systems

MDPI

Article

Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models

Archana Tikayat Ray [1,*], Bjorn F. Cole [2], Olivia J. Pinon Fischer [1,*], Anirudh Prabhakara Bhat [3], Ryan T. White [4] and Dimitri N. Mavris [1]

[1] Aerospace Systems Design Laboratory, School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA; dimitri.mavris@aerospace.gatech.edu
[2] Lockheed Martin Space, Littleton, CO 80127, USA; bjorn.f.cole@lmco.com
[3] Amazon, Toronto, ON M5H 4A9, Canada; anipbhat@gmail.com
[4] Neural Transmissions Laboratory, Department of Mathematical Sciences, Florida Institute of Technology, Melbourne, FL 32901, USA; rwhite@fit.edu
* Correspondence: atr@gatech.edu (A.T.R.); olivia.pinon@asdl.gatech.edu (O.J.P.F.)

Abstract: The increased complexity of modern systems is calling for an integrated and comprehensive approach to system design and development and, in particular, a shift toward Model-Based Systems Engineering (MBSE) approaches for system design. The requirements that serve as the foundation for these intricate systems are still primarily expressed in Natural Language (NL), which can contain ambiguities and inconsistencies and suffer from a lack of structure that hinders their direct translation into models. The colossal developments in the field of Natural Language Processing (NLP), in general, and Large Language Models (LLMs), in particular, can serve as an enabler for the conversion of NL requirements into machine-readable requirements. Doing so is expected to facilitate their standardization and use in a model-based environment. This paper discusses a two-fold strategy for converting NL requirements into machine-readable requirements using language models. The first approach involves creating a requirements table by extracting information from free-form NL requirements. The second approach consists of an agile methodology that facilitates the identification of boilerplate templates for different types of requirements based on observed linguistic patterns. For this study, three different LLMs are utilized. Two of these models are fine-tuned versions of Bidirectional Encoder Representations from Transformers (BERTs), specifically, aeroBERT-NER and aeroBERT-Classifier, which are trained on annotated aerospace corpora. Another LLM, called flair/chunk-english, is utilized to identify sentence chunks present in NL requirements. All three language models are utilized together to achieve the standardization of requirements. The effectiveness of the methodologies is demonstrated through the semi-automated creation of boilerplates for requirements from Parts 23 and 25 of Title 14 Code of Federal Regulations (CFRs).

Keywords: requirements engineering; Large Language Models (LLMs); transformer-based language models; Natural Language Processing (NLP); BERT; requirement boilerplates; model-based systems engineering; requirement tables

Citation: Tikayat Ray, A.; Cole, B.F.; Pinon Fischer, O.J.; Bhat, A.P.; White, R.T.; Mavris, D.N. Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models. *Systems* **2023**, *11*, 352. https://doi.org/10.3390/systems11070352

Academic Editor: Vladimír Bureš

Received: 14 May 2023
Revised: 2 July 2023
Accepted: 6 July 2023
Published: 10 July 2023

1. Introduction

Every successful system is deeply rooted in, and highly responsive to, the needs of its stakeholders. The traditional method for ensuring this is the practice of requirements engineering, where the need for a system, subsystem, or component is analyzed and used to create a set of requirements. These requirements are provided to organizations responsible for the build and verification of the system. The International Council of Systems Engineering (INCOSE) defines a requirement as "a statement that identifies a system, product, or process characteristic or constraint, which is unambiguous, clear, unique, consistent, stand-alone (not grouped), and verifiable, and is deemed necessary for stakeholder acceptability" [1]. A requirement should be necessary, clear, traceable, verifiable, and

complete [2,3]. Requirements are typically expressed in natural language (NL) to ensure that they are comprehensible to diverse stakeholders, including consumers, subcontractors, and equipment manufacturers, with varying degrees of expertise [4]. Nevertheless, the utilization of NL for the purpose of requirements elicitation may give rise to inconsistencies and ambiguities, as demonstrated by the following instances:

1. For the requirement, *"The system must have a user-friendly user interface"*, the word "user-friendly" can encompass various interpretations among different stakeholders.
2. Frequently, requirements incorporate abbreviations, which, if inadequately defined, can lead to ambiguity.
3. Inconsistent use of units associated with quantities/values can add ambiguity. Such an error led to the failure of the Mars Climate Orbiter in 1998.

