent the broader category of quality violations. Using word classification, heuristics can define generalized patterns to look for patterns that extend beyond pre-defined phrase lists. These heuristics enhance general applicability for automation using NLP-derived features, while helping to broadly apply automation to unseen text. This implementation also helps in enhancing the scope and ability of tools for analyzing requirements, especially when utilized in conjunction with modern NLP methods (Fabbrini et al., 2002; Génova et al., 2013; Zhao et al., 2021)

Semantical Analysis

Semantical analysis is a pivotal component of the automation process to analyze the meaning and context of text. Semantical analysis combines the morphological, lexical, and syntactical units of a sentence to derive meaning. Because older requirements automation tools relied on rule-based and heuristic-based methods for analysis, they often were unable to capture whether a quality violation indicator was truly an indicator of a quality violation. Semantical analysis helps to enhance the understanding of text, allowing for a broader automated analysis of quality violations and enhanced classification capabilities. Table 5 shows example components that are used to accomplish semantical analysis (Jurafsky et al., 2000; Nugues, 2024; Plag et al., 2007; Zhao et al., 2021)

Table 5 Example Semantical Analysis Components

Semantical Analysis Component	**Explanation**
Embeddings	A method that is used to capture the semantical meaning of text. Embeddings are a vector representation of meaning based on different relationships.
Features	Features can include various types of word-level embeddings or sentence-level embeddings. Features can also utilize methods such as Semantic Role Labeling (SRL), POS tagging, Named Entity Recognition, or Cosine Similarity.
Semantic Role Labeling	A method that can identify various relationships within a sentence. SRL allows for the conversion of text into a static structure that is useable for the creation of heuristics and features.

Named Entity Recognition	A method to determine the context and classification of proper nouns within a text. Useful for semantics during heuristic analysis or model training. Ultimately, NER helps to better understand the actual meaning of sentences through more accurate labeling of textual units.
Cosine Similarity	A method that can determine how closely related two distinct textual units are. Cosine similarity can determine similarity across smaller units such as words while also being able to analyze bigger units such as sentences or paragraphs, allowing for the encapsulation of context.
Question Answering (QA)	Question Answering methods can be useful in determining structures of requirements and requirement hierarchical structure within a given document. QA can also utilize methods to determine if terms or concepts are properly defined within requirements.
Retrieval-Augmented Generation	A way to return relevant information based on textual input. Used to help retrieve better context and relevant information when formulating a response. Helpful with determining more complicated quality violations by accessing a wide range of knowledge and context.
Abstractive Summarization	A way to summarize and shorten large amounts of text. Abstractive summarization helps with condensing text while retaining important information. Useful as input to requirement quality determinations, including

	quality attributes that require knowledge of the entire document.
Extractive Summarization	Similar to abstractive summarization but extracts out exact words and phrases that are important to capture the entire meaning of the text. This type of summarization is also helpful in making determinations about complex requirement quality criteria. Additionally, extractive summarization is especially helpful with requirement traceability tasks.

Semantical analysis is a step beyond lexical and syntactical analysis; ultimately, semantical analysis helps to encapsulate context and meaning before making a determination through classification models. Because of the modern capabilities of semantical analysis provided by NLP-based methods, utilizing semantical analysis helps to make quality analysis more reliable and accurate due to the gained meaning and context available. Feature generation for semantical analysis is even more impactful, allowing tasks to better analyze the interactions among words and sentences, while generating complicated vector embeddings to represent meaning. These features can inform both heuristics and model training initiatives in a way that extends beyond traditional rule-based methods. With the inclusion of other techniques such as abstractive and extractive summarization, question answering, cosine similarity, and retrieval-augmented generation, include more context and information before generating responses is possible. These modern methods allow for the analysis of highly complicated quality attribute criteria, making semantical analysis an important method for requirement quality automation.

Pragmatic Analysis

As a closely related concept to semantical analysis, pragmatic analysis helps to understand text beyond the immediate meaning of text; specifically, pragmatic analysis looks at the true meaning and intention behind words. When considering pragmatic analysis, the scope of analysis shifts to the intended and likely meaning of textual input. Pragmatic analysis is useful in determining the relevance of quality violations, allowing for a practical analysis of quality violations in relation to the context and domain of requirements and requirement documents. Because requirement documentation often is composed of unstructured language, deriving meaning and intent of text is critical for generating accurate feedback when analyzing quality violations.

As an example of how pragmatic analysis could be useful for requirements automation, a simple analysis of individual requirements typically needs to include references to different document text. For instance, references to other standards, terms, agents, and external systems need to be known for pragmatic analysis. With a purely semantical analysis, the observation of document-specific terminology or external resources may incorrectly flag quality violations. Pragmatic analysis can mitigate these issues by understanding the document context and related external information. Ultimately, requirements that may seem to violate certain quality attributes through semantical analysis may not have violations within the context of pragmatic analysis. Table 6 shows examples of components utilized to achieve pragmatic analysis (Jurafsky et al., 2000; Nugues, 2024; Plag et al., 2007; Zhao et al., 2021).

Table 6 Example Pragmatic Analysis Components

Pragmatic Analysis Component	**Explanation**

Embeddings	Embeddings are useful in capturing the context of text, and can help with pragmatic analysis through context analysis, cosine similarity, and as input to RAG systems.
Feature Engineering	Pragmatic analysis requires extraction of key features that are better indicative of quality violations; pragmatic-based features can include embeddings to help create distinct semantical features that are pragmatically generalizable. Feature engineering can also including document or text statistics to help better classify requirements.
Retrieval-Augmented Generation	RAG methods help to retrieve background information related to text. This method can help trained models achieve better inference based on relevant supporting text and documents. RAG is also helpful in determining complicated quality violations including completeness, feasibility, and ambiguity.
Abstractive Summarization	Used to create an abstract summary of text. Helps to condense text while also providing context for pragmatic analysis. Abstractive summarization is also useful in generating summaries that do not simply extract important phrases or words from the text. Ultimately, this type of summarization can serve as feature input for model training and context generation.
Extractive Summarization	Alternative type of summarization that extracts out the most important text to generate a summary. This type of

	summarization helps to identify the most important parts of the document while also serving as input for pragmatic analysis.
Prompt Engineering	Prompt engineering is a way to provide context, instructions, and examples to the model in order to achieve pragmatic analysis. The use of natural language can help the model understand the evaluation lens to utilize when analyzing text. Prompt engineering also allows for the addition of context and information related to the text to allow for better pragmatic analysis.
Large Language Models (LLMs)	LLMs are large models trained on a wide variety of data. LLMs can be helpful in analyzing text beyond semantical analysis by utilizing a large amount of knowledge to make better inferences based on textual input. LLMs utilize transformers and attention mechanisms to learn what parts of the text to pay attention to for each textual token. Recent LLMs allow for large context windows, allowing for the analysis of large amounts of text, enhancing the capabilities of pragmatic analysis.

Pragmatic analysis is a crucial component for requirement analysis automation. Primarily, pragmatic analysis allows for the understanding of text beyond the literal interpretation created through lexical, syntactical, and semantical relationships. This type of analysis has a variety of uses that extend beyond semantical analysis, including determining the usefulness of a quality violation and making more accurate quality analysis judgements. Through the use of methods such as RAG, Prompt Engineering, Summarization, Feature Engineering, and

Embeddings, pragmatic analysis can achieve better classification of violations and analysis of complicated quality attributes. Although some of the methods are difficult to implement, modern LLMs and transformers utilize methods that successfully promote the use of pragmatic analysis, making pragmatic analysis easier to implement in automated systems.

Combining Different Analysis Techniques

Different types of analysis techniques require different methods. All the analysis techniques can be useful depending on the situation and systematic need for automation. For lexical analysis, lists and heuristics are useful for quick reference and inference of quality violations. Although pre-defined lists of common, problematic words may not always prove useful when determining quality, lists are often useful for quick determination of requirement quality violations. By utilizing textual statistics, analyzing the occurrence of vague words and generalization measurement is possible when considering the entire document. Additionally, modern NLP methods have enhanced the generalizability of lexical analysis. By utilizing techniques such as cosine similarity and embeddings, making determinations on unseen text becomes significantly easier and more reliable. Further, methods such as Parts-of-Speech tagging help to formulate better heuristics that extend beyond simple string matching. Lexical analysis is a powerful tool when combined with NLP-based methods; other methods, such as syntactical and semantical analysis, extend automation capabilities even further.

As an alternative to lexical analysis, syntactical analysis provides a way to analyze the structure of text, creating a more complicated requirement evaluation. The heuristics associated with syntactical analysis pertain to the broader structure of the text. These heuristics are often created through the use of methods such as POS tagging, NER, and sentence statistics. Ultimately, heuristics can match expected patterns of word types and relationships, allowing for the creation of patterns that are generally indicative of quality violations. Although syntactical analysis is powerful, it does not take into account context within sentences. Regardless, the utilization of modern NLP-based methods helps to improve the usefulness and practicality of syntactical analysis. In conjunction with lexical heuristics, syntactical analysis can provide a critical analysis of quality for most quality attributes, while also minimizing the complexity of computation and model training.

As a deeper type of analysis, semantical analysis tries to understand the meaning of sentences beyond the syntactical and lexical formulations of words. Semantical analysis is a comprehensive approach that is useful when determining complicated quality attribute violations. Through semantical analysis, understanding the context of text becomes possible, allowing for the more accurate labeling of agents, actions, and constraints. As an example of the impact of semantical analysis on requirement automation, techniques such as NER, SRL, and POS tagging utilize semantical-based analysis to provide a better understanding and mapping to heuristics. Ultimately, the point of semantical analysis is to analyze the text holistically to determine literal meaning, providing a better understanding of quality violations through the formed understanding of context. Ultimately, semantical analysis looks at the actual relationships among words and is able to provide inference beyond lexical and syntactical methods. Additionally, embeddings provide a way to semantically understand words, sentences, and broader textual elements by representing context in a vector-based format. These embeddings can be used to determine quality violations through cosine similarity, model training, or feature engineering.

As the final form of analysis, pragmatic analysis looks at the actual meaning of text beyond semantics. Pragmatic analysis is a key component to requirement analysis, allowing for the understanding of practical and useful meanings associated with words and phrases. This type of analysis focuses on likely, intended meaning, creating a pivotal type of analysis for analyzing requirements in unstructured text. By deviating from literal meaning, pragmatic quality analysis can create useful feedback on quality violations. As methods to accomplish pragmatic analysis, embeddings, summaries, and RAG embeddings can help create more useful inferences by extending understanding beyond the immediate text scope. By providing context and supporting information, pragmatic analysis can look beyond literal meaning during automation. As an example, pragmatic analysis would likely understand document-specific notation, external rephrases, and unique words and phrases specific to requirement documents. Pragmatic analysis would also understand the practicality of requirements and whether the text is adequately conforming to the various quality attributes. Ultimately, this type of analysis helps to understand requirements based on context and supporting information, leading to a higher quality inference capability when compared to semantical, syntactical, and lexical analysis (Chantree et al., 2006; Landhaußer et al., 2015; Nugues, 2024; Plag et al., 2007).

Requirement Quality Criteria for Individual Requirements

Ultimately, requirement documents should be structured in a way that outlines both individual requirements and groups of requirements. Requirement groups typically define a broader feature, while individual requirements are the granular details that implement the feature defined by the requirement group. The IEEE 29148:2018 standard provides ideal characteristics of both entities, outlining quality attributes that all requirement text should ultimately adhere to. These characteristics help to improve the quality of the overall requirements document by eliminating requirements that negatively impact the overall quality of the document, including requirements that are ambiguous, unfeasible, or incomplete (“ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering,” 2018).

When considering quality violations outlined by the IEEE 29148:2018 standard, analyzing the practical importance of violations is pivotal. The standard specifies quality attributes that are relevant to requirement text, but the interpretations of high-quality requirements can be difficult. Understanding the relative impact of quality violations on the document's readability or the system's practicality is difficult; although a requirement may technically violate a quality attribute, the impact to the overall quality and understanding of the document or the requirement could be minimal depending on context and shared understanding. Additionally, certain quality attributes are easier to measure due to the limited scope of analysis. Other quality attributes require immense knowledge about the domain, document structure, and project practicality to properly understand whether a violation exists. Some requirements are measurable through simple heuristics, while others require more complicated methods for validation. Consequently, methods to determine quality violations differ based on the associated simplicity and scope provided by the quality standards.

Ambiguity

According to the IEEE 29148:2018 standard, requirements should generally be unambiguous. This means that the requirement should be easy to understand, while also only having one interpretation. To account for what constitutes an ambiguous requirement, the standard provides some examples. Vague terms, superlatives, and subjective language all affect the interpretability and ambiguity of the requirement. Terms such as 'best,' 'worst,' 'it,' 'this,' and 'that' could all lead to an ambiguous interpretation of the components within a requirement. The standard also suggests that certain phrases such as 'higher quality,' 'better than,' and 'as appropriate' could lead to ambiguous interpretation for requirements. Naturally, there are lists of words and phrases that generally constitute quality violations for ambiguity; many past automated systems have defined these words and lists to provide feedback on requirement improvement and readability issues ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Through the utilization of different types of grammatical analyses, an automated system can verify requirements for ambiguity at different levels of granularity. Some basic implementations of automation tools tend to utilize rule-based methods or heuristics to find ambiguity violations, while other systems utilize more complicated pattern analysis, NLP-based techniques, and Artificial Intelligence. Ultimately, analyzing ambiguity becomes a challenge when trying to determine the relative impact of potential violations. For instance, finding ambiguities that affect shared understanding can be difficult. Existing research uses syntactical heuristics to identify nocuous ambiguities, which are violations likely to negatively impact understanding. Their analysis primarily focuses on finding patterns within syntactical heuristics. Although their work provides a meaningful direction for understanding useful quality violations, identifying violations, such as ambiguity, is still a difficult task (Chantree et al., 2006; Zhao et al., 2021).

Because analyzing ambiguity can be complicated, existing works have utilized simple rule-based and heuristic-based systems to identify ambiguity violations with moderate success. Research has shown the implementation of different techniques to measure ambiguity; as a popular quality attribute, much existing work primarily focuses on ambiguity within requirements. As an example, existing research has investigated which textual indicators typical determine ambiguity; their feature set included heuristics such as number of word phrases and number of optional phrases. Other research looks at the utilization of formal methods to create a structure of requirements and dependencies, allowing for the use of formal language to analyze semantical ambiguity. Lastly, a survey of ambiguity-based analysis implementations shows that there are a variety of different ambiguities that are measurable through automation (Carlson & Laplante, 2014; Cavada et al., 2009; Gupta & Siddiqui, 2019).

Critically, these different types of ambiguity include previously discussed levels of analysis, including lexical, syntactical, and semantical analysis for ambiguity. The study also defines additional types of ambiguity beyond commonly used categories, all provided within Table 7. The study ultimately talks about requirement-specific ambiguities, which can comprise or aggregate the various different types of common grammatical ambiguities. These ambiguities are specific to requirements, and make use of common ambiguities that manifest as issues throughout requirement documents (Gupta & Siddiqui, 2019).

Table 7 Types of Ambiguity

Ambiguity Type	**Explanation**
Lexical Ambiguity	An ambiguity where there are multiple interpretations to a word.
Syntactic Ambiguity	When the structure of words can cause multiple interpretations. Although a requirement may be syntactically correct, the syntax could potentially lead to multiple meanings and interpretations.
Weighted Syntactic Ambiguity	A way to understand syntactic ambiguity based on its most likely reading or textual statistics. Helps mitigate misclassification of ambiguity through usage statistics.
Semantic Ambiguity	When the literal meaning of a text can be interpreted in multiple ways, leading to ambiguity in interpretation.
Weighted Semantic Ambiguity	Utilization of textual statistics or frequency-based methods to determine the most likely semantical meaning of a text. Helps to

	uncover useful ambiguities at the semantical level.
Pragmatic Ambiguity	When the actual meaning of the sentence is not easily understood. Includes interpretations beyond the literal interpretation of a text. Pragmatics helps to uncover actual meaning instead of literal meaning, often creating a problem of multiple interpretations.
Vagueness Ambiguity	It occurs when words, phrases, user-specific terms, or other document-specific items are syntactically unambiguous, but still have multiple interpretations, leading to a vague understanding of the requirement.
Generality Ambiguity	When terms can refer to multiple entities. Lack of definition leads to multiple interpretations of meaning, including agents, actions, and restrictions.
Genuine Ambiguity	It occurs when multiple interpretations of the text are correct, leading to ambiguous understanding. This type of ambiguity requires more context or clarification to further parse the understanding of the text.
Polysemy Ambiguity	It occurs when there are multiple, similar interpretations of textual units, leading to more complicated ambiguity. This type of ambiguity makes it difficult to utilize statistics to derive the most likely meaning, creating the need for a more complicated analysis.
Nocuous Ambiguity	It occurs when a text can convey multiple different meanings to different readers. Addressing nocuous ambiguity either means formulating text for the intended reader, or

	making sure that text is interpreted the same among different readers.
Anaphoric Ambiguity	It occurs when pronouns or referring words can point to multiple entities, creating ambiguous interpretation.
Incompleteness Ambiguity	It occurs when there are no structural ambiguities, but the information provided does not adequately convey the intended meaning of the text.
Referential Ambiguity	It occurs when the utilization of references within a text causes ambiguities for interpretation, such as pronouns potentially referring to multiple agents.
Scope Ambiguity	It occurs when the scope does not adequately define references within the text, leading to a misunderstanding of user or action references within requirements.

Given the multiple different types of ambiguities specified, the author also aggregates common ambiguities into requirements engineering-specific ambiguities. These RE-specific ambiguities are more practical by nature, allowing for the interpretation of ambiguities through a typical requirement understanding. Primarily, these Requirements Engineering (RE)--specific ambiguities work to combine or modify existing types of ambiguities to create useful interpretations of ambiguity within requirement documents. Some of these ambiguities, like Genuine RE Specific ambiguity, measure if there are multiple interpretations of requirements within the context of the requirement document. These ambiguities can also cover conflicting requirements, application-domain ambiguity, system-domain ambiguity, and development-domain ambiguity (Gupta & Siddiqui, 2019).

As mentioned previously, RE-specific ambiguity violations require an aggregation and practical analysis of individual ambiguity violations. Figure 1 illustrates the hierarchy of ambiguity violation categories. By utilizing these different kinds of ambiguity metrics, formulating broader determinations of ambiguity is possible through heuristics, feature engineering, or ML-based implementations. Primarily, these aggregations help to create better generalizations of requirements for quality attribute categories, including ambiguity (Gupta & Siddiqui, 2019).

Additional challenges associated with ambiguity include the utilization of context when determining individual ambiguity violations. The practical RE-specific ambiguity categories require domain understanding from the different levels of development relevant to the system. Ultimately, rule-based heuristics do not take context into account during inference. Many of the different types of ambiguity outlined previously require more information that is unavailable from the context alone, limiting the capability of existing systems. The most impactful types of ambiguity, require domain knowledge and higher-level analysis of requirements to make accurate determinations of ambiguity violations. Therefore, some of the techniques discussed previously for advanced NLP and ML-based processing are required to accomplish accurate automation of ambiguity detection (Chantree et al., 2006; Fabbrini et al., 2002; Kummler, 2021; Zhao et al., 2021).

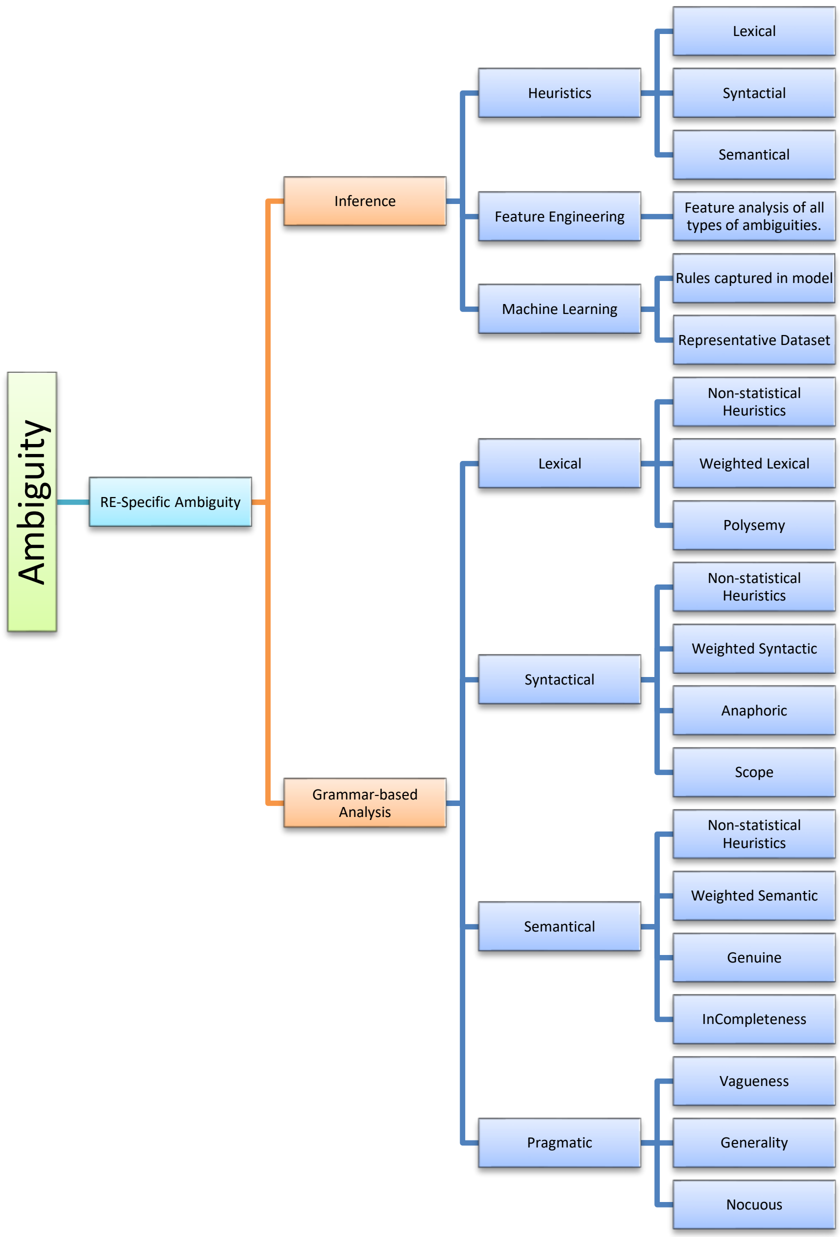

Figure 1 Hierarchy of Ambiguity

Even though ambiguity is difficult to measure, existing tools and research have implemented ways to measure likely violations. Many of these implementations, although complex, rely on rule-based and heuristic methods to provide quality determinations. Because of the wide variety of different ambiguities, measuring true ambiguity is difficult. Although utilizing textual indicators to broadly categorize RE-specific and pragmatic ambiguities is possible, these rule-based systems fail to apply domain-specific knowledge during the analysis process due to lack of domain and document context. Because of the failure of older systems to consider domain-specific knowledge and context, enhancements to analyze ambiguity could include the previously specified functions used for pragmatic and semantical analysis, such as abstract summarization, embeddings, AI and RAG systems. Ultimately, textual heuristics and rule-based systems can still be impactful when measuring ambiguity; further, heuristics and feature engineering can help uncover complicated and general patterns useful for classification. With the combination of modern methods such as heuristics, feature engineering, and complex NLP methods, providing adequate analysis of ambiguity is possible. Still, existing research fails to adequately address the topic of ambiguity detection outside of basic heuristics alone (Carlson & Laplante, 2014; Naeem et al., 2019; Zhao et al., 2021)

Conforming

When generating a requirement document, certain requirement syntaxes are commonly used to maintain compliance and readability. Although the IEEE 29148 standard suggests that requirements should be structured in a common format, the unstructured nature of requirement documents creates instances where requirement syntax tends to deviate from typical requirement templates. When analyzing conformance, an automated system would ideally look for conformance to standard templates and styles that are applicable across typical requirement notation and standards. Alternatively, an automated system could look for user-defined requirement patterns if the requirements in the document do not conform to a standard template, allowing for dynamic formulation of requirement templates. By providing the ability to look at or derive custom patterns, automated tools can provide more useful violation feedback about requirement conformance that can ultimately deviate from accepted standards ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Lami et al., 2019; Zhao et al., 2021). When analyzing a requirement document, conformance is a key quality attribute for many automated requirement tasks. To analyze most of the other quality attributes, making sure that requirements conform to a standard template can help simplify the automation and analysis processes. For requirement identification, requirement template conformance is pivotal for identifying all requirement-related text within the document. Ultimately, requirement template conformance allows for the use of automated methods as a dependent task for analyzing quality violations, allowing for the retrieval of all requirement-related text before quality analysis. In addition to requirement identification, requirement conformance helps with the automation of indexing requirement hierarchy. As an example, conforming to a certain labeling standard for requirement templates helps to determine the dependency and relationship among both requirements and requirement groups. Through the understanding of the requirement hierarchy, providing a holistic analysis of requirement quality relative to the requirement's associated position within the document hierarchy is ultimately possible. This type of analysis through requirement conformance can help to derive many important features for further quality analysis, including quality attributes that require contextual information about other requirement-related text. Figure 2 and Table 8 describe the dependency and impact of conformance with requirement automation, showing a pivotal relationship among both scopes of concern.

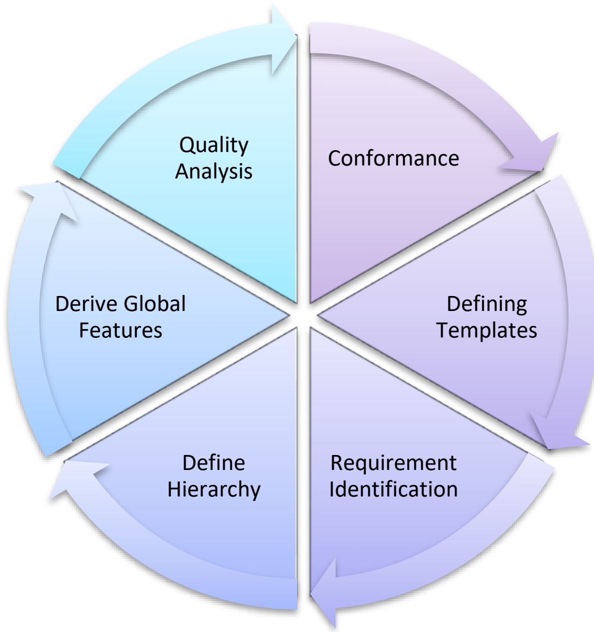

Figure 2 Conformance Impact for Automation

Table 8 Conformance-Related Tasks

Conformance Task	**Explanation**
Requirement Identification	Utilizing the expected templates, words, or phrases commonly associated with requirements to extract out all relevant requirement-related text in a document.
Defining Hierarchy	Using the process of requirement identification to understand the hierarchy and relationships among requirements. Useful as an input feature to other quality attributes where lower-level requirements face more scrutiny for quality adherence.
Global Features	Using the conformance of requirements to derive certain features including total number of requirements, levels and hierarchy of

	requirements, and common words or phrases within the requirement document.
Quality Analysis	Utilizing the global features derived from the requirement identification process to inform further quality analysis. For instance, analyzing common words and phrases could be useful when determining the relevance of quality attribute violations. The global features also help to better understand the project scope and context, which can improve the analysis of more complicated quality attributes.

Singularity

When considering requirement quality, specifying a single functionality is pivotal for simplicity and readability of requirement documents. According to the IEEE 29148:2018 standard, when defining requirements, the requirement should only define one capability, characteristic, constraint, or quality factor. Accordingly, requirements should be broken down into highly granular units in which each of these elements should each constitute their own requirement to fulfill conformance to the singularity quality attribute. When considering singularity, understanding requirement structure and conformance is important for adequate identification of potential violations. Ultimately, determining the capabilities, characteristics, constraints, and quality factors that belong to the requirement can help with the application of rules and heuristics to determine singularity quality violations ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

When considering how to analyze requirement singularity, breaking up the requirement into logical parts becomes an important factor for automation. Depending on if the requirement follows typical conformance patterns, heuristics and NLP methods become useful tools to help delineate the different parts of a requirement, including agent, action, and constraints. Requirements can be further broken down through the utilization of Question Answering techniques or Semantic Role Labeling. With the aggregation of grammatical and textual information, analyzing singularity adherence becomes more practical. Ultimately, the analysis of singularity can utilize lexical, syntactical, and semantical patterns during automation. Certain words and phrases can imply multiple actions, while semantics can help uncover implied and dependent actions of requirements, uncovering potential violations.

Like other quality attributes, pragmatic analysis can help improve the accuracy and usefulness of singularity quality attribute analysis. Ultimately, the pragmatic analysis of singularity should consider whether multiple units in a requirement are significant enough to be separated and whether any associated conditions should be moved into another requirement. Additionally, analyzing the pragmatic granularity of actions is important during the analysis process, further complicating the particular scope and practicality of useful analysis. Because of the complexity associated with pragmatic analysis for singularity, rules and heuristics could likely provide helpful feedback for potential violations through the detection of multiple verb phrases or conjunctions.

When considering the automation of singularity, understanding the importance of hierarchy within requirements can help better formulate quality attribute violations. For instance, a top-level requirement will likely contain broad specified functionalities which, under a normal analysis, could create a quality violation under a singularity analysis. When analyzing singularity, utilizing the level of the requirement in association to the document hierarchy is pivotal for accurate analysis, like other quality attributes. Ultimately, maintaining a practical balance of likely requirement reading, associated hierarchy, and the severity of the violation is crucial when analyzing singularity.

Verifiable

In addition to the previously mentioned quality attributes, verifiability is an important and key characteristic of requirements. Often, requirements take on lofty goals, unverifiable criteria, and unachievable functions, hurting their practicality for implementation. To mitigate problems with verifiability among requirements, fully defining and generating highly detailed requirements can help limit scope confusion. Like singularity, capabilities, characteristics, constraints, and quality factors should all specify measurable criteria that can be verified upon completion of a given system. In addition to granularity's impact on verifiability, traceability to test cases can also help improve verifiability among requirements; in general, documentation and explanatory text can help provide further context for the scope of requirements, helping to further specify the scope of requirements ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

To achieve verifiability, automation can help with determining the quality of requirements and associated test cases for measuring verifiability. NLP-based methods through previously mentioned implementations could help make determinations of verifiability. For instance, superlatives, subjective language, vague pronouns, and ambiguous terms make requirements difficult to verify. All of these categories are measurable through previously mentioned NLP-based methods, allowing for the formulation of heuristics to help automatically determine verifiability issues. Figure 3 shows the methods for achieving automated verifiability, including methods for testing and AI-based automation ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Critically, verifiability does typically require complicated analysis when trying to enhance accuracy of verifiability determinations. However, heuristics and rule-based systems can achieve a reasonable impact when analyzing verifiability. NLP-based methods can help uncover verifiability problems for lexical, syntactical, and semantical components through context, problematic words and phrases, and other types of NLP-based heuristics, including analysis of superlatives and subjective language. Often, these rules do not require additional context to help determine quality violations, making verifiability a useful quality attribute to analyze for automated systems.

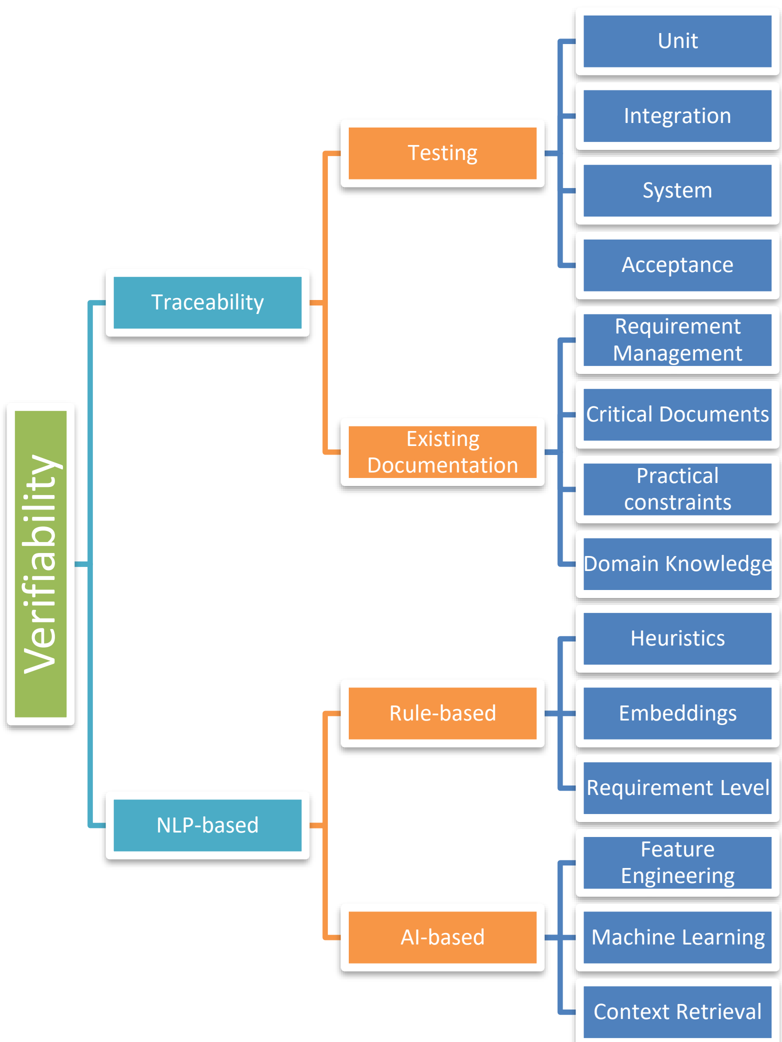

Figure 3 Components for Verifiability

Feasibility

Feasibility is defined as a requirement that is practical to implement given constraints such as cost, budget, technical, and deadlines related to the project. Naturally, feasibility includes determining scope creep, ambiguity, and verifiability of a requirement. Like previous quality attributes, previous rule-based and heuristic methods are available to help determine feasibility violations. However, rule-based methods may not necessarily prove useful when analyzing the entirety of the requirement in the context of its domain and the existing business constraints. To analyze feasibility at a surface level, textual statistics and other common textual categories such as ambiguity, superlatives, and ambiguous terms can help when automating violation detection for feasibility ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

For rule-based methods, certain features are potentially useful for determining relevant violations for feasibility. For instance, the level of the requirement makes a difference, as higher-level requirements tend to be general and all-encompassing of multiple, different features. Looking at the most granular requirements can help determine if a violation hurts the verifiability of the system requirements. Additional features such as linguistical indicators like superlatives, open-ended and non-verifiable terms, and terms that imply totality can also help to enhance the accuracy of inference for feasibility. At the surface level, analyzing feasibility is a practical task for automation systems, allowing for rule-based methods and heuristics to accurately determine feasibility of requirements. However, modern technology allows for a more complicated method of analysis for feasibility beyond heuristics ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

By utilizing the tacit knowledge of LLMs, analyzing feasibility beyond heuristics is possible through automation. Modern NLP and AI -based methods allow for the infusion of knowledge and context with model inference, allowing for a broader determination of quality violations. In particular, LLMs provide knowledge outside of the text that can help in determining domain practicality with the model's associated knowledge. Figure 4 and Figure 5 shows the process for AI-based inference and the associated features necessary to classify feasibility violations. Primarily, AI-based inference can determine the practicality of a requirement through both the tacit knowledge that LLMs contain, acquired context through summarization, and RAG-based methods that help to retrieve relevant documents pertaining to both the requirement and the requirement document. Additionally, utilizing prompt engineering and fine-tuning to guide the model's output through added context and quality attribute violation expectations is possible as well. Ultimately, methods such as prompt engineering can provide the ability to consider other information critical to automated analysis, including budgetary restrictions and timelines.

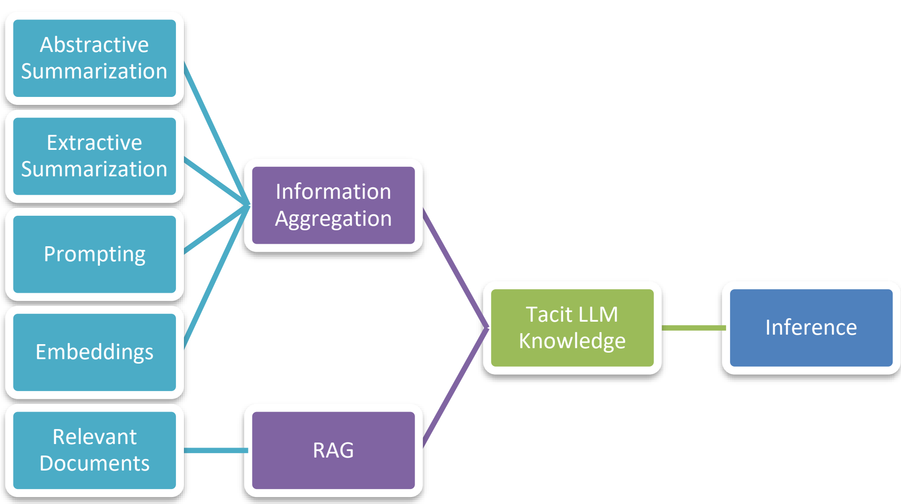

Figure 4 AI-based Inference for Feasibility

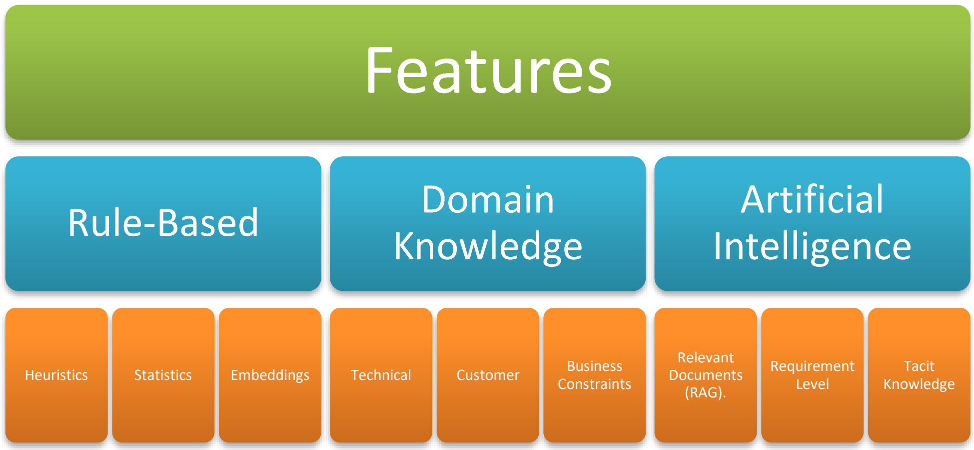

Figure 5 Features for Feasibility

Alternative Quality Attribute Criteria

Outside of the previous quality attribute criteria previously discussed, there are additional quality attributes that requirements should adhere to. Many of these quality criteria are closely related to the previously mentioned quality attributes and have close relationships among each other. Similarly, measuring violations associated with these additional quality criteria is possible through similar methods defined for both rule-based and AI-based automation systems. When considering these individual quality criteria, the IEEE 29148 standard provides a distinction across both individual and requirement group-based quality analysis. For individual requirements, the analysis is on the relationship between the individual requirement and the associated document. Individual requirement analysis measures attributes with regard to only that specific requirement. Group-based requirement analysis looks at the entire unit of requirements that comprise the group, making the analysis more difficult to measure and automate ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

In addition to the quality attributes already discussed, the IEEE 29148 standard specifies additional quality attributes for individual requirements. Of these quality attributes, the standard defines necessity, appropriateness, completeness, and correctness of requirements as important quality attributes for requirement quality. These quality attributes are also important to the overall quality of requirement documents, making their analysis even more important. Table 9 discusses the remaining individual quality attributes in detail. These attributes work to help define the scope, appropriateness, and contributions of individual requirements in relation to the actual needs of the system. Naturally, these requirements need complicated analysis implementations to be successfully automated. Although heuristics and rule-based methods can help determine potential violations, they are likely less impactful to these quality attributes due to the complicated and large scope of their analysis ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Table 9 Remaining Quality Attributes

Quality Attribute	**Definition**
Necessary	Whether the requirement meets an essential need for the system based on existing knowledge of intended user functionality. Without a necessary requirement, a deficiency would exist within the system.
Appropriate	Whether the requirement adequately expresses the functionality, characteristics, and constraints appropriately without being overly broad or overly restrictive. This helps to allow for flexibility with implementation during the evolution of the project.
Complete	Refers to whether the requirement adequately implements the intended feature without

	needing any further clarification or details about the feature. Specifies the complete functionality needed during implementation.
Correct	The requirement correctly describes the need in which it is trying to implement. Involves domain understanding, context, and communication with stakeholders to accurately implement features.

Conclusion and Practical Uses

When analyzing requirements, the IEEE 29148 standard provides guidelines that individual requirements should follow to enhance quality. These guidelines define quality attributes, a term used to describe basic quality criteria that requirements should ultimately follow within requirement documentation. The goal is to utilize automation through rule-based, NLP-based, or AI-based methods to make determinations of when there are quality attribute violations associated with the standard criteria. To accomplish automated analysis, a system can use different levels of grammatical analysis to determine if there are quality attribute violations throughout requirement text.

For instance, lexical analysis typically looks at common terms specified through lists of words that may indicate quality violations. Words and phrases that typically indicate quality violations could serve as quality violation indicators for lexical analysis. Further, utilizing NLP-based methods such as stemming, lemmatization, and embeddings to help further understand meaning related to words and phrases is possible, helping to improve the generalizability of lexical analysis. Furthermore, these methods could help with the derivation of lists and heuristics that could help systems perform broad automated analysis of requirements beyond pre-defined lists.

Syntactical analysis serves as a higher step of analysis, while also utilizing similar methods for lexical analysis. With syntactical analysis, automated systems look for violations through the structure and rules of the text. For this type of analysis, the utilization of Part-of-Speech tagging, Named Entity Recognition, and other NLP-based methods help to perform a holistic analysis of the text beyond pre-defined words and phrases. Not only can syntactical analysis help validate lexical violations, but it can also help with creating better, more generic heuristics that are largely applicable to requirements. These heuristics can be rules that specify typical sentence structure indicators of quality violations, types of words, generic phrases, and textual statistics that broadly apply to requirement violations.

Alternatively, semantic and pragmatic analysis provide more complicated methods to understand meaning within text and are useful when trying to perform higher-order quality attribute analysis. When used in analysis, semantic and pragmatic analysis can help both understand literal and intended meaning of text, providing analytical capabilities beyond lexical and syntactical-based approaches. Through the use of Semantic Role Labeling, Embeddings, Retrieval-Augmented Generation, and LLMs, an automated analysis can infuse context and knowledge in with the textual requirement to formulate better responses and inferences, allowing for the analysis of more complicated quality attributes such as completeness and feasibility.