Mistakes made during the requirement definition phase can have far-reaching implications for downstream tasks such as system architecture, design, implementation, inspection, and testing [5]. These issues can, in turn, have significant engineering and programmatic consequences if not addressed early in the product life cycle [6,7]. Requirements have great leverage over the final form of the system and, yet, cannot be fully validated until the system is actually built. Finding that the incorrect system has been built at this point clearly leads to expensive rework or even abandonment. Research efforts that can support earlier validation, thus, have great economic value. The work in this paper and its predecessors [8–10] aim to provide digital artifacts for such methods.

Increasing system complexity in aerospace has been driven by a desire for ever more functionality and variety in operating modes and environments. The growing complexity of modern-day systems, however, makes it difficult for individual designers to retain a holistic understanding of the system they are designing. As a result, savvy systems engineers are relying on a set of model-based environments, tools, and approaches to help them in their activities.

From a requirement engineering perspective, there is a logical need to automate the integration and translation of NL requirements into such model-based environments so as to facilitate and accelerate the validation of requirements earlier in the development process. Natural Language Processing (NLP), supported by recent advancements in Large Language Models (LLMs), provides the necessary capabilities to classify NL requirements and extract important entities (systems, conditions, etc.) from those requirements for further integration into model-based environments. This has been demonstrated in two recent papers from the authors through the development of aeroBERT-Classifier [9] and aeroBERT-NER [8].

Building on these past efforts, the objective of the present paper is to propose a pipeline for the development of requirement boilerplates, or templates. This pipeline, represented in Figure 1, is expected to help ingest NL requirements into digital workflows such as those in place to compare simulation results against requirements, perform logical consistency checks, or generate conforming examples to assess whether the collective requirement set shall be relaxed.

The main contributions of this study are as follows:

1. Implementation of an aerospace-domain-specific LM aeroBERT [8–10] to populate requirement tables with named entities and requirement types.
2. Development of a semi-automated methodology for the identification of boilerplate templates using aeroBERT LMs and flair/chunk-english [11].

This research presumes that well-structured requirements can be used to detect boilerplate templates. As such, difficulty identifying a regular pattern in a requirement that can fit into a boilerplate may indicate that said requirement is ill-formed. These templates can then be used by less experienced engineers to ensure consistency in their requirement compositions from the start.

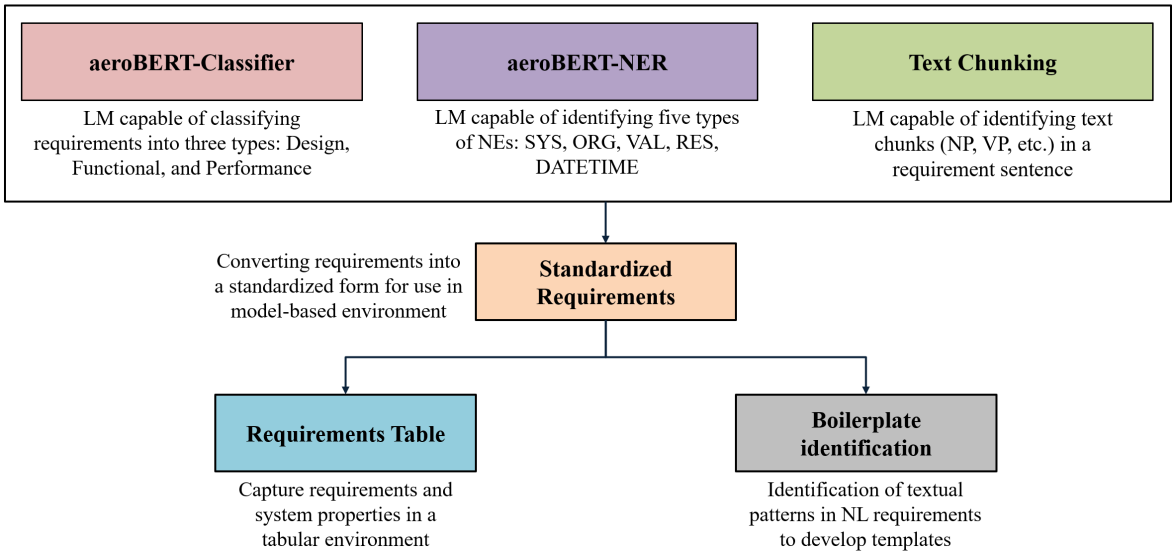


Figure 1. Pipeline for converting NL requirements to standardized requirements using various LLMs.

This paper is organized as follows: Section 2 outlines the need for digitized requirements, examines the enablers for this study, and explores methods for standardizing requirements. Section 3 introduces the datasets used, elucidates the role of various LLMs in shaping this study's methodology, and describes the process for creating requirement tables and identifying boilerplate templates. Section 4 delves into the pattern recognition process using sentence chunks and named entities, and demonstrates the creation of requirement tables and boilerplate templates using aerospace requirements. Lastly, Section 5 summarizes this research effort, discusses its limitations, and suggests potential avenues for future work.

2. Background

This section is articulated into three main subsections. The first identifies the state of the art of digital techniques in systems engineering, which are intended infusion points for this work. The second subsection presents an overview of the enablers that can be leveraged for the standardization of requirements. The third subsection discusses two approaches to standardizing requirements through the use of requirement tables and boilerplate templates.

2.1. Overview of Uses of Digitalized Requirements

Model-Based Systems Engineering (MBSE) has emerged as a solution to specifying more complex systems, involving the use of models to support system design processes in lieu of traditional document-based methods [12]. Models capture requirements and domain knowledge, making them accessible to all stakeholders [13,14]. However, the inherent ambiguities and inconsistencies of NL requirements hinder their direct conversion to models [15]. As creating models manually is both time-consuming and requires specialized expertise, there is a need to transform NL requirements into a machine-readable format for easier integration into an MBSE environment. The INCOSE's Requirements Working Group recognized the importance of accessing data within requirements, rather than treating them as separate entities, as discussed in a recent publication [16]. The document envisions an informational environment, where requirements are linked to one another as well as to test activities and architectural elements through a new type of requirement object. This object merges NL statements with machine-readable attributes, connecting to architectural entities such as interfaces and functions.

The use of machine-readable requirements is expected to greatly facilitate the integration of NL requirements in a model-based setting. Converting NL requirements to

machine-readable requirements can be achieved through the use of requirement tables or boilerplate templates, as shown in Figure 2. Requirement tables can be built by hand with NL statements (as is often conducted currently), but in this approach, the processed statements are data-rich enough to be further processed into system models (Step 3).

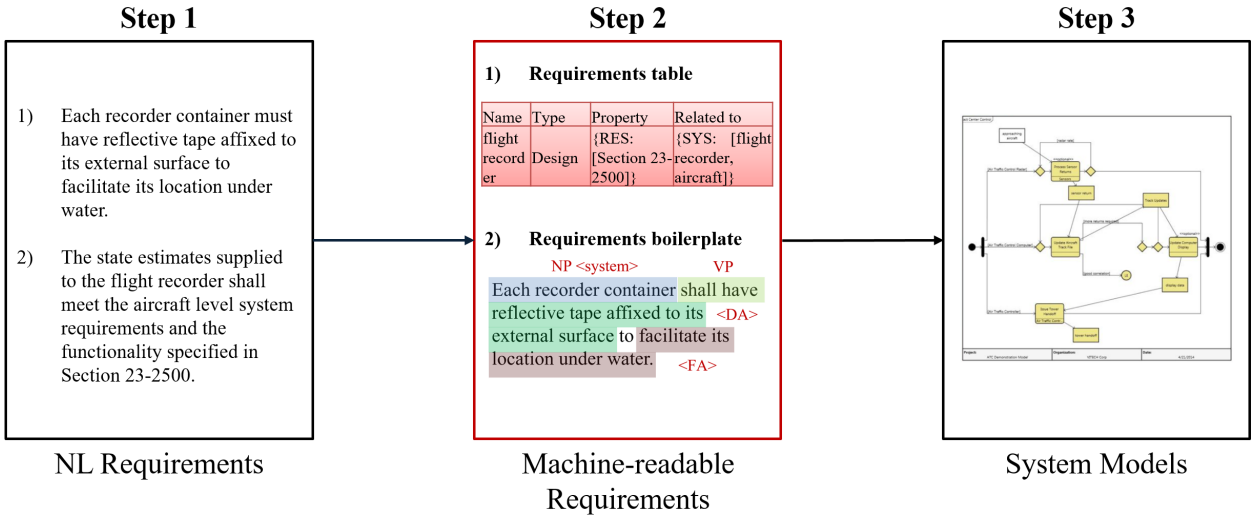


Figure 2. Steps of requirements engineering, starting with gathering requirements from various stakeholders, followed by using NLP techniques to standardize them, and, lastly, converting the standardized requirements into models. The main focus of this work is to convert NL requirements into machine-readable requirements (where parts of the requirement become data objects) as shown in Step 2.