When analyzing quality attributes, all of these methods of analysis can help to formulate better requirements, regardless of the analytical scope and complexity. Additionally, many of the quality attributes are measurable through simplistic rule-based and heuristics-based methods. Through the use of heuristics and textual statistics, an automated system can create aggregated assertations about the document’s overall quality. However, the utilization of NLP and AI-based methods allow for a better generalization of requirement quality violations that extend beyond lists of words and phrases. When trying to analyze quality attributes at a semantic or pragmatic level, utilizing context, summarization, and model knowledge can help generate better inferences, especially when an automated system is trying to determine the practical reading of a requirement. Further, simple implementations that can formulate better heuristics are also possible, allowing for complicated analysis with less computation. For automated quality analysis, the focus has primarily been on analyzing individual requirements. Notably, many of these quality attributes require the understanding of document context to perform an accurate analysis, suggesting that requirement groups could benefit from some of the same methods utilized by individual requirements ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Requirement Quality Criteria for Requirement groups

Input Features for Requirement groups

As discussed previously, quality attributes can also help to measure and improve groups of requirements. Even though requirement groups common quality attributes with individual requirements, the types of analysis are significantly different. When analyzing groups of requirements, for instance, the analysis should include the impact of each individual requirement or requirement group under it. Therefore, there are a variety of different considerations for the entities under a requirement group when trying to analyze quality conformance. As an example, Figure 7 shows a typical hierarchy within a requirement document in relation to individual requirements and requirement groups. As Figure 8 illustrates, requirement groups can have an associated document position, child requirement groups, and child requirements. Additionally, requirements typically serve as the terminal item within the hierarchical structure. When evaluating requirement groups, the input during analysis must include the position of the group and an analysis of all its children. For instance, a requirement group may specify a broad feature that is further specified among lower-level requirement groups and requirements. The aggregation of this information is necessary to decide on the quality adherence of the requirement group ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

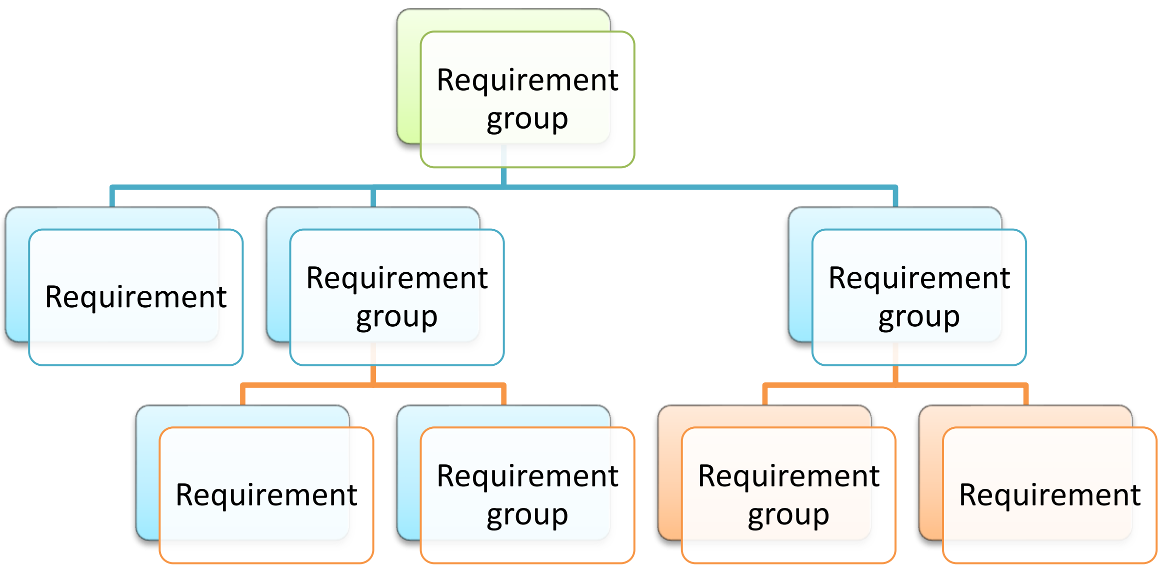

Figure 6 Requirement Group Hierarchy

When considering the complexity of analysis for requirement groups, an analytical tool should consider the implications of the input features needed for inference. Ultimately, analyzing requirement groups requires the deduction of document position, context, and the associated quality of all child requirements and requirement groups. This means that an automatic system would need to derive the structure of all requirements and groups within the document. As an added challenge, the analysis of requirement groups is both pragmatic and recursive, meaning that analyzing requirement groups requires extensive domain knowledge, external knowledge to measure quality attributes like feasibility, and knowledge of all child requirements and groups. Accordingly, an analytical system would need to utilize a custom formula for weighing the overall quality of a requirement group based on its input features. Figure 8 shows a mapping of the necessary features used to make a quality determination for requirement groups. As the figure demonstrates, the process of determining requirement group quality depends on the knowledge of hierarchical information within the document.

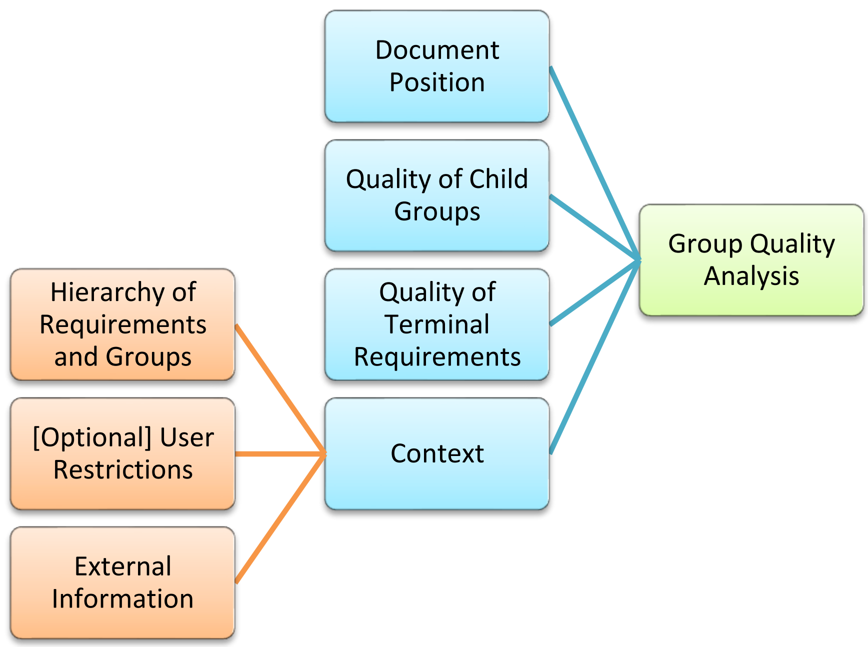

Figure 7 Requirement Group Features

Quality Attributes for Requirement groups

For requirement groups, the IEEE 29148 standard defines quality attributes that all requirement groups should adhere to. Ultimately, many of the quality attributes related to requirement groups require complicated features previously discussed. When considering how to analyze requirement groups based on certain quality criteria, utilizing contextual information about the document and the associated hierarchy is pivotal. In the standard, the quality attributes for requirement groups, as defined in Table 10, include *completeness, consistency, feasibility, comprehensibility, and ability to be validated*. Notice, a few of the quality attributes mimic the individual quality attribute definitions. However, they are still different due to the scope of the analysis and the complexity of the input features ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Table 10 Group Quality Attributes

Quality Attribute	**Explanation**
Completeness	It refers to whether the requirement set adequately implements the intended feature without needing any further clarification or details about the feature. Critically, the child requirements and groups should all be complete, while the current requirement or group should specify appropriate functionality relative to its document position.
Consistency	The set of requirements do not conflict. In essence, there is no duplication of functionality or concepts within the child requirements or groups. Additionally, the functionality for the current requirement or group should not duplicate other functionality at the same level within the hierarchy.
Feasible	The requirement group is feasible to implement when considering the child requirements and requirement groups based on schedule, cost, technical, and practical limitations.

Comprehensible	The requirement group clearly outlines its intended purpose and relevance within the system. All child requirements and groups should clearly and distinctively map back to the requirement group.
Able to be validated	The requirement group is testable and addresses the totality of the user need upon completion. The needs of the user are met based on the requirement group definition and its children.

An Automatic Process for Analyzing Requirement Groups

Practically, analyzing requirement groups is difficult due to the nature of the information needed to perform automated analysis. Through the utilization of modern methods, including the methods specified previously to perform semantic and pragmatic analysis, creating an automated analysis is possible depending on the utilized methods. For instance, Figure 8 shows a basic process that could potentially analyze requirement groups. Notice, there are severe upfront costs for deriving all the various features and knowledge needed to make quality determinations for requirement groups. To analyze requirement groups, an automated tool would need access to a summarization of the document, relevant documents and context, and an index determination that allows for the extraction of a requirement hierarchy ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018; Naeem et al., 2019; Ruan et al., 2023; Zhao et al., 2021). Alternatively, utilizing custom metrics that limit the required knowledge for quality attribute determinations among quality groups is possible. For instance, instead of utilizing RAG, summarization, and context, an automated system could potentially create heuristics that generalize the likelihood of quality violations for requirement groups without considering other, complicated information. For instance, requirement groups may have indicators of quality violations that do not require a holistic analysis of all requirements, groups, and context. In essence, an automated system could simply perform a similar analysis that is utilized under individual requirements to determine quality violations without analyzing all entities under a requirement group. Although this method of analysis is easier to perform, it is likely limited to only certain quality attributes. Therefore, the automated analysis would utilize the quality attributes specified for single requirements instead of analyzing requirement group-specific attributes. Practically, the same heuristics and rules that apply to single requirements could also apply to requirement groups with perhaps additional processing and considerations. Figure 9 shows how a requirement group analysis might work without recursive analysis.

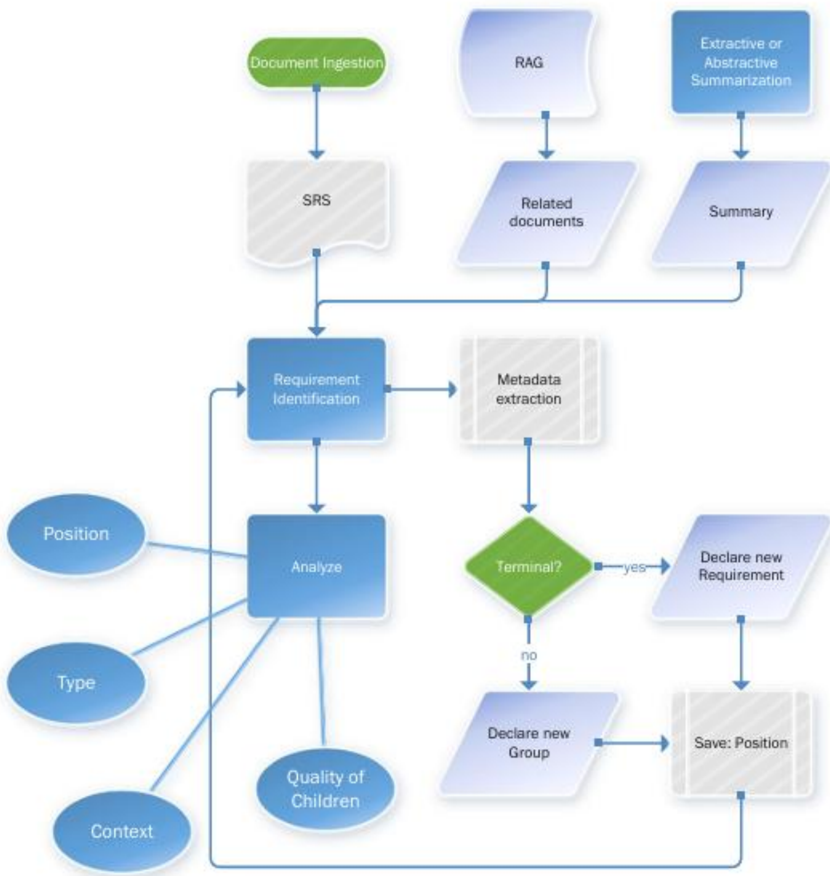

Figure 8 Process for Analyzing Requirement Groups

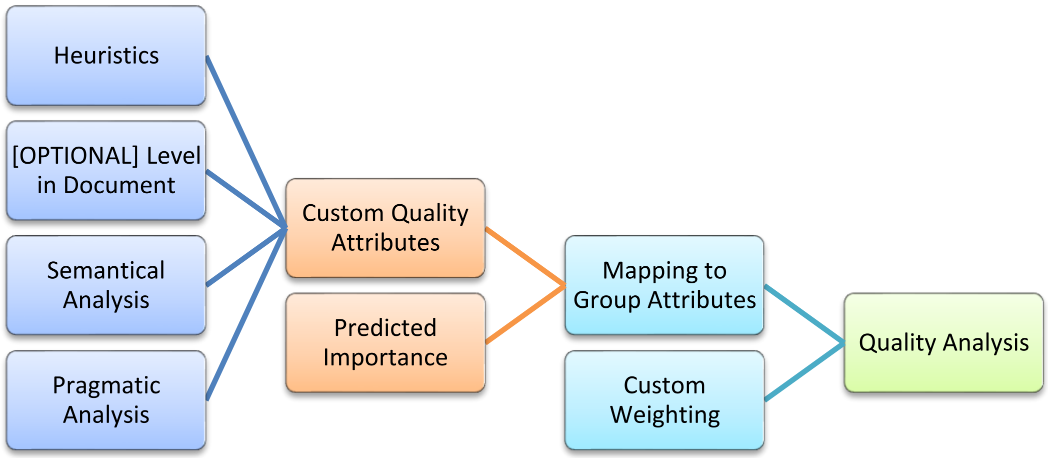

Figure 9 Requirement Group Analysis without Recursive Analysis

The simplification of requirement group quality analysis provides a good opportunity for practical systems. Although the method of quality analysis without recursive data about the document is promising, methods may also be limited due to the loss of contextual data. As a compromise, an analysis could utilize a hybrid approach for analysis by implementing individual requirement heuristics and, ultimately, creating broadly applicable heuristics. For instance, a large amount of unfeasible individual requirements would imply that the requirement group is likely unfeasible. Additionally, quality violations at a lower-level requirement group could indicate problems with the parent requirement. As a way to limit the need for complicated analytical techniques, an automatic system could potentially utilize statistics of group children while applying a custom equation for determining associated quality attributes violations. These aggregated statistics could map to the various different quality attributes associated with requirement groups, eliminating the need for complicated context and document parsing. Figure 10 shows the process to achieve group quality analysis through a simple analysis of individual requirements, eliminating the need for context-based information. Though this implementation would require the indexing of a document, the heuristics only require minimal information to determine the quality of requirement groups. Additionally, the implementation shows that heuristics would map individual quality attributes to group quality attributes. With heuristic mapping, the proposed automatic system would utilize statistics of violations and weighted importance to determine the likelihood of group attribute violations.

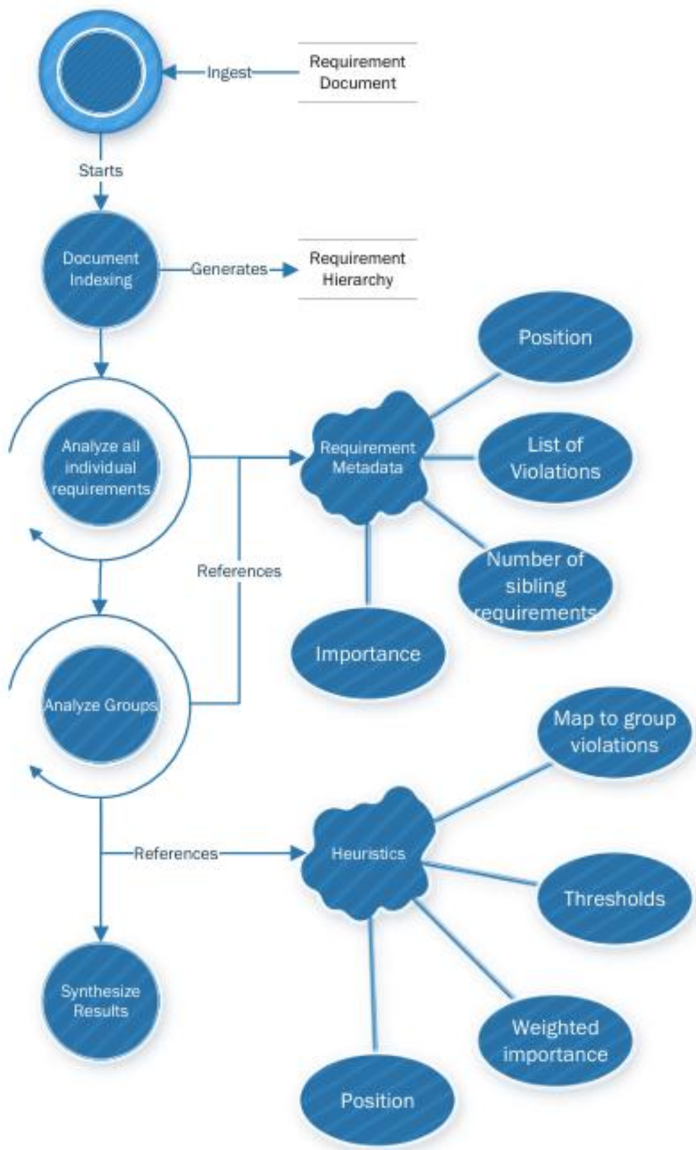

Figure 10 Analyzing Requirement Groups through Heuristics

Conclusion and Practical Uses

When compared to the automatic analysis of single requirements, analyzing requirement groups is considerably more difficult. When considering the features necessary to perform an accurate analysis of requirement groups, an automatic analysis would have to utilize automatic indexing of requirement hierarchy while synthesizing domain-related information. Further, analysis of requirement groups requires the recursive analysis of all child elements, making analysis dependent upon multiple entities within the document, complicating the automation process. Although the automated analysis of requirement groups is difficult, there are ways to mitigate the challenges through the utilization of hierarchical analysis and heuristics.

When considering ways to mitigate the complexity of analysis for requirement groups, an automated system could potentially utilize the impact of individual requirement quality violations or textual indicators within the group's text. Additionally, when discussing the analysis of individual requirements, an automatic system could derive heuristics that map individual requirement violations and thresholds to the quality attributes associated with requirement groups. This method requires an indexing of the hierarchy in the document, but eliminates the need for injection of context and knowledge. Alternatively, an automatic system could simply analyze requirement groups through the individual quality attribute criteria specified for individual requirements. This brute force approach eliminates the need for context and recursive analysis, simplifying automated implementations. Primarily, this method of analysis applies the same quality attributes to all requirement-related text, eliminating the delineation among requirements and requirement groups.

When analyzing requirement documents, an automated system can perform both individual and group-based analysis of requirements. Because most requirement documents exist in a hierarchy, analyzing individual requirements is more practical, as document indexing is difficult and unreliable, especially considering the unstructured nature of requirement documents. Although automated analysis is complicated for requirement groups, modern techniques and heuristics potentially enable the possibility of a system to perform a surface-level analysis of requirement groups.

Conclusion

The IEEE 29148 standard provides a way to analyze requirement documents through the various requirements and requirement groups within a document. Primarily, the standard focuses on the main quality attribute criteria that help to enhance the overall quality of requirement-related text. These criteria, such as ambiguity or singularity, help to provide a template for ideal requirements, providing guidance to eliminate misunderstandings and unfeasible functionality. Because the requirements process is complicated, and requirement-related documents tend to be unstructured, modern requirement analysis tools need to utilize complicated methods to accurately identify requirements and detect their associated quality issues.

Currently, most systems utilize rule-based and heuristic-based methods to identify requirement quality defects. Primarily, these defects pertain to common words or phrases that can often create quality violations within requirement documents. More advanced systems can utilize NLP-based methods to generate generic patterns that apply to unseen requirements, helping to enhance the applicability of heuristics to unseen text. These types of automatic analysis methods typically engage in lexical and syntactical processes to help determine violations. Even though lexical and syntactical indicators along with their associated heuristics are useful when determining quality violations, their scope is limited.

As an alternative to rule-based automation, methods that utilize AI-based or NLP-based solutions can help uncover textual meaning for performing semantic or pragmatic analysis. Additionally, techniques such as summarization, transfer learning, and prompting can help provide models with important context that is useful to evaluate complicated quality attributes. Because many of the quality attributes require domain knowledge and document-specific information, these techniques allow for more accurate analysis of requirement quality by integrating critical knowledge into models and other NLP implementation. Concepts such as RAG and embeddings help to provide high dimensionality and context to a model before making an inference, allowing for an automation that considers contextual and document-related information.

Lastly, automatic analysis for requirement groups is considerably more difficult due to the need for indexing requirement documents. Specifically, automated systems typically need to perform an analysis of the hierarchy and relationships of requirements to understand the various dependencies and levels necessary for analysis. Many of the requirement group quality attributes require a holistic analysis of their children before making a determination of quality. Because of the need for recursive analysis, an automated system would need to utilize domain and document knowledge to create an accurate analysis. Alternatively, an automated system could simply look at the text of the requirement group and analyze quality based on the criteria for individual requirements, mitigating the need for extensive parsing, knowledge, and indexing. With this method, requirement groups are treated as individual requirements instead of groups, mitigating the need for indexing and requirement identification. Lastly, an automated system could utilize a hybrid approach in which the document is indexed with regard to its hierarchy, but the quality of the requirement group is determined off of custom heuristics formulated by its children.

A practical system would utilize and combine all of the different methods specified to perform a holistic requirement analysis. Specifically, an automatic system could utilize rule-based and heuristic methodologies while also utilizing more complicated methods of semantic and pragmatic analysis. Ideally, the system would also accurately identify requirement-related text without indexing due to the added complexity and unreliability of computation and extractive inference. Lastly, an automated system would likely require minimal user input, retrieving the needed context and related documents through techniques like embeddings, RAG, or LLMs. The overall progress and complexity of AI-based methods allows for the practical achievement of most quality attributes on most grammatical levels, including pragmatics. Although complicated methods of analysis exist, simplistic methods of quality violation detection remain useful, especially as feature inputs to broader models. By utilizing and expanding upon the various methods of analysis that exist among current systems and implementations, automated systems can improve inference for requirements based on the IEEE 29148 quality model.

Chapter 3

Literature Review: Requirements Identification and Quality Analysis

To analyze a requirements document, it is necessary to both extract the requirements from the document and apply appropriate algorithms to analyze the associated quality of those requirements. Due to the unstructured nature of requirements documents, it becomes a difficult task to detect and extract out the requirements from within the document. Naturally, many studies have attempted to solve the issue of requirements identification and extraction through different and complicated implementations. Some of the existing implementations, for instance, utilize pre-labeled datasets to train various AI-based algorithms to classify whether a text is related to a requirement. Other implementations utilize a combination of AI-based and rule-based algorithms for requirement identification (Zhao et al., 2021).

Requirements Identification

Various works have attempted to address the issue of requirements identification. Among these topics, there are existing systematic literature reviews that address the various different types of automated requirements engineering tasks(Zhao et al., 2021). They define the different automated tasks as detection, extraction, classification, modeling, tracing and relating, and search and retrieval. Of the various different types of automated tasks, the systematic study includes extraction, classification, and modeling as requirements identification-related tasks. Table 11 provides more detail of these associated tasks and how they relate automated requirement extraction and identification (Zhao et al., 2021).

Table 11 Task Definitions (Zhao et al., 2021)

Task	Definition
Extraction	Used to extract project-specific concepts and terms from requirements. This task can help with modeling and consistency checking among differing requirements.
Classification	Classifying text into different categories. For instance, identifying requirements-related text, non-functional requirements, or requirement classification such as security-related or usability-related requirements.
Modeling	Extraction of requirements followed by the generation of associated models, such as UML diagrams. Helps support analysis of completeness and consistency of requirements and their associated relationships.

An Overview of NLP-based Methods for Requirements Identification

For requirements identification, it is important to differentiate between rule-based implementations for automatic tasks and the use of Machine Learning (ML) to delineate and extract requirements. For rule-based implementations, utilizing common patterns is possible to look for indicators that might indicate the presence of a requirement, or more granularly, an agent, action, or requirement restriction. We can use rule-based implementations as a general rule-of-thumb to detect and extract requirements-related text. We can further utilize NLP-based methods for rule-based systems through Part of Speech (POS) tagging, Semantic Role Labeling (SRL) and other methods to help with requirement identification, term and relationship extraction, and requirement classification tasks. Normally, these methods are utilized with pre-defined heuristics that help define common patterns that could delineate a requirement and its associated components.

In addition to rule-based methods, NLP can help with statistical and ML-based models. Typically, these types of implementations require training data, which further complicates the ability for accurate inference due to the lack of publicly available datasets for training. Although the training data requirement complicates automated requirement-based tasks, these methods can learn complicated patterns among text that better generalize against unseen examples when compared to rule-based patterns. Although different models have different requirements for training data, multiple preprocessing methods and new synthetic generation processes help to mitigate the limitations of dataset scarcity within requirement automation tasks. Additionally, data preprocessing techniques help train complicated models through concepts such as data augmentation, synthetic data generation, and feature engineering. Ultimately, text preprocessing can help provide adequate data for model training, allowing for more accurate and general patterns among data.

More complicated models that utilize Machine Learning have steeper data quantity and data quality requirements. Existing models such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Artificial Neural Networks (ANNs) are trained on existing, diverse and high-quality data to learn patterns that can help with classification. From the learned patterns, these models are able to capture complicated relationships among text that typically extend beyond statistical models. Other ML-based models utilize unsupervised and semi-supervised methods to achieve automated requirement classification. All of these different methods of classification are used to identify and classify requirements. In most cases, the effectiveness of these different ML methods will vary depending on the training data and learned patterns from the models. Ultimately, unsupervised, semi-supervised, and supervised methods have useful impacts on requirement identification; each method can help with the generation of synthetic data, automatic labeling, feature extraction, and inference.

To understand the potential impact of these different types of requirement identification methods, an analysis of existing tools will prove useful to help understand the existing state of methods for automation. Naturally, many different tools have a variety of different functions for requirement extraction and identification. Of the existing automated NLP-based tools, a systematic study found that extraction, classification, and modeling comprised a relatively high percentage of existing tools, with modeling scoring the highest in their study with 26.15% of all reviewed tools. Additionally, the study found that requirements specifications were the most common input documents for these tools, followed by User-Generated Text, General NL Documents, and Use Cases. For NLP-based methodologies, the modeling task primarily utilized Lexical Analysis, ML Classification, Regular Expressions, and Syntactic patterns, along with other common practices such as Stemming and Lemmatization for NLP. Classification and extraction utilized similar NLP-based methodologies. The existing tools outlined by the study show a wide range of implementations for requirement automation texts, among multiple different types of rule-based, NLP-based, and ML-based tasks (Zhao et al., 2021).

Ultimately, the existing tools have sought to address the automation of requirement-based tasks, including classification, modeling, and extraction through different methods across NLP and ML. Different implementations have unique positives and negatives depending on the dataset. Still, many existing tools and implementations struggle from dataset scarcity; to overcome the lack of labeled data, many tools utilize unique methods and processes to perform automatic analysis. Of the existing dataset limitations, a prime consideration of automation is the generalization among multiple different types of documents and domains. Therefore, it is useful to look at models and methods that can help not only identify requirements but identify requirements among a wide range of differing requirements documents.

Requirements Identification through Different Mediums

Often requirements are found in different mediums and forms. For instance, requirements can exist in structured documents, notes, or other forms of electronic communication. As a result, requirements often do not follow a pre-defined format before a formal identification process takes place, making rule-based templates less useful. The dynamic representation of requirements was a key limitation of previous, rule-based tools, and remains a difficult problem to solve even with NLP and ML-based methods. Therefore, an analysis of how different tools accomplish requirement identification across diverse representations is important. Because requirement documents tend to exist in unstructured language, deriving models that can find the patterns associated with requirements in different forms is important, even when requirement templates are consistent. Further, it is important that these patterns transcend typical syntactical formats for requirements. With non-rule-based implementations, automated systems can likely handle different permutations and templates for requirement identification, making the automated process less rigid and more generalizable.

As an example, online forums typically contain requirements for products. Often, these requirements come from multiple different people and contain different formats and representations. To handle requirement identification for an online forum, a systematic review looked at the NLP-based methodologies for identifying requirements among Stack Overflow (Ahmad et al., 2020). They look primarily at the existing ways to delineate both functional and non-functional requirements from the Stack Overflow website. The systematic study shows that the Support Vector Machine (SVM) and Latent Dirichlet Allocation (LDA) algorithms performed well for both functional and non-functional requirements identification across the various studies, with approximately 70% accuracy across both SVM and LDA methods for identification and categorization (Ahmad et al., 2020). The study also shows that methods of pre-processing of text such as Stop Word Removal, Tokenization, and Stemming were commonly used (Ahmad et al., 2020). Overall, the study indicates that ML-based and statistical algorithms can achieve reasonable accuracy across both functional and non-functional requirements, while also maintaining decent generalizability to unseen requirements.

Another study utilizes existing construction-based documents to detect and trace requirements (Jeon et al., 2021). The author proposed a way to extract quality inspection requirements from construction specifications using NLP. In the study, they extract individual sentences from the specification, preprocess the sentences through common NLP techniques, apply embedding algorithms like Word2Vec and Glove, and utilize machine learning to classify whether a sentence is a requirement or not. By utilizing CNNs and RNNs in conjunction with embeddings, they achieved a 95.49% accuracy with CNN and GloVe embeddings for training data, while also achieving a 91.85% accuracy on testing data. Ultimately, their training data was based on a Department of Transportation (DOT) Standard Specification that was mostly structured, limiting the scope and impact of the research. Due to these limitations, the generalizable of their training data is minimal (Jeon et al., 2021).

Ultimately, these studies experimented on different mediums, showcasing impressive results across different methods. SVM and LDA showed impressive results among a diverse set of requirements and sources, along with ML-based implementations with embeddings. Additionally, embeddings coupled with ML-based algorithms can perform classification among requirements, while learning complicated relationships that are beyond typical feature engineering and rule-based tasks. This is because embeddings help to understand context among words within a text. Instead of training a model on text, embeddings help a model to learn the semantic context in which requirements may exist. For many requirements, utilizing context is important to help identify supporting and requirement text outside of indicators. Importantly, SVM and similar methods rely on data preprocessing to achieve ideal classification performance.

Many of the preprocessing techniques simplify and normalize the text, limiting the amount of patterns a model has to learn. In essence, the transformation of requirements into a simpler form helps to limit the need for extensive training data and complicated pattern recognition. Alternatively, embeddings typically do not utilize preprocessing techniques, as their primary purpose is to capture the context and relationships among words. As two distinct approaches, these methods provide robust ways to further generalize models.

In a systematic study that tried to understand the existing work on requirement identification and other requirement automation tasks, existing tools and methods try to engage with both requirement extraction and requirement classification; requirement extraction deals with extracting the important terms and relationships out of requirements, whereas classification tries to identify requirements, place the requirements into categories based on similarity, or to determine the type of requirement category that the requirement belongs to. The vast majority of the input documentations used for these automated tools are from requirement specifications; as the study mentions, minimal implementations for extraction and classification deal with user-generated text, NL documents, Use Cases, and other diverse forms of documentation, leading to an issue with the generalizability of most tools and methods across different documentation types that contain requirements. Additionally, most implementations focus on extraction, as opposed to search, retrieval and classification. Primarily, this implies that there are minimal tools that work to find requirements within unstructured text, as most tools require a list of requirements in order to perform automated analysis. Therefore, the availability of requirement extraction tools is limited and are ultimately not trained on diverse documentation. Notably, the study indicates that classification and extraction tools utilize rule-based and ML-based implementations across the various implementations, suggesting that the methods of requirement identification are extensive (Zhao et al., 2021).

Requirements Identification, Data, and Relationships

The problem of requirements identification is further compounded with the tendency to nest requirements to arbitrary levels of granularity. Additionally, there can often be multiple requirements in a single text, making the requirement extraction process even more difficult. To maintain requirement traceability and dependencies, allowing for the maintenance of the requirement relationships when parsing unstructured text is important. Of course, this is a challenging task due to the complex representations of requirements. In the case of an expected format, a system could potentially utilize programming logic to identify requirements and sub-requirements, while also maintaining the hierarchy and traceability of the associated requirements. Because requirements often do not follow the same format, this approach will not extend to most situations, requiring more complicated implementations.

Alternatively, automatic AI-based approaches could help identify requirements and their associated relationships. For instance, question-answering approaches can help with identifying both requirements and their associated hierarchy (Akay et al., 2021). The study analyzes digital design documents with the aim to extract requirements and formulate a hierarchical relationship between all requirements in the document. To accomplish this, they utilize transfer learning and question-answering-based approaches to extract requirement relationships. They fine-tuned the Bidirectional Encoder Representations from Transformers (BERT) Large Language Model (LLM) to utilize question-answering for requirement identification and relationship extraction. To accomplish this task, the study fine-tuned the BERT LLM on the Stanford Question Answering Dataset (SQuAD). To implement the model, the authors formulated pre-defined question templates that were used recursively to help extract requirement dependencies. They refined their prompts through a prompt engineering process to help achieve better results. The question formulations help them recursively identify requirement dependencies and relationships within the document. They achieved a precision of 0.71 and 0.89 and a recall of 0.55 and 0.89 for their two abstracts during testing.

Although the previous study shows that ML-based approaches can identify requirements and the hierarchical relationships among them, identifying more granular aspects of requirements through question-answering is also possible. Due to the unpredictable format of unstructured text, the article tries to address the requirements elicitation process by extracting actors, actions, and objects from within the document while also creating an ontology of associated relationships. It uses question-answering techniques to extract and formulate relationships, while also providing a way to eliminate ambiguity and identify requirements from different types of unstructured text. Ultimately, the focus of the QUARE tool is to allow for the extraction of actors, actions, objects, and relationships among different writing styles and types of documents. Across the various identified actors, actions, objects, and relationships, the system achieved an average precision of 0.76, an average recall of 0.74, and an average F1 score of 0.76 (Calle Gallego & Zapata Jaramillo, 2023).

Another example showcases the utilization of deconstructing unstructured documents through a systematic process that utilizes both Natural Language Processing (NLP) and rule-based methodologies along with the question-answering implementation for the BERT model. The proposed pipeline can parse design guideline documents, extract and classify strings of information, and generate a knowledge base that helps formulate the design variables and their associated relationships and restrictions. According to the study, design guidelines can comprise regulatory documents, requirement documents, and other forms of text, making the proposed method of requirement identification diverse among documents. The study also allowed for the conversion of images and figures to text through the usage of Optical Character Recognition (OCR). The study ultimately utilized the SQuAD dataset to fine-tune the BERT model for the question-answering functionality. The proposed method can help with both requirement identification, consistency, and completeness. Overall, the study achieved an impressive accuracy for requirement extraction of 86.3% (Kwon et al., 2024).

For many requirements tasks, identifying requirements is a pivotal part of the process, but identifying the various relationships between them and the information contained within the requirement is important as well. Considering the existing difficulty of identifying requirements, considering the additional challenges of identifying relationships for metadata extraction traceability purposes is crucial. Regardless, identifying relationships among requirements is important to help better analyze requirement documents, while also helping to create a better identification result for use during the quality analysis process. Primarily, existing studies have utilized question-answering-based methods to help identify requirements and find requirement relationships. Although these methods are impactful, the accuracy of existing implementations is low, suggesting that there is significant room for improvement among both requirement identification and relationship extraction. Ultimately, question-answering methods seemingly help to identify requirements and formulate their associated relationships within a requirement hierarchy, creating an exciting advancement for automated requirement tasks.

Requirements Identification through Rule-Based Methods

As an example of a requirement identification process, rule-based methods utilize programming logic, common rules, and logic associated with NLP to extract and identify information from an unstructured document. Rule-based methods can follow certain patterns and extraction techniques that normally follow a reliable standard of representation among strings. For requirement documents, there are often many different formulations of patterns that help identify critical information related to a requirement. For instance, rule-based methods may utilize certain keywords or phrases that can help a program identify if a given text is related to a requirement or not. Rule-based methods can look for general patterns to help identify and extract associated requirements from requirement documentation. Methods such as POS tagging and SRL help identify the words and phrases indicative of pattern matching during the rule-based automation process.

As an example, certain implementations of rule-based patterns utilize NLP-based approaches and patterns to identify requirements within a requirement document. In an article, they utilize sentence segmentation, tokenization, and Part of Speech (POS) tagging via the spaCy NLP library to further refine and identify sentences and potential requirements. The authors define a set of five differing rules comprising certain combinations of word types to identify requirements, all of which are identified through the labeling and association of words to types of speech. The library can analyze relationships to help delineate among different word groups within a sentence. The study tested the implementation on five different requirements specifications, achieving a high precision and recall of 1.00 and 0.89, respectively. However, they also recorded a precision of and recall of 0.64 among different requirement documents, suggesting room for improvement among diverse requirement documentation (Haris & Kurniawan, 2020).

Although rule-based requirements may not necessarily prove highly effective with requirement identification on their own, they can help inform other methods, such as Machine Learning. As an extension of rule-based methods, textual statistics can help determine whether a given text is related to a requirement or not through the process of feature engineering. An article utilizes a myriad of different rule-based methods to parse and classify requirements. They utilize feature engineering across textual statistics in conjunction with ML-based methods to achieve requirement identification. The study takes a requirement specification as input to their system and breaks up the document into tokens, sentences, and tagged parts of speech. The system further breaks the text down into a variety of forms to help generate textual features. Specifically, they create token-based, syntactic and semantic, and frequency-based features as input to their model (Sallam et al., 2020).

Their token-based features include items such as number of tokens, number of alphabetic words, and whether the sentence starts with an id or trigger word. They also check to see if the sentence contains measurable units, which is something that is often present within requirements. The article then breaks down the syntactic features used during the requirement identification process. For instance, a requirement typically contains a verb and a modal verb. Given the POS tagging capabilities of modern NLP libraries, they can check for syntactic indicators for requirements automatically. The implementation also checks for other syntactic rules such as noun phrases and verb phrases that include model verbs, determiners, and conditionals. Finally, the article checks for passive voice and the associated tense of the sentence. All of these in culmination define the syntactic features that help classify requirement sentences through automation. Lastly, the semantic features check for cognition verbs, action verbs, or stative verbs. These types of verbs are often associated with requirements yet require additional semantical context to help with classification, allowing for an in-depth analysis beyond typical rule-based implementations. With these various textual statistics, they train a variety of different models for classification of requirements (Sallam et al., 2020).

To generalize these textual statistics, the article utilizes a variety of libraries and NLP methods (Sallam et al., 2020). Specifically, they utilize Aspose.Word for document parsing, LanguageToolSegmenter for tokenization, Apache OpenNLP for POS-Tagging and Text chunking, and other tools for the remaining tasks necessary for classification. For their ML-based classification, they utilize a variety of different algorithms, including Decision Trees, Logistic Regression, Random Forest, and SVMs. For all of the models, they implement both cost-sensitive learning and non-cost-sensitive learning. For non-cost-sensitive learning, Random Forest performed the best with a 94.4% accuracy and a 94.3% precision. For cost-sensitive learning, Random Forest also performed the best with a recall of 96.9% (Sallam et al., 2020).

In addition to the analysis of ML algorithms, the article tries to understand the corresponding features that made the biggest difference during inference. To derive important features, the article utilizes information gain to understand impactful characteristics of text that indicate requirements. They found that characteristics such as frequent model verbs and modal verb phrases contributed significantly to the accuracy of inference. They also found that the presence of a verb, the number of alphabetics, number of tokens, and the presence of a stative verb were also important for accurate classification. Although the other rule-based statistics helped with information gain, they were minimally impactful. Coupled with the feature engineering process, the article demonstrates the power of rule-based processes and textual statistics during the training and inference process of requirement identification (Sallam et al., 2020).

As shown from the existing work with rule-based and integrated methods for inference, these methods are a powerful way to help with most requirement automation tasks, including requirement identification. Although rule-based methods are limited and often tedious to accurately define and test, these methods can achieve high accuracy, while also serving as useful features for more complicated methods of inference. With the introduction of NLP-based methods such as POS tagging and SRL, checking for pre-defined patterns to help identify requirements have become easier and more generalizable. Because of the general nature of these modern NLP methods, rule-based heuristics can help inform textual analysis on both a syntactic and semantical basis. To help improve the effectiveness of rule-based methods, existing work has combined rule-based methods with ML-based methods during training and inference. As shown with existing literature, rule-based methods help extensively with feature engineering and statistic aggregation that can be used during the model training process. In conjunction with non-rule-based methods, existing literature shows promising results through the implementation of a hybrid approach for classification.

Requirements Identification through Unsupervised and Embedding-based Methods

As an alternative to rule-based methods, Machine Learning can serve as a helpful tool for most textual-based classification tasks. In the case of requirements identification, ML can help with identifying requirement and non-requirement-related text within diverse documentation. Unfortunately, Machine Learning often requires a large amount of labeled data to achieve accurate inference. Due to the need for large, labeled datasets, utilizing machine learning for many requirements-based tasks becomes challenging. Because labeled datasets for requirements are not widely available, ML-based methods are often limited in capability for requirement automation tasks. As an additional challenge, labeled datasets are scarce for requirement identification. Unsupervised learning is an alternative that allows for the mitigation or elimination of labeled datasets. Because of the ability for unsupervised learning to learn patterns without a labeled dataset, certain unsupervised learning algorithms can help derive important features of requirements without the need for extensive training. Additionally, unsupervised or semi-supervised learning algorithms can help with the generation of synthetic data, allowing for the tuning of complicated, models through supervised learning.

As an example of the utilization of unsupervised learning algorithms for requirement identification, a study shows that semantic similarity distance helped to identify and extract requirements from requirement documentation. As the study mentions, many existing implementations utilize supervised learning approaches for requirement classification. However, the study utilizes a different, unsupervised approach to achieve classification of non-functional requirements. In addition to utilizing unsupervised classification methods, their input data is parsed with tokenization, stop word removal, POS tagging, and other preliminary data parsing techniques. After the pre-processing of text, they utilize the Word2Vec model and semantic similarity distance to help determine if a text constitutes a non-functional requirement. The study utilizes the PROMISE NFR dataset for evaluation, while also utilizing data from Wikipedia for training. Ultimately, the study observed patterns associated with a variety of different types of NFRs, including Availability, Legal, Maintainability, and Scalability. They found that a threshold similarity of 0.61 yielded the ideal results, including an average precision, recall, and F1-score of 0.75, 0.59, and 0.64, respectively. They also found that there was a notable positive impact when utilizing pre-processing techniques for the unsupervised methods during inference. Ultimately, the results showcased reliable classification across a multi-labeled problem for unsupervised learning. Because NFRs are less likely to have structured syntax associated with them when compared to structural requirements, this implementation shows a broad generalization of categorization across different requirement syntaxes (Muhammad et al., 2020).

As an alternative to unsupervised learning methods, other implementations utilize embedding-based approaches in conjunction with supervised learning methods to achieve requirement identification and classification (Shreda & Hanani, 2021). Instead of utilizing existing datasets, the study created a manual dataset through the help of requirement practitioners. The labeling included the classification of a requirement and the degree of confidence associated with the classification. Further, the selected study participants had to determine what type of NFR category the text belonged to. Because the resulting dataset was imbalanced, the study utilized SMOTE to help balance the representation of the associated classes. During training, the authors utilized data pre-processing techniques in conjunction with Python's NLTK library to extract and tokenize sentences within the document. After additional steps such as stop word removal and lemmatization were applied, they utilized feature extraction through TF-IDF, Word2Vec, and BERT to create embeddings. With the various embeddings, they trained an SVM classifier, a

Naïve Bayes classifier, a Logistic Regression classifier, and a Convolutional Neural Network. They also introduced a fusion model that combines the different implementations of embeddings to simulate a model ensemble. The CNN consistently outperformed the other models across the various embedding implementations. With the BERT embeddings, the CNN was able to achieve a precision, recall, and F1-score of 0.922, 0.914, and 0.915, respectively. In a show of improvement, the fusion model improved the accuracy of the combined BERT and CNN implementation by 2.4% (Shreda & Hanani, 2021).

Many tools and implementations focus on simple requirement identification, while other, more advanced, implementations focus on traceability and support references through automation. This methodology helps to bridge the gap of different requirement text analysis through the use of modern NLP-based methods. Many of these different implementations focus on identifying and tracing text related to requirements to support documentation traceability; additionally, these implementations help to better fully define the definitions and scopes of requirements, allowing for better analysis. Primarily, identifying requirements typically requires finding related, supporting documentation to help understand critical context and external information relevant to its implementation. These types of implementations have to therefore identify related text through diverse types of text across different requirement documentation. As an example of this implementation, a study looks at identifying and tracing safety requirements for the construction industry. In the construction industry, requirements are often either informally recorded or documented in ways that do not conform to the standard requirements process. The author propose a project called Requirement Retrieval and Document Association (RRDA) that can extract safety requirements from unstructured construction documents and recordings, analyze them, and associate the requirements with relevant documentation. Ultimately, the RRDA system can link associated requirements to documents in order to help provide supporting documentation or clarification of the associated safety requirements. The

RRDA system accomplishes this mapping by extracting keywords from documents, formulating relationships between them, and comparing the semantic similarity of the requirement to the keywords of the document. The RRDA system then matches requirements to documents through keyword matching, semantic analysis of document topics, and emotional analysis (Wu & Ma, 2024).

In summary, the difficulty of requirement identification is mitigated through unsupervised and embedding-based methods. Many existing studies, implementations, and tools have found creative ways to limit the need for extensive datasets to accomplish accurate classification of requirements during inference. Because requirement documents come in various forms, the syntax of requirements is not always consistent among different requirements documents and domains. Unsupervised learning offers a way to help better generalize the classification and identification of requirements while also limiting the amount of labeled training data needed for accurate results. Often, unsupervised learning helps group unlabeled data together in common patterns, allowing for the discovery of critical features for requirement identification. In addition, the use of context-capturing embeddings can make an important difference with semantic understanding, allowing for the utilization of more generalizable features. A few studies show the implementation of unsupervised methods by utilizing the semantic similarity distance across words and sentences to help classify requirements. Other studies utilize embedding-based techniques to help enhance the results of supervised learning algorithms. Some of the current implementations utilize a variety of unsupervised and embedding-based methods to help both derive datasets and train models, while other studies try to utilize methods that help with broadly identifying requirements among different documents. Because of the capabilities of unsupervised and embedding-based methods, algorithms can derive new features that help to better encapsulate indicators of requirement text, mitigating the need for rule-based heuristics and extensive datasets.

Requirements Identification through Supervised Methods

As an alternative to unsupervised methods, supervised methods can also help identify and classify requirements. Although this method requires labeled training data, supervised learning often can yield better results if the dataset can capture the general patterns of most unseen classification instances. There are a variety of supervised learning methods that are well-suited for requirement identification tasks. For instance, some supervised learning methods utilize statistical-based methods, such as logistic regression, and Naïve Bayes. The statistics of the words within a text serve as input to these models, allowing for the creation of models that can perform classification. Other methods of supervised learning utilize decision trees, neural networks, and ensembles, all of which have a wide variety of variations, implementations, and hyperparameters to consider during the training process for ideal inference. Even though neural networks are often used with classification tasks, they often require a lot of training data to achieve accurate inference. As a powerful classification model type, ensemble methods combine the output of multiple, different models to make a prediction during the classification process, often increasing the reliability of predictions. Ensemble models can help with generalization and accuracy improvement, as different models tend to learn different patterns for the trained data. Lastly, Large Language Models (LLMs) are often useful for inference, as their extensive knowledge is useful for more complicated inference tasks. As an example, utilizing transfer learning to apply the knowledge of LLMs for accomplishing a specific task such as requirement identification and classification is possible and often helpful with general inference. Because LLMs are trained on a large amount of data, fine-tuning requires less labeled data, making the utilization of LLMs ideal for requirement identification (Kubat, 2021).

Certain studies focused on requirement identification have utilized different kinds of supervised learning to perform requirement identification. For instance, a study utilized supervised learning-based tasks to extract reporting requirements from construction documents. The study primarily focuses on the automatic extraction of reporting requirements from relevant construction documents, aiming to estimate the effort or cost associated with the project. To accomplish the estimation task, the study utilizes a variety of methods including rule-based and ML-based methods. For their rule-based implementation, they utilize uni-grams, bi-grams, tri-grams, and quad-grams to determine if a given sentence pertains to a reporting requirement. To determine the sequences most commonly associated with reporting requirements, the study labeled a dataset with requirement and non-requirement-related text. From the dataset, they constructed the gram sequences that were most common in both the true and false categories. Originally, they used rule-based processing to classify a given sentence without ML-based training. The bi-grams achieved high results with an F1-score, recall score, and precision score of 92%, 88%, and 96%, respectively for requirement identification, while achieving similar results for non-requirements. For the rule-based implementation, the study mentions that lemmatization and stop-word removal negatively impacted the results of longer gram sequences. This is presumably because of the loss of context associated with text normalization (Jafari et al., 2021).

In conjunction with a rule-based implementation for requirement identification, the study implemented a variety of different machine learning-based methods such as Naïve Bayes, Logistic Regression, Random Forest, and XGBoost, to classify reporting requirements (Jafari et al., 2021). Because of the wide variety of the models within the study, the results help to indicate broad requirement identification results for statistical and ML-based methods. Each of these models were able to achieve reasonable results when compared to rule-based methods. However, the results were not significantly different for rule-based and ML-based classification, suggesting a potential limitation in the labeled dataset. Ultimately, the study showed an increase in recall for ML-based classification methods, while performing marginally worse with accuracy. Another consideration was the utilization of their feature inputs for their models; primarily, the features comprised the N-gram indicators derived from rule-based methods. Although the study utilized ML-based algorithms, their features might not adequately capture patterns indicative of general requirement identification, limiting the validity of the results (Jafari et al., 2021).

Another study utilized an alternative supervised learning approach through the use of embeddings in conjunction with Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). The goal of their implementation was to help extract inspection requirements from construction documents (Jeon et al., 2021). The study primarily discusses the process via a typical NLP-based pipeline that consists of extraction, preprocessing, extracting vector features, model selection, and model training. For the extraction task, the study identifies individual sentences from requirement documentation, ultimately making the assumption that sentences typically comprise entire requirements. To mitigate information loss and enhance the functionality of sentence extraction, the implementation also tries to keep the requirement's associated structural position, including its division, section, subsection and paragraph location. To help identify sentences, the implementation utilized punctuation such as periods to divide text into small units. To train the models, the implementation utilized a manually labeled dataset that contained thousands of labeled examples from experts. Based on the dataset, the authors utilized both CNN and RNN algorithms, in conjunction with Word2Vec and GloVe embedding techniques to perform classification. The GloVe embedding method seemed to increase the performance of both the CNN and RNN models. The models achieved an impressive accuracy of 91% and 88% for CNN and RNN models, respectively. Their highest performing model was the CNN and GloVe embedding combination with a 91.85% accuracy. They also found that their model was generalizable to a different specification that was not utilized originally identified in the study (Jeon et al., 2021).

Many tools and implementations utilize supervised learning for most requirement automation tasks, including requirement identification. Unfortunately, the lack of labeled datasets makes it more difficult to train models that rely on supervised learning. Some existing studies utilize manually labeled datasets, while others rely on the generation of synthetic data or the utilization of existing datasets. Although many tools and implementations have utilized machine learning, the implementation of supervised learning methods is minimal for practical tools that automate requirement identification. Of the existing models, statistical models such as logistic regression, and Naïve Bayes show potential with a high accuracy for classification; additionally, these statistical models help mitigate the need for extensive training data, while also helping with the creation of features for more complicated ML models. Through the utilization of more complicated models, higher accuracy of requirement identification and broader generalization is potentially achieved. For instance, existing models show that CNNs and RNNs achieved high performance in comparison to statistical model alternatives. To enhance these models, the use of embeddings helped to better encapsulate the surrounding context of the text. Ultimately, supervised learning seemingly help to encapsulate textual patterns that may extend beyond traditional rule-based heuristics (Jeon et al., 2021).

Requirements Identification through Transfer Learning

With the recent advancements of Large Language Models (LLMs), utilizing complex models for requirement automation tasks is possible. This technique, called transfer learning, is applicable to automated requirement tasks, such as requirements identification; ideally, transfer learning helps to mitigate the upfront training cost and dataset requirements for accurate inference. Generally, Large Language Models provide a wide-variety of knowledge that is easily generalizable to most AI-based needs, as the models are typically trained on large and diverse datasets. Because of the immense capabilities of transfer learning for automated requirement tasks, fine-tuned models can potentially outperform other, alternative models and training methods. Additionally, fine-tuning smaller models or models that were trained for similar tasks is also possible during the transfer learning process. Often used in scenarios with limited dataset availability, few-shot methods are typically used to help train models on limited datasets. Some few-shot implementations can utilize prompt engineering to help guide the model's output, whereas other implementations of few-shot methods seek to learn the embedding similarities and differences among similar and dissimilar examples from the dataset. Often, providing definitive and broadly applicable examples of input and output can help guide the model to better results. Coupled with fine-tuning, few-shot learning, or prompt engineering, the models tasked with requirement automation can potentially achieve better generalization and benchmarks in most instances, especially when datasets are limited.

As an example of transfer learning, a study has utilized the existing BERT LLM to identify requirements from unstructured documents. In the study, the authors manually label the PURE dataset for 80 requirement documents. To achieve requirement identification, they fine-tuned the BERT LLM on their manually labeled dataset. Because their dataset spanned multiple requirements documents, the hope was that their model could learn better, more general patterns among diverse datasets. In the study, they utilized the fastText baseline model, ELMo with SVM classifier, and BERT for fine-tuning. Although their training efforts were limited, they were able to achieve an F1-score of 0.86, precision of 0.92, and a recall of 0.80 for the BERT model. Overall, BERT outperformed both the F1 and precision metrics, while fastText, and ELMo with SVM classifier outperformed for recall. For RFI documents, the results where similar across models. Ultimately, RFI-based documents showcased the ability of the BERT model to generalize to different, unstructured text and formats. Although this study's results showed a lower F1-score than other, comparable studies, their dataset was larger and more diverse, likely leading to a more generalizable model for unseen requirements text (Ivanov et al., 2022).

Another study decided to expand upon basic transfer learning model implementations for requirement identification. The study implements multiple models with both a pre-processing and non-pre-processing implementation to help understand how techniques such as stop-word removal, POS tagging, and lemmatization can affect model results. Their model focus is broad due to their focus on statistical models, traditional classifiers, and semantic representation-based classifiers. For the implemented models, the authors utilized dimensionality reduction techniques such as Principal Component Analysis (PCA) to help improve model results. To train their models, the authors utilized a public dataset that was further refined into requirement and non-requirement entries. Although the dataset was limited, they implemented methods such as fine-tuning and few-shot learning to help achieve generalization and accurate classification. As an additional step to mitigate the impact of a small dataset, they utilized five-fold sampling for cross-validation, a technique that can help with data generalization and serve to help expand limited datasets. Lastly, the study utilized a wide variety of LLM BERT variants on their labeled dataset, including SciBERT, RoBERTa, XRBERT, and XLNet. The traditional BERT uncased model outperformed every other model variant along with traditional classifiers. Although their BERT model performed the best, there were other models that performed similarly with a much faster execution time. Surprisingly, the study did not seem to indicate that preprocessing of text drastically improved the results for most of the traditional models. The results of preprocessing got worse in the case of LSTM-based RNNs and BERT-based models due to the sequential nature of data. Ultimately, their few-shot models showcased decent results among 10% and 20% training data, but were outperformed by models that did not use few-shot learning (Bashir et al., 2023).

Because most studies primarily focus on the identification of functional requirements, considering implementations that identify non-functional requirements is important. Although functional requirements typically follow a standardized format and have a strong, rigid syntax, the syntax of non-functional requirements can differ considerably. Therefore, to identify non-functional requirements, generalizability is a key factor. One study proposes utilizing models such as BERT and XLNet to detect non-functional requirements. The study tries to measure how impactful similar functional requirement-based methods can be for non-functional requirement identification. The study utilized a manually labeled dataset consisting of various non-functional requirement classes. Instead of identifying if text relates to a non-functional requirement, the study trains models to identify the various classes of non-functional requirements, including Availability, Maintainability, and Operability. To enhance the classification results, the authors further parsed the text through stopword removal, stemming, and regular expressions to remove unnecessary or redundant text. The study used a variety of models including BERT, Distilbert, Distilroberta, Electra-base, Electra-small, and XLNet. Of those models, XLNet performed the best with a 0.91 for accuracy, recall, and F1, showing an impressive performance among the 12 classification tasks. When compared to other models such as Logistic Regression, SVM, and CNNs, XLNet performed significantly better for all classification classes (Khan et al., 2023).

For most requirement identification tasks, automated systems analyze text on a per-sentence basis. A key issue that persists with requirement identification is the failure to break down requirements in a meaningful way beyond simple sentence segmentation. Although sentences can typically represent entire requirements, fully defined requirements can often span multiple sentences. To mitigate this issue, a study created an application called DRIP to help merge and detect multiple requirements within a paragraph (Zhao et al., 2023). The study addresses two primary issues for requirement identification. First, the algorithm addresses whether adjacent sentences belong to the same requirement. If the two sentences are related, the algorithm merges them. Additionally, the algorithm helps delineate the occurrence of a different requirement from within the same textual body, allowing for the output and distinction of multiple, different requirements in a paragraph. The novelty of this study is through the utilization of SBert and Siamese networks to help determine similarity among sentences. Naturally, the input to the DRIP system is a segmentation of paragraphs. The Siamese network determines semantic similarity among sentences. If there is similarity, the system does not necessarily combine the sentences. It uses a series of heuristics to check for incompleteness, ultimately deciding whether to combine or separate the requirements. The system also utilizes SRL with pre-defined rules to make a determination on completeness. If a requirement is incomplete, the system redefines the boundaries of the requirement. The DRIP application showcased notable increase in accuracy, precision, and recall among comparable tools, while also providing a better way to segment and extract requirements (Zhao et al., 2023).

Due to the limitations and availability of existing datasets, transfer learning is a useful tool that allows for the fine-tuning of LLMs and task-specific models without extensive dataset availability. Transfer learning helps to re-use existing knowledge from models to achieve better results for a given task, including requirement identification. Models such as BERT and its variations are trained on an enormous amount of data and parameters, allowing for the fine-tuning and better generalization of tasks across unseen data. Because LLMs are already trained on existing data, fine-tuning often requires a minimal dataset for effective performance. Alternative methods such as few-shot learning, k-fold cross validation, text preprocessing, and dimensionality reduction can also help models perform well with unseen data and minimal datasets. Ultimately, these techniques and strategies make the broad generalization and identification of requirements possible, even with minimal datasets. Further, transfer learning can utilize the knowledge of LLMs to move beyond heuristic limitations often associated with other models.

Requirements Identification with PURE and Derivates

Due to the limitations of publicly available datasets, many studies utilize the PURE dataset and other, similar variations. Primarily, the PURE dataset consists of requirements from different types of requirement documents and representations. As a benchmark, the dataset will serve as an important distinction for model performance and reliability of the applied component of the research. As mentioned previously, the PURE dataset has many derivatives, as the dataset can serve multiple tasks. For instance, certain studies seek to utilize the requirements found within the dataset to help label functional and non-functional requirements. Other uses include foundational delineation of requirement and non-requirement classes. Still, there exist different versions and representations of the PURE dataset, yet their consistent representation of requirements serves as an important baseline for requirement identification tasks. Therefore, the utilization of the PURE dataset or a closely related set is analyzed in depth to help understand practical model performance (Bashir et al., 2023; Ivanov et al., 2022; Zhao et al., 2021).

As discussed before, an existing study utilizes the PURE dataset in conjunction with modern implementations and models. For an existing study, they took the existing PURE dataset and manually labelled for requirement or non-requirement classifications. For the dataset provided, their training was implemented in conjunction with fastText, ELMo+SVM, and BERT; their results showcased that BERT outperformed among the PURE dataset with an F1 score of 0.86, a precision score of 0.92, and a recall of 0.80. The fastText model achieved a higher recall of 0.93. For further evaluation, the study utilized manual annotation and achieved superior results with the BERT model with an F1 score of 0.80, a precision of 0.90, and a recall of 0.81, where fastText again outperformed with a score of 0.82. Notably, their results seemingly diminished when testing model generalizability for the manual dataset (Ferrari et al., 2017; Ivanov et al., 2022).

Another article utilized the dataset provided by the previous study, allowing for the synthesis of all requirement and NFR classifications into a single category, creating a practical requirement identification implementation that does not consider other classes of NFRs. The task implemented by the article therefore simply considers a binary classification of requirement and non-requirement dataset instances. Ultimately, they implement more complicated models that help enhance the results specified previously. Notably, they implement auto-regressive LLMs, few-shot learning, and ensemble methods to help label requirements. In addition, they test these models on a Dronology dataset in conjunction with the PURE dataset. For the models, they use Llama2 as the auto-regressive LLM, which is a more complicated model than the typical BERT implementation. Their implementation of DeBERTa is a close BERT derivative that helps augment attention processing, allowing for a more refined understanding of word order and context. Lastly, they utilize Few-Shot learning, a technique that allows for the classification of data with minimal data instances, and an ensemble system, which combines and considers they output of all previous models (Wang et al., 2024a).

For the Dronology dataset, DeBERTa outperformed with an accuracy, precision, recall, and F1 of 0.87; whereas the macro average for F1, precision, and recall, were 0.84, 0.82, and 0.83, respectively. Llama2 achieved an accuracy, precision, recall, and F1 of 0.71, 0.69, 0.71, and 0.70, respectively. Next, BERT achieved an accuracy, precision, recall, and F1 of 0.85, 0.87, 0.85, and 0.85. Lastly, Few-shot learning performed the worst with an accuracy of 0.68, a precision of 0.78, a recall of 0.68, and an F1 score of 0.70. For the PURE dataset, that was also combined with the Dronology dataset, various ensemble models outperformed among the general statistics. For instance, DeBERTa + Llama2 ensemble implementation achieved an impressive accuracy of 0.95, a precision of 0.97, a recall of 0.95, and an F1 of 0.95. Their macro averages, however, were lower with DeBERTa achieving the best results among precision, recall, and F1 with scores of 0.91, respectively, suggesting a lower minority class performance. Other models showed lower accuracy, with DeBERTa achieving an accuracy of 0.91, Llama2 achieving an accuracy of 0.90, and Few-shot learning achieving an accuracy of 0.76 (Wang et al., 2024b).

Another approach utilizes a variety of meta-model implementations to help determine ideal models for requirement classification for the Dronology and PURE datasets. In conjunction with their use of meta-models, the authors implement a processing layer in the form of LSTM or GRU implementations to help identify sequential patterns across embeddings. Primarily, their main models consist of variations for ALBERT, BERT, Electra, Longformer, Roberta, and Xlnet, with the variations corresponding to model size and complexity. For the standalone implementations, they evaluate these results across the PURE, RFI, and Dronology datasets (Saleem et al., 2025)

For the PURE dataset, the best standalone model for accuracy was accuracy was Longformer (large) with an accuracy of 0.83. Longformer large also achieved the highest F1-weighted score with a result of 0.81. their highest results for sensitivity was achieved by the DeBERTa (base) model with a score of 0.82, while the Albert (base) model achieved a specificity of 0.94. Although there was variation among models, most standalone models performed consistency among the various statistics for the PURE dataset. Notably, the models achieved higher accuracy and general results for the Dronology dataset, including a high accuracy of 0.85 for the Longformer (large) model. The study goes on to further suggest notable increases in weighted F1 scores across standalone models when utilizing the LSTM or GRU layer implementations for sequence classification, suggesting a better performance among minority classes.

To choose potential outperforming ensemble methods, the study combines the previous results and incorporates the top performance within the proposed meta model. For their proposed meta model of ALBERT + BERT + XLNET + LSTM, they achieved an average accuracy of 0.87 for the PURE dataset. They also achieve precision, recall, and F1 scores of 0.88, 0.85, and 0.86, respectively, while achieving macro averages for these statistics of 0.83, 0.87, and 0.84, respectively. Generally, the proposed ensemble method achieved generally better results when compared to the FastTEXT, ELMo+SVM, and BERT implementations. Additionally, they also improved on previous work that utilized ChatGPT and Gemini for PURE classification purposes.

Notably, their ensemble model lagged behind the RFI dataset, but generally outperformed with the Dronology dataset, achieving an accuracy of 0.98 (Saleem et al., 2025).

Although there are limited implementations of the PURE dataset labeled specifically for Requirement and Non-Requirement tasks, existing results tend to showcase that ensemble models generally outperform when compared to standalone models. With the PURE dataset provided publicly, the representation of requirements is long and diverse across the dataset; perhaps more complicated models or ensembles better learned the representations of the data during the training process. Surprisingly, existing literature does not necessarily discuss the utilization and implementation of statistical or simpler models for requirement classification. The lack of model variety in the literature focus of complicated LLM and ensemble implementations creates a research gap for applied requirement automation. Specifically, most research utilized complicated, research-intensive implementations. Results may indicate that simpler models can adequately capture requirement representations without significant overhead. As an added limitation, existing studies combine their training with the Dronology dataset. Because of the Dronology dataset's consistent labeling, results may be skewed towards imbalance. Indeed, many of the macro averages across statistics did show a lowered ability to achieve comparable statistics across both classes for requirement identification.

Quality Analysis

As a natural next step to requirement identification, quality analysis provides a way to analyze individual requirements, groups of requirements, and overall document quality. Many tools utilize different methods and quality models to automate the analysis of quality violation detection. Through time, these tools have progressed, although many of the foundational methods and implementations still inform modern research. Through the progression of different types of methods, many of the historical rule-based and heuristic methods are used as feature inputs to new, more modern implementations for automated quality analysis. Although many of the quality attributes are different among systems, automated tools tend to measure similar metrics and violations, creating common rules and heuristics during the automated analysis process. Ultimately, quality analysis automation faces many of the limitations associated with requirement identification, including general inference limitations and differing syntaxes and textual patterns.

To classify common issues associated with requirements, various standards have helped define categories of quality violations. For instance, the general quality attributes, as mentioned in chapter 2, are provided in the IEEE 29148 standard. These quality attributes include the different criteria amongst both individual requirements and requirement groups, aggregating common best practices for requirement formulation. The standard in particular follows characteristics such as requirement necessity, appropriateness, ambiguity, and completeness. Primarily, the goal of the quality model is to make sure that requirements are well-specified and practical given system constraints. By following the characteristics provided by these standards, a requirement document is more likely to align with the needs of stakeholders, eliminating potential, costly problems during product development ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Quality Models, Attributes, and Indicators

Because of the complexity and unstructured nature of requirement documents, automated systems have sought to define their own quality models for measuring quality. These custom models are created by differing attributes, which are the specific categories of violations that constitute a well-formed requirement. Additionally, the quality attributes are typically predicted or inferred by indicators, which are features or textual patterns that typically represent potential violations associated with the requirement. Attributes make use of the aggregation of indicators, apply customized logic and heuristics, and determine violations relevant to the quality model. Different types of quality attributes utilize different indicators and heuristics during the automation process and have sought to utilize ideal heuristics to help measure requirement quality.

As an example of custom quality model generation, a seminal article helped define the basic foundation of quality attributes and serves as a key template for many attempts to improve requirement document quality. In the article, the author defines key requirement attributes that matter within the context of requirement quality. For instance, the author defines quality attributes such as ambiguity, completeness, and correctness as necessary for high document quality. Ultimately, they provide twenty-four key attributes that serve as quality templates during manual or automated analysis. The article also suggests that many quality attributes directly affect other quality attributes, making the process of quality attribute inspection more difficult and interconnected. Lastly, the article provides metrics for measuring the adherence of requirement documents with the outlined quality attributes. These metrics are not automated, however; they require a manual inspection, typically of user feedback or simplistic rule-based tools. Ultimately, they provide eighteen primary measurements that map to various quality attributes. The generation of the various indicators and attributes in this seminal article helped to formulate many automated system implementations for quality analysis (Davis et al., 1993).

To improve the automation capability of quality violation tasks, certain studies started to formalize the rule-based and heuristics process to measure quality. As an example, an article that inspired the NASA-based applications to analyze requirement documents showcased the potential of quality indicators to map to quality violations. In particular, the study utilized imperatives, directives, weak phrases, continuances, and options for individual requirements, while utilizing size, readability, specification depth, and text structure for document-wide indicators. Although these indicators may not necessarily always serve as an accurate metric for violations, they generally can indicate document and requirement problems, without needing additional context. As an example, weak phrases are phrases that often create ambiguity or hurt practicality of requirements, such as “adequate,” “as applicable,” or “as appropriate”. The heuristics that the tool utilizes are simple, but common problems associated with requirement documents; they are also easy to detect through automation. The article utilizes the document and requirement statistics, applies heuristics and ratios, and determines based on rules the likely quality violations and overall document quality (Carlson & Laplante, 2014; Wilson et al., 1997).

As a practical example of rule-based tool implementations, a study implemented a tool called *Smella*, a program that assists with manual analysis of requirements documents by utilizing dictionaries and other NLP-based methods such as POS tagging and Morphological analysis. The tool utilizes the heuristics defined in the IEEE 29148 standard, such as ambiguous adverbs, ambiguous adjectives, loopholes, superlatives, and non-verifiable terms, while also providing useful UI-based feedback that showcases the different violations throughout the document. Although the tool does not necessarily provide a complicated analysis of requirements, the tool is largely configurable and simplistic. Their results indicate that although requirement smells can be useful for quality analysis, there are potential issues with the accuracy and validity of certain indicator smells depending on project context. According to the study, the practitioners that utilized the tool found the smells to be helpful when analyzing requirement document quality. Lastly, some of the requirement smells were easier to detect than others or differed in commonality amongst the various requirement artifacts. Interestingly, existing research shows that there is subjective interpretation for quality attribute interpretation and perceived importance, further highlighting the difficulty of automation for requirement quality determination (Femmer et al., 2017; Kummler & Fromm, n.d.; Kummler et al., 2018).

As part of a way to understand the current implementation of requirement smells, a systematic study looked at nineteen different sources that utilized various types of requirement smells and indicators. The study defines the different types of smells utilized in the various methods, including morphological, lexical, analytical, and relational smell types. Although some of these smells were simplistic phrase and word-based components, other smells were aggregations of those individual indicators. For instance, component-based heuristics included verbal tense, passive voice, number of words, and number of paragraphs. Other smells looked at the actual relationship and analysis of requirements beyond words and phrases, such as missing conditions, semantic ambiguity, pragmatic ambiguity, and continuances or references. Most modern smell detection algorithms utilized NLP, Machine Learning, or both, while a considerable amount of implementations also utilized dictionaries and rule-based heuristics. For NLP-based techniques of smell detection, methods such as POS tagging, tokenization, and lemmatization were primary used to help derive analysis of associated NLP and ML-based rules. The study also gives a list of modern tools that utilized smells for requirement quality analysis, showing a notable gap in practical tools for more complicated ML detection methods (Alemneh & Berhanu, 2024).

Different Types of Quality Analysis

So far, rule-based and heuristic-based analysis are discussed as viable and common ways to measure requirement document quality. These methods of analysis are powerful, but limited due to the complexity of requirement documents, and the need for an understanding of domain-related and contextual-specific content. As mentioned before, some requirement quality analytical systems try to utilize analysis and relational heuristics to measure quality adherence, but many of the implementations are difficult through a simple rule-based method of analysis.

Throughout existing work, there are a wide variety of different types of analysis discussed for the existing literature and software implementations. An existing study cites that there are rule-based, formalization and modeling-based, and automation-based ways to conduct requirement quality analysis. They define rule-based methods as the typical heuristics that can indicate quality violations without analyzing further context or relationships. Formalization and modeling-based methods parse requirement artifacts, map existing requirements and their relationships, and typically generate a structure, such as an ontology, to detect violations based on requirement relationships. Lastly, automation systems utilize complex NLP-based and ML-based methods to detect violations within a requirement. Examples of ML-based methods include Decision Trees, Neural Networks, and Feature Engineering. These methods differ considerably, while also proving useful through a variety of different situations and functions (Kummler, 2021).

Of the existing tools utilized for quality violation detection, there are common processing techniques utilized across a variety of tools depending on the type of implementation. For instance, of the thirty-one tools specified in a systematic analysis of NLP for Requirements Engineering, techniques such as Bag-of-Word, Frequency Analysis, Keywords Searching, Lexical Analysis, POS Tagging, and SRL are used, often in deriving features for rule-based and heuristic implementations. Other tools utilize ML classification, Semantic Analysis, Annotation, Similarity Measurements, and Embeddings in the form of automation-based requirement analysis. Of the tools identified, the most commonly used NLP techniques were POS Tagging, Tokenization, Parsing, and Stop-Words Removal, which are all primarily feature engineering and rule-based methods. Techniques such as ML Classification, Semantic Analysis, and Embeddings were hardly utilized among existing tools. The study also mentions the notable lack of tooling availability, collaboration with practitioners, and utilization of modern ML-based methods (Zhao et al., 2021).

Existing Tools

Based on the general categories of automated tools, discussing the existing tools and their methodologies is useful to determine the research gap and avenues for potential improvement. As mentioned before, most of the tools cited are not necessarily available for download or are outdated and unusable due to lack of support. However, many methods exist outside of practical tooling that showcase and implement new, modern methods of quality analysis. Even with these implementations, there is a notable lack of tooling that incorporates these modern practices into automated analysis of requirements. Still, these modern methods can provide inspiration for understanding how to improve the practical implementation of tools or the development of new features and capabilities for requirement quality analysis (Zhao et al., 2021).

Automated Modeling Tools

For complicated quality attributes, many tools seek to provide visuals of requirement artifacts instead of automated requirement defect detection. Through the use of visuals, human practitioners can easily see dependencies and overall relationships from within the requirement documents. For instance, UML models and class diagrams are all artifacts that help to visually represent complicated relationships and actions within documents. Even without human practitioners, modeling helps to create an ontological mapping that a rule-based system could analyze. Of the existing automated tools, the vast majority of them automate modeling. Therefore, analyzing the usefulness of these tools helps to understand how modeling can translate to quality analysis (Zhao et al., 2021).

A few systematic studies have shown the specific impact of modeling tools. A more recent systematic study describes other tools that are practically implemented, along with various methods and concepts to analyze requirements. These tools and methods help to achieve automatic analysis through different implementations. As an example, the study cites a few practical tools that deal with the generation of models to help with the requirement analysis phase. *DoMoBot*, a tool cited by the systematic study, transforms domain problem descriptions into domain models. To accomplish the transformation, the tool utilizes a combination of rule-based methods, semantic similarity, and Machine Learning. Rule-based methods help to transform and extract the metadata of sentences to formulate concepts, while embeddings and ML-based methods help to understand the association and relationships among the different domain entities (Saini et al., 2020). Another tool called *TRAM* helps with the generation of models and the automation of *Model Driven Development*. The tool takes requirement text, parses it through rule-based actions, and outputs associated with UML diagrams and a traceability report that helps map the requirement text to the UML elements. As an alternative to other tools, *TRAM* utilizes multiple different templates when analyzing requirements to better determine model relationships; these templates, called Semantic Object Models (SOMs), are then used to generate the various dependencies and models present in a UML class diagram. Lastly, the tool generates a report via *RTracer* that helps with forward and backward traceability (Letsholo et al., 2013). Other tools specified in the systematic review such as *Sugar, CM-Builder, aToucan, Use Case Diagram Generator, UML Model Generator, Sequence Diagram Generator, and Conceptual Modeling* all work to transform plain text requirement artifacts into structured, model-based representations (Umar & Lano, 2024).

Automated Extraction Tools

When considering model generation and other automated requirement tasks, there are a variety of tools and methods that help with automation and quality analysis. Ultimately, model generation is a useful way to analyze complicated quality attributes through the use of powerful visuals and ontologies. Although model generation and quality analysis are distinct tasks, model generation can typically help with requirement evaluation; For instance, requirement models can create visuals that help with the analysis of completion, consistency, traceability, or ambiguity (Cavada et al., 2009). Alternatively, model-based automation systems can help generate requirements with models as input (Gupta & Siddiqui, 2019). According to a systematic study, modeling-based tools made up a wide-variety of the existing tools for automation at 26.15%, with detection of defects at 23.85% (Zhao et al., 2021).

Because of its close relationship to modeling, extraction tools are often pivotal in the automation process to facilitate the generation of artifacts. Extraction-based tools focus on establishing glossaries that include domain-specific terms and phrases. This task focuses on extracting out the text that creates actual relationships and is requirement-based. Primarily, extraction serves as the input to many other requirement automation tasks, including detection, classification, and modeling. Typically, NLP-related methods are used to determine the entities and relationships needed to achieve extraction. Extraction makes an important impact with modeling, and often serves as a necessary precursor to the generation of artifacts. Therefore, there are a variety of different tools that provide API capabilities for extraction. In essence, extraction tools can help refine the document, while eliminating irrelevant text. Of the existing tooling, extraction makes up approximately 18.46% of tools (Zhao et al., 2021).

Automated Quality Tools

As its own separate category, there are a variety of tools that automate detection of quality attribute violations for various different requirement artifacts. A systematic study looks at the various types of tooling available, ranging from older tools to more recent applications. These tools differ widely in their implementations and methodologies for accomplishing automated analysis. Additionally, the quality models used for automation also differ significantly. When considering the various tooling that is available, this analysis looks at rule-based tools and AI-based tools, which are two common, distinct categories used throughout the available tools for automated requirement analysis (Zhao et al., 2021).

For existing rule-based tools, the Systemized Requirements Engineering Environment tool (SREE) helps to measure ambiguity through rule-based indicators, primarily focusing on 100% recall for violations at the cost of precision. SREE relies on dictionaries that are user expandable, focusing on simplistic words and phrases instead of complicated heuristics. In essence, the tool tries to allow for dynamic list expansion while also utilizing common quality violation indicators (Tjong & Berry, 2013). A more complicated tool, called the Requirements Quality Analyzer (RQA), analyzes text on a morphological, lexical, analytical, and relational basis. The RQA tool provides feedback to the user based on the specific quality violation by utilizing the indicators as a way to map to error feedback messages. Uniquely, the RQA tool utilizes a step function that determines if quality indicators positively or negatively affect quality (Génova et al., 2013). As another rule-based tool, *LELIE* utilizes common rule-based heuristics to detect lexical, syntactical, semantical, and discourse-based errors. As an alternative to other rule-based tools, *LELIE*, adjusts its heuristics based on user feedback, allowing for the dynamic update of rules to better accommodate domain-specific knowledge (Saint Dizier & Kang, 2015).

There are many rule-based tools that focus on simplistic easy-to-use implementations and continually make a practical impact on existing tooling for the current iteration of automated quality tools. For instance, the Automated Requirements Measurement tool (ARM) is a tool that utilizes aggregated textual statistics within requirements such as continuances, directives, and weak phrases to determine overall document quality. The tool provides references to the potential violations within the output, allowing for an intuitive explanation of how to enhance document quality (Carlson & Laplante, 2014). The SRR-Director tool analyzes a custom quality model, including accuracy, ambiguity, and verifiability. The tool supports MS Word, Excel, structured XML, and text files for input. Additionally, it utilizes lexical analysis, syntactical analysis, and dictionaries to generate conclusions about requirement quality (Berrocal Rojas & Barrantes Sliesarieva, 2010). The Quality Analyzer for Requirements Specifications (QuARS) is a popular tool that measures linguistic issues associated with ambiguity, incompleteness, and inconsistency. The tool defines a new quality model that specifies heuristics for ambiguity, specification, and understandability quality attributes. The scope of the tool focuses on lexical, syntactical, and semantical heuristics for the quality attributes within the custom model. The tool also provides a UI-based implementation that highlights the location of specific indicator violations, while providing dynamic dictionaries that allow for the adjustment of indicators (Lami et al., 2019). Lastly, REGICE is a tool used to extract out glossary terms for requirement artifacts. The tool helps to store, organize, and combine related glossary terms, serving as a preliminary and informatory tool to many other, practical quality tools. The tool ultimately helps determine features and potential quality violations such as inconsistent terminology (Arora et al., 2017). These tools are widely useful and informative to many modern implementations, serving as methodological foundations or tools to help further parse requirement documents.

Other, more complicated, rule-based tools go beyond simple heuristics to detect quality violations. Some of these methods use formal-based implementations and abstractions to help create a structured format that enables quality analysis of complicated quality attributes. For instance, *AnaCon* is a Controlled Natural Language (CNL) tool that takes in a strict linguistic structure, generates formal logic, and detects logical conflicts. The tool requires strict, rigid structures, but helps to analyze conflicts through obligations and prohibitions. Through the utilization of a custom tool, AnaCon gives counterexamples to show the reasoning behind suspected violations (Angelov et al., 2013). An alternative tool, the Automated Inconsistency

Checker (MaramaAIC), is a rule-based tool that abstracts individual requirements to general categories and forms of requirements. By reducing the requirement specificity and applying a general mapping to generic requirement components, the tool can check for consistency, correctness, completeness, and traceability for requirement documents. The tool generates visual outputs in the form of models and HTML components that help convey the dependencies and potential conflicts among requirements. All of the tool's heuristics are based on the generic forms of requirements, instead of the actual requirement. Although information is lost during the conversion, the tool is able to detect broad requirement inconsistencies while also enhancing readability (Kamalrudin et al., 2017). *Dowser,* another model-based tool, analyzes ambiguities, inconsistencies, and underspecifications within requirement documents. The tool turns the requirement-related text into a visual presentation of requirements in model form, including classes, methods, variables, and associates. Accordingly, the tool can utilize pre-defined logic and human inspection to identify complicated quality attribute violations (Popescu et al., 2008). As an additional example of formal methods, the Knowledge reuse-oriented safety analysis (KROSA) tool analyzes potential hazard and safety issues based on an ontology and previous examples provided by the user. The tool expands on the upfront investment for user input to provide better, more accurate results. KROSA utilizes NLP methods to break down requirements, reference relevant ontologies and domain knowledge, and match to potential problems through similarity metrics (Daramola et al., 2013).

Other tools that rely on formal-based methods and strict input are also specified within the systematic review. The Requirement Assessment Tool (RAT). Extends upon legacy tool functionality from GVscrib, Innoslate, QuOD, and RQA. The tool supports the analyses of various file types, including doc, docx, txt, and pdf. It also utilizes dictionaries to measure the quality adherence to the IEEE 29148 standard. Additionally, the tool analyzes both individual requirements and global document quality (Naeem et al., 2019). As a way to provide system compatibility through API access, the *AQUSA* tool exposes an API that is broadly useable for quality analysis. The tool checks user stories for template conformance and adherence to the QUS framework for quality. The tool also utilizes rule-based heuristics through the use of NLP to analyze syntactic, semantic, and pragmatic quality violations. To analyze pragmatic and semantic violations, the tool utilizes the aggregations of various different analyzers to determine quality violations. The tools API support allows for a deeper analysis of requirements due to the formal notation and processing available with integration (Lucassen et al., 2016).

There are also many tools that utilize unique methods of analysis beyond typical extraction and quality analysis techniques. These tools primarily focus on ways to integrate with the experience of the user, allowing for more creative and broad solutions. For instance, the Computer-Aided Requirements Logic (CARL) tool focuses on defining UI-centered approaches to abstract away formalization logic; the tool does not force changes based on formal logic results to help facilitate the natural evolution of requirements. Functionally, CARL creates an abstraction of the formal logic utilized with associated intuitive UI elements. The tool focuses more on practical requirement development, instead of encouraging near-perfect requirement documents (Gervasi & Zowghi, 2005). Another unique tool, *NARCIA,* deals with change management of requirements in an automated fashion. The tool utilizes phrase similarity via WordNET path similarity metrics to determine of other requirements are likely affected by a change. The tool compares requirements across phrases or sentence components to determine likely impact from a change. NARCIA does not require traceability across requirements due to the method of semantic analysis utilized by the tool (Arora, Sabetzadeh, Goknil, et al., 2015). Lastly, *QAMinor*, focuses on extracting out quality attributes from use cases. The tool extracts out early aspects, which are verb-object pairs from requirements, maps the early aspects to a general quality attribute, and stores the early aspect and associated quality attribute in an ontology. The tool utilizes semantic similarity of existing matches found through the ontology to determine accurate placement of an early aspect. Utilizing the aggregation of the ontology results, the system extracts the most important quality attributes that most commonly appear in the document (Rago et al., 2013).

Lastly, there are a variety of tools that utilize NLP, AI, and ML-based methods to achieve automatic analysis and extraction of requirement documents. Although these tools are available and useful, they often contain small scopes of focus due to limited training data within the requirement analysis domain. A tool called *RETA* primarily focuses on requirement template conformance through the use of NLP-based strategies and pre-defined templates. The tool breaks requirements into their appropriate grammar type, consolidates the individual elements using text chunking, and then fills in slots corresponding to Rupp's and EARS templates. Ultimately, the tool avoids the need for complicated sentence parsing and contextual understanding by mapping parts of the sentence into pre-defined categories that are used to measure template conformance (Arora, Sabetzadeh, Briand, et al., 2015). Another NLP-based tool, called Detect Nominalizations (DeNom), checks nominalizations within requirements to determine if they are fully specified. Nominalizations occur when verbs are converted to nouns; problems with nominalizations exist when the full components of the nominalization are not understood within the requirement context. Therefore, the tool checks to see if the nominalization errors are useful, eliminating examples that are easily understood and common. The tool achieves the automatic analysis through NLP and rule-based methods, including tagging, dependency graphs, and ontologies. By using the pre-built Cyc ontology, DeNom can analyze requirements without the need for document context (Landhaußer et al., 2015).

Additional tools utilize pure AI and ML methodologies to achieve requirement quality analysis. For instance, the Nocuous Ambiguity Identification Tool (NAI) identifies nocuous ambiguities through ML-based methods. Instead of focusing on all ambiguities, NAI focuses on ambiguities that will likely have a strong impact on quality. The tool utilizes basic NLP methods, rule-based heuristics, and aggregated heuristics to detect ambiguity through Machine Learning.

These rule-based elements serve as features for the ML model, allowing the model to learn off of concepts such as coordination frequency, distributional similarity, colocation frequency, morphological similarity, and semantic similarity. These heuristics utilize document-wide statistics to determine relative impact of ambiguity, leading to more relevant results (Yang et al., 2010). Next, the Requirements Specification Ambiguity Checker (ReqSAC) is a classification tree-based tool that combines heuristics and ML to make inference about ambiguity for requirements. The tool trains the classifier off of examples provided by experts, allowing for the derived surface-level features to be used as an ML-based concept. The tool utilizes sentence-level and discourse-level features to train the model. Because of the dynamic nature of the tool, the results showed a higher agreement with human annotators (Hussain et al., 2007).

Current State of Automation Tools

According to a recent systematic study, there are multiple avenues of automatic tools that further explore requirement quality automation through various methods, including ML-based and rule-based methods. These tools that share commonality with the previous systematic study discussed show a variety of different methods and purposes of automatic tools across multiple domains of operation. For instance, tools specified under model generation include unique ways to extract models and visuals from different requirement documents. Some of the output artifacts include class models, UML models, sequence diagrams, and goal models. The majority of these tools rely on AI and ML-based methods along with NLP to help preprocess and generate artifacts. Other tools allow for the generation of requirement specifications through the input of formal grammar or pre-defined requirements, mitigating the issue of natural language (Umar & Lano, 2024).

Additionally, the study also talks about requirement elicitation, a pivotal preliminary step to most automated requirement processing techniques. Some of these tools work to extract requirements from user sessions and feedback, instead of trying to parse a requirement document. Because requirements are often stored in multiple places, these tools try to extract out the requirement-related task from multiple sources of information. Some of these tools provide support for requirement extraction from meetings and audio as well, allowing for the real-time storage of requirements from meetings and external documents. Importantly, some of these tools also support requirement documents as input during the requirement extraction process; these implementations help to facilitate the preliminary process of requirement extraction for requirement documents, allowing for quality analysis and eliminating the need for re-inventing the process of identification. These processes primarily utilize NLP-based methods and the generation of ontologies and formal structures to create artifacts, suggesting that NLP is typically enough to extract out the information related to requirements necessary to generate artifacts. Seemingly, formal languages seem to act as an intermediary step for both the generation of models and the generation of requirement documents from models (Umar & Lano, 2024).

For the vast majority of quality analysis tools and methods, the primary implementations contain rule-based and NLP-based methods. Although there are tools that implement ML-based methods, few practical tools utilize Machine Learning to detect quality attribute violations. Additionally, most of the tools primarily focus on inconsistency or ambiguity; few tools measure multiple quality attributes, meaning there is not an availability of general-purpose tools for quality automation. Of the existing tools provided, many seek to determine if there are duplicate requirements or requirements-based text through the use of semantic analysis. Additionally, the method of requirement similarity matching helps with traceability and consistency within the quality framework outlined previously. Still, these tools do not implement quality models beyond similarity matching, leaving a severe gap in the availability of practical tools for quality analysis.

Seemingly, many of the practical tools specified in the different systematic literature reviews indicate that most tools focus on specific tasks, such as ambiguity detection or conflict analysis. Of the existing tools provided in both systematic studies, many are unavailable, implemented in an academic setting, or describe only theoretical concepts based on the rules of a method's preliminary analysis.

Lastly, the systematic study talks about tools that are specifically dedicated to classification of requirements. Some of these classifications include determining if user text constitutes a requirement, while other implementations talk about trying to determine if a requirement is functional or non-functional. The determination of requirement type is a pivotal preliminary step to quality analysis, making many of these tools an important part of the pipeline for requirement automation processing. Additionally, some of the tools outlined help to determine if requirements belong to a similar group, allowing for complicated quality analysis, such as completeness. These determinations, similar to other methods of automation, utilize semantic similarity of requirements to determine appropriate placement. Also, various tools focus on requirement triage, a method to determine the relative impact and importance of a given requirement. As a result, triage tools can help inform quality analysis tools by placing weight on certain requirements depending on relative impact. Overall, these tools allow for the generation of metadata for requirement documents, helping to improve the overall process of requirement analysis. Table 12 shows the various different tools provided by the systematic study; the categorization includes model generation, elicitation, quality assurance, and classification.

Table 12 Existing Tools by Category

Category	Tools
Model Generation	**DoMoBot** (Saini et al., 2020), **TRAM** (Letsholo et al., 2013), **SUGAR** (Kumar & Sanyal, 2008), **CM-Builder** (Harmain &

	Gaizauskas, 2000), **aToucan** (Yue et al., 2015), **Use Case Diagram Generator** (Vemuri et al., 2017), **UML Model Generator** (Deeptimahanti & Sanyal, 2011), **Sequence Diagram Generator** (Thakur & Gupta, 2014), **Conceptual Modeling** (Vidya Sagar & Abirami, 2014), **Secure Tropos Model Generation** (Kiyavitskaya & Zannone, 2008), **Two-Level Grammar Parser** (Lee & Bryant, 2004), **Class Diagram Extractor** (Ibrahim & Ahmad, 2010), **ER Diagram Generator** (Kuk et al., 2019), **Arabic Use Case Modeler** (Jabbarin & Arman, 2014), **Ontology-Based Class Diagrammer** (Jyothilakshmi & Samuel, 2012), **NL to Class Diagram** (Sharma et al., 2015), **Sequence Diagram** (Alami et al., 2017), **Sequence Generator** (Segundo et al., 2007).
Elicitation	**Requirements-Collector** (Panichella & Ruiz, 2020), **Requirement Mining Framework** (Pinquié et al., 2018), **Data Mining for RE** (Qureshi et al., 2021), **Graph-Based Elicitation** (Wang et al., 2021), **GOORE** (Shibaoka et al., 2007), **Ontology-Guided Elicitation** (Farfeleder et al., 2011), **JIT Requirements Capturer** (Reddivari et al., 2019), **Requirements Recommender** (Castro-Herrera et al., 2009), **Mobile App Security RE** (Yusop et al., 2016).
Quality Assurance, review & Consistency	**InConsistency Checker** (Kamalrudin et al., 2010), **InConsistency Handling Assistant**

	(Bhatia et al., 2013), **Duplicate Use Case Detector** (Rago et al., 2016), **Linguistic Merge Tool** (Umar & Lano, 2024), **RENDEX** (Antinyan & Staron, 2017), **CIRCE** (Ambriola & Gervasi, 2006), **Ambiguity Detector** (Ferrari & Esuli, 2019), **EA-Analyzer** (Sardinha et al., 2013), **REInDetector** (Nguyen et al., 2012), **MaramaAIC** (Kamalrudin et al., 2017), **Spec Review & Test** (Miao et al., 2016), **Quality User Story** (Lucassen et al., 2016), **Requirements Quality Classifier** (Parra et al., 2015), **Aspect Mining for Quality Attributes** (Rago et al., 2013).
Classification	**Crowd RE Request Classifier** (C. Li et al., 2018), **F/NFR Classifier** (Kurtanović & Maalej, 2017), **NFR Classifier** (Haque et al., 2019), **CNN Requirements Classifier** (Winkler & Vogelsang, 2016), **Rule-Based RE Discovery** (Vlas & Robinson, 2011), **Requirements Classification & Reuse** (Cybulski & Reed, 2000), **Requirements Prioritization** (Duan et al., 2009).