A requirement table, used in areas such as software development and systems engineering [17,18], consolidates system, product, or process requirements, displaying related properties such as requirement ID, description, source, priority, verification method, and status. A given table can be stored in dedicated requirement management tools, such as DOORS, or be collocated with architecture models in SysML tools. It not only manages and monitors requirements throughout the development process but also assists in identifying dependencies, conflicts, redundancies, and potential oversights. The information needed to construct the table is obtained from NL requirements. Although requirements are often written in NL for their universal appeal, the introduction of requirement boilerplates, or pre-defined linguistic patterns, can enhance requirement quality by reducing ambiguities and inconsistencies and boosting readability. These boilerplates standardize the NL format of aerospace requirements for automated analysis while facilitating understanding among stakeholders. Despite their relatively rigid structure and limited customization flexibility for varying requirement types across different organizations, boilerplates like Rupp's [19,20] and EARS [21] templates are widely utilized.

2.2. Overview of Enablers

NL is predominantly employed in writing requirements, ensuring accessibility for stakeholders with varied understandings of requirement engineering processes [22], while tools such as the Structured Analysis Design Technique (SADT) and the System Design Methodology (SDM) [23] have been used to facilitate requirements elicitation and management since the 1970s, they have been limited by the lack of advanced NL technologies and their inability to generalize well to new requirements. However, current NLP capabilities have substantially improved, as evidenced by the availability of numerous open-source NLP tools, libraries, and pre-trained LLMs. These advanced machine-learning-based methods offer significant enhancements in generalization and efficacy, as well as cost and time efficiency, contributing greatly to the field of Natural Language Processing for Requirements Engineering (NLP4RE). Despite these advancements, a gap persists in NLP4RE

research and its industrial implementation [24,25]. This gap is attributed to the fragmented approach to sharing NLP4RE knowledge and a lack of open-source datasets. Unlike most studies in NLP4RE that focus on software engineering, this work seeks to harness these advancements to benefit the aerospace domain.

LLMs are central to modern NLP. The introduction of neural LMs, using neural networks to learn lower-dimensional word embeddings and simultaneously estimate the conditional probability using gradient-based supervised learning was a game changer in the field of NLP [26]. Further advancements, such as self-supervision, multi-headed self-attention [27], and improved computational parallelization, have led to the development of state-of-the-art LLMs like BERT LM [28], T5 character-level language model [29], and the Generative Pre-trained Transformer (GPT) family [30,31] of auto-regressive LMs. These models, pre-trained on large text corpora such as the Book Corpus and English Wikipedia, can be fine-tuned on smaller labeled datasets for various NLP tasks such as text classification, named entity recognition (NER), part-of-speech tagging (POS), and text chunking [11], to name a few. Training LLMs from scratch is expensive, mostly because of the extensive computational resources required. Pre-trained LLMs from entities such Google, Meta, and OpenAI are, however, freely accessible on the Hugging Face platform [32] through the *transformers* library.

As noted previously, LLMs are frequently trained on general-purpose text corpora (e.g., news articles and Wikipedia). As a result, they do not perform well on technical texts containing specialized jargon, such as aerospace requirements, which are the focus of this study. To overcome this issue, the authors have fine-tuned BERT to develop two LMs: aeroBERT-NER [8] and aeroBERT-Classifier [9]:

- **aeroBERT-NER** [8] can identify named entities belonging to five distinct categories (as shown in Table 1), which were selected based on their frequency and importance to aerospace texts. The fine-tuning process involved using annotated aerospace text from various sources, as well as requirements from Parts 23 and 25 of Title 14 of the Code of Federal Regulations (CFRs), to fine-tune BERT and generate aeroBERT-NER.
- **aeroBERT-Classifier** [9] can classify aerospace requirements into three categories: design, functional, and performance requirements. To train aeroBERT-Classifier, the authors fine-tuned BERT using annotated aerospace certification requirements from Parts 23 and 25 of Title 14 of the CFRs. The ability to classify requirements is crucial when analyzing NL requirements, particularly in large-scale systems where there are a significant number of requirements to be defined [33].