The tools outlined in the systematic study show a wide range of functionality and use across different requirement automation domains. These tools utilize a wide-variety of methods, including NLP, ML, and rule-based implementations to accomplish many different automation tasks. Many of these tools show the immense potential of traditional NLP processing techniques, especially when the tool's purpose is limited and well-defined. Often, many of these tools provide the ability to act as a preliminary step in the automation process for informing other tools. For each part of the pipeline, there is a specialized tool that can perform data pre-processing. For instance, many tools help perform requirement elicitation and extraction, while other tools help to take pre-processed requirements and generate models and diagrams. Notably, current tools have not utilized modern methods of AI/ML. As an additional problem, many of the practical tools are not publicly available. Of the existing and available tools, many only focus on a few quality attributes. Additionally, there is a severe lack of practical tool implementation based on practitioner feedback. Because most existing tools do not address the entire process of requirements engineering, the existing applications fail to consolidate the approaches of requirement automation processing in a meaningful way. Additionally, the limitations of dataset availability for automated requirement processes make modern tools difficult to produce.

The consolidation of the various tools and methods for requirement automation should include modern methods of processing with the support of automating most major requirements engineering activities necessary to perform quality analysis. For instance, a tool should utilize modern methods to allow for both requirement identification and quality analysis for requirement-related documents; the processing of both identification and analysis limits the need for multiple systems and data pipelines that make the automation process tedious. This system should also measure quality attributes from the IEEE 29148 standard, while also providing meaningful feedback to the user about quality violations. Lastly, research on how to mitigate the lack of dataset availability for requirement automation tasks remains an important factor for practical tool implementation. For the generated data, modern methods should take into account the diversity of domain and form present within requirements. As a notable limitation of existing tools, rule-based methodologies remain prevalent in practical implementations of requirement tools. Ideally, a new requirement tool would also utilize methods that allow for context and pragmatic understanding of context, allowing for the achievement of analysis beyond heuristics.

Conclusion

In this chapter, both requirement identification and requirement quality analysis were discussed. These two automatic processes are pivotal for the requirement automation process; often, identification and quality analysis are related for the practical analysis of requirements. When talking about requirement identification, there are a wide range of ways to identify requirements within text. Some methods utilize rule-based methods to identify requirements, such as looking for common terminology or phrases that are often associated with requirement text. Other implementations utilize Machine Learning to adequately classify requirements. Of these ML techniques, studies have looked at unsupervised and supervised learning methods. Critically, datasets like PURE allow for the training of ML models for requirement identification, making the process of model training more practical. Unfortunately, other automated requirement tasks, such as quality analysis, do not have the same availability of labeled data.

Quality Analysis of requirements is complicated due to the lack of available training sets. As a consequence, the multiple types of analysis and quality models further complicate the automation process. This chapter showed that the existing tools available for quality analysis are minimal and vastly unavailable. Unlike implementations for requirement identification, methods of quality analysis mostly relied on typical rule-based and heuristic methods to perform automation. Current tools still have not implemented new ML-based methods to perform quality analysis, leaving a considerable research gap for automated requirement quality analysis.

Chapter 4

ARQM: A New Tool for Requirements Detection and Analysis

The ARQM tool was developed as a practical tool to help mitigate the challenges associated with the requirements engineering process. Ultimately, the tool's purpose was to automate many of the existing and tedious processes to ultimately help generate better requirement artifacts, simplifying the necessary user's work during the requirement process. Primarily, the tool's intention was to provide an easy-to-use interface with broad support across general requirement documentation. Additionally, the tool focus included modern NLP and AI-based methods for requirement quality analysis, expanding upon existing tool capabilities for feedback and quality violation detection. ARQM ultimately included both requirement quality analysis and identification during its evolution as a requirement automation solution. The different evolutions of ARQM's development, including the original add-in application to the simplified ingestion, helped to further consolidate the accuracy and usefulness of the tool, allowing for a general-purpose solution for most types of requirement documentation analysis. The tool was strongly inspired by past tools, including the ARM and ARM reconstruction tools. As the tool's development progressed, many different tools and implementations helped to inform the tool's scope and operation. This chapter discusses the progression and final implementation details of the ARQM tool, discussing the details of how the practical application utilizes theoretical concepts to accomplish automatic requirement identification and quality analysis (Carlson & Laplante, 2014).

History of Development

The ARQM tool has developed and changed significantly throughout its implementation. Originally, the tool's focus was to serve as an interface for Microsoft Word documents. Critically, the idea of interfacing with Word was to utilize the application-specific APIs in the hope of accomplishing broad and extensive use cases for requirement automation. Additionally, a Word-specific application provided unique abilities to provide UI-based feedback, while increasing the intuition of analytical results through inline comments. The original scope of the ARQM application was extensive, supporting a broad range of session management functionalities and broad requirement actions. As the tool was developed, the implementation shifted to a simple API-based program that minimized UI feedback. With the API-based implementation, requirement identification became a pivotal factor for adequate processing, while ultimately simplifying the overall complexity of the application and its associated implementation.

Microsoft Word Add-in

The Add-in implementation for ARQM was meant to combine both user input with the strong and extensive capabilities of AI-based methods. Primarily, the add-in was meant to allow a user to identify the requirements and requirement groups through the ARQM user interface, while providing automated quality feedback based on the declared requirement-related text. By allowing the user to declare requirements and requirement groups, the tool mitigated the parsing complexity of requirement documents. Additionally, by eliminating the need for automated requirement identification, the ARQM system could derive critical features that were necessary for analyzing complicated quality attributes among both individual and group requirements. Ultimately, the manual declaration of requirement allowed ARQM to bypass the unstructured nature of requirement documents, while also allowing for the formulation of a requirement hierarchy.

Originally, the intention of the ARQM add-in implementation was to provide general feedback for all quality attributes among both individual and group requirements. Because of the system access to requirement hierarchy, analyzing complicated quality attributes for requirement documents was possible, even among requirement groups. For instance, the scope of ARQM included quality attributes that required knowledge of all existing requirements, their associated hierarchy, and document context. Even though measuring quality attributes such as completeness, consistency, and feasibility are complicated, ARQM helped consolidate and simplify the documentation within requirement artifacts to make an automated analysis possible. Figure 11 demonstrates how the ARQM Add-In primarily works for common requirement-related tasks.

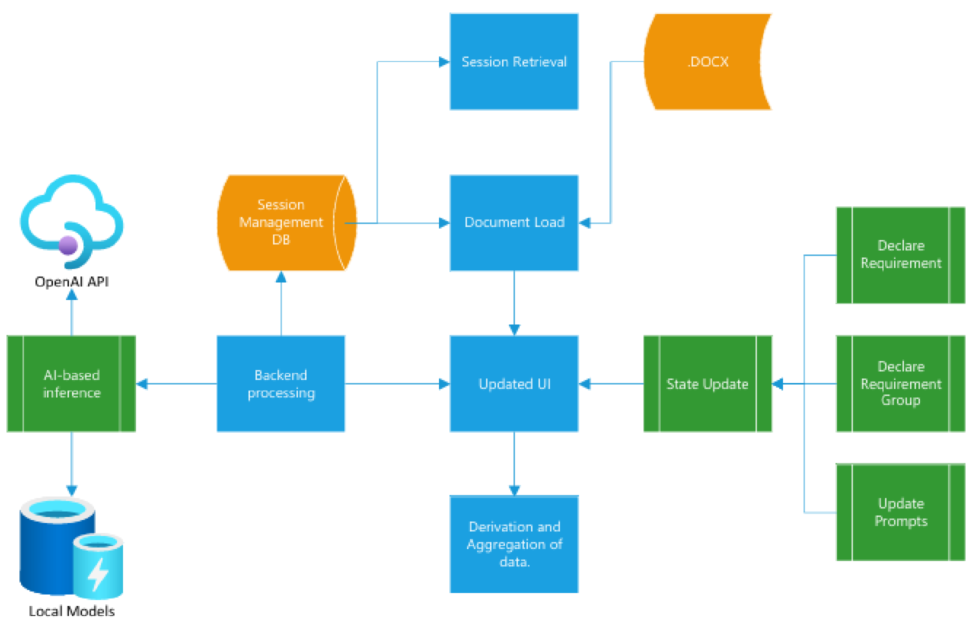

Figure 11 Basic Add-In Implementation

As shown by the previous example, the ARQM system takes a Word document as input and, through context and session management injection, loads the application state for that specific document. Once loaded, the user can update the application through requirement and requirement group declarations or prompt updates. Prompt updates are manual user input that help provide pivotal context to the application during model inference, allowing for the creation of external knowledge and user-defined limitations before implementing quality evaluation. Any of these three actions will update the application state, save the relevant information to the user session, and aggregate results accordingly. Additionally, the ARQM Add-in application utilizes a variety of different model implementations to help with inference. Primarily, the application was concerned with quality attribute determinations while also providing feedback based on the inference results. To engage in quality violation, the system requires certain features from the existing user session. Figure 12 shows the input information necessary to accomplish automated inference of quality for requirements and requirement groups within the ARQM application.

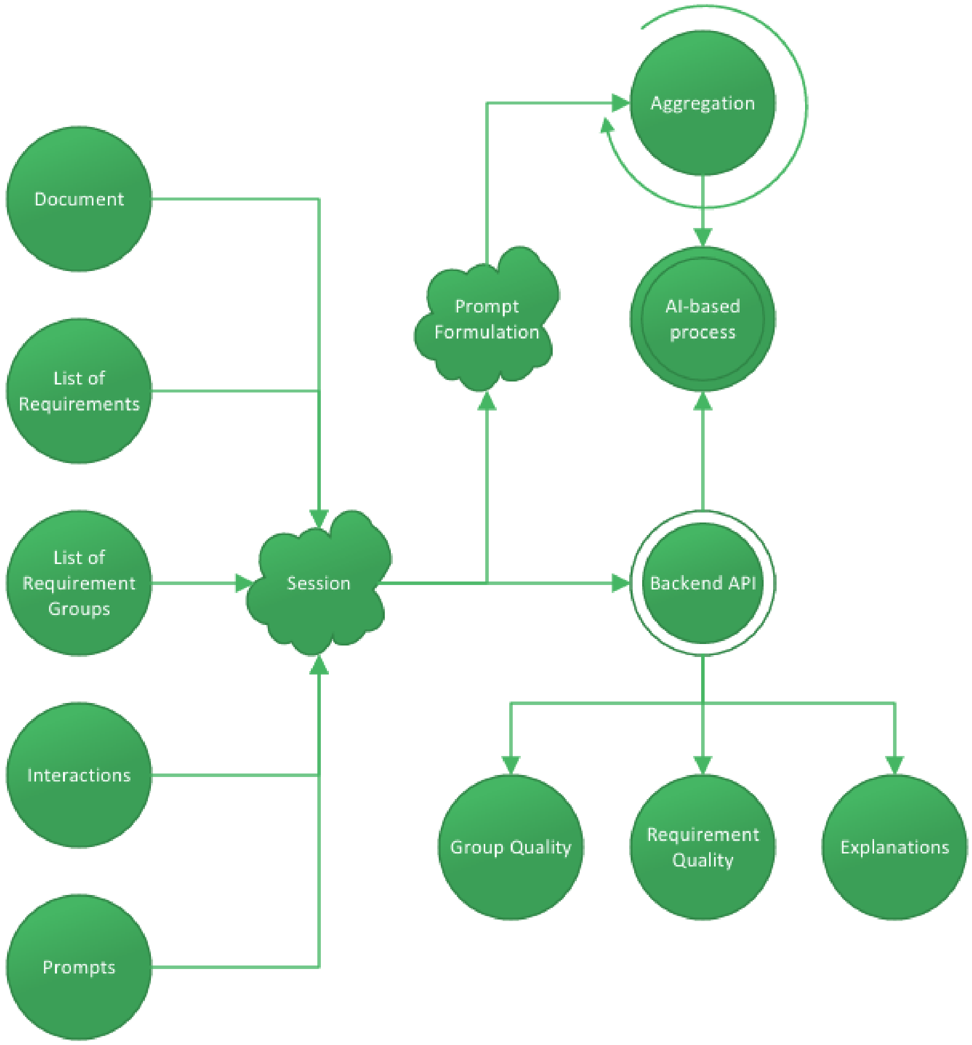

Figure 12 Add-In Data Flow

To address some of the important capabilities of an automated requirement analysis tool, ARQM focuses on a wide range of features which are engrossed- primarily on requirement management, traceability, and automated quality analysis. The core features of the application provided the user with the ability to define the requirement structure, analyze the associated quality of requirement text, and interact with the various quality results. These features combine the necessity of user input with the automated ability of ARQM to measure requirement quality.

Critically, the application measured all quality attributes, while providing granular explanations for potential violations among requirements and requirement groups. Table 13 lists the main features provided within the ARQM application.

Table 13 ARQM Add-In Features

Feature	**Explanation**
Aggregated Statistics	ARQM provides a way to aggregate quality data based on requirement and requirement group results. The aggregated statistics give a visualization of the overall document quality based on the declared requirement text.
Requirement Hierarchy	It allows for the generation and visualization of requirements through a hierarchy of groups and individual components. ARQM allows for a visualization of requirement dependencies while also enabling smarter quality processing.
Quality Evaluation	ARQM analyzes the quality attributes of both individual requirements and requirement groups. The tool primarily utilizes the IEEE 29148 quality model for analysis during the automated feedback process. ARQM also utilizes the defined requirement hierarchy to provide context and other important relationship information for quality evaluation.
Custom Prompts	ARQM utilizes custom prompts, a pivotal way for the user to provide context to the system before evaluation. Within the interface, the user can provide both AI-specific instructions and project context to

	consider before analyzing quality. These inputs allow for the injection of additional context and limitations that may exist within the project.
MS Word Integration	The ARQM application is integrated primarily with MS Word through the add-in API. The components of Word make the application processing linked to user actions, simplifying the process of requirement identification, while providing more intuitive feedback. ARQM utilizes word-based components such as comments, highlighting, and inline feedback that are native to the Word interface.
Tool Communication	ARQM provides the ability for users to respond to quality violation responses from the system while also maintaining a communication history for all requirements and requirement groups. By allowing user feedback, added context allows for better model responses and inference.
Custom Accounts	All interaction history, quality results, and statistics are saved within the user account context. A user can utilize their login information to continue analyzing their requirement documents.

When considering the variety of different AI-based implementations available, ARQM utilized various types of data processing, analysis, and inferences to achieve classification and feedback of requirement quality. Initially, the intent of the application was to utilize the new advancements in AI technology among LLMs. Through the use of prompts and generative AI, achieving both classification and explanation of generation was possible. The primary goal was to formulate a prompt that considered the document context, requirements, and the associated hierarchy before generating an inference. For individual documents, the prompt included the custom user prompts, the actual requirement for evaluation, and any associated requirement interactions. For groups, the prompt included the various requirement group children, interactions, and user prompts. Although the prompts did not necessarily follow the typical template of prompt formats, results indicated relative success for both quality evaluation and explanation generation. Figure 13 shows the different features and prompt templating processes utilized during both individual requirement and requirement group processing.

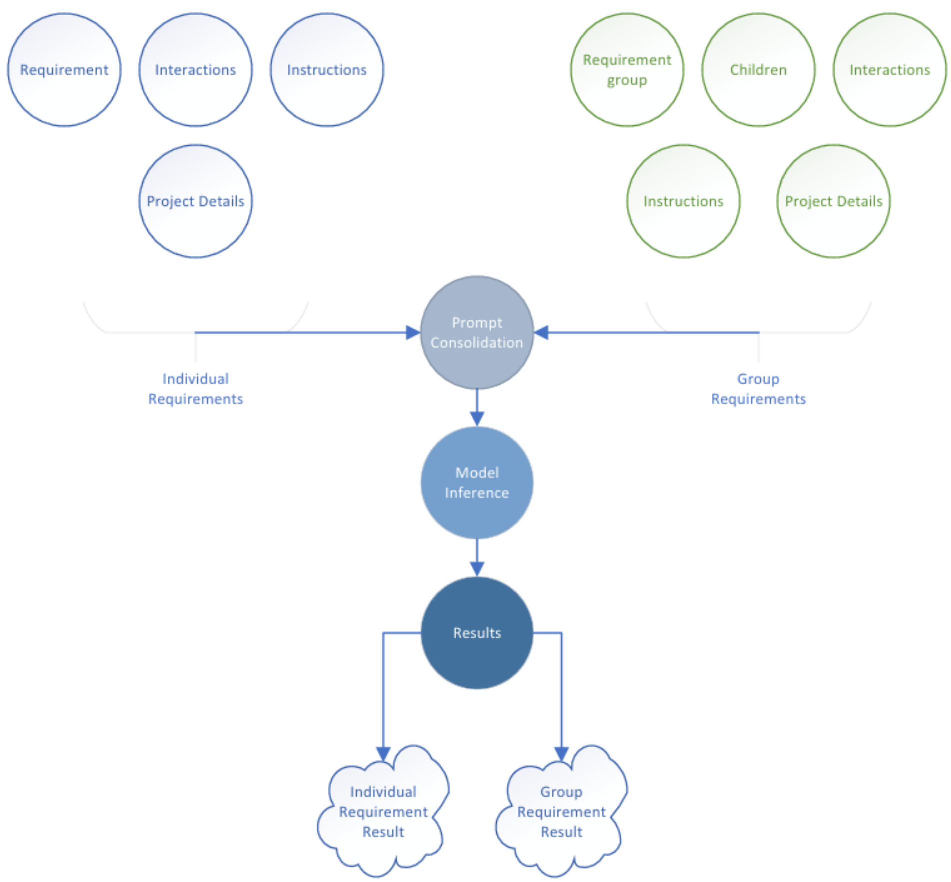

Figure 13 ARQM Add-In Prompt Components

The ARQM Add-in ultimately utilized a wide variety of model generation processes that were pivotal to successful responses within the application. For instance, the tool implemented different sequence generation models to help both the classification of quality violations and the generation of explanations for violations. Initially, the tool utilized the GPT 3.5 API for result generation; during the analysis of the results, the tool utilized prompt engineering to help fine-tune the model output for requirements analysis. Because of the limitation of training data, the purpose of the application's use of the GPT model was to utilize the tacit knowledge of LLMs to help generate accurate results instead of providing a manually labeled dataset. Although the GPT models achieved accurate results, there were immense limitations for both cost and training efficiency. As the application progressed, the server logic started to utilize alternative models from HuggingFace to limit the computational burden and cost of model training and fine-tuning.

Although the Add-in methodology showed promising results, the application was not completed or validated due to limitations for automation. For instance, the application was limited due to its inability to automatically identify requirements within requirement documents; even though individuals could manually declare requirements, the tool did not support automatic ingestion and automation of results without manual human interaction within the application. Additionally, the limitation of available training data caused a severe limitation for the validation of results; primarily, labeled data required both quality defects and explanations, making accurate requirement automation difficult and limited. As a result, much of the research for the Add-in would become foundational for a simpler, ingestion-based tool that did not require human intervention for requirement declaration.

Ingestion-Based Tool

The ARQM ingestion tool was created after the partial development of the Add-in-based tool. The goal was to simplify the process of requirement quality automation through the utilization of an API-based tool. This tool allows for the automatic evaluation of a requirement document without the need for human intervention or declaration of requirement text. To automate the requirement document evaluation, the tool needed to gain the ability to identify and structure the requirements and evaluate their associated quality. Therefore, the implementation needed to include automated requirement identification as part of the implementation, introducing a challenging aspect to requirement document processing.

Requirement Identification

To accomplish requirement identification, the goal was to divide the associated requirement document into meaningful textual units. Due to limitations of scope for the ARQM application, the primary focus was to extract information from requirement documents in the form of sentences, even though requirements can often span multiple sentences, paragraphs, and pages. To parse existing documents, the ARQM application utilized the *fitz* python library to convert PDF documents to a textual representation. After the conversion, the ARQM application addressed the issue of sentence parsing through the utilization of regex and rule-based pattern matching. Once the application successfully splits the document into sentences, the program would utilize pre-defined models to determine which sentences were requirement-related text. Figure 14shows the overall process of transforming requirement documents into a list of requirements for quality analysis.

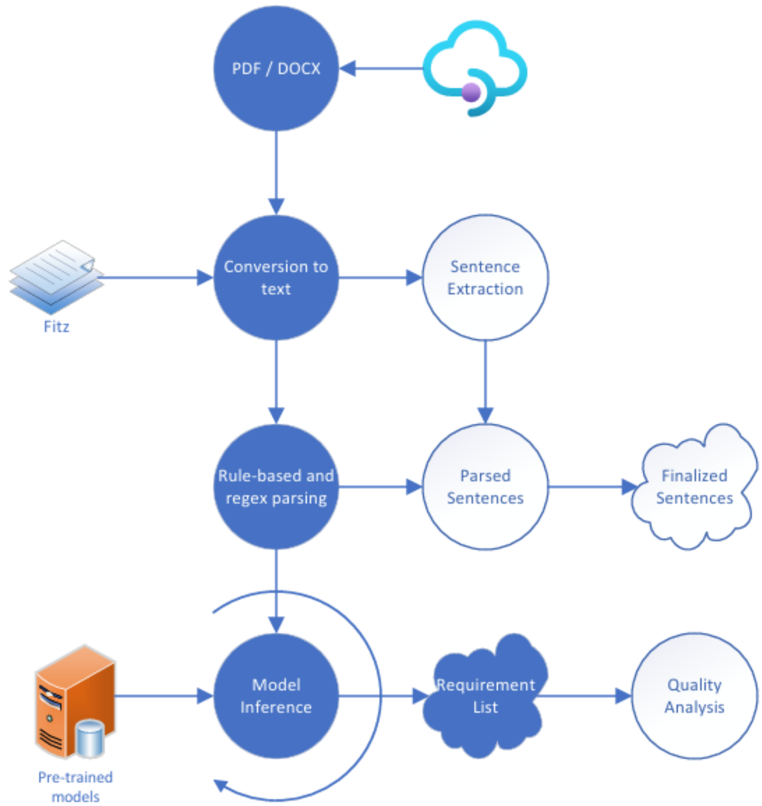

Figure 14 Requirement Identification Process

To accomplish model inference for requirement identification, the ARQM tool utilized existing pre-labeled datasets such as PURE to train models for automation. ARQM also utilized a wide range of different models and methods to achieve requirement identification. Primarily, the goal was to create a model that accurately identified requirements across different types of requirement documents. The PURE dataset contains thousands of entries of both requirement and non-requirement related text among a diverse range of requirement documents, allowing for adequate training of requirement identification. Table 14 discusses the different types of implementations used to achieve automated requirement identification (Ivanov et al., 2021)

Table 14 Algorithms Utilized for Requirement Identification

Model	**Explanation**
Logistic Regression	Logistic regression is utilized typically during categorical classifications, such as requirement identification as an alternative to linear regression. Similarly to linear regression, the hope was to utilize a simpler model while extracting out important textual features.
Naïve Bayes	Helpful when dealing with high-dimensional data and classification tasks. Naïve Bayes also took the textual statistics from TF-IDF to determine classification. Given the complicated dataset representations within the labeled dataset, Naïve Bayes served as a useful model implementation for classification of requirement text.
Support Vector Machine	This method helps to draw a hyperplane in high dimensional spaces; often SVMs can be trained on limited datasets. The SVM model can also help uncover new features within high dimensional data. This model was useful during the initial implementation of ARQM given dataset limitations.
Random Forest	An ensemble method that combines the predictions of multiple decision trees to perform classification. Ultimately, this model helps with more accurate predictions and feature engineering by utilizing multiple models.

KNN	Uses existing examples by finding the nearest Euclidean distance and providing a likely classification based on the closest neighbors of an entry. This algorithm can be helpful in the case of strong data separation, such as the PURE dataset.
Artificial Neural Network	Machine learning model that learns through labeled training data and back propagation. Primarily, ANNs help capture more complicated patterns that simpler models tend to miss, allowing for more complicated feature engineering.
Convolutional Neural Network	A machine learning algorithm that works well with images or text. The algorithm looks for patterns to help determine the classification result. Helpful to make determinations for blocked textual data in the form of statistics. For instance, arrays of vectors representing words in a requirement are useable within a CNN.
Recurrent Neural Network	A machine learning algorithm that primarily focuses on sequential data, allowing for the understanding of previous context within a given text. RNNs can help with classification through a holistic understanding of text based on order and key logic gates.
Ensemble-based Voting	This method takes the predictions of multiple models when considering the actual classification result. Helps to consider multiple models and feature representations of the data to improve accuracy, especially among diverse requirement datasets.

Meta Models	A way to train a model from the predictions of other models. For instance, training a logistic regression model based on the predictions of other pre-trained models and the associated results of the data. Helps to define the importance and contributions of models based on the textual representations of requirements.
BERT	An LLM that understands textual context through forward and backward embedding representations. BERT is a general purpose model that can be applied to requirement identification through fine-tuning. The BERT models are able to encapsulate critical requirement context and tacit knowledge relevant to make classification decisions.
BERT Siamese Network	A Siamese network is a way to help create a discriminator for positive and negative examples. During training, the Siamese network adjusts identical BERT models to create vector representations of requirements and non-requirements. When updated, the model learns useful vector representations that help to discriminate against positive or negative examples.

For requirement identification, each method specified has advantages and disadvantages for the automation process (see Table 14); while some implementations are light weight, more impactful implementations require more computational resources and broader data representation of requirements. The automation process does not discriminate across different types of requirements, such as nonfunctional and functional requirements. Instead, the process aims to find general textual representations of generic requirement text, simplifying the classification process. Ultimately, these models help to uncover features that are pivotal for requirement identification; these features can help achieve broad applicability and translatability to rule-based methods for faster inference, although complicated models may not fully encapsulate simplistic rule-based heuristics.

Although model inference is important for requirement identification, addressing the explanatory part related to the inference was even more crucial to finalizing some of the ARQM features. In particular, understanding the reasons behind model inference was important to help provide user feedback and gauge representations of critical words or phrases that represented requirements. Based on existing methods outlined in a systematic study, implementations for Explanatory Artificial Intelligence (XAI), a way to explain AI model inference, was pivotal to achieving most ARQM features. When considering the types of models utilized for requirement identification, different types of XAI methods could work for analyzing model decisions. For simplistic models such as linear and logistic regression, coefficients, and metadata about the model can help derive important features without model augmentation and analysis. More complicated methods require input manipulation, output analysis, and derivation of meta models that help synthetize the importance of features. The different types of XAI methods are defined in Table 15.

Table 15 XAI Methods for Analysis

XAI Method	Explanation
SHAP	Observes the results of input augmentation for the base model to generate feature importance for model inference. Helps to assign values of importance to all features through input changes.

Lime	Generates a meta model that helps explain the most important features for black box models. The meta model is generated and trained off of the predictions of the base model through micro input changes.
Anchors	This method looks at similar classes of examples, derives rule-based heuristics that hold true for most examples, and provides more intuitive explanations through the rule-based derivations.
Permutation Importance	This method shuffles features to determine their relative importance during model inference; the features are shuffled across the dataset to understand relative impact on the prediction.
Integrated Gradients	Creates a baseline gradient to gradually adjust to the input gradient. During the adjustment, the algorithm looks at the model inference results, allowing for the creation of a predictive gradient and the derivation of feature importance. Typically used for Neural Networks and Black Box models.
DeepLIFT	Utilizes a baseline gradient to help understand the difference between the models prediction for the original input and a base input; the results help to inform the back propagation process while also generating feature importance in a single inference pass.
Guided Backpropagation	A way to analyze the positive contributions of neurons through the backpropagation process. Guided backpropagation helps generate a

	heatmap of feature inputs based on positive contributions for inference.
Model Distillation	A way to extract the knowledge from a complex model to a simpler model that may be more interpretable. This method helps to synthesize the essential features of predictions from complex models while also mitigating the computation required for inference. Additionally, model distillation can help inform models that are directly observable to help derive critical rule-based features.
Rule extraction	Synthesis of model complexity into rule-based rules that generally work for inference. By simplifying to a rule-based representation of logic, intuitive explanations for model predictions become easier to implement.
Prototypes	Providing similar examples to the model inference results in helping to illustrate the reason behind inference. Typically, utilizing prototypes can help when visualizing common errors and problems associated with similar requirements through the use of human knowledge and intuition.
Counterfactuals	Helps to show how small changes could help impact model inference results. Typically, counterfactuals can illustrate potential input adjustment to help generate model results, instead of showing direct feature impact.
Partial Dependence Plot	A way to help understand and visualize the relationship among a single or pair of features and the related model output. This method is

	good for fast inference and interpretability for feature contribution in inference.
Attention Rollout	A way to analyze models such as transformers that utilize attention layers. The algorithm analyzes the multiple different attention layers, aggregates the importance in relation to the input, and creates a determination of feature importance.

For general requirement identification, there are adequate datasets for model training; the PURE dataset helps to provide models with a broad representation of requirements from a range of requirement documents, enhancing the potential for generalizable results during the requirement identification process. To accomplish requirement identification, ARQM utilized pre-existing libraries to parse requirement documents, synthesize the information into candidate sentences, and then utilize models trained from the PURE dataset to determine if a sentence was requirement-related or not. The requirements then serve as a necessary precursor to the quality analysis portion of the tool, utilizing other models to help analyze requirements off of certain quality attribute criteria. Through this implementation of requirement identification, the tool did not require manual human input for declaring requirements, significantly enhancing the automation of the ARQM tool.

Requirement Quality Analysis

As the natural next step from requirement identification, the ARQM tool also provides the ability to analyze the quality of requirements based on the IEEE 29148 standard. Like requirement identification, the quality analysis implementation for ARQM relies upon Artificial Intelligence and Machine Learning instead of rule-based heuristics. Because of the severe limitations of existing practical tools for requirement quality analysis, ARQM focuses on utilizing modern methods of Machine Learning to provide better, context-focused analysis for quality determinations. Due to the limitations of available datasets for requirement quality violations, ARQM utilized various data augmentation and generation methods to help simulate additional data.

Initially for dataset generation, ARQM utilized manually labeled datasets to help show proof-of-concept. The entries for the manual dataset were generated from extracted requirement documents. These datasets were labeled from the quality criteria definitions provided through the IEEE 29148 standard. Of the quality definitions selected, approximately two hundred examples were generated for each quality violation class. Data imbalances for each class were addressed through data pre-processing specific to the model analysis type. For instance, single classification models represented both a positive and negative class for the respective quality violations. For multi-label classification, equal representation of each representative class permutation was considered during data pre-processing. For instance, four quality classifications constituted sixteen classification classes.

Of the existing quality attributes specified in the IEEE 29148 standard, the ARQM tool focused on four primary violations. Specifically, the tool focuses on *singularity, feasibility, verifiability,* and *ambiguity*. These quality attributes were chosen due to their relative simplicity and applicability to single requirement quality analysis. Ultimately, the tool only analyzed single requirement violations, limiting the scope of quality analysis for other quality attributes and requirement groups entirely. Of the quality analysis data labeling process, the positive violation class considers lexical, syntactical, semantical, and pragmatic violations; ultimately, the tool's purpose is to showcase the ability for modern AI-based models to capture complicated quality violations that are not easily represented through heuristics.

As an added consideration for the quality attribute selection process, the ingestion tool did not contain the document metadata and structure that the Add-in application did. As a result, there was limited context and ability to analyze more complicated quality attributes. Primarily, the selection of quality attributes was made in a way to allow for the analysis of individual quality attributes without context. Singularity, for instance, is a quality attribute that is easily verifiable on a per-requirement-unit basis, as the violation pertains to the specification of multiple needs in one requirement. Heuristics among lexical, syntactical, and semantical considerations would be possible to learn through model training due to the simplicity of the quality attribute. In contrast, ambiguity is an important and widely-measured quality attribute for most quality analysis tools (Zhao et al., 2021). Additionally, there are many considerations of heuristics that are both lexical and syntactical that could widely pertain to ambiguity without the need for context (Carlson & Laplante, 2014). As mentioned in the IEEE 29148 standard, superlatives, subjective language, vague pronouns, and other types of patterns can indicate ambiguity; as a result, the quality attribute was well-formulated for NLP-based tasks ("ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering," 2018).

Verifiability and feasibility are generally only specified in the IEEE 29148 specification; however, many different quality models tend to encapsulate their essence. Ultimately, these quality attributes are also well-specified in heuristics. Common terminology and phrases allow the analysis process to make determinations from a high-level perspective without the need for context, making these quality attributes ideal for an ingestion-based software. Other quality attributes, such as completeness, would require further project-specific context that would have drastically impacted the tool reliability. For instance, adoption of some of these quality attributes would have added a necessity for metadata document processing. Due to the unstructured nature of requirement documents, this was unfeasible without user intervention (Alemneh & Berhanu, 2024; Lami et al., 2019; Umar & Lano, 2024).

Although the manual process of requirement labeling was initially utilized for model training, other methods were also utilized to help generate additional data. For instance, multiple data augmentation techniques were used that would alter requirements through basic or complicated manipulation strategies. For instance, the nlpaug library provides the ability to insert or replace words through context embeddings within a given text. By utilizing contextual embeddings, the sentence augmentations become meaningful and contextually relevant. By utilizing simplistic augmentations for sentences, generating synthetic datasets became possible, allowing models to generalize more to unseen data. Table 16 discusses the various data augmentation techniques available to help with synthetic data generation (Bayer et al., 2022; *Nlpaug 1.1.11 Documentation*, 2019).

Table 16 Data Augmentation Techniques

Method	**Explanation**	**Limitations**
Nlpaug Library	A library that utilizes context embeddings to more accurately substitute, insert, or delete words.	Replacing or inserting words could affect the quality violations present within the original text.
Pre-Processing	Removing irrelevant or low impact tokens	Most tokens are important for sequential analysis of requirements. For instance, the requirement identification number could help determine the granularity of analysis. Additionally, removal of punctuation and normalization of words could eliminate important context.
Synonyms	Replacing words in a sentence with a similar word.	This method could hurt the understanding and context of

	Through contextual embeddings and modern NLP methods, accurate replacement based on POS is possible	the text, limiting the ability of analytical techniques such as BERT. In particular, violations on a lexical or syntactical basis or vulnerable.
Language Back Translation	Converting a sentence to a different language and then converting the text back to the original language, allowing for a different textual representation.	Could eliminate all types of violations, including semantical and pragmatic violations. Language translation can correct violations based on model knowledge and assumptions, limiting the validity of the dataset.
Summarization	Augmenting the text by summarizing and extracting the key parts out. Helpful with both data augmentation and retaining context.	Similar to Language Back Translation, the process could augment the text in a way that eliminates critical context while filling in knowledge from model inference.
Noise Insert	Inserting random text into a sentence in a way that does not alter the understanding; useful when textual context is important for accurate classification.	This method may have a difficult time helping models converge on critical parts of the sentence necessary for quality analysis. Basic noise can help the model learn important patterns, but does not suffice as an adequate text manipulation technique for broader purposes.

LLM Data Generation	Utilizing LLMs to help modify sentences. LLMs can also help with the generation of synthetic data.	Similar to other techniques, data generation is vulnerable to knowledge injection and loss of context relative to the requirement document. Prompt engineering and manual validation can help validate the manipulation of data.

Due to the limitations of available data, ARQM relied on the broad manipulation of manual data through some of the previously mentioned techniques. Primarily, ARQM utilized the Nlpaug library to help manipulate existing labeled requirement text. Although this technique is powerful due to the utilization of embedding techniques, the sentence transformations were limited when trying to represent different types of quality violations. Often, transformations may create problems with interpretability and loss of context. Still, data augmentation did help the model generalize unseen data, even with imperfect data augmentation.

Architecture Overview

The ARQM system had many complicated implementations that were deeply inspired by existing research and the evolution of its development. Based on the previously discussed architecture for requirement identification, requirement quality analysis implemented both similar and differing methods to achieve automation of requirement analysis. As mentioned previously, the requirement analysis portion of the tool is a postprocessing step to requirement identification, focusing primarily on both classification and explanation-oriented steps for automation. The explanation focus of requirement quality analysis was crucial for the success of the tool. In particular, providing feedback based on the requirement determinations of the tool was a key factor for many of the base features that ARQM provided.

To provide an explanation to the user for quality violations, many methods were available. Of these existing methods, sequence-to-sequence models and LLMs initially provided promising results for quality violation explanations. Because sequence-to-sequence models required significant training data, multiple different techniques and implementations were utilized by ARQM to help facilitate better results. Of these techniques, prompt engineering, dataset consolidation and normalization, and fine-tuning were pivotal tasks to help with explanation generation. Prompt engineering, for instance, helped with focusing on the model to help hone the output accuracy, while also providing critical context, examples, and scopes for explanation accuracy. Dataset consolidation and normalization consisted of simplifying the explanation results to a small set of textual, categorical responses. Lastly, fine-tuning was used to help implement vast LLM knowledge into textual responses. Fine-tuning helped by limiting the amount of necessary data and by utilizing the tacit knowledge of the LLMs for a specific task. Although many of these solutions were valid implementations that showed promising results, the generalizability and consistency of output was a problem.

Figure 15 discusses the first implementation of the quality analysis method for LLMs. This implementation specifically focused on consolidating the complexity of input, while also guiding the model to the appropriate output based on manually defined data. Notice, the implementation shows that the model was responsible for both the classification of all quality attribute violations and their associated explanation. Figure 16 shows the evolution of the quality analysis process; specifically, the new architecture simplified the concerns of the LLMs by mitigating their classification and output responsibility. With the new implementation, each LLM would be responsible for only one quality attribute determination and the associated explanatory output.

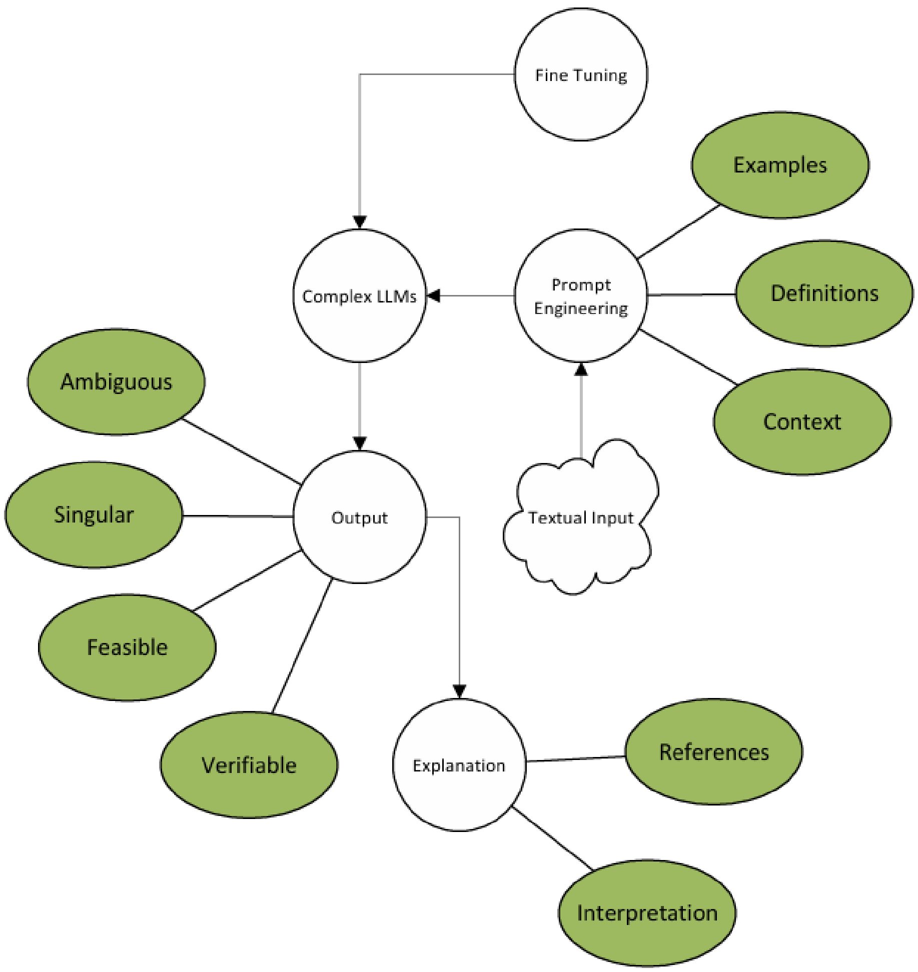

Figure 15 ARQM LLM Implementation Iter 1

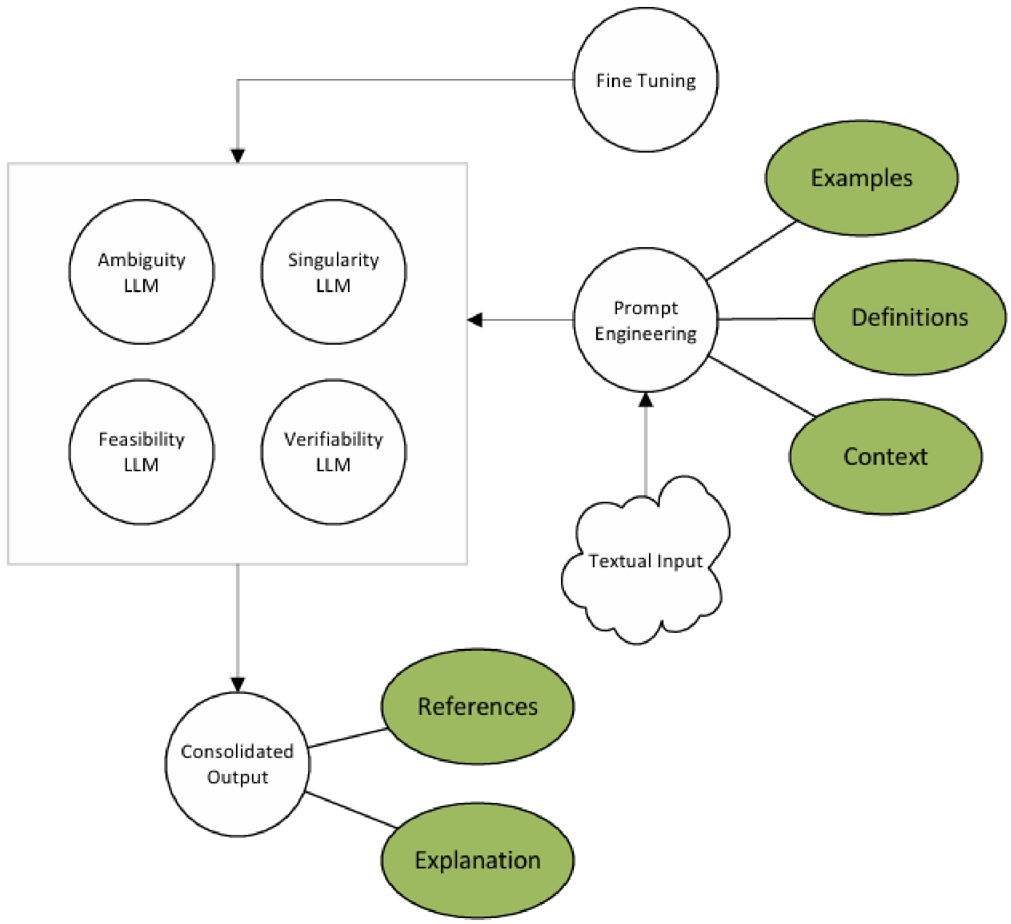

Figure 16 ARQM LLM Implementation Iter 2

The final iteration of the ARQM automation analysis focused primarily on further simplifying the output concerns of models. Specifically, this iteration focused on eliminating sequence generation entirely, while still focusing on generating explanations. As mentioned previously, XAI can help to analyze the reason for model decisions by analyzing their output. For instance, through the utilization of a pure classification model, XAI analytical techniques could help provide explanations of inferences without the need to generate text. Ultimately, the final iteration of the ARQM application relied on XAI models to generate explanations, significantly simplifying the model and dataset complexity requirements for quality analysis. Figure 17 shows the final implementation of the ARQM quality process. Through the utilization of SHAP, ARQM visualizes the importance of individual tokens for each quality violation without needing to utilize sequence generation models.

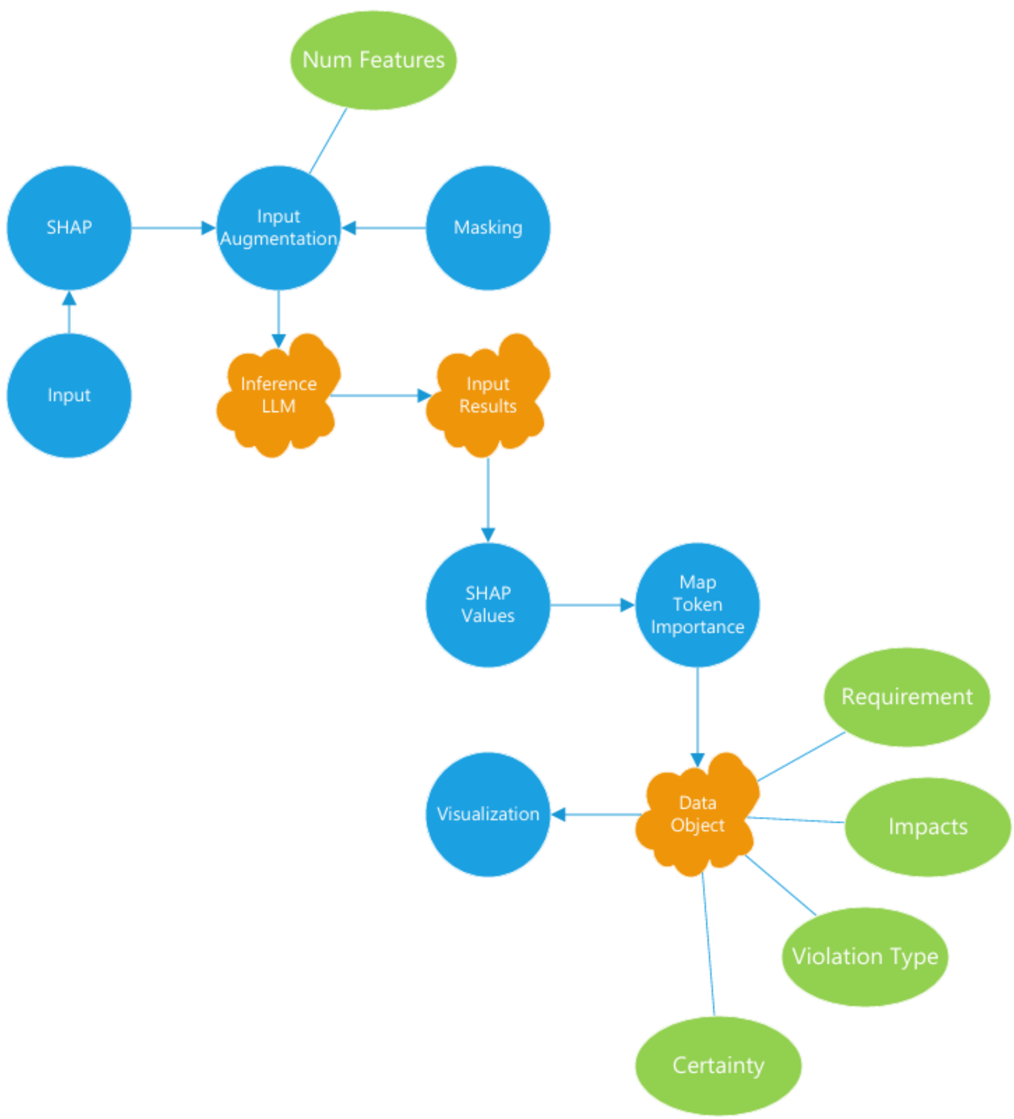

Figure 17 Final Quality Analysis Architecture

The implementation of XAI within the ARQM implementation was critical to achieving the completion of many features. Through the SHAP library, visualizing the reasoning behind the model inference became possible. By simply analyzing the model through input augmentation and XAI algorithms, the complexity of the training data was significantly reduced. SHAP utilizes input augmentation and game theory to help understand the impact of each input token for the model inference (details of implementation provided in

Table 15). Although the scope of the ARQM project for XAI was limited, other methods of XAI are likely to work with uncovering helpful visualizations of model decisions. Critically, the SHAP implementation provides token importance relative to the inference, allowing for visualization of high impact text (Lundberg, 2018).

ARQM Visualization

As the last component of the ARQM automation process, visualization of the results became an important consideration when providing explanations of quality violations to the user. For the current implementation of ARQM, visualization primarily utilizes a PDF generator to help visualize the results from the quality analysis process. During the inference and analysis stages of quality analysis, the system produces a structure that represents all of the quality violations and associated requirements. With the result object, ARQM generates a PDF that visualizes the overall number of quality violations, the requirements with quality violations, and the visualization of contributory text. The overall process of PDF generation is outlined in Figure 18. The process consolidates the results of requirement identification, quality analysis, and SHAP analysis to generate an aggregated PDF document. The PDF document result is then sent in the response of the API request.

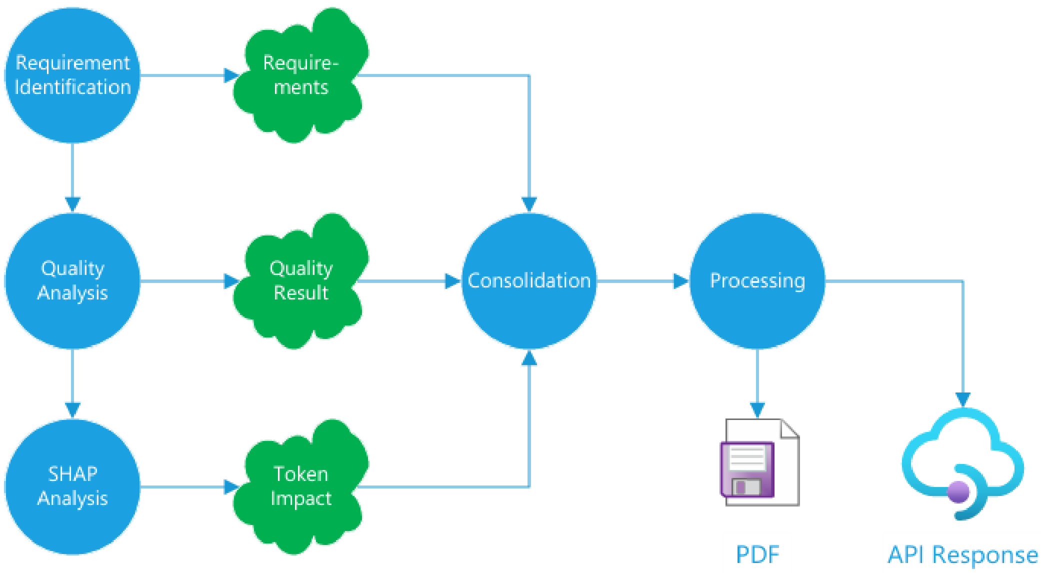

Figure 18 ARQM PDF Generation

As an example of the report results, Figure 19 shows an overview of the requirement document analysis for the *Reqview* requirement document. Based on the analysis, the document provides the number of violations for each quality attribute violation category. Figure 20 shows the quality analysis result of an individual requirement, stating that there were feasibility and singularity concerns related to the requirement text. The visualization shows the parts of the text that were most impactful for each quality attribute violation to help the user understand how to change the requirement for overall quality improvement. The average impact category shows the text that was most responsible and common across all of the relevant quality attribute violations. Additionally, the visualization shows the certainty of the model for a given quality violation, allowing the user to prioritize their requirement changes. Lastly, the report provides the visualization of the requirement for inline visualization of violations and requirement mapping.

From the SHAP value token mappings, the user can ultimately visualize the specific words and phrases of the requirement that contributed to the quality violation (Bus, 2021).

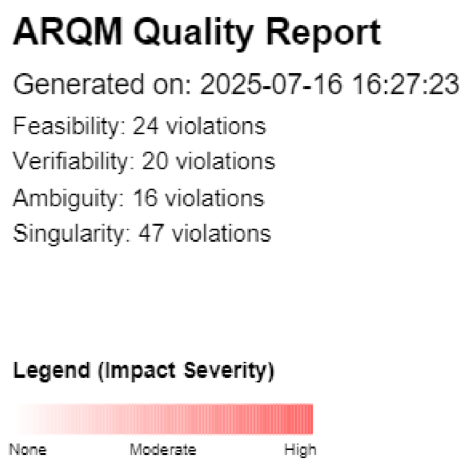

Figure 19 Aggregated ARQM Results

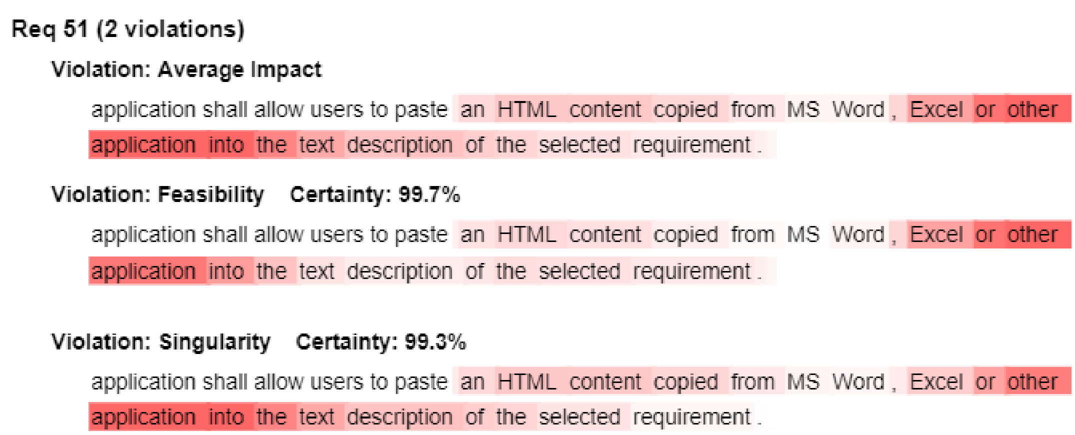

Figure 20 Quality Violation Visualization

The output of the ARQM application is meant to convey problems related to requirement text. Utilizing all three methods of requirement identification, quality analysis, and SHAP analysis, the tool generates a PDF file that visualizes the requirements that have potential quality violations. Ultimately, the tool visualizes quality violations for singularity, ambiguity, feasibility, and verifiability using XAI, a method for analyzing the reason behind model inference. The tool also analyzes the requirements on an individual basis without consideration of the entire document context, limiting the scope of analysis. To deal with potential information loss, the utilization of LLMs helps inject tacit knowledge into the inference process without the consolidation of a document structure. As an additional consideration, the tool does not measure the quality of requirement groups but does provide general statistics of violations to help indicate the potential document quality. Ultimately, the ARQM ingestion tool was designed for simplicity and verification of quality with minimal user input, improving upon the original design of the ARQM Add-in tool while also utilizing modern AI-based techniques.

Conclusion

The ARQM tool faced many challenges during the research and development process. Originally, the tool focused on providing a user interface and complicated session management using the Microsoft Add-in API. The Add-in implementation allowed users to declare requirements and requirement groups within a document, creating a document structure and mapping that was pivotal to requirement traceability. The format provided by the manual annotation of requirements would help to generate quality assessments of attributes for both individual requirements and requirement groups. The Add-in tool provided many different customization features and functionalities that allowed for more dynamic interaction and feedback during the analysis process. Additionally, features such as session management, custom prompts, and interactions allowed for significant customization of the requirement quality experience. Ultimately, the tool faced practical limitations and added complexity, creating the need for a simplistic ingestion-based tool.

The ARQM ingestion tool simplified the process of requirement automation by minimizing human input. To successfully develop the ingestion tool, requirement identification and quality analysis were important considerations for document processing. For requirement identification, defining rule-based methods in conjunction with the use of external libraries helped to extract sentences from requirement documents. Further research experimented with different models of varying types and complexity to accomplish the identification of requirements. These models utilized the PURE dataset, a publicly available dataset that provided requirement and non-requirement text across diverse requirement documents. Unlike quality analysis, requirement identification had a large availability of labeled datasets, significantly simplifying the training process.

During the ARQM ingestion quality analysis process, scarce dataset availability became a problem for model training. Originally, the models were trained on manually labeled datasets; as research progressed, different data augmentation tasks were utilized to help improve the generalizability of the dataset. The methods of data augmentation included the use of embeddings to help make changes to a given text while also considering pivotal context. Although many methods of data augmentation exist, the scope of the research was limited. As an alternative to dataset augmentation, multiple different models and techniques were used to help perform quality analysis. Initially, ARQM utilized sequence-to-sequence models with a focus on prompt engineering for providing quality analysis results. Due to the complexity of training data for sequence-to-sequence models, alternative methods were necessary to achieve requirement quality automation. As the application evolved, XAI became a pivotal task for visualizing the reasons behind model inference. Of the existing XAI implementations, the ARQM application utilized SHAP, a game theory-based method, to visualize the importance of tokens within the input sequence.

After the quality analysis process, ARQM focused on generating a readable, intuitive document that provided quality feedback information. The generated PDF includes aggregated statistics of each quality violation, the requirements with violations, and a visualization of contributory text for each violation. Additionally, the visualization also shows the average impact of tokens across the different quality attribute violations. Ultimately, the visualization utilized the XAI SHAP method to place importance behind each individual token for the model inference result. With the PDF document, the user can analyze the highlighted text, make changes to the associated requirement, and run the ARQM application again to mitigate quality concerns.

Chapter 5

ARQM Validation and Evaluation

Evaluation of the ARQM tool consists of two main components; first, the evaluation includes the performance of requirement identification; second, the evaluation analyzes the effectiveness of quality attribute detection across requirements. For both sections, the analysis includes a description of the methods, validation of results, and datasets for obtaining the associated results. Ultimately, this chapter shows the results across requirement identification for all generic requirement classifications, while also analyzing select quality attribute criteria previously discussed. For the general dataset transformations, all transformations were made after the Train/Test split to avoid leakage. Primarily, the Train/Test split created distinct, separate datasets that trained and validated the model. The statistics needed for model training were then derived from the training set instead of the test set to avoid data leakage and to validate the generalizability of the models. In many instances, the transformations did not affect data leakage regardless of augmentation patterns. For instance, BERT embeddings, stop words, and lemmatization were static transformations that did not learn from the corpus. Other methods such as TF-IDF and Counter Vectorizer were used on the corpus of training data with up to 1000 features; the experiment did test different values of representations to help converge on ideal model performance. Ultimately, the Train/Test split utilized a 70/30 ratio for training and validation.

For the PURE dataset, there were a total of 2833 requirements and 2475 non-requirement entries. During the training process, we utilize SMOTE oversampling of the minority class to address dataset disparities. Similarly, the reviewer datasets had a proportion of 222 requirements and 198 non-requirements based on user input. For ambiguity, the proportion was 162 Ambiguous requirements and 258 non-Ambiguous requirements. For feasibility, the proportion was 407 feasible requirements and 13 non-feasible requirements. Notably, the dataset was not balanced for feasibility, creating challenges with model training. The problem of dataset inbalance was mitigated through the use of methods such as class oversampling and undersampling. For singularity, the proportion was 321 singular requirements, and 99 non-singular requirements and verifiability had a proportion of 290 verifiable requirements and 130 non-verifiable requirements. Due to the high disparity of class representation for feasibility, the quality attribute has been mostly eliminated from the analysis. The other quality attributes utilized SMOTE to help balance the datasets.

Research Questions

For requirement identification, the methodology includes research questions that mainly relate to the practical performance of the models and the associated uncovered features that boosted inference results. A key part of the analysis is the generalizability of the requirement identification process to both human-annotated data and the PURE dataset. Because the PURE dataset includes requirements from differing requirement documents, the focus includes how well the ARQM tool can generalize requirement and non-requirement text across different documents and requirement representations. Therefore, the following research questions for the requirement identification component are proposed:

1. *How can a practical tool identify requirement-related text in differing unstructured documents?*
2. *How can existing tools and methods address the lack of training data for requirement identification?*

3. *How effectively can the best algorithm identify requirements based on accuracy, precision, recall, and generalization across different documents?*
4. *What are the roles of lexical, syntactical, and semantical textual features in performing requirement identification?*
5. *What features were useful during the model inference process for requirement identification?*

Next, the analysis shifts to questions over requirement quality. Due to the complex nature of the quality attributes specified in the IEEE 29148 standard, the analysis only focuses on certain quality attributes that generally do not require document or requirement context outside of the textual unit. Still, the analysis performs higher-order language processing for classification but is limited by the tacit knowledge and intuition of the associated models via transfer learning. Lastly, the analysis explores means of visualizing requirement quality defects as a means of practical application for the ARQM tool. The following research questions are proposed:

1. *How can a practical tool identify quality defects in unstructured documents based on various quality attributes outlined in the IEEE 29148:2018 standard?*
2. *What quality attributes are measurable with AI-based methods?*
3. *What role does context play in determining the quality of a requirement across the various quality attributes?*
4. *How can existing tools and methods address the lack of training data for requirement quality analysis?*
5. *How effectively can the best algorithm identify requirement quality defects based on accuracy, precision, recall, and generalization across different documents?*
6. *What methods exist to help with the generation of explanations for quality inferences?*

Requirement Identification

In this section, the analysis looks at two primary datasets; for the first dataset, the section analyzes the results of the ARQM tool classification results for the PURE dataset. For the second dataset, the analysis focuses on the tool's accuracy for the human-annotated dataset, along with rater agreement and other statistical considerations. The classification process focuses on two classes, requirement and non-requirement labels. After the analysis of the classification, the section discusses the common features that helped achieve inference through various XAI methods discussed previously.

PURE Dataset Analysis for Requirement Identification

As previously discussed, the PURE dataset has diverse requirement representations with thousands of classified examples for requirement and non-requirement-related text. Many studies base their results primarily on this dataset due to its broad representation of requirement text classifications. Primarily, the dataset synthesizes all requirement types, functional and nonfunctional, into a single requirement category, while all other text is considered non-requirement text for classification purposes The results show the Precision, Recall, F1-Score, and Accuracy of each associated model and its inference class. Equation 1 Accuracy, Equation 2 Precision, Equation 3 Recall, and Equation 4 F1-Score highlight the mathematical formulas behind each metric (*Classification*, 2025).

Accuracy helps to measure the number of correct predictions relative to all predictions made by the model. Ultimately, accuracy is an important metric to understand the overall classification abilities of the model with respect to all classification possibilities. Precision helps to understand the abilities of the model for each inference class while Recall uncovers the accurate inferences in relation to the entire inference class. Lastly, F1-Score helps to achieve the harmonic precision of precision and recall as a higher analysis for model performance. These metrics serve as a critical part of model evaluation and are therefore utilized when analyzing the various models for requirement identification.

Equation 1 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation 2 Precision

$$Precision = \frac{TP}{TP + FP}$$

Equation 3 Recall

$$Recall = \frac{TP}{TP + FN}$$

Equation 4 F1-Score

$$Recall = 2 * \frac{Precision * Recall}{Precision + Recall}$$

For the analysis provided in Table 17, the results vary based on the two inference classes. Class *N* indicates that a text was not requirement-related, where *Y* indicates that the text was a requirement. For the analysis, multiple different models of varying complexity were used on the PURE dataset. Primarily, simpler models such as Logistic Regression, KNN, and SVM were used as basic, statistical models for the result and analysis. Deep Learning models were also utilized for the analysis, including ANNs, CNNs, and RNNs; for more complicated analysis, LLMs were utilized in the form of transfer learning. Additionally, ensemble methods were used to help enhance the generalizability of the inference results. The tacit knowledge of the ensemble models was then synthesized into meta-models by generating training data from the predictions of the ensemble models. Instead of utilizing average or majority methods to determine a classification, the meta-model learned the resulting patterns of each model, their importance, and the appropriate patterns for inference. To help consolidate on an average, each experiment included 10 model results, and the associated average and standard deviation is included. To help improve the randomness and assessment of the models, the data was generated off a different data randomization seed for each run, allowing for the training and convergence of models on unseen data. Due to computational limitations, 10 results were chosen as a reasonable limitation to help offer insight into model results. Additionally, the dataset utilized minority class undersampling to help even the data for all models. Because of the large entries for the PURE dataset, a 20% data selection technique was ultimately used to help mitigate model training overhead. Further analysis is provided in the form of significance testing to help understand which models likely outperformed. Still, further research should generate more results to help broadly understand model performance.

Table 17 PURE Requirement Identification Results

Model	Class	Precision	Recall	F1-Score	Accuracy
ANN	N	0.83 ± 0.01	0.91 ± 0.03	0.87 ± 0.02	0.86 ± 0.01
CNN	N	**0.91 ± 0.01**	0.94 ± 0.01	**0.92 ± 0.00**	**0.92 ± 0.01**
XLNet	N	0.83 ± 0.06	0.83 ± 0.05	0.83 ± 0.03	0.83 ± 0.03
DistilBERT	N	0.90 ± 0.01	0.94 ± 0.01	**0.92 ± 0.01**	**0.92 ± 0.01**
DistilBERT Few-Shot	N	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.86 ± 0.01
DistilBERT Siamese	N	0.75 ± 0.07	0.72 ± 0.16	0.72 ± 0.10	0.75 ± 0.06
Ensemble (Average)	N	0.86 ± 0.01	0.95 ± 0.01	0.90 ± 0.01	0.90 ± 0.01
Ensemble (Majority)	N	0.85 ± 0.01	0.94 ± 0.00	0.89 ± 0.01	0.89 ± 0.01
KNN	N	0.78 ± 0.02	0.91 ± 0.02	0.84 ± 0.01	0.83 ± 0.01
Logistic Regression	N	0.86 ± 0.01	0.92 ± 0.00	0.89 ±0 .01	0.88 ± 0.01
Meta-Model (Gradient Boost)	N	0.87 ± 0.02	0.95 ± 0.02	0.91 ± 0.01	0.91 ± 0.01
Meta-Model (Logistic Regression)	N	0.90 ± 0.02	0.94 ± 0.01	**0.92 ± 0.01**	0.91 ± 0.01
Random Forest	N	0.89 ± 0.02	**0.96 ± 0.01**	0.86 ± 0.01	**0.92 ± 0.01**

RNN	N	**0.91 ± 0.01**	0.94 ± 0.01	**0.92 ± 0.01**	**0.92 ± 0.01**
Support Vector Machine	N	0.85 ± 0.00	0.92 ± 0.01	0.89 ± 0.00	0.88 ± 0.00
XLNet	Y	0.83 ± 0.04	0.83 ± 0.07	0.83 ± 0.03	0.83 ± 0.03
ANN	Y	0.91 ± 0.02	0.81 ± 0.02	0.86 ± 0.01	0.86 ± 0.01
CNN	Y	0.94 ± 0.01	**0.91 ± 0.02**	**0.92 ± 0.01**	**0.92 ± 0.01**
DistilBERT	Y	0.94 ± 0.03	0.90 ± 0.01	**0.92 ± 0.00**	**0.92 ± 0.01**
DistilBERT Few-Shot	Y	0.87 ± 0.00	0.87 ± 0.00	0.87 ± 0.01	0.86 ± 0.01
DistilBERT Siamese	Y	0.77 ± 0.09	0.78 ± 0.10	0.77 ± 0.05	0.75 ± 0.06
Ensemble (Average)	Y	0.94 ± 0.02	0.85 ± 0.01	0.89 ± 0.01	0.90 ± 0.01
Ensemble (Majority)	Y	0.93 ± 0.01	0.84 ± 0.01	0.88 ± 0.00	0.89 ± 0.01
KNN	Y	0.90 ± 0.02	0.75 ± 0.02	0.82 ± 0.02	0.83 ± 0.01
Logistic Regression	Y	0.92 ± 0.01	0.84 ± 0.01	0.88 ± 0.01	0.88 ± 0.01
Meta-Model (Gradient Boost)	Y	0.95 ± 0.02	0.86 ± 0.03	0.90 ± 0.01	0.91 ± 0.01
Meta-Model (LR)	Y	0.93 ± 0.01	0.89 ± 0.02	0.91 ± 0.01	0.91 ± 0.01
Random Forest	Y	**0.96 ± 0.02**	0.88 ± 0.02	0.91 ± 0.01	**0.92 ± 0.01**
RNN	Y	0.94 ± 0.01	**0.91 ± 0.02**	**0.92 ± 0.01**	**0.92 ± 0.01**
Support Vector Machine	Y	0.91 ± 0.01	0.85 ± 0.00	0.88 ± 0.00	0.88 ± 0.01

As part of the analysis, Table 17 shows the models utilized for requirement identification and the associated classification results. Random Forest, RNN, DistilBERT and CNN all outperformed for accuracy with minimum variation at 0.92. The other models such as Logistic Regression, SVM, and the Meta-Models performed similarly as well, suggesting an ability for simplistic models to perform on par with more complicated models for the same accuracy performance. For the F1-Score, RNN , DistilBERT and CNN outperformed for the positive class, whereas RNN, LR Meta-Model, DistilBERT, and CNN outperformed for the negative class, all with a score of 0.92. With the exception of the DistilBERT Siamese implementation, the F1 score was relatively consistent across models. For recall, RNN and CNN outperformed for the positive class with a score of 0.91, whereas other models like Distilbert and the Meta-Models performed similarly. For the negative class, Random Forest outperformed with a surprising performance of 0.96. The Meta-Models and CNN implementation performed similarly to Random Forest as well. Lastly, Random Forest outperformed for Precision for the positive class whereas CNN outperformed for the negative class. Overall, the results seem to indicate that most models were able to broadly learn and generalize across the data.

Figure 21 Inference for Non-Requirements

Figure 21 shows a clear picture of the success of model inference for identifying non-requirement text among the various models. Notably, model performance remained relatively consistent across statistical and complicated models. For a few instances, certain models achieved better results depending on the metric. For example, CNN and RNN outperformed for precision; Average Ensemble, GB Meta-Model, and Random Forest outperformed for recall; CNN, Random Forest, and GB Meta-Model outperformed for F1-Score; and the most accurate models were CNN, DistilBERT, Random Forest and RNN. For the positive class results outlined in Figure 22, the results show similar clustering and performance across models. With Random Forest achieving a precision of 0.96, CNN and RNN achieving a recall of 0.91, XLNet, Distilbert, and

RNN achieving an F1-Score of 0.92, and multiple models achieving an accuracy of 0.92, most of the statistical models learned the data representations for the positive class as well.

Figure 22 Inference for Requirements

For significance testing, the results showcase the various model groupings and a 95% confidence interval. The testing utilizes Tukey, which is a way to determine which groups are significantly different. If groups share the same letter, they are not necessarily significantly different. Figure 23 shows the results for the negative class. For precision, CNN, BERT, Logistic Regression, Random Forest, and Gradient Boost performed the best in comparison to XLNET and below based on mean results. CNN outperformed RNN and below with a mean of 0.91 and a single grouping of A. The worst performing models included XLNET, ANN, KNN, Siamese

BERT, and Naïve Bayes. For recall, Random Forest outperformed RNN and below, but most models were in the same statistical grouping. The worst performing models were Naïve Bayes and below based on mean score. Next, CNN and Random Forest outperformed ANN and below for recall. Notably, recall had a variety of statistical groups, suggesting stratified performance based on model. The worst performing models were XLNet, Naïve Bayes, and the Siamese implementation. Lastly, CNN against outperformed for accuracy, beating Majority Vote Ensemble and below. BERT, Random Forest, Logistic Regression, Gradient Boost, and Averaging Ensemble performed statistically similar to CNN, but had lower overall means based on the experimental data. Overall, CNN and Random Forest seemingly outperformed for the negative class.

For the positive class (Figure 24), different statistical results were achieved. For precision, there was common top performer grouping with Random Forest to Logistic Regression. Random Forest performed better than ANN and below. For recall, CNN to Gradient Boost outperformed, with CNN achieving the highest average mean and outperforming Averaging Ensemble and below. F1 showed similar results with CNN outperforming Average Ensemble and below. BERT, Random Forest, Logistic Regression, and gradient boost performed statistically similarly. The positive class seemingly consolidated top results for the Random Forest, CNN, BERT, and Logistic Regression models. Other models such as RNN, ANN, and Naïve Bayes performed statistically worse.

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 0)

Factor	N	Mean	Grouping
CNN	10	0.91066	A
BERT	10	0.90021	A B
LOG REG ENSEMBLE	10	0.89634	A B C
Random Forest	10	0.88539	A B C D
Gradient Boost	10	0.87202	A B C D E
RNN	10	0.86756	B C D E
Average Ensemble	10	0.86001	B C D E
SVM	10	0.85632	C D E
LOG REG META	10	0.85610	C D E
MAJ. VOTE ENSEMBLE	10	0.85281	D E
BERT FEW SHOT	10	0.85179	D E
XLNET	10	0.8327	E
ANN	10	0.83107	E
KNN	10	0.78343	F
Siamese	10	0.7549	F
Naive Bayes	10	0.74879	F

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.96159	A
Gradient Boost	10	0.95161	A B
Average Ensemble	10	0.94846	A B
MAJ. VOTE ENSEMBLE	10	0.94051	A B
BERT	10	0.93905	A B
CNN	10	0.93901	A B
LOG REG ENSEMBLE	10	0.93547	A B
SVM	10	0.92170	A B C
LOG REG META	10	0.92167	A B C
ANN	9	0.91802	A B C
KNN	10	0.91737	A B C
RNN	10	0.88966	B C D
Naive Bayes	10	0.86310	C D
BERT FEW SHOT	10	0.85480	C D
XLNET	10	0.8336	D
Siamese	10	0.7190	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 0)

Factor	N	Mean	Grouping
CNN	10	0.92451	A
Random Forest	10	0.92177	A
BERT	10	0.91913	A B
LOG REG ENSEMBLE	10	0.91533	A B
Gradient Boost	10	0.90972	A B C
Average Ensemble	10	0.90191	A B C
MAJ. VOTE ENSEMBLE	10	0.89438	A B C D
SVM	10	0.88769	A B C D E
LOG REG META	10	0.88758	A B C D E
RNN	10	0.87824	B C D E
ANN	10	0.87034	C D E F
BERT FEW SHOT	10	0.85328	D E F
KNN	10	0.84500	E F
XLNET	10	0.83077	F G
Naive Bayes	10	0.80159	G
Siamese	10	0.7230	H

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 0)

Factor	N	Mean	Grouping
CNN	10	0.92320	A
BERT	10	0.91865	A B
Random Forest	10	0.91843	A B
LOG REG ENSEMBLE	10	0.91376	A B C
Gradient Boost	10	0.90588	A B C D
Average Ensemble	10	0.89724	A B C D
MAJ. VOTE ENSEMBLE	10	0.88935	B C D E
SVM	10	0.88348	C D E
LOG REG META	10	0.88330	D E
RNN	10	0.87676	D E
ANN	10	0.86423	E
BERT FEW SHOT	10	0.86357	E
KNN	10	0.83192	F
XLNET	10	0.83059	F
Naive Bayes	10	0.78655	G
Siamese	10	0.7540	H

Means that do not share a letter are significantly different.

Figure 23 Significance Testing for PURE (Class 0)

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.95804	A
Gradient Boost	10	0.94748	A B
Average Ensemble	10	0.94301	A B C
BERT	10	0.93820	A B C
CNN	10	0.93693	A B C
MAJ. VOTE ENSEMBLE	10	0.93420	A B C
LOG REG ENSEMBLE	10	0.93294	A B C
SVM	10	0.91532	A B C D
LOG REG META	10	0.91519	A B C D
ANN	10	0.90561	B C D
KNN	10	0.90066	C D
RNN	10	0.88691	D
BERT FEW SHOT	10	0.87388	D E
Naive Bayes	10	0.83888	E
XLNET	10	0.8345	E
Siamese	10	0.7748	F

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 1)

Factor	N	Mean	Grouping
CNN	10	0.90726	A
BERT	10	0.89905	A B
LOG REG ENSEMBLE	10	0.89253	A B C
Random Forest	10	0.87547	A B C D
BERT FEW SHOT	10	0.87116	A B C D
RNN	10	0.86420	A B C D E
Gradient Boost	10	0.86082	A B C D E
Average Ensemble	10	0.84668	B C D E
SVM	10	0.84543	C D E
LOG REG META	10	0.84498	C D E
MAJ. VOTE ENSEMBLE	10	0.83884	D E
XLNET	10	0.8258	D E F
ANN	10	0.81427	E F
Siamese	10	0.7846	F G
KNN	10	0.74673	G H
Naive Bayes	10	0.71027	H

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 1)

Factor	N	Mean	Grouping
CNN	10	0.92173	A
BERT	10	0.91812	A B
Random Forest	10	0.91472	A B
LOG REG ENSEMBLE	10	0.91212	A B C
Gradient Boost	10	0.90164	A B C D
Average Ensemble	10	0.89206	B C D E
MAJ. VOTE ENSEMBLE	10	0.88380	C D E F
SVM	10	0.87885	D E F
LOG REG META	10	0.87857	D E F
RNN	10	0.87517	D E F
BERT FEW SHOT	10	0.87250	E F
ANN	10	0.85724	F
XLNET	10	0.8275	G
KNN	10	0.81633	G
Siamese	10	0.7731	H
Naive Bayes	10	0.76879	H

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 1)

Factor	N	Mean	Grouping
CNN	10	0.92320	A
BERT	10	0.91865	A B
Random Forest	10	0.91843	A B
LOG REG ENSEMBLE	10	0.91376	A B C
Gradient Boost	10	0.90588	A B C D
Average Ensemble	10	0.89724	A B C D
MAJ. VOTE ENSEMBLE	10	0.88935	B C D E
SVM	10	0.88348	C D E
LOG REG META	10	0.88330	D E
RNN	10	0.87676	D E
ANN	10	0.86423	E
BERT FEW SHOT	10	0.86357	E
KNN	10	0.83192	F
XLNET	10	0.83059	F
Naive Bayes	10	0.78655	G
Siamese	10	0.7540	H

Means that do not share a letter are significantly different.

Figure 24 Significance Testing for PURE (Class 1)

PURE Dataset Preprocessing Considerations

Although minimal data preprocessing manipulation techniques were used during the analysis, the TF-IDF algorithm was utilized for simplistic and statistical-based models, while basic tokenization was used for models like CNNs and RNNs. Lastly, the analysis utilized native tokenization methods for the various LLMs implemented for testing. Ultimately, data preprocessing techniques could eliminate critical context necessary for determining requirement classifications. Models such as BERT, RNNs, and CNNs rely on context, making data preprocessing difficult. Table 18 shows the results of the basic models before and after the utilization of lemmatization as a preprocessing technique in conjunction with TF-IDF.

Importantly, the process applies the data augmentation transformations after the Train/Test split as well for validation purposes. Notice, minimal improvement was achieved through Lemmatization, while many instances of performance decrease were observed. For certain models like Logistic Regression, there were marginal improvements across Precision, Recall, F1-Score, and Accuracy. For Logistic Regression, there were generally no decrease in improvement across the statistics. Further, KNN and SVM saw similar marginal performance increases, suggesting that statistical models may benefit from lemmatization.

*Table 18 PURE Requirement Identification Results Lemmatization (Before Vs. After Method Applied) * Data Augmentation Applied After Train/Test Split*

Model	**Class**	**P_1**	**P_2**	**R_1**	**R_2**	**F1_1**	**F1_2**	**A_1**	**A_2**
ANN	N	0.89 ± 0.02	0.83 ± 0.01	0.93 ± 0.02	0.91 ± 0.03	0.91 ± 0.01	0.87 ± 0.02	0.91 ± 0.01	0.86 ± 0.01
KNN	N	0.78 ± 0.02	0.78 ± 0.02	0.92 ± 0.02	0.91 ± 0.02	0.84 ± 0.01	0.84 ± 0.01	0.83 ± 0.01	0.83 ± 0.01
Logistic Regression	N	**0.85 ± 0.02**	**0.86 ± 0.01**	**0.91 ± 0.02**	**0.92 ± 0.00**	**0.88 ± 0.02**	**0.89 ± 0 .01**	0.88 ± 0.01	0.88 ± 0.01
Random Forest	N	0.89 ± 0.02	0.89 ± 0.02	0.96 ± 0.01	0.96 ± 0.01	0.92 ± 0.01	0.86 ± 0.01	0.92 ± 0.01	0.92 ± 0.01
SVM	N	0.86 ± 0.02	0.85 ± 0.00	0.92 ± 0.01	0.92 ± 0.01	0.89 ± 0.02	0.89 ± 0.00	0.88 ± 0.01	0.88 ± 0.00
ANN	Y	0.93 ± 0.02	0.91 ± 0.02	0.89 ± 0.03	0.81 ± 0.02	0.91 ± 0.01	0.86 ± 0.01	0.91 ± 0.01	0.86 ± 0.01
KNN	Y	0.90 ± 0.02	0.90 ± 0.02	**0.73 ± 0.02**	**0.75 ± 0.02**	**0.81 ± 0.01**	**0.82 ± 0.02**	0.83 ± 0.01	0.83 ± 0.01
Logistic Regression	Y	**0.91 ± 0.01**	**0.92 ± 0.01**	0.84 ± 0.02	0.84 ± 0.01	**0.87 ± 0.01**	**0.88 ± 0.01**	0.88 ± 0.01	0.88 ± 0.01
Random Forest	Y	0.96 ± 0.01	0.96 ± 0.02	0.88 ± 0.02	0.88 ± 0.02	0.92 ± 0.01	0.91 ± 0.01	0.92 ± 0.01	0.92 ± 0.01
SVM	Y	0.91 ± 0.01	0.91 ± 0.01	**0.84 ± 0.02**	**0.85 ± 0.00**	0.88 ± 0.01	0.88 ± 0.00	0.88 ± 0.01	0.88 ± 0.01

As an added consideration of data inference, the utilization of stop words could also potentially hurt the necessary context for some of the various statistical models to achieve inference; to understand the impact of stop words, all non-ML models were compared with and without stop words during training. Like before, the data was augmented after the Train/Test split to avoid leakage. The goal of this comparison was to understand the impact of removing common words for requirement text, as requirements typically contain common syntax and, by extension, common words. As Table 19 shows, there was a lack of improvement across most models, with exception of ANN, KNN, and Random Forest. For most of the models, there were decreases in performance, suggesting that the removal of stop words did not help improve the results of the statistical models. For ANNs, there were increases across all statistics when utilizing stop words.

Table 19 PURE Requirement Identification Results Stop Words (Before Vs. After Method Applied)
** Data Augmentation Applied After Train/Test Split*

Model	Class	P_1	P_2	R_1	R_2	F1_1	F1_2	A_1	A_2
ANN	N	**0.83 ± 0.01**	**0.88 ± 0.02**	**0.91 ± 0.03**	**0.93 ± 0.02**	**0.87 ± 0.02**	**0.90 ± 0.01**	**0.86 ± 0.01**	**0.90 ± 0.01**
KNN	N	0.78 ± 0.02	0.77 ± 0.03	**0.91 ± 0.02**	**0.92 ± 0.01**	0.84 ± 0.01	0.84 ± 0.02	0.83 ± 0.01	0.82 ± 0.02
Logistic Regression	N	0.86 ± 0.01	0.83 ± 0.02	0.92 ± 0.00	0.91 ± 0.01	0.89 ±0 .01	0.87 ± 0.01	0.88 ± 0.01	0.86 ± 0.01
Random Forest	N	0.89 ± 0.02	0.89 ± 0.03	0.96 ± 0.01	0.94 ± 0.01	**0.86 ± 0.01**	**0.91 ± 0.02**	0.92 ± 0.01	0.91 ± 0.02
SVM	N	0.85 ± 0.00	0.83 ± 0.02	0.92 ± 0.01	0.91 ± 0.02	0.89 ± 0.00	0.87 ± 0.01	0.88 ± 0.00	0.86 ± 0.01
ANN	Y	**0.91 ± 0.02**	**0.93 ± 0.02**	**0.81 ± 0.02**	**0.87 ± 0.03**	**0.86 ± 0.01**	**0.90 ± 0.01**	**0.86 ± 0.01**	**0.90 ± 0.01**
KNN	Y	0.90 ± 0.02	0.90 ± 0.01	0.75 ± 0.02	0.73 ± 0.03	0.82 ± 0.02	0.81 ± 0.02	0.83 ± 0.01	0.82 ± 0.02
Logistic Regression	Y	0.92 ± 0.01	0.90 ± 0.02	0.84 ± 0.01	0.82 ± 0.02	0.88 ± 0.01	0.85 ± 0.02	0.88 ± 0.01	0.86 ± 0.01
Random Forest	Y	0.96 ± 0.02	0.94 ± 0.02	0.88 ± 0.02	0.88 ± 0.03	0.91 ± 0.01	0.91 ± 0.02	0.92 ± 0.01	0.91 ± 0.02
SVM	Y	0.91 ± 0.01	0.90 ± 0.02	0.85 ± 0.00	0.81 ± 0.02	0.88 ± 0.00	0.85 ± 0.02	0.88 ± 0.01	0.86 ± 0.01

PURE Dataset Features for Requirement Identification

As part of the analytical process, understanding the features responsible for the model inference is crucial; through the utilization of these features, deriving the reasons behind model inference, in the form of various XAI methods, becomes possible. For the statistical models, the translation of the associated coefficients to word importance can help uncover important features. For more complicated models, the analysis process synthesizes the knowledge of the DL models into a meta-model to help uncover important features. Although analyzing DL models is possible through other methods, such as Integrated Gradient or SHAP implementations, a baseline analysis of word contribution can suffice for feature engineering at the lexical and syntactical level of requirement identification. Table 20 and Table 21 shows various results among the models for the highest impact words among both classes.

Table 20 Model-Specific Word Impact Results (Class 0)

Model	Classification	Impact 1	Impact 2	Impact 3	Impact 4	Impact 5
Linear Regression	Negative	processes	permissive	low	figure	administrati
Logistic Regression	Negative	processes	requirements	specification	administration	xml
Naïve Bayes	Negative	nanc	transportation	xml	layer	srs
SVM	Negative	processes	administration	release	found	specificatior
Meta Random Fores	Negative	processes	requirements	specification	administration	xml
Meta KNN	Negative	processes	administration	well	document	online
Meta ANN	Negative	processes	specification	administration	requirements	them
Meta CNN	Negative	administration	deliverable	describes	file	amount
Meta RNN	Negative	administration	document	requirements	file	police

For the negative class, there are consistent results among the topmost impactful words; specifically, the highest impact words among the various models include “processes” and “administration” for non-requirement classification. Although these words often occur in requirement-based text, their collective contribution skewed towards a negative classification, suggesting the importance of sequences, context, and order instead of individual word units. As indicated by Figure 25, models relied on many common words for classification, suggesting a potential pattern among non-requirement text that is transferable to rule-based methods.

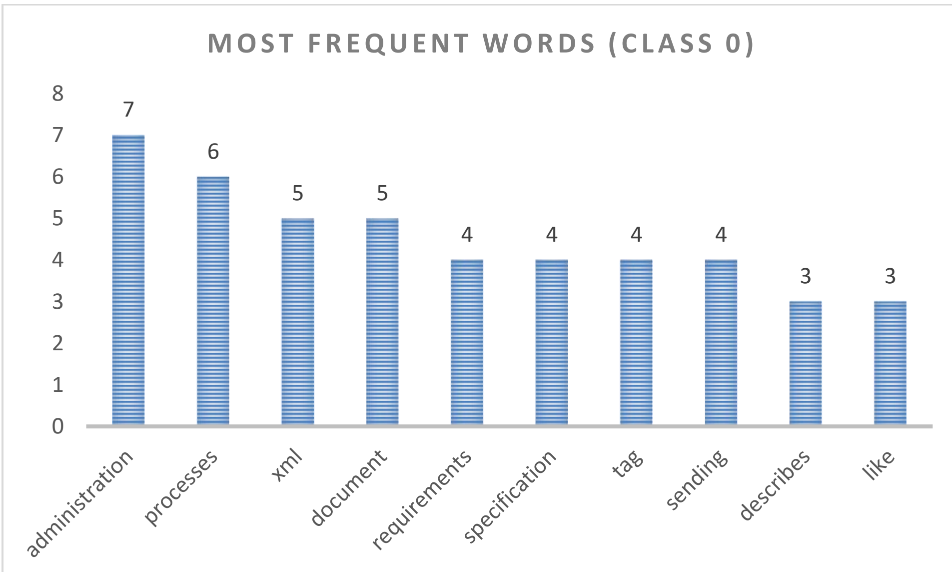

Figure 25 Most Frequent Words for Non-Requirements

For the positive class, Table 21 Model-specific Word Impact Results (Class 1) shows the top common words associated with requirement identification across the various models. According to the results, requirement-related text shares more overlap for common words during the classification process. Words such as ‘shall’ and ‘should’ were primary indicators of requirement-related text. In a similar way to the negative class, these indicators seemingly work in conjunction as contributory factors to the positive class; their accuracy likely improves in specific sequences, patterns, or in the instance of multiple common word indicators.

Table 21 Model-specific Word Impact Results (Class 1)

Model	Classification	Impact 1	Impact 2	Impact 3	Impact 4	Impact 5
Logistic Regression	Positive	shall	should	must	option	able
Naïve Bayes	Positive	rr3	previously	smss	shall	137
SVM	Positive	shall	should	must	option	able
Meta Random Fores	Positive	shall	shall	should	must	option
Meta KNN	Positive	shall	able	should	plant	must
Meta ANN	Positive	shall	should	must	able	previously
Meta CNN	Positive	shall	should	must	account	acquisition
Meta RNN	Positive	shall	should	must	display	able
Meta BERT	Positive	shall	will	must	should	may

Figure 26 shows the words that were the most responsible for the positive classification. As mentioned previously, words such as 'shall' and 'should' were indicators, along with other common requirement terminology. Notably, requirement common words constitute a certain formality or obligation. Words such as 'should' and 'able' imply language that bonds entities and actions, while seemingly following the broad syntactical framework of general requirements forms specified in the IEEE 29148 standard. Similar to the negative class common words, most models converged on similar terminology to classify positive instances of requirements.