Table 1. aeroBERT-NER is capable of identifying five types of named entities. The BIO tagging scheme was used for annotating the NER dataset [8].

Category	NER Tags	Example
System	B-SYS, I-SYS	nozzle guide vanes, flight recorder, fuel system
Value	B-VAL, I-VAL	5.6 percent, 41,000 feet, 3 s
Date time	B-DATETIME, I-DATETIME	2017, 2014, 19 September 1994
Organization	B-ORG, I-ORG	DOD, NASA, FAA
Resource	B-RES, I-RES	Section 25–341, Sections 25–173 through 25–177, Part 25 subpart C

The aforementioned LMs have the capability to extract valuable information from aerospace requirements written in NL. This information can then serve as input for either rule-based methods or machine learning (ML) models [34]. The ability to extract relevant information is crucial for ensuring that requirements are standardized, which, in turn, aids in their integration into a model-based environment.

2.3. Methods for Requirements Standardization

In the context of this research, there are two methods for standardizing requirements. The first involves the construction and use of a requirements table, while the second entails the development of standardized templates, referred to as boilerplates.

2.3.1. Requirement Table

A spreadsheet-like environment can be used to create a requirement table, wherein each requirement is represented by a row, and its associated attributes or system model elements are captured in columns. This table not only allows for the filtering of requirements and their related properties but is also easily exportable to Microsoft Excel and other spreadsheet/tabular environments due to its tabular structure [17]. It can assist in the automated analysis and modeling of requirements, resulting in cost and time savings as requirements change over time. Riesener et al. [35] presented a method to generate such a table using a dictionary-based approach, where domain-specific keywords are added for extraction from mechatronics requirement texts. However, the creation and updating of this dictionary can be labor-intensive, and its applicability might be limited to specific projects. Therefore, using LLMs for extracting pertinent entities proves to be more efficient and widely applicable/generalizable. Additionally, the automation of requirement table creation through the use of LLMs minimizes potential human errors, reduces repetitive tasks in column population, and is more customizable, thereby streamlining the development of SysML tables. Figure 3 provides a sample requirements table.

#	Name	Paragraph Text	Traced Elements
1	§ 25.1457a	Each cockpit voice recorder required by the operating rules of this chapter must be approved and must be installed so that it will record the following:	6 Approved Cockpit Voice Recorder
2	§ 25.1457a1	Voice communications transmitted from or received in the airplane by radio.	Voice Communications via Radio
3	§ 25.1457a2	Voice communications of flight crewmembers on the flight deck.	Voice Communications between Crewmembers
4	§ 25.1457a3	Voice communications of flight crewmembers on the flight deck, using the airplane's interphone system.	Voice Crewmember Communication via Interphone
5	§ 25.1457a4	Voice or audio signals identifying navigation or approach aids introduced into a headset or speaker.	Voice or Audio Signals for Navigation or Approach Aids
6	§ 25.1457a5	Voice communications of flight crewmembers using the passenger loudspeaker system, if there is such a system and if the fourth channel is available in accordance with the requirements of paragraph (c)(4)(ii) of this section.	Voice Communications via Loudspeaker
7	§ 25.1457a6	If datalink communication equipment is installed, all datalink communications, using an approved data message set. Datalink messages must be recorded as the output signal from the communications unit that translates the signal into usable data.	Datalink Communications 7 Datalink Message Recording Signals

Figure 3. A SysML requirement table with three columns, namely, Name, Paragraph Text, and Traced Elements. More columns can be added to capture other properties pertaining to the requirements.

2.3.2. Requirement Boilerplates

Boilerplates, which are pre-defined linguistic patterns for standardizing requirements, serve as the second method for improving quality, reducing ambiguity, and promoting uniformity in NL requirements. Notable generalized boilerplates include Rupp's [19,20] and the Easy Approach to Requirements Syntax (EARS) [21]. These boilerplates, however, can pose restrictions on specific functional and non-functional requirements and fail to incorporate constraints. Rupp's boilerplate may even cause ambiguity and inconsistency due to its structural rigidity, which excludes ranges of values, bi-conditionals, and references to external systems. An enhanced boilerplate proposed by Mazo et al. [20] addresses some of Rupp's limitations but its complexity can be daunting. Consequently, it might be ideal to develop templates for each different type of requirement after observing the patterns in the NL requirement text for that particular type.

Arora et al. [36] highlight the necessity of assessing the conformance of NL requirements with a selected boilerplate template for quality assurance. They propose using text chunking to verify conformance with Rupp's boilerplate and promote this approach as dependable in the absence of a glossary. Furthermore, they developed a methodology to check the conformance of Rupp's and EARS boilerplates using an NLP pipeline. Despite these advances, the development of boilerplates is not universal due to variances in organizational and industrial needs and the connection between requirement articulation

and verification. Different technical requirements may be better suited to various modes of demonstration, inspection, or testing, depending on the industry context. Consequently, a flexible approach that can generate boilerplates for diverse types of requirements is crucial. This paper suggests a structured, scalable, and repeatable method for creating customized boilerplates using LLMs, which helps overcome the difficulties and time-consuming nature of manual boilerplate creation.

3. Materials and Methods

This section outlines the dataset and LLMs employed in this study, with an emphasis on their practical application. It also explains the methodology used to construct a requirements table and boilerplate templates based on the results generated from the LLMs.

3.1. Dataset

This study used a collection of 310 requirements (the list of the requirements can be found in https://huggingface.co/datasets/archanatikayatray/aeroBERT-classification, accessed on 10 May 2023, Ref. [9]) categorized into design (149), functional (99), and performance (62) requirements. Since requirement texts are almost always proprietary, all the requirements used for this work were obtained from Parts 23 and 25 of Title 14 of the Code of Federal Regulations (CFRs) [37], while these requirements are mostly used for verifying compliance during certification, they were used in this work to establish a methodology for the conversion of NL requirements into machine-readable requirements. The methodology proposed herein can be reproduced and applied to proprietary requirements.

3.2. Information Extraction from Aerospace Requirements Using LLMs

The methodology for standardizing requirements requires the use of three LLMs to extract information from NL aerospace requirements. The importance of these models to the methodology, along with their capabilities, are outlined in the following subsections.

3.2.1. aeroBERT-NER

In its current form, aeroBERT-NER [8] is employed to identify five categories of named entities (Table 1). The entities extracted in this process serve to populate various columns of a requirements table, while aeroBERT-NER currently recognizes five entity types, it can be trained further to recognize additional named entity categories as needed by various organizations.

3.2.2. aeroBERT-Classifier

aeroBERT-Classifier [9] classifies requirements into one of three types, with the classification results being included in the requirements table. The aeroBERT-Classifier model can be expanded to classify additional types of requirements as needed. The classification of NL requirements is also key to the identification of boilerplates that are specific to each requirement type. For the purpose of this research, which is to to devise and demonstrate a methodology for standardizing requirements using LLMs, only three types of requirements are considered. The intent is eventually to effectively reduce the manual workload for systems engineers.

3.2.3. Text Chunking

Standardizing requirements necessitates identifying recurrent linguistic patterns within natural language requirements, which, in turn, demands a comprehensive understanding of the utilized syntax. Helpful techniques for this purpose include parts-of-speech (POS) tagging and sentence chunking. Words in a sentence can be grouped into various POS or lexical categories. When these POS tags are combined, sentence chunks are obtained, which are non-overlapping segments [38], as depicted in Figure 4. POS tags on their own can be noisy due to their individual association with each word, making them less beneficial for

pattern identification in requirements. In contrast, sentence chunks offer more utility in reorganizing requirements into standardized templates.