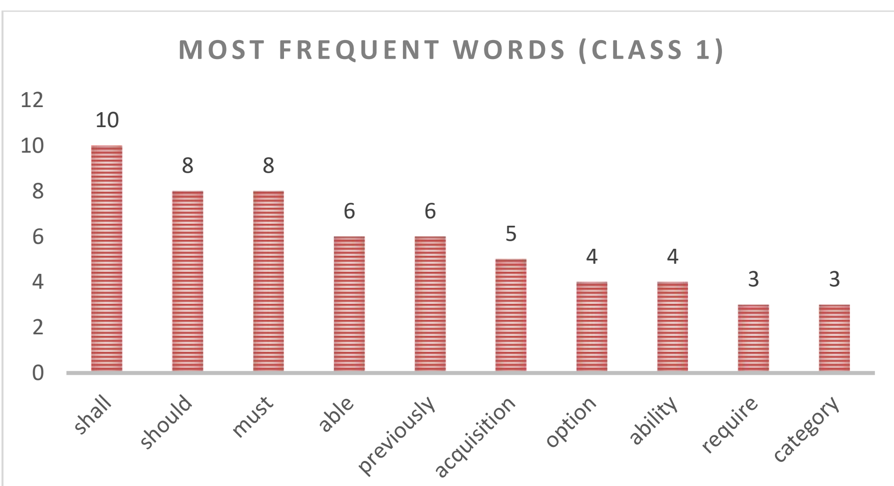

Figure 26 Most Frequent Words for Requirements

Based on the analysis of both classes, words alone do not seemingly explain the reasons for classification. With exception, positive requirement classification did show results akin to the expected requirement syntax broadly outlined among accepted requirement syntaxes. Still, statistical models achieved limited accuracy based on the aggregations of these particular terms. The analysis suggests that lexical units alone are not enough to help inform an accurate rule-based algorithm for classification; instead, a combination of the common word inclusions or positions could help better explain and understand what constitutes a requirement. With the results, complicated models such as BERT converged on similar words for importance among the classes, even though meta-models were utilized to help synthesize and understand model knowledge.

Human-Annotated Dataset Analysis for Requirement Identification and Quality Analysis

To further validate the ARQM application, the study analyzes human-annotated datasets. These datasets were created from reviewers tasked with both requirement identification and quality analysis of requirements. For the manually annotated datasets, the study compiled the aggregated unlabeled datasets from diverse requirement documents. Each reviewer annotated a list of random text to identify requirements and the associated requirement quality. Because the datasets were synthesized from disparate requirement documents, context was not provided to the reviewer.

In general, there was low to moderate agreement among the reviewers for both requirement identification and quality analysis tasks. To determine the agreement, overlapping datasets were given to the reviewers to help generate Kappa agreement results. Further, these agreements showed there was a diversity of opinion across the various tasks, leading to some limitations and high variation of model results. Ultimately, due to the limitation of the dataset size, data control was also more limited. To train the models, the data consisted of the overlapping data in which the users rated consistently. In instances where this would have led to an impractically small dataset for training, other data from the non-overlapping datasets were added to the training process. Additionally, Reviewer 1 tended to rate differently than the other reviewers. As a result, they were often removed as an outlier when appropriate due to low agreement. Figure 27 shows the process of manual annotation for requirement identification amongst the various datasets.

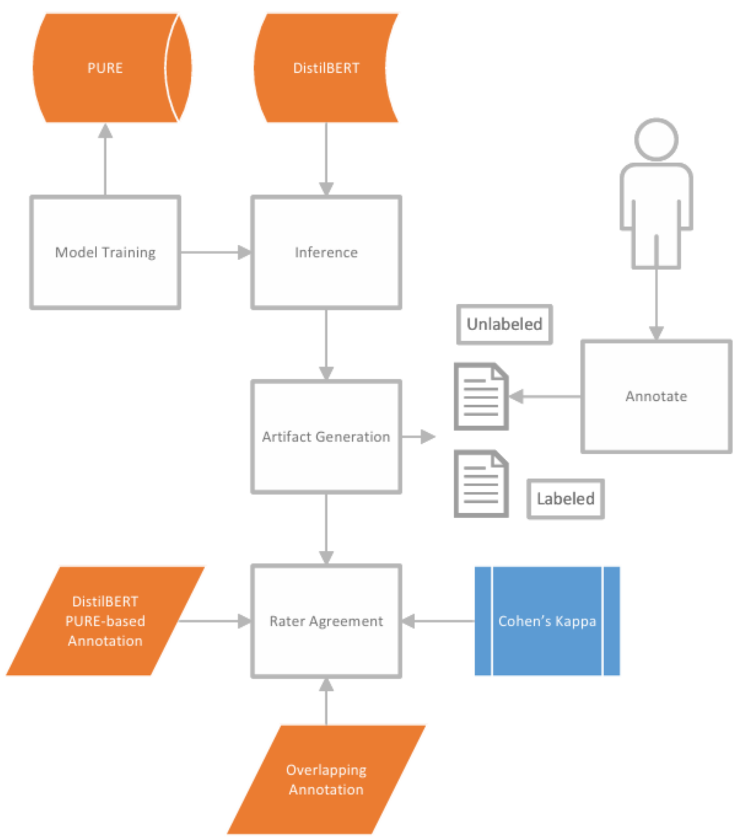

Figure 27 Manual Requirement Identification Process

As the figure shows, the automated artifact generation process utilizes pre-trained models that were tuned from the PURE dataset. Through inference, these trained models help to create both unlabeled and labeled datasets. Ultimately, the unlabeled dataset was manually labeled by a human annotator, allowing for the determination of agreement among both human annotated reviewers and the DistilBERT model predictions. Using Cohen's Kappa, a technique that measures agreement among multiple classes, the analysis shows how often the raters agreed based on similar data. Additionally, the agreement also outlines how accurately the trained model agreed with the human annotators, helping to understand the practical limitations of AI models and their associated generalizability.

Requirement Identification

For requirement identification, various models achieved success with the classification of requirement and non-requirement text. Notably, there was a high correlation between the PURE dataset labeling logic and the reviewers' responses. Table 22 shows the agreement among reviewers with respect to the PURE dataset; There was moderate agreement among reviewers, but notable disagreement for what constitutes a requirement, suggesting multiple interpretations for requirement identification.

Table 22 Kappa Reviewer Agreement with PURE

Source	Identification
Dataset 1	0.286
Dataset 2_1	0.343
Dataset 2_2	0.286
Dataset 3_1	0.429
Dataset 3_2	0.4
Dataset 4	0.371
Dataset 5_1	0.371

Dataset 5_2	0.457
Dataset 6_1	0.457

Table 23 shows the agreement for requirement identification where relevant; Although the reviewer agreement was significant, there was clear disagreement with Annotator 1, suggesting outlier data. The remaining reviewers agreed on what constituted a requirement, with a moderate Kappa agreement. To understand the implications of the reviewer agreement, the results were also analyzed against the PURE model results to understand the generalizability of the PURE dataset with the manual dataset. The results are provided in Table 23 and Table 24.

Table 23 Reviewer Agreement

	Annotator_1	Annotator_2	Annotator_3	Annotator_4	Annotator_5
Annotator_1	X	0.169	X	X	X
Annotator_2	0.169	X	0.441	X	X
Annotator_3	X	0.441	X	X	X
Annotator_4	X	X	X	X	0.6
Annotator_5	X	X	X	0.6	X

The BERT model achieved relatively high accuracy among all datasets but failed to generalize beyond low 70% accuracy. Although the results differed across datasets and reviewers, the model achieved sufficient accuracy for both classes, suggesting moderate generalizability between the PURE dataset and the annotated dataset. Although the model was moderately accurate, the analysis compares the models across manual data as well. These results are provided in Table 26.

Table 24 Negative Class PURE Accuracy

	Precision_0	**Recall_0**	**F1_0**	**Accuracy**
Dataset_1	0.66	0.6	0.63	0.64
Dataset_2_1	0.7	0.6	0.65	0.67

Dataset_2_2	0.69	0.51	0.59	0.64
Dataset_3_1	0.74	0.66	0.7	0.71
Dataset_3_2	**0.77**	0.57	0.66	0.7
Dataset_4	0.66	0.77	0.71	0.69
Dataset_5_1	0.64	0.86	0.73	0.69
Dataset_5_2	0.69	0.83	0.75	**0.73**
Dataset_6	0.66	**0.94**	**0.78**	**0.73**

Table 25 Positive Class PURE Accuracy

	Precision_1	Recall_1	F1_1	Accuracy
Dataset_1	0.63	0.69	0.66	0.64
Dataset_2_1	0.65	0.74	0.69	0.67
Dataset_2_2	0.61	0.77	0.68	0.64
Dataset_3_1	0.69	0.77	**0.73**	0.71
Dataset_3_2	0.66	**0.83**	**0.73**	0.7
Dataset_4	0.72	0.6	0.66	0.69
Dataset_5_1	0.78	0.51	0.62	0.69
Dataset_5_2	0.79	0.63	0.7	**0.73**
Dataset_6	**0.9**	0.51	0.65	**0.73**

In Table 26, the results show a variety of models and their associated results among inference for the manual datasets. For Accuracy, Logistic Regression surprisingly outperformed with a score of 0.89, while other models such as ANN, CNN, XLNet, DistilBERT, and Random Forest performed similarly for accuracy. For F1-Score, Logistic Regression again outperformed for both the positive and negative classes with scores of 0.88 and 0.89, respectively. Similar results were also achieved among models such as ANN, CNN, and XLNet. For Recall, KNN outperformed for the negative class with a score of 0.92, while ANN and CNN outperformed for the positive class with scores of 0.89. Lastly, Logistic Regression outperformed for precision within the positive class with a score of 0.90, and ANN outperformed for precision with a score of 0.89 among the negative class. Although there were similar performance across statistics for the different models, the degree of variation was much higher when compared to the PURE dataset. Naturally, the manual datasets contained significantly less entries, likely leading to higher variation of performance.

Table 26 Requirement Identification for Manual Datasets

Model	Class	Precision	Recall	F1-Score	Accuracy
ANN	N	**0.89 ± 0.08**	0.84 ± 0.14	0.85 ± 0.08	0.86 ± 0.07
CNN + Embeddings	N	0.88 ± 0.07	0.84 ± 0.10	0.85 ± 0.06	0.86 ± 0.06
XLNet	N	0.87 ± 0.04	0.88 ± 0.06	0.87 ± 0.03	0.87 ± 0.03
DistilBERT	N	0.85 ± 0.05	0.85 ± 0.07	0.85 ± 0.04	0.86 ± 0.04
DistilBERT Few-Shot	N	0.76 ± 0.01	0.89 ± 0.03	0.82 ± 0.01	0.79 ± 0.01
DistilBERT Siamese	N	0.77 ± 0.09	0.73 ± 0.17	0.74 ± 0.11	0.74 ± 0.09
Ensemble (Average)	N	0.84 ± 0.04	0.87 ± 0.06	0.86 ± 0.04	0.85 ± 0.03
Ensemble (Majority)	N	0.84 ± 0.04	0.88 ± 0.06	0.86 ± 0.03	0.85 ± 0.03
KNN	N	0.70 ± 0.10	**0.92 ± 0.07**	0.79 ± 0.08	0.76 ± 0.09
Logistic Regression	N	0.88 ± 0.07	0.90 ± 0.04	**0.89 ± 0.04**	**0.89 ± 0.04**
Meta-Model (Gradient Boost)	N	0.85 ± 0.05	0.80 ± 0.05	0.82 ± 0.02	0.82 ± 0.02
Meta-Model (Logistic Regression)	N	0.84 ± 0.04	0.86 ± 0.05	0.85 ± 0.03	0.85 ± 0.03
Random Forest	N	0.86 ± 0.06	0.90 ± 0.08	0.87 ± 0.04	0.87 ± 0.05
RNN	N	0.88 ± 0.08	0.79 ± 0.09	0.83 ± 0.07	0.84 ± 0.05
Support Vector Machine	N	0.87 ± 0.07	0.82 ± 0.09	0.84 ± 0.05	0.85 ± 0.05
ANN	Y	0.85 ± 0.13	**0.89 ± 0.08**	0.86 ± 0.07	0.86 ± 0.07
CNN + Embeddings	Y	0.85 ± 0.10	**0.89 ± 0.06**	0.86 ± 0.06	0.86 ± 0.06
XLNet	Y	0.87 ± 0.06	0.85 ± 0.05	0.86 ± 0.03	0.87 ± 0.03
DistilBERT	Y	0.86 ± 0.06	0.85 ± 0.05	0.86 ± 0.04	0.86 ± 0.04

DistilBERT Few-Shot	Y	0.84 ± 0.03	0.68 ± 0.03	0.75 ± 0.02	0.79 ± 0.01
DistilBERT Siamese	Y	0.73 ± 0.12	0.74 ± 0.14	0.72 ± 0.09	0.74 ± 0.09
Ensemble (Average)	Y	0.87 ± 0.04	0.83 ± 0.04	0.85 ± 0.03	0.85 ± 0.03
Ensemble (Majority)	Y	0.87 ± 0.05	0.82 ± 0.05	0.85 ± 0.03	0.85 ± 0.03
KNN	Y	0.89 ± 0.11	0.60 ± 0.14	0.71 ± 0.11	0.76 ± 0.09
Logistic Regression	Y	**0.90 ± 0.06**	0.87 ± 0.07	**0.88 ± 0.05**	**0.89 ± 0.04**
Meta-Model (Gradient Boost)	Y	0.81 ± 0.04	0.84 ± 0.06	0.82 ± 0.03	0.82 ± 0.02
Meta-Model (Logistic Regression)	Y	0.85 ± 0.04	0.83 ± 0.04	0.84 ± 0.03	0.85 ± 0.03
Random Forest	Y	0.89 ± 0.09	0.85 ± 0.06	0.86 ± 0.05	0.87 ± 0.05
RNN	Y	0.81 ± 0.08	0.89 ± 0.07	0.84 ± 0.06	0.84 ± 0.05
Support Vector Machine	Y	0.83 ± 0.09	0.88 ± 0.06	0.85 ± 0.05	0.85 ± 0.05

Based on significance testing, there was low consensus on ideal model performance for class 0 (Figure 28). For precision, the vast majority of the models, with exception of Siamese BERT, BERT Few-Shot, and KNN performed statistically similar. ANN through XLNET achieved high mean averages but failed to outpace each other in significance. For recall, all models besides RNN and Siamese BERT achieved statistical similarity. For F1, the results are similar with all models performing statistically similar with RNN and Siamese BERT excluded. Lastly, accuracy shows a similar story with statistically similar results among models with exception of BERT Few-Shot, KNN, and Siamese BERT. Based on the high variation in data results, the significance testing makes sense. The low amount of data and data agreement likely minimized the ability for models to learn useful patterns.

The significance testing for class 1 (Figure 29) details a similar story as class 0. For precision, all models besides the Siamese BERT implementation were statistically similar, with

Logistic Regression through ANN performing in the top class. For recall, there was more class distribution, where the top class outperformed Siamese BERT, BERT Few-Shot, and KNN. CNN, ANN, RNN, SVM, Logistic Regression, BERT, and XLNet all performed in the top class and achieved the highest means. For F1, most models performed similarly, while BERT Few-Shot, Siamese BERT, and KNN performed the worst based on statistical significance. These results do show some group spread across the different statistics, but the results seemingly indicate that both simplistic and complicated models performed similarly.

When compared to the PURE dataset, there was lower performance and higher variation. Naturally, the manual dataset consisted of significantly less entries and less diverse requirement specification artifacts during the dataset generation. Still, there was relatively high agreement from the PURE-trained model when trying to predict the manual dataset results, suggesting some useful data pattern extraction. Further, there was moderate to high-moderate agreement among annotators 2, 3, 4, and 5, with exception of annotator 1. However, given the disagreement among reviewers for what constitutes a requirement, the PURE-trained model was unable to achieve high metrics across the statistical points of analysis when identifying the manual identification dataset results.

Due to the PURE datasets strong and diverse dataset, the agreement and results are still promising, even though the model did not encapsulate the manual reviewer patterns well. Ultimately, the limitations and variability of the datasets show that perhaps the PURE and manual datasets differ in feature representation and reviewer opinion. Additionally, the small entries within the manual dataset provide further limitations for eliminating extremities, likely hurting the models capabilities for generalization further. The problem is likely not a generalization issue due to the vast entries of the PURE dataset. Further additions to the manual dataset would likely help bolster better results and patterns for model inference. Notably, Table 34 showed that the models did learn similar patterns across the manual identification dataset. Additionally, the model agreement was higher than the corresponding quality attribute entries. Lastly, the reviewer instructions differed significantly in relation to the PURE dataset instructions. Specifically, the reviewers were instructed to label an entry as a requirement without regard to granularity. Scope of reference and requirement qualification likely differed across the two studies.

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 0)

Factor	N	Mean	Grouping
ANN	10	0.8882	A
CNN	10	0.8817	A
RNN	10	0.8774	A
LOG REG ENSEMBLE	10	0.8750	A
SVM	10	0.8691	A
XLNET	10	0.8660	A
Random Forest	10	0.8553	A B
BERT	10	0.8515	A B
Gradient Boost	10	0.8453	A B
Average Ensemble	10	0.8432	A B
LOG REG META	10	0.8427	A B
MAJ. VOTE ENSEMBLE	10	0.8381	A B
Siamese	10	0.7677	B C
BERT FEW SHOT	10	0.76027	B C
KNN	10	0.7010	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 0)

Factor	N	Mean	Grouping
KNN	10	0.9229	A
LOG REG ENSEMBLE	10	0.9019	A B
Random Forest	10	0.8958	A B
BERT FEW SHOT	10	0.88806	A B
MAJ. VOTE ENSEMBLE	10	0.8824	A B
XLNET	10	0.8789	A B
Average Ensemble	10	0.8707	A B
LOG REG META	10	0.8575	A B C
BERT	10	0.8548	A B C
CNN	10	0.8383	A B C
ANN	10	0.8360	A B C
SVM	10	0.8220	A B C
Gradient Boost	10	0.8047	A B C
RNN	10	0.7929	B C
Siamese	10	0.7311	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 0)

Factor	N	Mean	Grouping
KNN	10	0.9229	A
LOG REG ENSEMBLE	10	0.9019	A B
Random Forest	10	0.8958	A B
BERT FEW SHOT	10	0.88806	A B
MAJ. VOTE ENSEMBLE	10	0.8824	A B
XLNET	10	0.8789	A B
Average Ensemble	10	0.8707	A B
LOG REG META	10	0.8575	A B C
BERT	10	0.8548	A B C
CNN	10	0.8383	A B C
ANN	10	0.8360	A B C
SVM	10	0.8220	A B C
Gradient Boost	10	0.8047	A B C
RNN	10	0.7929	B C
Siamese	10	0.7311	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 0)

Factor	N	Mean	Grouping
LOG REG ENSEMBLE	10	0.8852	A
Random Forest	10	0.8691	A B
XLNET	10	0.86642	A B
CNN	10	0.8603	A B
ANN	10	0.8578	A B
BERT	10	0.8552	A B
Average Ensemble	10	0.8530	A B
MAJ. VOTE ENSEMBLE	10	0.85299	A B
LOG REG META	10	0.84701	A B
SVM	10	0.8467	A B
RNN	10	0.8399	A B
Gradient Boost	10	0.82388	A B C
BERT FEW SHOT	10	0.79127	B C D
KNN	10	0.7616	C D
Siamese	10	0.7352	D

Means that do not share a letter are significantly different.

Figure 28 Significance Testing for Manual Identification (Class 0)

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 1)

Factor	N	Mean	Grouping
LOG REG ENSEMBLE	10	0.8958	A
KNN	10	0.8920	A
Random Forest	10	0.8892	A
MAJ. VOTE ENSEMBLE	10	0.8734	A
XLNET	10	0.8709	A
Average Ensemble	10	0.8656	A
BERT	10	0.8632	A
LOG REG META	10	0.8527	A
ANN	10	0.8526	A
CNN	10	0.8462	A B
BERT FEW SHOT	10	0.84415	A B
SVM	10	0.8314	A B
RNN	10	0.8116	A B
Gradient Boost	10	0.8081	A B
Siamese	10	0.7289	B

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 1)

Factor	N	Mean	Grouping
CNN	10	0.8906	A
ANN	10	0.8892	A
RNN	10	0.8870	A
SVM	10	0.8750	A
LOG REG ENSEMBLE	10	0.8690	A
BERT	10	0.8546	A
XLNET	10	0.8530	A
Random Forest	10	0.8477	A B
Gradient Boost	10	0.8435	A B
LOG REG META	10	0.8341	A B
Average Ensemble	10	0.8328	A B
MAJ. VOTE ENSEMBLE	10	0.8215	A B
Siamese	10	0.7399	B C
BERT FEW SHOT	10	0.68136	C D
KNN	10	0.6032	D

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 1)

Factor	N	Mean	Grouping
LOG REG ENSEMBLE	10	0.8806	A
Random Forest	10	0.8640	A
CNN	10	0.8634	A
ANN	10	0.8616	A
XLNET	10	0.8601	A
BERT	10	0.8571	A
SVM	10	0.8489	A
Average Ensemble	10	0.84799	A
MAJ. VOTE ENSEMBLE	10	0.8451	A
RNN	10	0.8448	A
LOG REG META	10	0.84261	A
Gradient Boost	10	0.82328	A B
BERT FEW SHOT	10	0.75331	B C
Siamese	10	0.7237	C
KNN	10	0.7116	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 1)

Factor	N	Mean	Grouping
LOG REG ENSEMBLE	10	0.8852	A
Random Forest	10	0.8691	A B
XLNET	10	0.86642	A B
CNN	10	0.8603	A B
ANN	10	0.8578	A B
BERT	10	0.8552	A B
Average Ensemble	10	0.8530	A B
MAJ. VOTE ENSEMBLE	10	0.85299	A B
LOG REG META	10	0.84701	A B
SVM	10	0.8467	A B
RNN	10	0.8399	A B
Gradient Boost	10	0.82388	A B C
BERT FEW SHOT	10	0.79127	B C D
KNN	10	0.7616	C D
Siamese	10	0.7352	D

Means that do not share a letter are significantly different.

Figure 29 Significance Testing for Manual Identification (Class 1)

Ultimately, models trained on manual data achieved better generalization than the PURE derived model. With the achievement of high metrics, the models learned patterns representative of the user annotations, suggesting consistency among raters. Additionally, simpler models seemingly achieved comparable or greater performance when compared to complicated models, implying that features from human annotators may be less complicated. Overall, models were able to generalize the raters' patterns for determining requirement and non-requirement text among the datasets.

Quality Analysis

Among the existing quality attributes from the IEEE 29148 standard, the annotated dataset consisted of four items including ambiguity, feasibility, singularity, and verifiability. These items were evaluated for each requirement by human annotators using the IEEE 29148 standard definition for violations. The analysis compares the rater agreement through Cohen's Kappa to determine rater consistency. Additionally, the analysis looks into the ability of models to learn the pattern representations of the manually annotated datasets.

Ambiguity

For ambiguity, there was general agreement among most reviewers for what constitutes an ambiguous requirement. For Cohen's Kappa, the results are provided in Table 27. There was no agreement between Annotator 1 and Annotator 2, but there was agreement among Annotator 2, 3, 4, and 5. High and moderate Kappa agreements of 0.304 and 0.711 suggest an agreement among ambiguity violations for requirements. To further show the agreement of ambiguity within the datasets, Table 28 shows the results of model inference.

Table 27 Ambiguity Rater Agreement

	Annotator_1	**Annotator_2**	**Annotator_3**	**Annotator_4**	**Annotator_5**
Annotator_1	X	-0.068	X	X	X
Annotator_2	-0.068	X	0.304	X	X
Annotator_3	X	0.304	X	X	X
Annotator_4	X	X	X	X	0.711
Annotator_5	X	X	X	0.711	X

Table 28 shows a slight inability for models to learn the general patterns of ambiguity from the human annotated dataset. For Accuracy, Random Forest outperformed with a result of 0.84. For the F1-Score, Random Forest also outperformed among the negative and positive classes with scores of 0.84. For Recall, XLNet generalized the best for the negative class with a score of 0.87, while Random Forest achieved the best score for the positive class with a score of 0.82. Lastly, CNN outperformed for Precision among the negative class with a score of 0.84, while XLNet achieved a score of 0.85 for the positive class. In general, Random Forest seemingly outperformed most models, even though other statistical models showcased moderate or below-average performance. Even though Random Forest did not necessarily perform the best for every statistic, the model generally performed well regardless of the class, suggesting that the Random Forest model would be ideal for Ambiguity detection. Lastly, other, more complicated models, such as DistilBERT seemingly failed to encapsulate the patterns of the data. Naturally, there was disagreement across the various annotators; further extrapolation and processing of the dataset could help models learn the patterns better. Figure 32 Ambiguity Embedding Space shows the associated embedding separation among the two classes.

Table 28 Ambiguity Model Results

Model	**Class**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
ANN	N	0.79 ± 0.10	0.75 ± 0.11	0.77 ± 0.09	0.78 ± 0.07
CNN + Embeddings	N	**0.84 ± 0.09**	0.75 ± 0.05	0.79 ± 0.06	0.80 ± 0.06
XLNet	N	0.70 ± 0.05	**0.87 ± 0.07**	0.78 ± 0.04	0.76 ± 0.05
DistilBERT	N	0.71 ± 0.04	0.76 ± 0.10	0.73 ± 0.04	0.72 ± 0.02
DistilBERT Few-Shot	N	0.60 ± 0.02	0.81 ± 0.04	0.69 ± 0.02	0.57 ± 0.03
DistilBERT Siamese	N	0.61 ± 0.03	0.53 ± 0.12	0.56 ± 0.08	0.50 ± 0.05
Ensemble (Average)	N	0.76 ± 0.07	0.70 ± 0.06	0.73 ± 0.05	0.74 ± 0.03
Ensemble (Majority)	N	0.76 ± 0.07	0.72 ± 0.06	0.73 ± 0.04	0.75 ± 0.03
KNN	N	0.60 ± 0.08	0.92 ± 0.05	0.73 ± 0.07	0.66 ± 0.07

Logistic Regression	N	0.77 ± 0.09	0.73 ± 0.12	0.75 ± 0.09	0.76 ± 0.07
Meta-Model (Gradient Boost)	N	0.75 ± 0.05	0.69 ± 0.13	0.71 ± 0.08	0.74 ± 0.04
Meta-Model (Logistic Regression)	N	0.77 ± 0.07	0.68 ± 0.05	0.72 ± 0.05	0.74 ± 0.03
Random Forest	N	0.83 ± 0.06	0.86 ± 0.06	**0.84 ± 0.06**	**0.84 ± 0.05**
RNN - LSTM + Embeddings	N	0.78 ± 0.08	0.69 ± 0.09	0.73 ± 0.08	0.75 ± 0.06
Support Vector Machine	N	0.78 ± 0.09	0.72 ± 0.12	0.75 ± 0.10	0.76 ± 0.07
ANN	Y	0.77 ± 0.08	0.80 ± 0.09	0.78 ± 0.07	0.78 ± 0.07
CNN + Embeddings	Y	0.76 ± 0.08	0.85 ± 0.09	0.80 ± 0.08	0.80 ± 0.06
XLNet	Y	**0.85 ± 0.07**	0.64 ± 0.09	0.73 ± 0.06	0.76 ± 0.05
DistilBERT	Y	0.75 ± 0.06	0.69 ± 0.09	0.71 ± 0.03	0.72 ± 0.02
DistilBERT Few-Shot	Y	0.45 ± 0.07	0.22 ± 0.02	0.29 ± 0.03	0.57 ± 0.03
DistilBERT Siamese	Y	0.38 ± 0.04	0.46 ± 0.10	0.41 ± 0.05	0.50 ± 0.05
Ensemble (Average)	Y	0.74 ± 0.04	0.78 ± 0.06	0.76 ± 0.03	0.74 ± 0.03
Ensemble (Majority)	Y	0.74 ± 0.03	0.78 ± 0.06	0.76 ± 0.02	0.75 ± 0.03
KNN	Y	**0.85 ± 0.08**	0.40 ± 0.07	0.54 ± 0.07	0.66 ± 0.07
Logistic Regression	Y	0.75 ± 0.09	0.79 ± 0.07	0.77 ± 0.07	0.76 ± 0.07
Meta-Model (Gradient Boost)	Y	0.74 ± 0.07	0.79 ± 0.06	0.75 ± 0.02	0.74 ± 0.04
Meta-Model (Logistic Regression)	Y	0.72 ± 0.03	0.81 ± 0.06	0.76 ± 0.02	0.74 ± 0.03
Random Forest	Y	**0.85 ± 0.07**	**0.82 ± 0.05**	**0.84 ± 0.06**	**0.84 ± 0.05**
RNN - LSTM + Embeddings	Y	0.72 ± 0.06	0.81 ± 0.06	0.76 ± 0.05	0.75 ± 0.06
Support Vector Machine	Y	0.75 ± 0.09	0.80 ± 0.07	0.77 ± 0.07	0.76 ± 0.07

For Ambiguity, statistical significance tended to be more stratified and pronounced when compared to the manual requirement identification result. Figure 30 shows the results of the significance testing for the negative class. For precision, CNN through Gradient Boost outperformed. In particular, CNN and Random Forest performed better than BERT, XLNet, Naïve Bayes, Siamese BERT, KNN, and BERT Few-Shot. For recall, KNN surprisingly performed the best by mean, statistically outperforming Naïve Bayes through Siamese BERT. The worst performing models included SVM through Siamese BERT. F1 had general model agreement, whereas Random Forest outperformed almost every model, including Majority Vote Ensemble and below. For accuracy, Random Forest again outperformed similarly, with CNN, ANN, SVM, and Logistic Regression achieving similar results.

For the positive class (Figure 31), there continues to be broad distribution across statistical model results. For precision, Random Forest outperformed Majority Vote Ensemble and below, with KNN and XLNet performing similarly. Recall had more model agreement, with CNN, Random Forest, Logistic Regression, RNN, and SVM outperforming BERT and below. For F1, Random Forest again outperformed with a superior performance to Gradient Boost and below. Overall, Random Forest seemingly outperformed in most categories and statistics. Other models, such as KNN, performed uniquely well off of certain statistics, while CNN also performed well consistently. Generally, there was significant changes in top model results based on the metric.

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 0)

Factor	N	Mean	Grouping
CNN	10	0.8432	A
Random Forest	10	0.8272	A
ANN	10	0.7894	A B
SVM	10	0.7827	A B
RNN	10	0.7765	A B
LOG REG META	10	0.7723	A B
LOG REG ENSEMBLE	10	0.7722	A B
MAJ. VOTE ENSEMBLE	10	0.7576	A B
Average Ensemble	10	0.7568	A B
Gradient Boost	10	0.7538	A B
BERT	10	0.7130	B C
XLNET	10	0.7028	B C D
Naive Bayes	10	0.6808	B C D
Siamese	10	0.6075	C D
KNN	10	0.6050	C D
BERT FEW SHOT	10	0.59594	D

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 0)

Factor	N	Mean	Grouping
KNN	10	0.9246	A
XLNET	10	0.8714	A B
Random Forest	10	0.8611	A B C
BERT FEW SHOT	10	0.8122	A B C D
BERT	10	0.7595	B C D E
ANN	10	0.7517	B C D E
CNN	10	0.7453	B C D E
Naive Bayes	10	0.7346	C D E
LOG REG ENSEMBLE	10	0.7328	C D E
SVM	10	0.7200	D E
MAJ. VOTE ENSEMBLE	10	0.7168	D E
Average Ensemble	10	0.7050	D E
Gradient Boost	10	0.6894	D E
RNN	10	0.6887	D E
LOG REG META	10	0.6751	E
Siamese	10	0.5267	F

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.8430	A
CNN	10	0.7895	A B
XLNET	10	0.7763	A B C
ANN	10	0.7651	A B C
LOG REG ENSEMBLE	10	0.7491	A B C
SVM	10	0.7470	A B C
MAJ. VOTE ENSEMBLE	10	0.7340	B C
BERT	10	0.7305	B C
KNN	10	0.7292	B C
RNN	10	0.7286	B C
Average Ensemble	10	0.7272	B C
LOG REG META	10	0.7185	B C
Gradient Boost	10	0.7132	B C
Naive Bayes	10	0.7041	B C
BERT FEW SHOT	10	0.68730	C
Siamese	10	0.5594	D

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.8411	A
CNN	10	0.8002	A B
ANN	10	0.7754	A B C
SVM	10	0.7637	A B C
LOG REG ENSEMBLE	10	0.7618	A B C
XLNET	10	0.7561	B C
RNN	10	0.7499	B C
MAJ. VOTE ENSEMBLE	10	0.74903	B C
Average Ensemble	10	0.74452	B C
LOG REG META	10	0.74387	B C
Gradient Boost	10	0.7381	B C D
BERT	10	0.72323	B C D
Naive Bayes	10	0.6960	C D
KNN	10	0.6612	D
BERT FEW SHOT	10	0.56667	E
Siamese	10	0.5021	E

Means that do not share a letter are significantly different.

Figure 30 Significance Testing for Ambiguity (Class 0)

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.8526	A
KNN	10	0.8493	A B
XLNET	10	0.8464	A B C
ANN	10	0.7678	A B C D
CNN	10	0.7645	A B C D
LOG REG ENSEMBLE	10	0.7525	A B C D
BERT	10	0.7501	A B C D
SVM	10	0.7485	B C D
MAJ. VOTE ENSEMBLE	10	0.7450	C D
Average Ensemble	10	0.7377	D
Gradient Boost	10	0.7351	D
RNN	10	0.7246	D
LOG REG META	10	0.72448	D
Naive Bayes	10	0.7140	D
BERT FEW SHOT	10	0.4542	E
Siamese	10	0.3833	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 1)

Factor	N	Mean	Grouping
CNN	10	0.8529	A
Random Forest	10	0.8211	A
LOG REG META	10	0.8099	A
RNN	10	0.8061	A
SVM	10	0.8000	A
ANN	10	0.7959	A B
LOG REG ENSEMBLE	10	0.7868	A B
Gradient Boost	10	0.7861	A B
Average Ensemble	10	0.7844	A B
MAJ. VOTE ENSEMBLE	10	0.7820	A B
BERT	10	0.6856	B C
Naive Bayes	10	0.6620	C
XLNET	10	0.6430	C
Siamese	10	0.4630	D
KNN	10	0.4002	D
BERT FEW SHOT	10	0.21731	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.8356	A
CNN	10	0.8049	A B
ANN	10	0.7786	A B C
SVM	10	0.7707	A B C
LOG REG ENSEMBLE	10	0.7666	A B C
LOG REG META	10	0.76334	A B C D
RNN	10	0.7624	A B C D
MAJ. VOTE ENSEMBLE	10	0.76087	A B C D
Average Ensemble	10	0.75800	A B C D
Gradient Boost	10	0.75480	B C D
XLNET	10	0.7267	B C D
BERT	10	0.71032	C D
Naive Bayes	10	0.6835	D
KNN	10	0.5402	E
Siamese	10	0.4142	F
BERT FEW SHOT	10	0.29301	G

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.8411	A
CNN	10	0.8002	A B
ANN	10	0.7754	A B C
SVM	10	0.7637	A B C
LOG REG ENSEMBLE	10	0.7618	A B C
XLNET	10	0.7561	B C
RNN	10	0.7499	B C
MAJ. VOTE ENSEMBLE	10	0.74903	B C
Average Ensemble	10	0.74452	B C
LOG REG META	10	0.74387	B C
Gradient Boost	10	0.7381	B C D
BERT	10	0.72323	B C D
Naive Bayes	10	0.6960	C D
KNN	10	0.6612	D
BERT FEW SHOT	10	0.56667	E
Siamese	10	0.5021	E

Means that do not share a letter are significantly different.

Figure 31 Significance Testing for Ambiguity (Class 1)

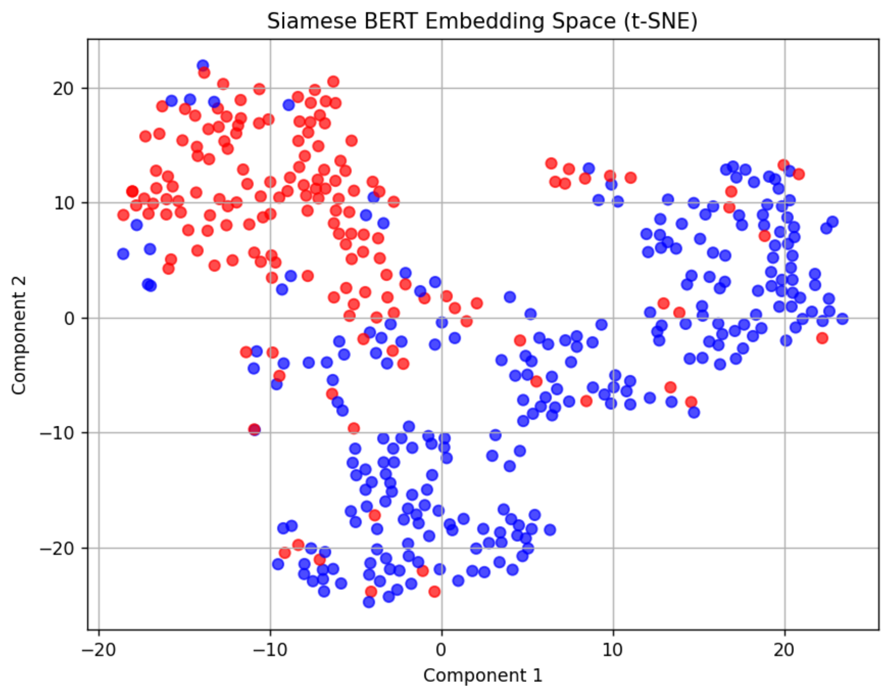

Figure 32 Ambiguity Embedding Space

Feasibility

Similarly to ambiguity, the analysis looks at the reviewer agreement among feasibility violations for requirements. Additionally, the reviewers were instructed to utilize the IEEE 29148 standard definition of feasibility to inform their decisions. Although there was high agreement among most reviewers, there was disagreement among Annotator 1 and Annotator 2, similarly to the ambiguity Kappa agreement. For Annotator 2 and 3, there was a high Kappa agreement of 0.66, while Annotators 4 and 5 had a Kappa agreement of 0.317, indicating a moderate agreement. The results are provided in Table 29 Feasibility Kappa Agreement.

Table 29 Feasibility Kappa Agreement

	Annotator_1	Annotator_2	Annotator_3	Annotator_4	Annotator_5
Annotator_1	X	0	X	X	X
Annotator_2	0	X	0.66	X	X

Annotator_3	X	0.66	X	X	X
Annotator_4	X	X	X	X	0.317
Annotator_5	X	X	X	0.317	X

Notably for the feasibility annotations among reviewers, there was a high bar to indicate a violation. Most of the entries were determined to be feasible, even with the IEEE 29148 standard definition provided to the users. As mentioned previously, the users were not given the context of the project associated with each requirement, limiting their ability to make an adequate decision on whether a requirement was feasible. Due to the lack of a balanced dataset for both non-feasible and feasible requirements, training AI models was not practical for this particular quality attribute. Figure 33 shows the separation result of the data. Of the two classes, the data is well separated, but the data limitations affect the generalizability of the separation results.

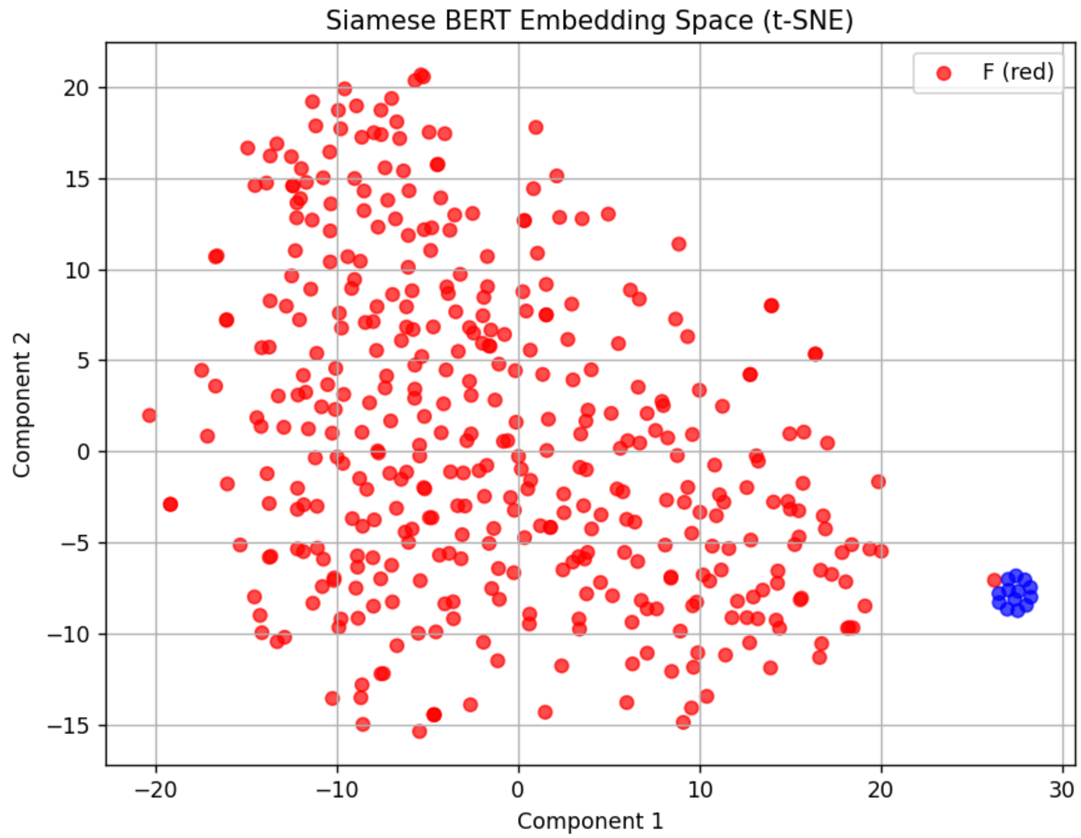

Figure 33 Feasibility Embedding Space

Singularity

Unlike ambiguity and feasibility, there was agreement among all reviewers for what constituted a singularity violation within a requirement. Annotators 1 and 2 had a Kappa agreement of 0.333, Annotators 2 and 3 had a kappa agreement of 0.151, and Annotators 4 and 5 had a Kappa agreement of 0.204. Although there was agreement among reviewers, the agreement was only moderate, suggesting that does not have a simplistic definition among raters. The analysis of singularity is further extended through model inference and how well simple and complicated models could learn the human patterns provided from the datasets. The Kappa agreement is provided in Table 30 whereas the model results are provided in Table 31.

Table 30 Singularity Kappa Agreement

	Annotator_1	Annotator_2	Annotator_3	Annotator_4	Annotator_5
Annotator_1	X	0.333	X	X	X
Annotator_2	0.333	X	0.151	X	X
Annotator_3	X	0.151	X	X	X
Annotator_4	X	X	X	X	0.204
Annotator_5	X	X	X	0.204	X

Unlike ambiguity, the models achieved high achievement among metrics for singularity, suggesting more consistent data labeling and patterns. For Accuracy, Random Forest outformed with a score of 0.93. Similarly, Random Forest also outperformed for F1-Score with scores of 0.92 for both the positive and negative classes. For Recall, DistilBERT Few-Shot achieved a high recall of 0.97 for the negative class, whereas Random Forest outperformed again with a score of 0.94 for the positive class. Lastly, Random Forest outperformed for precision among the negative class with a score of 0.94, while KNN outperformed for the positive class with a score of 0.92. Unlike the ambiguity dataset, singularity had less variation and higher agreement among reviewers, likely allowing for better model performance and consistency. Figure 36 shows a healthy separation of classes amongst embeddings.