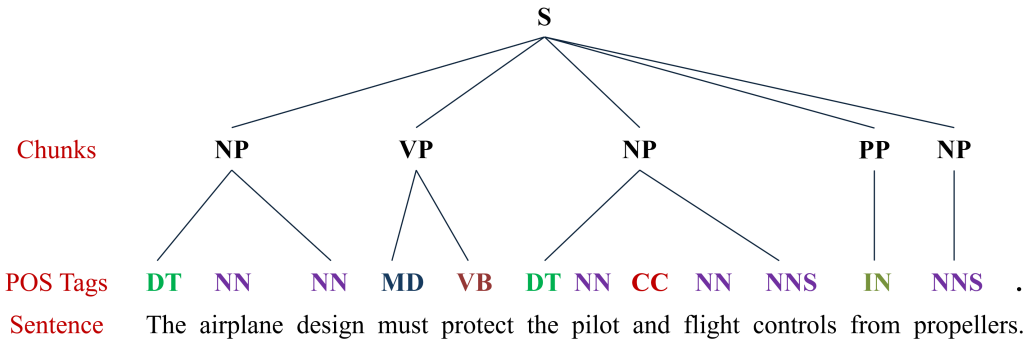


Figure 4. An aerospace requirement along with its POS tags and sentence chunks. Each word has a POS tag associated with it, which can then be combined together to obtain a higher-level representation called sentence chunks (NP: Noun Phrase; VP: Verb Phrase; PP: Prepositional Phrase).

A variety of sentence chunk types are enumerated in Table 2. The LLM flair/chunk-english [11] is used for identifying sentence chunks within aerospace requirements and no specific fine-tuning was performed.

Table 2. A subset of text chunks along with definitions and examples is shown here [11,39]. The blue text highlights the type of text chunk of interest. This is not an exhaustive list.

Sentence Chunk	Definition and Example
Noun Phrase (NP)	Consists of a noun and other words modifying the noun (determinants, adjectives, etc.); **Example:** The airplane design must protect the pilot and flight controls from propellers.
Verb Phrase (VP)	Consists of a verb and other words modifying the verb (adverbs, auxiliary verbs, prepositional phrases, etc.); **Example:** The airplane design must protect the pilot and flight controls from propellers.
Subordinate Clause (SBAR)	Provides more context to the main clause and is usually introduced by subordinating conjunction (because, if, after, as, etc.) **Example:** There must be a means to extinguish any fire in the cabin such that the pilot, while seated, can easily access the fire extinguishing means.
Adverbial Clause (ADVP)	Modifies the main clause in the manner of an adverb and is typically preceded by subordinating conjunction; **Example:** The airplanes were grounded until the blizzard stopped.
Adjective Clause (ADJP)	Modifies a noun phrase and is typically preceded by a relative pronoun (that, which, why, where, when, who, etc.); **Example:** I can remember the time when air-taxis didn't exist.

3.3. Methodology for Creating Requirement Tables

Requirement tables are helpful for filtering requirements of interest and performing requirements analysis. For the purpose of this work, five columns were chosen to be included in the requirement table, as shown in Table 3. The *Name* column was populated by the system name (SYS-named entity) identified by aeroBERT-NER [8]. The *Type of Requirement* column was populated after the requirement was classified as a design, functional, or performance requirement by aeroBERT-Classifier [9]. The *Property* column was populated by all the named entities identified by aeroBERT-NER [8] (except for SYS) Lastly, the *Related to* column was populated by system names identified by aeroBERT-NER.

Table 3. List of language models used to populate the columns of requirement table.

Column Name	Description	Method Used to Populate
Name	System (SYS named entity) that the requirement pertains to	aeroBERT-NER [8]
Text	Original requirement text	Original requirement text
Type of Requirement	Classification of the requirement as design, functional, or performance	aeroBERT-Classifier [9]
Property	Identified named entities belonging to RES, VAL, DATETIME, and ORG categories present in a requirement related to a particular system (SYS)	aeroBERT-NER [8]
Related to	Identified system named entity (SYS) that the requirement properties are associated with	aeroBERT-NER [8]

The flowchart in Figure 5 illustrates the procedure for generating a requirements table for a single requirement utilizing aeroBERT-NER and aeroBERT-Classifier. In particular, it shows how the LMs are used to extract information from the requirement and how this information is employed to populate various columns of the requirements table, while this process is demonstrated on a single requirement, it can be used on multiple requirements simultaneously.