Table 31 Singularity Model Results

Model	**Class**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
ANN	N	0.90 ± 0.07	0.75 ± 0.10	0.81 ± 0.07	0.83 ± 0.06
CNN + Embeddings	N	0.90 ± 0.04	0.82 ± 0.12	0.85 ± 0.06	0.86 ± 0.05
XLNet	N	0.90 ± 0.04	0.82 ± 0.07	0.85 ± 0.04	0.86 ± 0.04
DistilBERT	N	0.89 ± 0.04	0.86 ± 0.04	0.87 ± 0.02	0.88 ± 0.02
DistilBERT Few-Shot	N	0.73 ± 0.00	**0.97 ± 0.02**	0.83 ± 0.01	0.71 ± 0.01

DistilBERT Siamese	N	0.75 ± 0.07	0.72 ± 0.16	0.72 ± 0.10	0.75 ± 0.06
Ensemble (Average)	N	0.90 ± 0.03	0.90 ± 0.04	0.90 ± 0.02	0.90 ± 0.02
Ensemble (Majority)	N	0.89 ± 0.03	0.91 ± 0.04	0.90 ± 0.02	0.90 ± 0.02
KNN	N	0.72 ± 0.06	0.95 ± 0.04	0.82 ± 0.03	0.79 ± 0.04
Logistic Regression	N	0.86 ± 0.07	0.86 ± 0.05	0.86 ± 0.05	0.86 ± 0.05
Meta-Model (Gradient Boost)	N	0.91 ± 0.03	0.84 ± 0.08	0.87 ± 0.05	0.88 ± 0.04
Meta-Model (Logistic Regression)	N	0.92 ± 0.04	0.87 ± 0.04	0.89 ± 0.02	0.89 ± 0.02
Random Forest	N	**0.94 ± 0.03**	0.91 ± 0.04	**0.92 ± 0.02**	**0.93 ± 0.02**
RNN	N	0.92 ± 0.06	0.85 ± 0.06	0.88 ± 0.03	0.88 ± 0.04
Support Vector Machine	N	0.90 ± 0.05	0.81 ± 0.06	0.85 ± 0.04	0.86 ± 0.04
ANN	Y	0.78 ± 0.10	0.91 ± 0.06	0.84 ± 0.07	0.83 ± 0.06
CNN + Embeddings	Y	0.84 ± 0.09	0.90 ± 0.06	0.87 ± 0.05	0.86 ± 0.05
XLNet	Y	0.83 ± 0.08	0.90 ± 0.04	0.86 ± 0.05	0.86 ± 0.04
DistilBERT	Y	0.87 ± 0.03	0.90 ± 0.04	0.88 ± 0.02	0.88 ± 0.02
DistilBERT Few-Shot	Y	0.40 ± 0.11	0.05 ± 0.02	0.09 ± 0.03	0.71 ± 0.01
DistilBERT Siamese	Y	0.77 ± 0.09	0.78 ± 0.10	0.77 ± 0.05	0.75 ± 0.06
Ensemble (Average)	Y	0.90 ± 0.04	0.89 ± 0.04	0.90 ± 0.02	0.90 ± 0.02
Ensemble (Majority)	Y	0.91 ± 0.03	0.89 ± 0.04	0.90 ± 0.02	0.90 ± 0.02
KNN	Y	**0.92 ± 0.09**	0.62 ± 0.08	0.74 ± 0.08	0.79 ± 0.04
Logistic Regression	Y	0.86 ± 0.07	0.86 ± 0.08	0.85 ± 0.07	0.86 ± 0.05
Meta-Model (Gradient Boost)	Y	0.85 ± 0.06	0.92 ± 0.03	0.89 ± 0.04	0.88 ± 0.04
Meta-Model (Logistic Regression)	Y	0.87 ± 0.03	0.92 ± 0.04	0.90 ± 0.02	0.89 ± 0.02

Random Forest	Y	0.91 ± 0.04	**0.94 ± 0.03**	**0.92 ± 0.03**	**0.93 ± 0.02**
RNN	Y	0.86 ± 0.07	0.92 ± 0.06	0.88 ± 0.05	0.88 ± 0.04
Support Vector Machine	Y	0.82 ± 0.08	0.91 ± 0.04	0.86 ± 0.05	0.86 ± 0.04

Unlike Ambiguity, Singularity had less group distribution for statistical significance. In Figure 34, the results for precision show that Random Forest outperformed Logistic Regression and below, but performed statistically similar to all other models. For recall, BERT Few-Shot outperformed along with KNN when compared to Gradient Boost and below. For F1, Random Forest again outperformed with the highest mean, while also outperforming statistically with CNN, SVM, BERT Few-Shot, KNN, ANN, and Siamese BERT. For accuracy, Random Forest again outperformed most models including CNN and below. Seemingly, Random Forest and Ensemble-based models outperformed in general based on the data. Similarly to Ambiguity, Random Forest generally performed the best among models based on Mean, with exception to the recall results.

For class 1, provided in Figure 35, the results indicate that Random Forest again outperformed among Recall, F1, and general accuracy. For precision, KNN, Majority Vote Ensemble, Random Forest, and Average Ensemble performed in the top class. However, there was a large clustering of statistically similar models, showing that only ANN, Siamese BERT, and BERT Few-Shot performed worse. For Recall, Random Forest outperformed Logistic Regression and below, while Gradient Boost through Majority Vote Ensemble performed statistically similarly. Again, there was a large group overlap, suggesting most models achieved statistically similar results, especially with the high variation detailed in the data. For F1, Random Forest outperformed for the mean result and achieved significantly better performance than ANN, Siamese BERT, KNN, and BERT Few-Shot. Still, there was strong model overlap. Even though the results were less clear for the positive class, Random Forest and Ensemble models seemingly outperformed.

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.94243	A
LOG REG META	10	0.9161	A B
RNN	10	0.9153	A B
Gradient Boost	10	0.9144	A B
CNN	10	0.9026	A B
SVM	10	0.9021	A B
ANN	10	0.8964	A B
XLNET	10	0.8959	A B
Average Ensemble	10	0.8958	A B
MAJ. VOTE ENSEMBLE	10	0.89325	A B
BERT	10	0.8916	A B
LOG REG	10	0.8619	B
Siamese	10	0.7549	C
BERT FEW SHOT	10	0.726299	C
KNN	10	0.7174	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 0)

Factor	N	Mean	Grouping
BERT FEW SHOT	10	0.96813	A
KNN	10	0.9533	A
MAJ. VOTE ENSEMBLE	10	0.9101	A B
Random Forest	10	0.9065	A B
Average Ensemble	10	0.9039	A B
LOG REG META	10	0.8666	A B C
LOG REG	10	0.8650	A B C
BERT	10	0.8585	A B C
RNN	10	0.8541	A B C
Gradient Boost	10	0.8380	B C
XLNET	10	0.8215	B C D
CNN	10	0.8165	B C D
SVM	10	0.8086	B C D
ANN	10	0.7547	C D
Siamese	10	0.7190	D

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.92320	A
MAJ. VOTE ENSEMBLE	10	0.90088	A B
Average Ensemble	10	0.89873	A B C
LOG REG META	10	0.88960	A B C
RNN	10	0.8811	A B C D
BERT	10	0.87331	A B C D
Gradient Boost	10	0.8722	A B C D
LOG REG	10	0.8623	A B C D
XLNET	10	0.8548	A B C D
CNN	10	0.8516	B C D
SVM	10	0.8511	B C D
BERT FEW SHOT	10	0.82991	C D
KNN	10	0.8168	D
ANN	10	0.8147	D
Siamese	10	0.7230	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.92524	A
MAJ. VOTE ENSEMBLE	10	0.90000	A B
Average Ensemble	10	0.89845	A B
LOG REG META	10	0.89275	A B
RNN	10	0.8847	A B C
Gradient Boost	10	0.8793	A B C
BERT	10	0.87876	A B C
CNN	10	0.8630	B C
LOG REG	10	0.8613	B C
SVM	10	0.8598	B C
XLNET	10	0.8596	B C
ANN	10	0.8300	C D
KNN	10	0.7881	D E
Siamese	10	0.7540	E F
BERT FEW SHOT	10	0.71349	F

Means that do not share a letter are significantly different.

Figure 34 Significance Testing for Singularity (Class 0)

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 1)

Factor	N	Mean	Grouping
KNN	10	0.9204	A
MAJ. VOTE ENSEMBLE	10	0.9091	A
Random Forest	10	0.9074	A
Average Ensemble	10	0.9045	A
LOG REG META	10	0.87415	A B
BERT	10	0.8709	A B
RNN	10	0.8568	A B
LOG REG	10	0.8556	A B
Gradient Boost	10	0.8541	A B
CNN	10	0.8369	A B
XLNET	10	0.8314	A B
SVM	10	0.8230	A B
ANN	10	0.7835	B
Siamese	10	0.7748	B
BERT FEW SHOT	10	0.4027	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.94191	A
Gradient Boost	10	0.92186	A B
LOG REG META	10	0.9198	A B
RNN	10	0.9191	A B
SVM	10	0.9124	A B
ANN	10	0.9075	A B
XLNET	10	0.9015	A B
CNN	10	0.9012	A B
BERT	10	0.8973	A B
Average Ensemble	10	0.8935	A B
MAJ. VOTE ENSEMBLE	10	0.8902	A B
LOG REG	10	0.8553	B C
Siamese	10	0.7846	C
KNN	10	0.6208	D
BERT FEW SHOT	10	0.05143	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.92379	A
MAJ. VOTE ENSEMBLE	10	0.89877	A B
Average Ensemble	10	0.89780	A B
LOG REG META	10	0.89551	A B
Gradient Boost	10	0.8851	A B
RNN	10	0.8846	A B
BERT	10	0.88284	A B
CNN	10	0.8652	A B
SVM	10	0.8633	A B
XLNET	10	0.8629	A B
LOG REG	10	0.8545	A B
ANN	10	0.8377	B C
Siamese	10	0.7731	C D
KNN	10	0.7384	D
BERT FEW SHOT	10	0.08969	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.92524	A
MAJ. VOTE ENSEMBLE	10	0.90000	A B
Average Ensemble	10	0.89845	A B
LOG REG META	10	0.89275	A B
RNN	10	0.8847	A B C
Gradient Boost	10	0.8793	A B C
BERT	10	0.87876	A B C
CNN	10	0.8630	B C
LOG REG	10	0.8613	B C
SVM	10	0.8598	B C
XLNET	10	0.8596	B C
ANN	10	0.8300	C D
KNN	10	0.7881	D E
Siamese	10	0.7540	E F
BERT FEW SHOT	10	0.71349	F

Means that do not share a letter are significantly different.

Figure 35 Significance Testing for Singularity (Class 1)

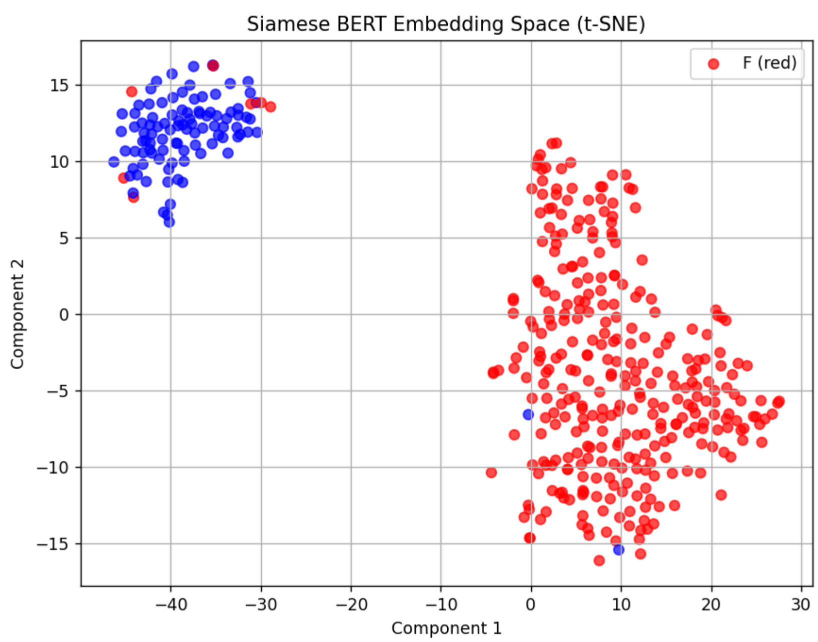

Figure 36 Singularity Embedding Space

Verifiability

Lastly, the analysis looks at the verifiability quality attribute. Unlike the previous quality attributes, there was no agreement for what constituted a verifiability violation among requirements. There was no agreement among Annotators 1 and 2, a Kappa agreement of 0.172 for Annotators 2 and 3, and a Kappa agreement of 0.117 for annotators 4 and 5. Additionally, there was a high tendency towards the negative class among reviewers, limiting the ability to train models from the reviewer responses. Ultimately, the high propensity towards the negative classification among reviewers likely meant that there was not adequate representation of the positive class, leading to a low agreement. Perhaps, stronger prompting, examples, and clarification could help with consolidating the definition of verifiability violations among reviewers. Through the use of oversampling for the minority class, the models were trained on the user responses for an even dataset. The results for the Kappa agreement are provided in Table 32, while the model inference results are provided in Table 33.

Table 32 Verifiability Kappa Agreement

	Annotator_1	Annotator_2	Annotator_3	Annotator_4	Annotator_5
Annotator_1	X	0	X	X	X
Annotator_2	0	X	0.172	X	X
Annotator_3	X	0.172	X	X	X
Annotator_4	X	X	X	X	0.117
Annotator_5	X	X	X	0.117	X

Even though there was annotator disagreement, the models seemingly were able to learn the patterns of the dataset. For accuracy, the Random Forest model achieved an accuracy of 0.85. For F1-Score, Random Forest again outperformed for the positive and negative classes with scores of 0.85. For Precision, KNN outperformed for the negative class with a score of 0.93, while CNN outperformed for the positive class with a score of 0.88. Lastly, Random Forest achieved the best precision for the negative class with a score of 0.86, while KNN achieved a score of 0.89 for the positive class. In general, Random Forest seemingly outperformed most models based on the results. Additionally, the ensemble implementations seemingly performed well in general across both classes. Other models such as ANN and CNN also performed well, suggesting that statistical and DL models performed similarly. Figure 39 shows the embedding space for this quality attribute. Seemingly, the embedding implementation struggled with separating the two classes.

Table 33 Verifiability Model Results

Model	Class	Precision	Recall	F1-Score	Accuracy
ANN	N	0.82 ± 0.09	0.78 ± 0.06	0.80 ± 0.05	0.80 ± 0.06
CNN	N	0.87 ± 0.08	0.73 ± 0.11	0.78 ± 0.06	0.80 ± 0.06
XLNet	N	0.70 ± 0.06	0.86 ± 0.04	0.77 ± 0.03	0.75 ± 0.04
DistilBERT	N	0.72 ± 0.06	0.73 ± 0.13	0.71 ± 0.05	0.72 ± 0.03
DistilBERT Few-Shot	N	0.64 ± 0.01	0.92 ± 0.04	0.76 ± 0.02	0.61 ± 0.03
DistilBERT Siamese	N	0.75 ± 0.07	0.72 ± 0.16	0.72 ± 0.10	0.75 ± 0.06
Ensemble (Average)	N	0.83 ± 0.04	0.77 ± 0.05	0.80 ± 0.04	0.81 ± 0.04
Ensemble (Majority)	N	0.82 ± 0.04	0.79 ± 0.06	0.80 ± 0.04	0.81 ± 0.04
KNN	N	0.68 ± 0.09	**0.93 ± 0.04**	0.78 ± 0.05	0.74 ± 0.05
Logistic Regression	N	0.80 ± 0.10	0.78 ± 0.08	0.79 ± 0.07	0.79 ± 0.07
Meta-Model (Gradient Boost)	N	0.83 ± 0.06	0.70 ± 0.15	0.76 ± 0.12	0.78 ± 0.09
Meta-Model (Logistic Regression)	N	0.84 ± 0.05	0.74 ± 0.07	0.79 ± 0.05	0.80 ± 0.05
Random Forest	N	**0.86 ± 0.08**	0.84 ± 0.06	**0.85 ± 0.05**	**0.85 ± 0.05**

RNN	N	0.82 ± 0.08	0.79 ± 0.08	0.80 ± 0.04	0.80 ± 0.05
Support Vector Machine	N	0.83 ± 0.08	0.78 ± 0.10	0.80 ± 0.07	0.81 ± 0.06
ANN	Y	0.79 ± 0.08	0.82 ± 0.10	0.80 ± 0.07	0.80 ± 0.06
CNN	Y	0.76 ± 0.10	**0.88 ± 0.08**	0.81 ± 0.07	0.80 ± 0.06
XLNet	Y	0.83 ± 0.04	0.66 ± 0.08	0.73 ± 0.05	0.75 ± 0.04
DistilBERT	Y	0.74 ± 0.08	0.71 ± 0.11	0.71 ± 0.04	0.72 ± 0.03
DistilBERT Few-Shot	Y	0.22 ± 0.17	0.04 ± 0.04	0.07 ± 0.06	0.61 ± 0.03
DistilBERT Siamese	Y	0.77 ± 0.09	0.78 ± 0.10	0.77 ± 0.05	0.75 ± 0.06
Ensemble (Average)	Y	0.79 ± 0.05	0.84 ± 0.05	0.81 ± 0.04	0.81 ± 0.04
Ensemble (Majority)	Y	0.80 ± 0.06	0.83 ± 0.04	0.81 ± 0.04	0.81 ± 0.04
KNN	Y	**0.89 ± 0.07**	0.56 ± 0.08	0.68 ± 0.05	0.74 ± 0.05
Logistic Regression	Y	0.79 ± 0.08	0.80 ± 0.11	0.79 ± 0.08	0.79 ± 0.07
Meta-Model (Gradient Boost)	Y	0.75 ± 0.09	0.87 ± 0.04	0.80 ± 0.07	0.78 ± 0.09
Meta-Model (Logistic Regression)	Y	0.77 ± 0.06	0.86 ± 0.05	0.81 ± 0.05	0.80 ± 0.05
Random Forest	Y	0.84 ± 0.07	0.86 ± 0.09	**0.85 ± 0.07**	**0.85 ± 0.05**
RNN	Y	0.79 ± 0.09	0.81 ± 0.09	0.80 ± 0.07	0.80 ± 0.05
Support Vector Machine	Y	0.80 ± 0.09	0.84 ± 0.08	0.82 ± 0.06	0.81 ± 0.06

Based on the statistical significance results of the various models for class 0 (Figure 37), there was broad distribution of classes. For precision, CNN and Random Forest outperformed Siamese BERT and below based on statistical significance. CNN through Logistic Regression performed similarly but with a notably decreasing mean result. Still, the high variability within the data, as was the case with the other quality attributes, makes it difficult to determine the top performing models. For Recall, KNN outperformed the vast majority of models including Majority Vote Ensemble and Below. KNN performed statistically similarly to BERT Few-Shot, XLNet, and Random Forest. BERT Few-Shot also outperformed Average Ensemble and below, a notable deviation from past performance across quality attributes. For F1, Random Forest outperformed Siamese BERT and BERT, but most models performed statistically similar. Accuracy showcased a different story with Random Forest outperforming Siamese BERT, XLNet, KNN, BERT, and BERT Few-Shot. Additionally, Random Forest performed statistically similar to the other models but did achieve the top mean result.

Based on Figure 38, there was similar distribution for the positive class. For Precision, KNN outperformed Gradient Boost, BERT, and BERT Few-Shot, but there was strong model agreement and results with the rest performing statistically similarly, even with strong mean differences. For recall, CNN, Gradient Boost, Random Forest, Logistic Regreesion, Average Ensemble, and SVM outperformed BERT, XLNet, KNN, and BERT Few-Shot. For F1, Random Forest outperformed XLNet, BERT, KNN, and BERT Few-Shot, while all other models performed statistically similarly. Overall, the distribution for the positive class did not show a clear outperformer among statistics. More data exploration is necessary to understand which models truly better learn the data associated with the quality attribute.

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 0)

Factor	N	Mean	Grouping
CNN	10	0.8652	A
Random Forest	10	0.8646	A
LOG REG META	10	0.8404	A B
Gradient Boost	10	0.8345	A B
SVM	10	0.8327	A B
Average Ensemble	10	0.8320	A B
MAJ. VOTE ENSEMBLE	10	0.8228	A B C
ANN	10	0.8207	A B C
RNN	10	0.8177	A B C
LOG REG	10	0.8021	A B C D
Siamese	10	0.7549	B C D E
BERT	10	0.7185	C D E F
XLNET	10	0.6992	D E F
KNN	10	0.6761	E F
BERT FEW SHOT	10	0.64152	F

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 0)

Factor	N	Mean	Grouping
KNN	10	0.9309	A
BERT FEW SHOT	10	0.9195	A B
XLNET	10	0.8562	A B C
Random Forest	10	0.8371	A B C D
MAJ. VOTE ENSEMBLE	10	0.7890	B C D
RNN	10	0.7868	B C D
ANN	10	0.7845	B C D
SVM	10	0.7833	B C D
LOG REG	10	0.7830	B C D
Average Ensemble	10	0.7745	C D
LOG REG META	10	0.7398	C D
CNN	10	0.7292	C D
BERT	10	0.7275	C D
Siamese	10	0.7190	C D
Gradient Boost	10	0.7006	D

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.8485	A
MAJ. VOTE ENSEMBLE	10	0.8044	A B
SVM	10	0.8037	A B
Average Ensemble	10	0.8014	A B
ANN	10	0.7985	A B
RNN	10	0.7971	A B
LOG REG	10	0.7883	A B
LOG REG META	10	0.7853	A B
CNN	10	0.7841	A B
KNN	10	0.7788	A B
XLNET	10	0.7675	A B
Gradient Boost	10	0.7563	A B
BERT FEW SHOT	10	0.75556	A B
Siamese	10	0.7230	B
BERT	10	0.7138	B

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 0)

Factor	N	Mean	Grouping
Random Forest	10	0.8517	A
SVM	10	0.8138	A B
MAJ. VOTE ENSEMBLE	10	0.8086	A B
Average Ensemble	10	0.8080	A B
CNN	10	0.8017	A B
ANN	10	0.8017	A B
RNN	10	0.8000	A B
LOG REG META	10	0.7989	A B C
LOG REG	10	0.7914	A B C
Gradient Boost	10	0.7828	A B C
Siamese	10	0.7540	B C
XLNET	10	0.7517	B C
KNN	10	0.7397	B C
BERT	10	0.71724	C
BERT FEW SHOT	10	0.61349	D

Means that do not share a letter are significantly different.

Figure 37 Significance Testing for Verifiability (Class 0)

Grouping Information Using the Tukey Method and 95% Confidence

Precision (Class 1)

Factor	N	Mean	Grouping
KNN	10	0.8876	A
Random Forest	10	0.8399	A B
XLNET	10	0.8323	A B
SVM	10	0.8038	A B
MAJ. VOTE ENSEMBLE	10	0.7976	A B
Average Ensemble	10	0.7886	A B
RNN	10	0.7886	A B
ANN	10	0.7870	A B
LOG REG	10	0.7861	A B
Siamese	10	0.7748	A B
LOG REG META	10	0.7687	A B
CNN	10	0.7640	A B
Gradient Boost	10	0.7508	B
BERT	10	0.7376	B
BERT FEW SHOT	10	0.2189	C

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Recall (Class 1)

Factor	N	Mean	Grouping
CNN	10	0.8768	A
Gradient Boost	10	0.8664	A
Random Forest	10	0.8630	A
LOG REG META	10	0.8583	A
Average Ensemble	10	0.8425	A
SVM	10	0.8380	A
MAJ. VOTE ENSEMBLE	10	0.8301	A B
ANN	10	0.8165	A B
RNN	10	0.8145	A B
LOG REG	10	0.7975	A B
Siamese	10	0.7846	A B
BERT	10	0.7092	B C
XLNET	10	0.6551	C D
KNN	10	0.5592	D
BERT FEW SHOT	10	0.0432	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

F1 (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.8494	A
SVM	10	0.8159	A B
Average Ensemble	10	0.8139	A B
MAJ. VOTE ENSEMBLE	10	0.8125	A B
CNN	10	0.8114	A B
LOG REG META	10	0.8100	A B
Gradient Boost	10	0.8019	A B C
ANN	10	0.7988	A B C
RNN	10	0.7974	A B C
LOG REG	10	0.7872	A B C
Siamese	10	0.7731	A B C
XLNET	10	0.7296	B C D
BERT	10	0.7125	C D
KNN	10	0.6799	D
BERT FEW SHOT	10	0.0705	E

Means that do not share a letter are significantly different.

Grouping Information Using the Tukey Method and 95% Confidence

Accuracy (Class 1)

Factor	N	Mean	Grouping
Random Forest	10	0.8517	A
SVM	10	0.8138	A B
MAJ. VOTE ENSEMBLE	10	0.8086	A B
Average Ensemble	10	0.8080	A B
CNN	10	0.8017	A B
ANN	10	0.8017	A B
RNN	10	0.8000	A B
LOG REG META	10	0.7989	A B C
LOG REG	10	0.7914	A B C
Gradient Boost	10	0.7828	A B C
Siamese	10	0.7540	B C
XLNET	10	0.7517	B C
KNN	10	0.7397	B C
BERT	10	0.71724	C
BERT FEW SHOT	10	0.61349	D

Means that do not share a letter are significantly different.

Figure 38 Significance Testing for Verifiability (Class 1)

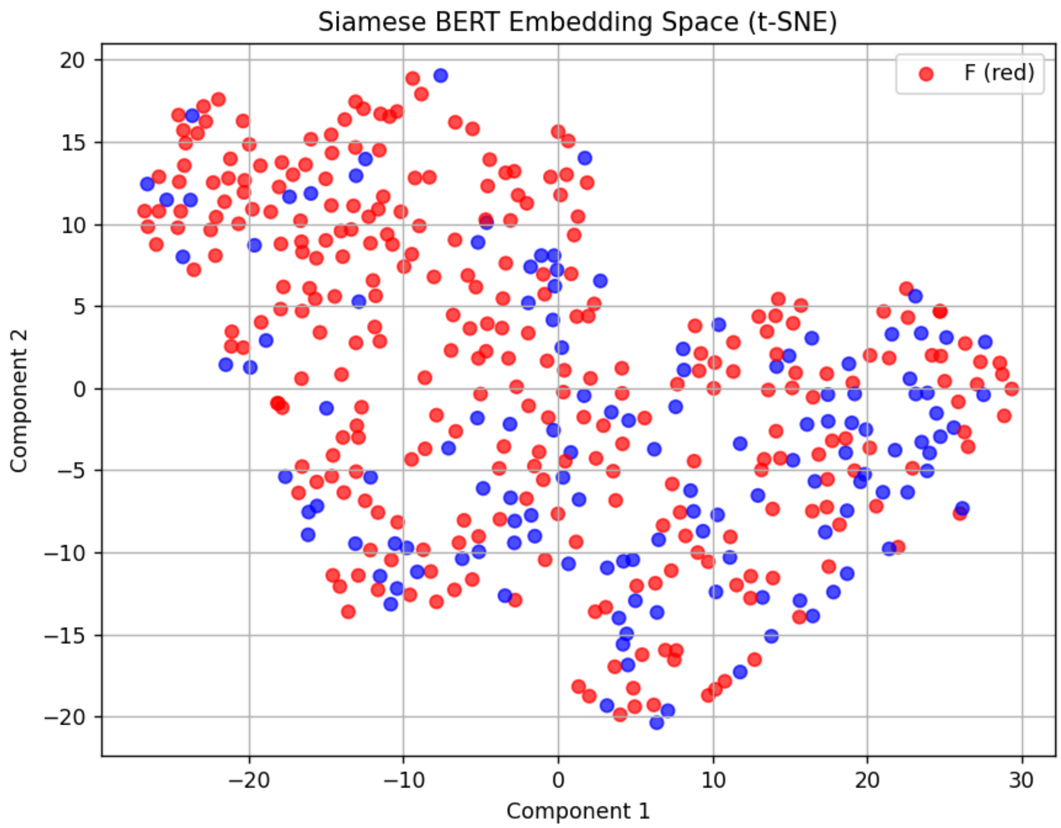

Figure 39 Verifiability Embedding Space

Conclusion

In this chapter, the discussion provided insight into the practical applications of NLP and AI for requirement automation processes. Specifically, the analysis looked at a variety of topics ranging from requirement identification to requirement quality analysis. Based on the results of the PURE dataset, high metrics were achieved for requirement identification among quality models. From these models, the features were synthesized into rule-based heuristics to help understand what ultimately delineated a requirement. The results showed that common template requirement template words did typically indicate a requirement. Additionally, formally language often contributed to the positive class of requirement identification. Although these results were helpful in understanding the formulations of requirements, the results showed overlap among words, suggesting that requirement-specific features likely were derived from syntactic and semantical features.

Complicated models such as DistilBERT were able to learn the broad patterns of the PURE dataset with high accuracy. Simpler models such as Logistic Regression also achieved high accuracy, suggesting a simpler pattern complexity. Although, larger models were required to achieve the highest metrics, smaller models achieved adequate accuracy for most practical implementations. For the manually annotated dataset, there was more consistency among model performance. Larger models did not necessarily outperform smaller models. With a relatively large Kappa agreement, AI models learned the patterns from the datasets to achieve high accuracy. Notably, there was disagreement among the PURE dataset and the user annotated dataset, suggesting a slight disconnect between the perception of requirements. The analysis further validates the ability of the models to learn user annotations through the Fleiss' Kappa agreement provided in Table 34.

Lastly, the results analyzed the quality results of the human annotators and the ability of AI models to learn from the labeled datasets. For the analysis, the class labels were relatively consistent among ambiguity, singularity, and verifiability. The reviewers disproportionately labeled requirements as feasible for the datasets, limiting the ability to train the associated models to predict quality attribute violations. For the model results among the other quality attributes, the analysis shows a strong ability to learn from the user dataset. Often, the Siamese BERT implementation, a discriminator that optimizes embeddings, outperformed for most of the tasks. Similarly to Requirement Identification, there was not significant model result difference for large or small models. With the added Kappa agreement among quality attributes, there is a practical ability to encapsulate patterns of most quality attribute violations.

Table 34 Fleiss' Kappa Agreement

Method	**Fleiss' Kappa**
PURE Requirement Identification	0.749
Human Requirement Identification	0.703
Ambiguity	0.546
Singularity	0.641
Verifiability	0.542

Chapter 6

Results and Discussion

In this praxis, the primary focus was the creation of a practical tool that could analyze requirement documents. Based on the progression of the research, the practical tool, ARQM, evolved into many forms for both UI implementation, structure, and methodological implementation. Primarily, the prototype of the application started as a Microsoft Word Add-In and required extensive user input and data labeling to adequately analyze requirements. The original intent of the application was to incorporate user input and generate a meta structure to help better analyze requirement documents, ultimately mitigating the need for automatic parsing of requirements while still utilizing the relevant AI methodologies for analysis. As the tool evolved to an automated form and mitigated the necessity of user input, the necessity of automated requirement identification and quality analysis became more apparent.

To accomplish an automated tool for requirement analysis, requirement identification and quality analysis became pivotal for the automation process. When considering the different formats and structures of documents, document parsing and structuring became an added consideration to help identify and analyze requirements. With some basic processing and AI-based implementation, the ARQM tool can ingest regular requirement documents and analyze the associated requirements. The associated limitations and advantages of the tool were seen with the added complexities of model tuning and providing context before inference. To understand the effectiveness of the methodologies behind the tool, the analysis looks at the associated research questions for both requirement identification and quality analysis along with the practical implementations for both categories.

Requirement Identification: Research Questions and Practicality

To appropriately understand the results, the analysis looks at the research questions provided previously to help better understand the practical limitations and advantages of the ARQM application. The research questions require not only an in-depth understanding of the results, but also the associated features that helped for inference among both classes. Naturally, requirement identification was a simpler task due to the availability of labeled data and simplistic inference standards. Below are the answers to the research questions originally provided.

1. *How can a practical tool identify requirement-related text in differing unstructured documents?*

Through an understanding of the various research and limitations of string process, the common process of document parsing was complicated. For the practical implementation of the ARQM tool, the implementation broke up the document by sentences and then performed inference on each individual unit through trained models. Although this method had limitations, determining continuity of requirements was difficult, and there were not a lot of pre-existing implementations, datasets, or models that could aid with the task. The ARQM tool did utilize native libraries to help combine sentences together, mitigating the separation of individual requirements, but existing implementations seemingly had limitations.

Ultimately, the ARQM tool utilized simple pre-processing processes and libraries, including pattern recognition, to help parse unstructured data. The results indicated that most requirement units were kept in a whole unit through the use of simple heuristics, allowing for better processing and inference. Additionally, simplistic and complicated models were able to identify requirements with high accuracy. When trained on the PURE dataset, AI-based models achieved impressive inference for requirement identification, with the highest achieved accuracy of 0.92 and an F1-Score of 0.92 among certain models for both classes. Other models were able to achieve low to mid 90s for Recall and Precision, suggesting a strong ability to learn the data patterns. The results also indicated success among the human-annotated dataset, suggesting a high agreement and ability for models to learn patterns for automated requirement identification. Of the existing datasets, the manual requirement identification dataset had some of the highest agreement among annotators, with a maximum Kappa of 0.457. Even though there was high agreement, the results indicated lower success among models and higher variation. For instance, the best performing model, Logistic Regression, achieved an 0.89 accuracy. High F1-Scores were 0.89 and 0.88 for Logistic Regression as well, whereas ANN, CNN, and KNN outperformed for recall. Lastly, Logistic Regression and ANN outperformed for Precision. In general, Logistic Regression learned the patterns better than the other models and other, more complicated models did not necessarily improve performance. Ultimately, different models may be suited for certain use cases depending on the practical application.

For the statistical significance testing, CNN, Random Forest, and BERT generally outperformed among the various results for the PURE dataset. Logistic Regression, Gradient Boost, and the various ensemble methods also performed statistically similar, suggesting that different models were able to learn the representations of the data appropriately. In general, CNN outperformed for both classes, suggesting that the CNN model may have encapsulated the data better for the PURE dataset. Importantly, CNN was able to uncover the highest recall for the positive class, making it the superior model for detecting all requirement text. Random Forest was better with identifying the recall for the negative class, making it the superior model for identifying non-requirements. Of the other models, complicated models generally performed at or below most statistical models. For instance, the Siamese BERT model, XLNet, and BERT Few-Shot were consistently the worst performers, suggesting simpler models may better identify requirement text.

For the manual requirement identification dataset, there was large group overlap for performance. Generally, most of the models performed statistically the same, even though their means differed drastically. The high variation suggests that more research is needed to understand the data along with methods to help control outliers. Again, the worst performing models included Siamese BERT, BERT Few-Shot, and other LLMs. Surprisingly, there was notable diversity for the top mean scores among models, but their performance cannot be ruled as significant according to the Tukey test. Due to the high variation of mean results, there was no clear superior performers for the manual requirement identification dataset.

Based on the original results outlined in the PURE dataset study, the results were able to achieve better precision, recall, and F1 scores across the different classes. Based on the study, their highest results included an F1 score of 0.86, a precision of 0.92, and a recall of 0.82 among the various classes (Ivanov et al., 2022). For the model results, this study showcased a precision of 0.96, a recall of 0.91, and an F1 score of 0.92. Another study utilized a combination of the Dronology and PURE datasets. Although the results are not necessarily comparable, they did perform important work utilizing LLMs and auto-regressive models. Ultimately, their macro results performed well with an accuracy of 0.95, a precision of 0.97, a recall of 0.95, and an F1 score of 0.95. Ultimately, the study showcased better results among the dataset when compared to the previously outlined results in this study. Aside from their DeBERTa + Llama2 ensemble implementation, their other models drastically underperformed the benchmarked results. Ultimately, their results did not utilize significance testing, and their PURE dataset was combined with the Dronology dataset, limiting the result comparison validity (Saleem et al., 2025). Because this study utilized more diverse and representative models, the results seemingly indicated that statistical and simpler models were able to achieve at or above the existing literature results. Critically, this suggests that simpler models can be used for requirement identification tasks, limiting the computational resources needed for an applied requirement application.

2. *How can existing tools and methods address the lack of training data for requirement identification?*

For requirement identification, there was minimal issue with dataset availability. The PURE dataset provided thousands of examples across multiple requirement documents, allowing for the broad training of AI models. Of these entries among the PURE dataset, requirements were well represented among multiple domains and syntaxes. Although the dataset was extensive, the types of requirements among non-functional and other syntaxes were not well represented.

To improve the generalization of the dataset during training, many methods exist to help models better generalize. For instance, pre-processing techniques, limiting model complexity, feature engineering, and other normalization tasks can help reduce the overfitting of models. Additionally, manipulation of requirement text could also help via synonym replacement or abstract summarization. Additional techniques such as manual annotation and sequence generation from LLMs can also be associated with generating new data for requirement identification.

Ultimately, the features of the associated models were simplistic. Common requirement words such as 'Shall' and 'Should' were often present and crucial for model inference. By utilizing these features, simpler models achieved high accuracy among the PURE dataset, suggesting an ability for generalization. With the extraction of critical features present within the PURE dataset labeled entries, simplistic rules and heuristics could likely also generalize unseen formats of requirements. Additionally, the results showed a high agreement among the PURE dataset and the human-annotated dataset for inference.

To achieve higher accuracy among requirements, utilizing features for syntactic and lexical analysis is crucial. Additionally, semantic and pragmatic representations of text could also prove helpful when helping to identify requirements. Through features representative of textual semantics, the generalization of requirement identification is likely improved. Based on the results, there were common words that indicated requirements, but there was also a broader pattern beyond simple words and phrases present during the inference, suggesting the need for more complicated heuristics.

3. *How effectively can the best algorithm identify requirements based on accuracy, precision, recall, and generalization across different documents?*

Based on the results for the PURE dataset, the highest accuracy was achieved with just Lemmatization, along with certain DL models and statistical models. CNN, DistilBERT, Random Forest, and RNN all achieved an accuracy of 0.92 with minimal variation. Generally, RNN outperformed among both closes for Recall and F1-Score, making the model superior in most cases. For Precision, CNN, RNN, and Random Forest generally also outperformed. Based on the results of the models, there was higher performance in the negative class for Recall. For instance, Random Forest achieved recall of 0.96 for the negative class, whereas the model achieved 0.91 for the positive class. This suggests that it may be generally easier to identify non-requirements. With the minimal variation, most models were able to achieve higher results among the PURE dataset, mitigating the need for complicated models. Lastly, there were marginal improvements across select statistical models when utilizing Lemmatization, even though the result improvement was minimal. Stop Words improved the results notably across ANN, KNN, Random Forest, and ANN. This suggests that some pre-processing may help achieve better results.

For the human-annotated dataset, Logistic Regression outperformed for accuracy with a score of 0.89. With moderate exception, Logistic Regression also outperformed for the statistics among the positive class. The highest results for F1-Score was 0.89 and 0.88 for the negative and positive classes, respectively. ANN, CNN, and KNN outperformed for recall; ANN and CNN achieved a score of 0.89 for the positive class and KNN achieved a score of 0.92 for the negative class. Lastly, Logistic Regression achieved a score of 0.90 for the positive class among precision entries, while ANN achieved the high score of 0.89 for the negative class. Notably, there was higher variation among the results, indicating less consistent convergence of models based on different data divisions. For the statistical significance testing, the results showed that there were clear model best performers among the PURE dataset, with CNN, BERT, Logistic Regression, and Random Forest outperforming. Primarily, CNN seemingly learned the data representations better based on the mean result. For the human annotated dataset, there were not clear performers based on the statistical significance results of the models. Due to the large overlap of models, the high deviation of the results suggests more entries are needed.

4. *What are the roles of lexical, syntactical, and semantical textual features in performing requirement identification?*

Based on the analysis of the features for requirement identification, there were certain syntactical and lexical elements that typically helped to indicate a requirement or non-requirement classification. Typically, formal language often indicated requirement text. Words such 'Shall' and 'Should' often were indicative of positive inference among most models. For the negative class, less formal language was seemingly suggestive of non-requirement text. Words such as 'administration' and 'process' typically contributed negatively to the classification but also had overlap among both classes.

Based on the feature synthesis and results from the PURE dataset, more complicated structures were likely indicative of requirement delineators. Specifically, features on the syntactical and semantical analysis likely helped to determine the classification of text. Although the analysis of these features was outside the scope of this analysis, models that relied on embeddings and semantical representations of text typically outperformed when compared to the statistical models for the large PURE dataset. However, most statistical models achieved comparable accuracy to these more complicated models for the manual datasets, suggesting that requirements may be easily represented through common patterns and rules instead of complicated LLM analysis.

Ultimately, the analysis utilized pre-processing and embedding techniques for all models. For the statistical models, pre-processing and embeddings did not drastically improve the outcome; in some cases, the resultant accuracy was worse. Other models such as CNNs and RNNs benefited from embeddings for accuracy but still performed similarly to statistical models. Lastly, the LLM implementations such as BERT utilized embeddings and achieved the best performance among both the PURE and human-annotated datasets.

5. *What features were useful during the model inference process for requirement identification?*

Primarily, the analysis utilized pre-processing techniques among the different models. Methods such as stemming, lemmatization, or TF-IDF did not significantly improve the model results, although some model results were positively affected. Other models such as CNNs and RNNs utilized both vectorization and embedding during the inference process. For more complicated models, the embeddings seemingly helped improve model accuracy. Further fine-tuning of the associated hyper-parameters could yield better results.

As mentioned previously, there was formal language associated with requirement text. Typical requirement syntax words helped inform the models during classification but were also limited among the entire datasets. The model features seemingly learned features through the entire representation of sentences. Because pre-processing did not have a significant impact on accuracy, the inference process for all models likely depended on the entire sentence context instead of a few words or phrases.

Quality Analysis: Research Questions and Practicality

1. *How can a practical tool identify quality defects in unstructured documents based on various quality attributes outlined in the IEEE 29148:2018 standard?*

The Quality attributes specified in the IEEE 29148:2018 standard apply to multiple levels of analysis. Based on the type of analysis conducted, AI models seem to be capable of analysis on a lexical, syntactical, semantical, and pragmatic basis. To adequately train models to detect quality attribute violations, the utilization of datasets becomes a problem. As part of the experimentation, the analysis provided multiple reviewers with requirements. Utilizing the definitions of quality violations, the reviewers marked quality violations associated with each requirement. From these results, pre-processing and data normalization techniques helped to enhance the data generalizability for unseen requirements.

Based on practical AI model implementations, the ARQM application was able to uncover and predict potential violations through the learned textual representations and features provided through the datasets. Through the additional utilization of overfitting reduction techniques and feature engineering, the ARQM application synthesized the user classifications into broad patterns and features, allowing for an automated process to take place when analyzing requirements. Because of the dataset limitations, explanation generation was an additional challenge. Methods available via XIA provided an opportunity to analyze models to help provide predictions of importance on a per-token basis, completing the automated analysis for requirement quality.

Dataset limitations were a primary concern for the training of quality attribute violation detection. To mitigate the problem, common techniques previously discussed for synthetic data generation could prove useful. Unfortunately, techniques such as synonym replacement and abstract summarization could eliminate pivotal context necessary for quality attribute detection, making text transformation difficult. Adjusting most parts of the sentence could potentially affect the meaning, rendering the data less useful. Methods such as automatic labeling through semi-supervised learning could work, but there are further restrictions that may mitigate the accuracy of models. To determine useful quality attribute violations, human-annotated datasets were particularly useful.

2. What quality attributes are measurable with AI-based methods?

Ultimately, every quality attribute is measurable with AI-based methods, although some quality attributes require additional context. For instance, completeness requires an understanding of the document metadata and associated requirements and structure; to derive these items, extensive and difficult processing is likely required. Still, many of these quality attributes can be analyzed at some granular level regardless of the context. There are a variety of words, phrases, and semantical representations that, regardless of the metadata manifestation, would likely constitute a violation. However, these types of analytical techniques have a smaller scope and are likely less useful.

For the practical implementation of the ARQM project, the analysis provided insight into four quality attributes: ambiguity, singularity, feasibility, and verifiability. Based on the results of the model training, AI models were able to measure and accurately predict ambiguity, singularity, and verifiability. For Ambiguity, the highest accuracy was achieved by the Random Forest model with a score of 0.84; the highest accuracy achieved under Singularity was 0.93 by the Random Forest model; the highest accuracy achieved under Verifiability was 0.85 by the Random Forest model, with other models achieving notably less accuracy. Other mid-range values for Recall, Precision, and F1-Score seem to indicate that the quality attributes are generally measurable and detectable with both statistical and DL models. Seemingly, Random Forest generally outperformed for the manual datasets among most quality attributes.

For feasibility, the raters agreed that most of the requirements provided were feasible. Primarily, this suggests that feasibility has a higher and looser interpretation during the annotation process. Additionally, the lack of context could have also impacted the reviewers' perceptions of potential violations. Because of the lack of balanced data, the models were unable to train and predict potential feasibility violations. For further research, analyzing feasibility would likely require a better understanding of context and domain-specific information for human annotators. With the associated results of the feasibility attribute analysis, many other quality attributes are likely limited during the human annotation process.

Based on the significance results, Random Forest, CNN, and KNN were typically the top performing models. For Ambiguity, there were a variety of class delineations, suggesting genuinely different model performance. Random Forest generally outperformed for the positive class, whereas different models such as CNN and KNN outperformed for the negative class. The ensemble models performed well, with statistical similarity to the top results. Other, more complicated models, generally performed worse. Similarly for Singularity, Random Forest outperformed for the positive class. Surprisingly, Random Forest also outperformed for the negative class, suggesting significant model performance based on the highly deviated results across models. Other top performers included BERT Few-Shot and KNN. For Verifiability, Random Forest performed well, but did not necessarily outperform in most cases. For Precision and Recall of the positive class, KNN and CNN outperformed. For the negative class, CNN and KNN outperformed among Precision and Recall as well. Although Random Forest did not perform the best across all metrics, the model was able to outperform in most cases among the various quality attributes.

3. *What role does context play in determining the quality of a requirement across the various quality attributes?*

Context is a pivotal element for accurate requirement analysis. However, the limitations of consolidating and providing context for model inference is difficult. To mitigate this issue, the tacit knowledge provided within LLMs and other model pre-processing techniques can help overcome the need for immediate context. For instance, multiple NLP techniques, such as SLR and POS tagging can help a model better understand the associated relationships between words from within text. With this critical knowledge, the models can likely generalize even when context is not necessarily provided. As an additional step, LLM tacit knowledge can help apply context based on its own extensive training to incorporate better inference results.

Even with the results of the extended models, there are limitations when models do not receive the context of the requirement document. Alternative methods such as RAG and Abstractive Summarization can help with providing models with critical information, but metadata document collection is still a challenge due to the unstructured nature of requirement documents. Without the context provided to models, there are also limitations for the scope of analysis. Seemingly, most quality attributes are measurable without context for lower levels of analysis. For instance, lexical analysis often does not require context, with exception to references and links.

Even though the encapsulation of document context is possible, there are extensive processing requirements and challenges. Instead of generating a structured representation of requirement documents, this analysis utilized quality attribute violations for individual requirements in conjunction with manual annotation. Based on the results, there were significant violations uncovered during the labeling process, suggesting that context does not necessarily matter for surface level or certain semantical components of violations. Ultimately, the reviewers did not have access to document context, likely creating simplistic quality violations during the annotation process.

4. How can existing tools and methods address the lack of training data for requirement quality analysis?

Unlike requirement identification, there was a lack of dataset representation for quality attribute violations. The study mitigates this issue through the utilization of human annotated datasets, but further limitations still exist. As mentioned previously, data augmentation is difficult for requirement quality violations due to the textual dependencies that often constitute violations. Seemingly, manual annotation is the strongest way to generate requirement quality violations but remains an expensive and tedious task.

One of the key issues with the ARQM implementation was providing the user with feedback for violations. Originally, the application simplified the potential explanations for quality attribute violations. For sequence models, the datasets were not adequate to consistently generate useful explanations. When the problem was synthesized into a simple classification task instead of a generation task, the accuracy improved drastically. Through the use of XAI, the ARQM application simply analyzes the model's inference importance among tokens. Through SHAP, the generative task was no longer required, ultimately providing a clear visual of high impact violations within sentences.

5. How effectively can the best algorithm identify requirement quality defects based on accuracy, precision, recall, and generalization across different documents?

For ambiguity, the highest accuracy achieved among both classes was from the Random Forest implementation with an accuracy of 0.84. The model also achieved a Precision, Recall, and

F1-Score of 0.84, 0.87, and 0.84 for the negative class, and a Precision, Recall, and F1-Score of 0.85, 0.82, and 0.84 for the positive class. For singularity, Random Forest achieved an accuracy of 0.93. Additionally, the models achieved scores of 0.94, 0.97, and 0.92 for Precision, Recall, and F1, respectively for the negative class and 0.92, 0.94, and 0.92 for the positive class. Lastly for verifiability, the Random Forest model achieved the highest accuracy with an accuracy of 0.85. The highest performing models achieved scores of 0.86, 0.93, and 0.85 for Precision, Recall, and F1, respectively, while the models also achieved scores of 0.89, 0.88, and 0.85 for the positive class. In general, there was higher variation among the different quality attribute results. Random Forest seemingly performed the best across most metrics, while other models performed better for different metrics.

Based on the results of model inference, singularity was the quality attribute with the highest accuracy. Other quality attributes like ambiguity and verifiability achieved high accuracy among models, but were ultimately limited. Their associated Kappa scores indicate that although there was agreement, there was also moderate disagreement for what constituted a violation. Ultimately, the Random Forest model outperformed in most cases among these various quality attributes. Ultimately, simpler models like Random Forest and Logistic Regression were able to perform on-par or even outperform more complicated models consistently.

6. *What methods exist to help with the generation of explanations for quality inferences?*

Originally, the ARQM application primarily utilized sequence generation models to help with explanations. Due to the limitations of datasets, this method proved unfeasible and highly impractical. As a result, the implementation utilized XAI to help generate explanations for quality violations. Utilizing SHAP, the algorithm analyzes how the model was impacted based on slight changes in the input. Through this method, the SHAP algorithm utilizes Game Theory to help determine token importance relative to the classification results.

Once the token importance was generated, the ARQM algorithm synthesized the results into a data structure for visualization. Through various PDF generation techniques, the ARQM application utilized colors to indicate the most impactful parts of sentences for a specific quality attribute violation. With this particular method, the ARQM application visualizes the parts of sentences that contributed negatively to a violation, allowing the user to visualize and understand ways to correct the associated requirement. Through this method sequential generation became unnecessary, allowing for the full automation of requirement analysis without extensive datasets.

Additional methods previously discussed serve as potential ways to visualize token impact. Each of these methods, such as Integrated Gradients, utilize different techniques that are helpful with model inference visualization. Further research could look into these methods to help understand the usefulness of token visualizations in relation to user understanding. Lastly, simpler methods exist to help understand model reasoning. For instance, meta models can help synthesize LLMs into smaller models. From these smaller models, various coefficients could help provide an algorithm with the information necessary for visualization. Additionally, these methods could help extract critical rule-based heuristics. Based on these heuristics, the implementation could map specific violations to explanatory messages.

Chapter 7

Conclusion

Summary

Ultimately, the analysis indicates that automation can achieve requirement identification and quality analysis. With public datasets such as PURE, models achieved high accuracy for identifying requirements. Although limitations exist, the PURE dataset was well representative of multiple different requirement syntaxes and domain representations, increasing the generalizability of requirement inference for models. Additionally, the PURE dataset had a high correlation with the manually annotated dataset, suggesting a high agreement for what constitutes requirement and non-requirement text.

Through basic NLP techniques and textual consolidation, ARQM achieved broad processing of unseen requirement documents. As a preliminary step, the ARQM application identified entire multi-sentence requirements through simplistic heuristics and pre-processing. After processing, the ARQM tool was able to analyze the units and components to extrapolate requirements from unstructured text. For both the PURE and human-annotated datasets, ARQM achieved high accuracy among both classes for requirement identification.

For quality analysis, dataset scarcity is becoming an issue for model inference. Through the use of human annotated datasets, the analysis provided insight into ambiguity, feasibility, verifiability, and singularity. Of these associated quality attributes, ambiguity, verifiability, and singularity had balanced datasets, allowing for adequate model training. Due to the loose interpretation of feasibility, raters generally agreed that most requirements were feasible, limiting the analysis of violations for that specific quality attribute. Based on the results, the Siamese

BERT implementation often outperformed for all quality attribute inferences among both classes. Although the accuracy was highest among singularity, accuracy for other quality attributes was limited. Ultimately, the tool was rigorously tested. Models achieved high accuracy across the various datasets and generalized reasonably well to the annotator data among requirement identification and quality analysis. Due to the data limitations of the data, additional challenges and opportunities remain for requirement quality automation.

Threats to Validity

Internal Threats to Validity

I1: Reviewer Data Limitations

For the study, there are a few limitations that mitigate the generalizability of the datasets and the tool implementation. For instance, the manual datasets included a total of five reviewers. As a result of the minimal reviewers, there was limited agreement and consistency among the various datasets. The limitation was shown through some of the lower Kappa result scores, indicating that more reviewers could potentially help improve the training data and bolster the agreement. Additionally, usage of mitigation methods such as minority oversampling became necessary due to the dataset limitations from limited reviewers, further affecting the results.

For Requirement Identification, the highest agreement was a Kappa of 0.60 with a low result of 0.169; For Ambiguity, Verifiability, and Feasibility, there was no agreement between Annotator 1 and Annotator 2; and there was limited agreement among all annotators for the Verifiability task. The Singularity labeling task achieved low to moderate Kappa, but the agreement remained limited. Based on the Kappa results across the dataset, there was limited agreement for the reviewers, limiting the results of the ARQM tool. Ultimately, further studies should improve upon base the datasets, the associated imbalances, and the reviewer disagreements to help achieve better results.

I2: Unbalanced Data

For requirement quality, reviewers were given the opportunity to determine violations based on the IEEE 29148:2018 standard definitions for quality attributes. Although the results of the quality dataset were relatively balanced, many of the quality attributes did not have balanced data. To mitigate this concern, as a result, the use of minority dataset oversampling helped to balance the associated datasets, limiting the model generalizability. For certain quality attributes such as feasibility, there were not enough representative samples among both classes to train models, limiting the impact of model inference. Additionally, there was low to moderate Kappa agreement for the associated quality datasets, suggesting that there were minimal agreements for certain quality attributes. Although models achieved high accuracy for these specific quality attributes, the data may not properly encapsulate the appropriate general patterns for quality analysis.

External Threats to Validity

E1: Dataset Origins

The review process consisted of requirement documents that were generated through academic repositories, limiting the generalizability of the results. Primarily, these requirement documents were retrieved from a corpus of previous students tasked with generating requirement documents. Further, these requirement documents likely did not represent the typical industry formats of requirement documents, limiting the results of the ARQM tool. As an added consideration, the majority of the datasets were limited in size as well. With exception to PURE, the manual identification dataset and the corresponding quality violation datasets had minimal entries. As a limitation of this study, higher entry datasets could help significantly improve the model generalization results while also improving the understanding of model performance.

E2: Dataset Availability Limitations

For requirement quality analysis, there is a distinct lack of labeled datasets, prompting the generation of the various academic datasets used in the study. As a consequence, the results are limited to the data used during the training process. Broader analysis and development of quality datasets is needed to help validate the experimentation and practicality of the ARQM system. Although comparisons to other tools for requirement identification were used in the study, there were no comparisons due to the lack of reproducibility for quality analysis tasks across other applications.

E3: Existing Dataset Limitations

For the ARQM tool validation, the PURE dataset was used to measure the reliability of the requirement identification process. As a primary indicator of the tool's task ability, there was a high reliance on the PURE dataset for tool measurement. Therefore, the associated limitations of the PURE dataset could impact the generalizability of the ARQM tool for requirement identification tasks. Although the results were also validated with annotator-specific datasets, the

PURE dataset had the most entries and impact on the evaluation process. Given the minimal link between the annotator agreement and the PURE dataset, the ARQM tool's results are limited.

Construct Validity

C1: Requirement Segmentation Complexity and Loss of Information

As an added consideration, requirement segmentation, even with mitigation methods, proved highly difficult. There was not necessarily consideration on supporting information as well for the requirement, likely leading to a loss of context during the document ingestion process. With better processing of multi-sentence requirements, along with the aggregation of supporting requirement information, the results likely could have been improved. Ultimately, the inferences of the associated ARQM models had to use model-specific knowledge instead of the project-specific parameters to determine violations.

Additionally, the study only focuses on a select amount of quality attributes during the reviewer process. The quality attributes, Singularity, Feasibility, Verifiability, and Ambiguity were selected due to the relevant literature and practicality of single requirement evaluation. Although these quality attributes are often simpler to analyze, the attributes may have required additional context to make a useful evaluation to help bolster the data reliability from the annotations. The lack of background information related to the requirement could have led to a different interpretation of context that was critical for the annotator evaluation (Alemneh & Berhanu, 2024; Lami et al., 2019; Umar & Lano, 2024).

C2: Unrepresentative Data

For the data generation process, there were many complex controls in place to help extract requirements for user annotation. Ultimately, it is likely that this process was somewhat inaccurate to the representation of typical requirements. Often, requirements are complex and span multiple sentences. Although segmentation methods were used with NLP, the requirement entities provided to the reviewers were likely incomplete. This is a limitation due to the loss of data representation that could help users determine requirement classification.

Ultimately, the ARQM system relied on pre-defined heuristics of syntactical structure to help identify continuing requirements, along with various rule-based methods. More work could help inform the practicality and accuracy of these methods for typical requirement extraction. Also, the tools primarily relied on pre-defined patterns that may not necessarily represent requirements text due to the dynamic nature of requirement documents. Extensibility and utilization of ML methods could help better indicate requirement extensions.

C3: Unbalanced Data

The datasets were relatively unbalanced for some of the quality attributes. These unbalanced datasets made the process of achieving reliable, general inference of quality violations difficult. To overcome the imbalance, the process utilized minority oversampling, but there was still a lack of representation for useful data among negative and positive classes. In an attempt to overcome these limitations, the ARQM tool utilized methods such as few-shot training, but did not look into other ways of data replication that could have proven useful during the data training process.

Discipline Contribution

Due to the added complexity of existing engineering projects, requirements engineering has become a pivotal task for mitigating cost overruns and user dissatisfaction. With the added project complexity, requirements are also becoming more complicated, limiting the ability of human practitioners to manually analyze and correct requirement documents. As a result, current research tries to address and simplify the complicated requirements engineering process. Of the existing tools, many implementations realy on extensive user input or rule-based methodologies.

With the introduction of NLP and AI, there is a tremendous opportunity for practical tools to help analyze requirement documents at a deeper level. Even with the introduction of new methods and research, the limitations of practical tools and practitioner input leave a significant potential research gap. To address this practical research gap, the new tool named ARQM utilizes modern AI and NLP methods to process unstructured requirement documents. The final implementation of the ARQM tool focused primarily on simple document ingestion, requirement identification, and quality analysis, limiting complexity, and required human input.

As a primary focus of the research, the main purpose of the ARQM tool aims to simplify the process of requirement analysis. Through the utilization of AI, the tool can analyze most requirement documents, identify potential requirements, and analyze their associated quality, mitigating human involvement, and further simplifying the automated process. The analysis looked at the results of these elements and tasks, showing high accuracy among publicly available and human-annotated datasets. With the high accuracy among both requirement identification and quality analysis, the ARQM tool was able to automate the requirement document analysis process for both requirement identification and analysis. Additionally, the tool provides useful user feedback through the use of XAI, a method to visualize token importance in relation to model inference results.

Ultimately, the ARQM tool was informed by the research and past implementations of tools and methods. The tool also utilized practical input from human annotators by garnering human feedback on requirement identification and quality analysis among labeled and unlabeled datasets. The associated ARQM tool accomplished high accuracy among the human-annotated datasets, showing the ability for AI-based methods to learn and generalize to practical human input. As an added consideration, the ARQM application provides a user-centric focus through the use of XAI, PDF visualization of violations, and user-specific model training, allowing for a simplistic and intuitive application for the requirements engineering process.

Future Work

Due to the multiple iterations of the ARQM application, there was limited time to hone the technology, frameworks, and methods utilized. For instance, the model analysis for visualization utilizes SHAP, whereas other implementations may provide better, more promising results for explanatory text generation. Future work should focus on consolidating the ideal models and frameworks to help better train, interpret, and visualize requirement violations.

As an added consideration for future work, additional training data from human reviewers could help the model better learn general patterns for quality violations. The datasets used in this study were limited. With the ARQM basic implementation completed, further surveys about the User Interface (UI) and general experience and requirements could help further improve the practical product for use. Ideally, the model should closely align to user expectations. Further qualitative research should be done to understand ways to improve ARQM for both functionality and appearance.

References

Ahmad, A., Feng, C., Khan, M., Khan, A., Ullah, A., Nazir, S., & Tahir, A. (2020, July 15). *A Systematic Literature Review on Using Machine Learning Algorithms for Software Requirements Identification on Stack Overflow—Ahmad—2020—Security and Communication Networks—Wiley Online Library*. https://onlinelibrary.wiley.com/doi/10.1155/2020/8830683

Akay, H., Yang, M., & Kim, S.-G. (2021). Automating Design Requirement Extraction From Text With Deep Learning. *Volume 3B: 47th Design Automation Conference (DAC)*, V03BT03A035. https://doi.org/10.1115/DETC2021-66898

Alami, N., Arman, N., & Khamyseh, F. (2017). A semi-automated approach for generating sequence diagrams from Arabic user requirements using a natural language processing tool. *2017 8th International Conference on Information Technology (ICIT)*, 309–314. https://doi.org/10.1109/ICITECH.2017.8080018

Alemneh, E., & Berhanu, F. (2024). Software Requirement Smells and Detection Techniques: A Systematic Literature Review. *Cybernetics and Information Technologies*, *24*(4), 78–107. https://doi.org/10.2478/cait-2024-0037

Ambriola, V., & Gervasi, V. (2006). On the Systematic Analysis of Natural Language Requirements with CIRCE. *Automated Software Engineering*, *13*(1), 107–167. https://doi.org/10.1007/s10515-006-5468-2

Angelov, K., Camilleri, J. J., & Schneider, G. (2013). A framework for conflict analysis of normative texts written in controlled natural language. *The Journal of Logic and Algebraic Programming, Formal Languages and Analysis of Contract-Oriented Software (FLACOS'11)*, *82*(5), 216–240. https://doi.org/10.1016/j.jlap.2013.03.002

Antinyan, V., & Staron, M. (2017). Rendex: A method for automated reviews of textual requirements. *Journal of Systems and Software*, *131*, 63–77. https://doi.org/10.1016/j.jss.2017.05.079

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2015). Automated Checking of Conformance to Requirements Templates Using Natural Language Processing. *IEEE Transactions on Software Engineering*, *41*(10), 944–968. https://doi.org/10.1109/TSE.2015.2428709

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2017). Automated Extraction and Clustering of Requirements Glossary Terms. *IEEE Transactions on Software Engineering*, *43*(10), 918–945. https://doi.org/10.1109/TSE.2016.2635134

Arora, C., Sabetzadeh, M., Goknil, A., Briand, L. C., & Zimmer, F. (2015). NARCIA: An automated tool for change impact analysis in natural language requirements. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 962–965. https://doi.org/10.1145/2786805.2803185

Bashir, S., Abbas, M., Saadatmand, M., Enoiu, E. P., Bohlin, M., & Lindberg, P. (2023). Requirement or Not, That is the Question: A Case from the Railway Industry. In A. Ferrari & B. Penzenstadler (Eds.), *Requirements Engineering: Foundation for Software Quality* (pp. 105–121). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-29786-1_8

Bayer, M., Kaufhold, M.-A., & Reuter, C. (2022). A Survey on Data Augmentation for Text Classification. *ACM Comput. Surv.*, *55*(7), 146:1-146:39. https://doi.org/10.1145/3544558

Berrocal Rojas, A., & Barrantes Sliesarieva, E. G. (2010). Automated Detection of Language Issues Affecting Accuracy, Ambiguity and Verifiability in Software Requirements Written in Natural Language. In T. Solorio & T. Pedersen (Eds.), *Proceedings of the*

NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas (pp. 100–108). Association for Computational Linguistics. https://aclanthology.org/W10-1614/

Bhatia, J., Sharma, R., Biswas, K. K., & Ghaisas, S. (2013). Using Grammatical Knowledge Patterns for structuring requirements specifications. *2013 3rd International Workshop on Requirements Patterns (RePa)*, 31–34. https://doi.org/10.1109/RePa.2013.6602669

Bus, E. (2021). *ReqView Software Requirements Specification Example*.

Calle Gallego, J. M., & Zapata Jaramillo, C. M. (2023). QUARE: Towards a question-answering model for requirements elicitation. *Automated Software Engineering*, *30*(2), 25. https://doi.org/10.1007/s10515-023-00386-w

Carlson, N., & Laplante, P. (2014). The NASA automated requirements measurement tool: A reconstruction. *Innovations in Systems and Software Engineering*, *10*(2), Article 2. https://doi.org/10.1007/s11334-013-0225-8

Castro-Herrera, C., Duan, C., Cleland-Huang, J., & Mobasher, B. (2009). A recommender system for requirements elicitation in large-scale software projects. *Proceedings of the 2009 ACM Symposium on Applied Computing, SAC '09*, 1419–1426. https://doi.org/10.1145/1529282.1529601

Cavada, R., Cimatti, A., Mariotti, A., Mattarei, C., Micheli, A., Mover, S., Pensallorto, M., Roveri, M., Susi, A., & Tonetta, S. (2009). Supporting Requirements Validation: The EuRailCheck Tool. *2009 IEEE/ACM International Conference on Automated Software Engineering*, 665–667. https://doi.org/10.1109/ASE.2009.49

Chantree, F., Nuseibeh, B., De Roeck, A., & Willis, A. (2006). Identifying Nocuous Ambiguities in Natural Language Requirements. *14th IEEE International Requirements Engineering Conference (RE'06)*, 59–68. https://doi.org/10.1109/RE.2006.31

Classification: Accuracy, recall, precision, and related metrics | Machine Learning. (2025). Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

Cybulski, J. L., & Reed, K. (2000). Requirements Classification and Reuse: Crossing Domain Boundaries. In W. B. Frakes (Ed.), *Software Reuse: Advances in Software Reusability* (pp. 190–210). Springer. https://doi.org/10.1007/978-3-540-44995-9_12

Daramola, O., Stålhane, T., Omoronyia, I., & Sindre, G. (2013). Using Ontologies and Machine Learning for Hazard Identification and Safety Analysis. In *Managing Requirements Knowledge* (pp. 117–141). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34419-0_6

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., & Theofanos, M. (1993). Identifying and measuring quality in a software requirements specification. *[1993] Proceedings First International Software Metrics Symposium*, 141–152. https://doi.org/10.1109/METRIC.1993.263792

Deeptimahanti, D. K., & Sanyal, R. (2011). Semi-automatic generation of UML models from natural language requirements. *Proceedings of the 4th India Software Engineering Conference, ISEC '11*, 165–174. https://doi.org/10.1145/1953355.1953378

Duan, C., Laurent, P., Cleland-Huang, J., & Kwiatkowski, C. (2009). Towards automated requirements prioritization and triage. *Requirements Engineering*, *14*(2), 73–89. https://doi.org/10.1007/s00766-009-0079-7

Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2002). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. *Proceedings 26th Annual NASA Goddard Software Engineering Workshop*, 97–105. https://doi.org/10.1109/SEW.2001.992662

Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Omoronyia, I., & Zojer, H. (2011). Ontology-Driven Guidance for Requirements Elicitation. *The Semanic Web: Research and Applications*, 212–226. https://doi.org/10.1007/978-3-642-21064-8_15

Femmer, H., Méndez Fernández, D., Wagner, S., & Eder, S. (2017). Rapid quality assurance with Requirements Smells. *Journal of Systems and Software*, *123*, 190–213. https://doi.org/10.1016/j.jss.2016.02.047

Ferrari, A., & Esuli, A. (2019). An NLP approach for cross-domain ambiguity detection in requirements engineering. *Automated Software Engg.*, *26*(3), 559–598. https://doi.org/10.1007/s10515-019-00261-7

Ferrari, A., Spagnolo, G. O., & Gnesi, S. (2017). PURE: A Dataset of Public Requirements Documents. *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 502–505. https://doi.org/10.1109/RE.2017.29

Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., & Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, *18*(1), 25–41. https://doi.org/10.1007/s00766-011-0134-z

Gervasi, V., & Zowghi, D. (2005). Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology*, *14*(3), 277–330. https://doi.org/10.1145/1072997.1072999

Gupta, A., & Siddiqui, S. (2019). A SURVEY OF SOFTWARE REQUIREMENTS SPECIFICATION AMBIGUITY. *Journal of Engineering and Applied Sciences*, *14*, 3046–3061.

Haque, Md. A., Abdur Rahman, Md., & Siddik, M. S. (2019). Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study. *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, 1–5. https://doi.org/10.1109/ICASERT.2019.8934499

Haris, M. S., & Kurniawan, T. A. (2020). Automated requirement sentences extraction from software requirement specification document. *Proceedings of the 5th International Conference on Sustainable Information Engineering and Technology*, 142–147. https://doi.org/10.1145/3427423.3427450

Harmain, H. M., & Gaizauskas, R. (2000). CM-Builder: An automated NL-based CASE tool. *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, 45–53. https://doi.org/10.1109/ASE.2000.873649

Hussain, I., Ormandjieva, O., & Kosseim, L. (2007). Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier. *Seventh International Conference on Quality Software (QSIC 2007)*, 209–218. https://doi.org/10.1109/QSIC.2007.4385497

Ibrahim, M., & Ahmad, R. (2010). Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques. *2010 Second International Conference on Computer Research and Development*, 200–204. https://doi.org/10.1109/ICCRD.2010.71

ISO/IEC/IEEE International Standard—Systems and software engineering – Life cycle processes – Requirements engineering. (2018). *ISO/IEC/IEEE 29148:2018(E)*, 1–104. ISO/IEC/IEEE 29148:2018(E). https://doi.org/10.1109/IEEESTD.2018.8559686

Ivanov, V., Sadovykh, A., Naumchev, A., Bagnato, A., & Yakovlev, K. (2022). Extracting Software Requirements from Unstructured Documents. In E. Burnaev, D. I. Ignatov, S. Ivanov, M. Khachay, O. Koltsova, A. Kutuzov, S. O. Kuznetsov, N. Loukachevitch, A. Napoli, A. Panchenko, P. M. Pardalos, J. Saramäki, A. V. Savchenko, E. Tsymbalov, & E. Tutubalina (Eds.), *Recent Trends in Analysis of Images, Social Networks and Texts* (pp. 17–29). Springer International Publishing. https://doi.org/10.1007/978-3-031-15168-2_2

Ivanov, V., Sadovykh, A., Naumchev, A., Yakovlev, K., & Bagnato, A. (2021). *ReqExp: BERT-based ML Model for Extracting Software Requirements* [Dataset]. Zenodo. https://zenodo.org/records/4630687

Jabbarin, S., & Arman, N. (2014). Constructing use case models from Arabic user requirements in a semi-automated approach. *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 1–4. https://doi.org/10.1109/WCCAIS.2014.6916558

Jafari, P., Hattab, M. A., Mohamed, E., & AbouRizk, S. (2021). Automated Extraction and Time-Cost Prediction of Contractual Reporting Requirements in Construction Using Natural Language Processing and Simulation. *Applied Sciences*, *11*(13), 6188. https://doi.org/10.3390/app11136188

Jeon, J., Xu, X., Zhang, Y., Yang, L., & Cai, H. (2021). Extraction of Construction Quality Requirements from Textual Specifications via Natural Language Processing. *Transportation Research Record*, *2675*(9), 222–237. https://doi.org/10.1177/03611981211001385

Jurafsky, D., Jurafsky, D., & Martin, J. H. (2000). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall.

Jyothilakshmi, M. S., & Samuel, P. (2012). Domain ontology based class diagram generation from functional requirements. *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, 380–385. https://doi.org/10.1109/ISDA.2012.6416568

Kamalrudin, M., Grundy, J., & Hosking, J. (2010). Managing Consistency between Textual Requirements, Abstract Interactions and Essential Use Cases. *2010 IEEE 34th Annual Computer Software and Applications Conference*, 327–336. https://doi.org/10.1109/COMPSAC.2010.40

Kamalrudin, M., Hosking, J., & Grundy, J. (2017). MaramaAIC: Tool support for consistency management and validation of requirements. *Automated Software Engineering*, *24*(1), 1–45. https://doi.org/10.1007/s10515-016-0192-z

Khan, M. A., Khan, M. S., Khan, I., Ahmad, S., & Huda, S. (2023). Non Functional Requirements Identification and Classification Using Transfer Learning Model. *IEEE Access*, *11*, 74997–75005. https://doi.org/10.1109/ACCESS.2023.3295238

Kiyavitskaya, N., & Zannone, N. (2008). Requirements model generation to support requirements elicitation: The Secure Tropos experience. *Automated Software Engineering*, *15*(2), 149–173. https://doi.org/10.1007/s10515-008-0028-6

Korzyński, P., Mazurek, G., Krzypkowska, P., & Kurasinski, A. (2023). Artificial intelligence prompt engineering as a new digital competence: Analysis of generative AI technologies such as ChatGPT. *Entrepreneurial Business and Economics Review*, *11*(3), 25–37. https://doi.org/10.15678/EBER.2023.110302

Kubat, M. (2021). *An Introduction to Machine Learning*. Springer International Publishing. https://doi.org/10.1007/978-3-030-81935-4

Kuk, K., Angeleski, M., & Popovic, B. (2019). A Semi-automated generation of Entity-Relationship Diagram based on Morphosyntactic Tagging from the Requirements Written in a Serbian Natural Language. *2019 IEEE 19th International Symposium on Computational Intelligence and Informatics and 7th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics (CINTI-MACRo)*, 000085–000092. https://doi.org/10.1109/CINTI-MACRo49179.2019.9105162

Kumar, D. D., & Sanyal, R. (2008). Static UML Model Generator from Analysis of Requirements (SUGAR). *2008 Advanced Software Engineering and Its Applications*, 77–84. https://doi.org/10.1109/ASEA.2008.25

Kummler, P., & Fromm, H. (n.d.). *A Closer Look on the Difficulties to Determine the Quality of Software Requirements*.

Kummler, P. S. (2021). *Automated Quality Assessment of Natural Language Requirements*. https://doi.org/10.5445/IR/1000137191

Kummler, P. S., Vernisse, L., & Fromm, H. (2018). How Good are My Requirements?: A New Perspective on the Quality Measurement of Textual Requirements. *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 156–159. https://doi.org/10.1109/QUATIC.2018.00031

Kurtanović, Z., & Maalej, W. (2017). Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning. *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 490–495. https://doi.org/10.1109/RE.2017.82

Kwon, B., Kim, J., Lee, H., Suh, H.-W., & Mun, D. (2024). Construction of design requirements knowledgebase from unstructured design guidelines using natural language processing. *Computers in Industry*, *159–160*, 104100. https://doi.org/10.1016/j.compind.2024.104100

Lami, G., Fusani, M., & Trentanni, G. (2019). QuARS: A Pioneer Tool for NL Requirement Analysis. In *From Software Engineering to Formal Methods and Tools, and Back* (pp. 211–219). Springer, Cham. https://doi.org/10.1007/978-3-030-30985-5_13

Landhaußer, M., Korner, S. J., Tichy, W. F., Keim, J., & Krisch, J. (2015). DeNom: A tool to find problematic nominalizations using NLP. *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 1–8. https://doi.org/10.1109/AIRE.2015.7337623

Laplante, P. A., & Kassab, M. H. (2022). *Requirements Engineering for Software and Systems*. Auerbach Publishers, Incorporated. http://ebookcentral.proquest.com/lib/pensu/detail.action?docID=6927367

Lee, B.-S., & Bryant, B. R. (2004). Automation of Software System Development Using Natural Language Processing and Two-Level Grammar. *Radical Innovations of Software and Systems Engineering in the Future*, 219–233. https://doi.org/10.1007/978-3-540-24626-8_15

Letsholo, K. J., Zhao, L., & Chioasca, E.-V. (2013). TRAM: A tool for transforming textual requirements into analysis models. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 738–741. https://doi.org/10.1109/ASE.2013.6693146

Li, C., Huang, L., Ge, J., Luo, B., & Ng, V. (2018). Automatically classifying user requests in crowdsourcing requirements engineering. *Journal of Systems and Software*, *138*, 108–123. https://doi.org/10.1016/j.jss.2017.12.028

Li, Y., Schulze, S., & Saake, G. (2018). Extracting features from requirements: Achieving accuracy and automation with neural networks. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 477–481. https://doi.org/10.1109/SANER.2018.8330243

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., & Brinkkemper, S. (2016). Improving agile requirements: The Quality User Story framework and tool. *Requirements Engineering*, *21*(3), 383–403. https://doi.org/10.1007/s00766-016-0250-x

Lundberg, S. (2018). *Welcome to the SHAP documentation—SHAP latest documentation*. https://shap.readthedocs.io/en/latest/

Luttmer, J., Prihodko, V., Ehring, D., & Nagarajah, A. (2023). Requirements extraction from engineering standards – systematic evaluation of extraction techniques. *Procedia CIRP, The 33rd CIRP Design Conference*, *119*, 794–799. https://doi.org/10.1016/j.procir.2023.03.125

Miao, W., Pu, G., Yao, Y., Su, T., Bao, D., Liu, Y., Chen, S., & Xiong, K. (2016). Automated Requirements Validation for ATP Software via Specification Review and Testing. In K. Ogata, M. Lawford, & S. Liu (Eds.), *Formal Methods and Software Engineering* (pp. 26–40). Springer International Publishing. https://doi.org/10.1007/978-3-319-47846-3_3

Muhammad, Y., A, J. D. N., Imran, G., & Shah, M. A. (2020). Extraction of non-functional requirement using semantic similarity distance. *Neural Computing & Applications*, *32*(11), 7383–7397. https://doi.org/10.1007/s00521-019-04226-5

Naeem, A., Aslam, Z., & Shah, M. A. (2019). Analyzing Quality of Software Requirements; A Comparison Study on NLP Tools. *2019 25th International Conference on Automation and Computing (ICAC)*, 1–6. https://doi.org/10.23919/IConAC.2019.8895182

Nguyen, T. H., Vo, B. Q., Lumpe, M., & Grundy, J. (2012). REInDetector: A framework for knowledge-based requirements engineering. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. https://doi.org/10.1145/2351676.2351754

Nlpaug 1.1.11 documentation. (2019). https://nlpaug.readthedocs.io/en/latest/augmenter/word/context_word_embs.html

Nugues, P. M. (2024). *Python for Natural Language Processing: Programming with NumPy, scikit-learn, Keras, and PyTorch*. Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-57549-5

Panichella, S., & Ruiz, M. (2020). Requirements-Collector: Automating Requirements Specification from Elicitation Sessions and User Feedback. *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 404–407. https://doi.org/10.1109/RE48521.2020.00057

Parra, E., Dimou, C., Llorens, J., Moreno, V., & Fraga, A. (2015). A methodology for the classification of quality of requirements using machine learning techniques. *Information and Software Technology*, *67*, 180–195. https://doi.org/10.1016/j.infsof.2015.07.006

Pinquié, R., Véron, P., Segonds, F., & Croué, N. (2018). A requirement mining framework to support complex sub-systems suppliers. *Procedia CIRP, 28th CIRP Design Conference 2018, 23-25 May 2018, Nantes, France*, *70*, 410–415. https://doi.org/10.1016/j.procir.2018.03.228

Plag, I., Braun, M., Lappe, S., & Schramm, M. (2007). *Introduction to English Linguistics* (1st. ed). De Gruyter Mouton. https://research.ebsco.com/linkprocessor/plink?id=e9515b8e-5e23-3a7b-b6c1-51b36aa515dd

Popescu, D., Rugaber, S., Medvidovic, N., & Berry, D. M. (2008). Reducing Ambiguities in Requirements Specifications Via Automatically Created Object-Oriented Models. *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, 103–124. https://doi.org/10.1007/978-3-540-89778-1_10

Qureshi, M. Z., Azhar, A., Abubakar, Q., Rana, T. A., & Maqbool, A. (2021). Capturing Users Requirements Using a Data Mining Approach. *2021 International Conference on Communication Technologies (ComTech)*, 49–54. https://doi.org/10.1109/ComTech52583.2021.9616939

Rago, A., Marcos, C., & Diaz-Pace, J. A. (2013). Uncovering quality-attribute concerns in use case specifications via early aspect mining. *Requirements Engineering*, *18*(1), 67–84. https://doi.org/10.1007/s00766-011-0142-z

Rago, A., Marcos, C., & Diaz-Pace, J. A. (2016). Identifying duplicate functionality in textual use cases by aligning semantic actions. *Software & Systems Modeling*, *15*(2), 579–603. https://doi.org/10.1007/s10270-014-0431-3

Reddivari, S., Bhowmik, T., & Hollis, C. (2019). Automated support to capture verbal just-in-time requirements via audio mining and cluster-based visualization. *Journal of Industrial Information Integration*, *14*, 41–49. https://doi.org/10.1016/j.jii.2018.06.001

Ruan, K., Chen, X., & Jin, Z. (2023). Requirements Modeling Aided by ChatGPT: An Experience in Embedded Systems. *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 170–177. https://doi.org/10.1109/REW57809.2023.00035

Saini, R., Mussbacher, G., Guo, J. L. C., & Kienzle, J. (2020). DoMoBOT: A bot for automated and interactive domain modelling. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '20*, 1–10. https://doi.org/10.1145/3417990.3421385

Saint Dizier, P., & Kang, J. (2015). LELIE - An Intelligent Assistant for Improving Requirement Authoring. *International Journal on Requirement Engineering*, *2015–02*, 1–9.

Saleem, S., Asim, M. N., & Dengel, A. (2025). *Reqnet: A Large Language Model Driven Computational Framework for Automated Requirements Extraction from Unstructured Documents* (SSRN Scholarly Paper No. 5122555). Social Science Research Network. https://doi.org/10.2139/ssrn.5122555

Sallam, A., Chetan, A., Mehrdad, S., Briand, L. C., & Traynor, M. (2020). Automated demarcation of requirements in textual specifications: A machine learning-based approach. *Empirical Software Engineering*, *25*(6), 5454–5497. https://doi.org/10.1007/s10664-020-09864-1

Sardinha, A., Chitchyan, R., Weston, N., Greenwood, P., & Rashid, A. (2013). EA-Analyzer: Automating conflict detection in a large set of textual aspect-oriented requirements. *Automated Software Engineering*, *20*(1), 111–135. https://doi.org/10.1007/s10515-012-0106-7

Segundo, L. M., Herrera, R. R., & Herrera, K. Y. P. (2007). UML Sequence Diagram Generator System from Use Case Description Using Natural Language. *Electronics, Robotics and Automotive Mechanics Conference (CERMA 2007)*, 360–363. https://doi.org/10.1109/CERMA.2007.4367713

Shanthamallu, U. S., & Spanias, A. (2022). *Machine and Deep Learning Algorithms and Applications*. Springer International Publishing. https://doi.org/10.1007/978-3-031-03758-0

Sharma, R., Srivastava, P. K., & Biswas, K. K. (2015). From natural language requirements to UML class diagrams. *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 1–8. https://doi.org/10.1109/AIRE.2015.7337625

Shibaoka, M., Kaiya, H., & Saeki, M. (2007). GOORE: Goal-Oriented and Ontology Driven Requirements Elicitation Method. *Advances in Conceptual Modeling – Foundations and Applications*, 225–234. https://doi.org/10.1007/978-3-540-76292-8_28

Shreda, Q. A., & Hanani, A. A. (2021). Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches. *IEEE Access*, 1–1. IEEE Access. https://doi.org/10.1109/ACCESS.2021.3052921

Sonbol, R., Rebdawi, G., & Ghneim, N. (2022). Learning software requirements syntax: An unsupervised approach to recognize templates. *Knowledge-Based Systems*, *248*, 108933. https://doi.org/10.1016/j.knosys.2022.108933

Thakur, J. S., & Gupta, A. (2014). Automatic generation of sequence diagram from use case specification. *Proceedings of the 7th India Software Engineering Conference, ISEC ’14*, 1–6. https://doi.org/10.1145/2590748.2590768

Tjong, S. F., & Berry, D. M. (2013). The Design of SREE — A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned. *Requirements Engineering: Foundation for Software Quality*, 80–95. https://doi.org/10.1007/978-3-642-37422-7_6

Umar, M. A., & Lano, K. (2024). Advances in automated support for requirements engineering: A systematic literature review. *Requirements Engineering*, *29*(2), 177–207. https://doi.org/10.1007/s00766-023-00411-0

Vemuri, S., Chala, S., & Fathi, M. (2017). Automated use case diagram generation from textual user requirement documents. *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1–4. https://doi.org/10.1109/CCECE.2017.7946792

Verma, K., & Kass, A. (2008). Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, & K. Thirunarayan (Eds.), *The Semantic Web—ISWC 2008* (pp. 751–763). Springer. https://doi.org/10.1007/978-3-540-88564-1_48

Vidya Sagar, V. B. R., & Abirami, S. (2014). Conceptual modeling of natural language functional requirements. *Journal of Systems and Software*, *88*, 25–41. https://doi.org/10.1016/j.jss.2013.08.036

Vlas, R., & Robinson, W. N. (2011). A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. *2011 44th Hawaii International Conference on System Sciences*, 1–10. https://doi.org/10.1109/HICSS.2011.28

Wang, J., & Chen, Y. (2023). *Introduction to Transfer Learning: Algorithms and Practice* (1st 2023.). Springer Nature Singapore. https://doi.org/10.1007/978-981-19-7584-4

Wang, K., Zhang, F., & Sabetzadeh, M. (2024a). *Automated Requirements Demarcation using Large Language Models: An Empirical Study*.

Wang, K., Zhang, F., & Sabetzadeh, M. (2024b). *Automated Requirements Demarcation using Large Language Models: An Empirical Study*.

Wang, Z., Chen ,Chun-Hsien, Zheng ,Pai, Li ,Xinyu, & and Khoo, L. P. (2021). A graph-based context-aware requirement elicitation approach in smart product-service systems. *International Journal of Production Research*, *59*(2), 635–651. https://doi.org/10.1080/00207543.2019.1702227

Wilson, W. M., Rosenberg, L. H., & Hyatt, L. E. (1997). Automated analysis of requirement specifications. *Proceedings of the 19th International Conference on Software Engineering - ICSE '97*, 161–171. https://doi.org/10.1145/253228.253258

Winkler, J., & Vogelsang, A. (2016). Automatic Classification of Requirements Based on Convolutional Neural Networks. *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, 39–45. https://doi.org/10.1109/REW.2016.021

Wu, Z., & Ma, G. (2024). NLP-based approach for automated safety requirements information retrieval from project documents. *Expert Systems with Applications*, *239*, 122401. https://doi.org/10.1016/j.eswa.2023.122401

Yang, H., Willis, A., De Roeck, A., & Nuseibeh, B. (2010). Automatic detection of nocuous coordination ambiguities in natural language requirements. *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, ASE '10*, 53–62. https://doi.org/10.1145/1858996.1859007

Yue, T., Briand, L. C., & Labiche, Y. (2015). aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. *ACM Trans. Softw. Eng. Methodol.*, *24*(3), 13:1-13:52. https://doi.org/10.1145/2699697

Yusop, N., Kamalrudin, M., Yusof, M. M., & Sidek, S. (2016). Meeting Real Challenges in Eliciting Security Attributes for Mobile Application Development. *Journal of Internet Computing and Services*, *17*(5), 25–32. https://doi.org/10.7472/jksii.2016.17.5.25

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. *ACM Comput. Surv.*, *54*(3), 55:1-55:41. https://doi.org/10.1145/3444689

Zhao, Z., Zhang, L., Lian, X., & Lv, H. (2023, December 19). *DRIP: Segmenting individual requirements from software requirement documents*. https://onlinelibrary-wiley-com.ezaccess.libraries.psu.edu/doi/full/10.1002/spe.3303

Appendix

Reviewer Instructions

You have been given two datasets to evaluate:

- dataset_{number}_quality_unlabeled.xlsx (1)
- dataset_{number}_unlabeled_req.xlsx (2)

Please follow the instructions below:

Dataset 1

Read the sentence entry as a requirement and use the drop-down box to choose either True or False as to whether there exists a violation for the qualities defined blow from IEEE 29148:2018 standard: Ambiguity, feasibility, singularity, and verifiability.

Note:

- TRUE indicates there is a violation for that specific quality attribute,
- FALSE indicates no violation
- Do not leave any cells blank.
- Even if you do not believe the sentence is a requirement, please complete the evaluation for each quality.

Ambiguity

The requirement is not ambiguous if: "*The requirement is stated in such a way so that it can be interpreted in only one way. The requirement is stated simply and is easy to understand.*"

Feasibility

The requirement is feasible if: "*The requirement can be realized within system constraints (e.g., cost, schedule, technical) with acceptable risk*."

Singularity

The requirement <u>is singular</u> if: "*The requirement states a single capability, characteristic, constraint or quality factor.*"

Verifiability

The requirement <u>is verifiable</u> if: "*The requirement is structured and worded such that its realization can be proven (verified) to the customer's satisfaction at the level the requirements exist. Verifiability is enhanced when the requirement is measurable.*"

<u>Dataset 2</u>

Determine if the sentence is a requirement as defined blow from IEEE 29148:2018 standard and use the drop-down box to choose either True or False.

Note:

- Do not leave any entries blank.
- Do not consider the type of requirement. For instance, non-functional and functional requirements both count as requirements.

When determining if a sentence is a requirement, utilize the following definitions for the associated quality attribute:

Requirement Identification

A text is a requirement if it *"is a statement that translates or expresses a need and its associated constraints and conditions."*

VITA

Dylan (June) Porter

Education

Penn State University – University Park, PA
Doctor of Engineering in *Engineering – Artificial Intelligence* Spring 2026

Penn State University – Malvern, PA
Certificate in AI Engineering Fall 2024

Penn State University – Malvern, PA
Master of Science in *Software Engineering* Fall 2021

Duquesne University – Pittsburgh, PA
Bachelor of Science in *Computer Science* Spring 2019

Experience

Senior Software Engineer – Solventum / 3M 01/2022 – 11/2025

Software Engineer – General Dynamics Mission Systems 06/2019 – 01/2022

Software Developer Intern – UPMC 06/2018 – 08/2018

Software Developer Intern – CORE 05/2017 –08/2017

Publications

Berdik, D., Otoum, S., Schmidt, N., Porter, D., & Jararweh, Y. (2021). A Survey on Blockchain for Information Systems Management and Security. Information Processing & Management, 58(1), 102397. https://doi.org/10.1016/j.ipm.2020.102397

Porter, D. (2024). Requirements specification automated quality analysis: Past, present, and future. IEEE Computer. https://doi.org/10.1109/MC.2024.3480629