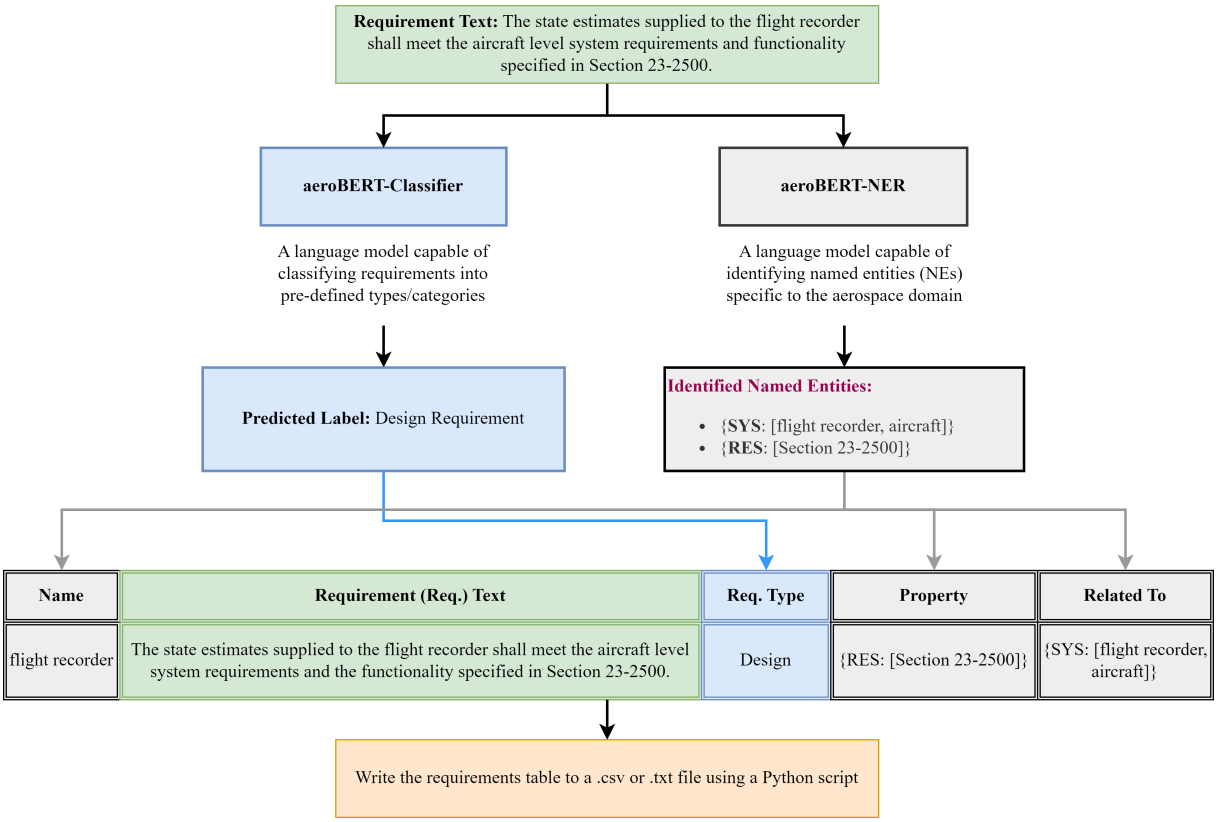


Figure 5. Flowchart showcasing the creation of the requirements table for the requirement "*The state estimates supplied to the flight recorder shall meet the aircraft level system requirements and functionality specified in Section 23–2500*" using two LMs, namely, aeroBERT-Classifier [9] and aeroBERT-NER [8], to populate various columns of the table. A zoomed-in version of the figure can be found here and more context can be found in [10].

3.4. Methodology for Identification of Standardized Boilerplates for Requirements

The methodology starts with using aeroBERT-Classifier for classifying requirements into various categories. This is followed by obtaining the sentence chunks and named entities present in the requirements. Various boilerplate elements are then identified based on the patterns observed in regard to sentence chunks and the presence/absence of certain named entities. The patterns observed in the sequence of boilerplate elements are aggregated to obtain boilerplate templates for different types of requirements. The flowchart in Figure 6 summarizes the process for identifying boilerplate templates from well-written requirements. The example shown in the flowchart illustrates the steps involved in using a single requirement. It is crucial to note that in order to generate boilerplate templates that can be applied more broadly, common patterns observed across multiple requirements need to be identified.

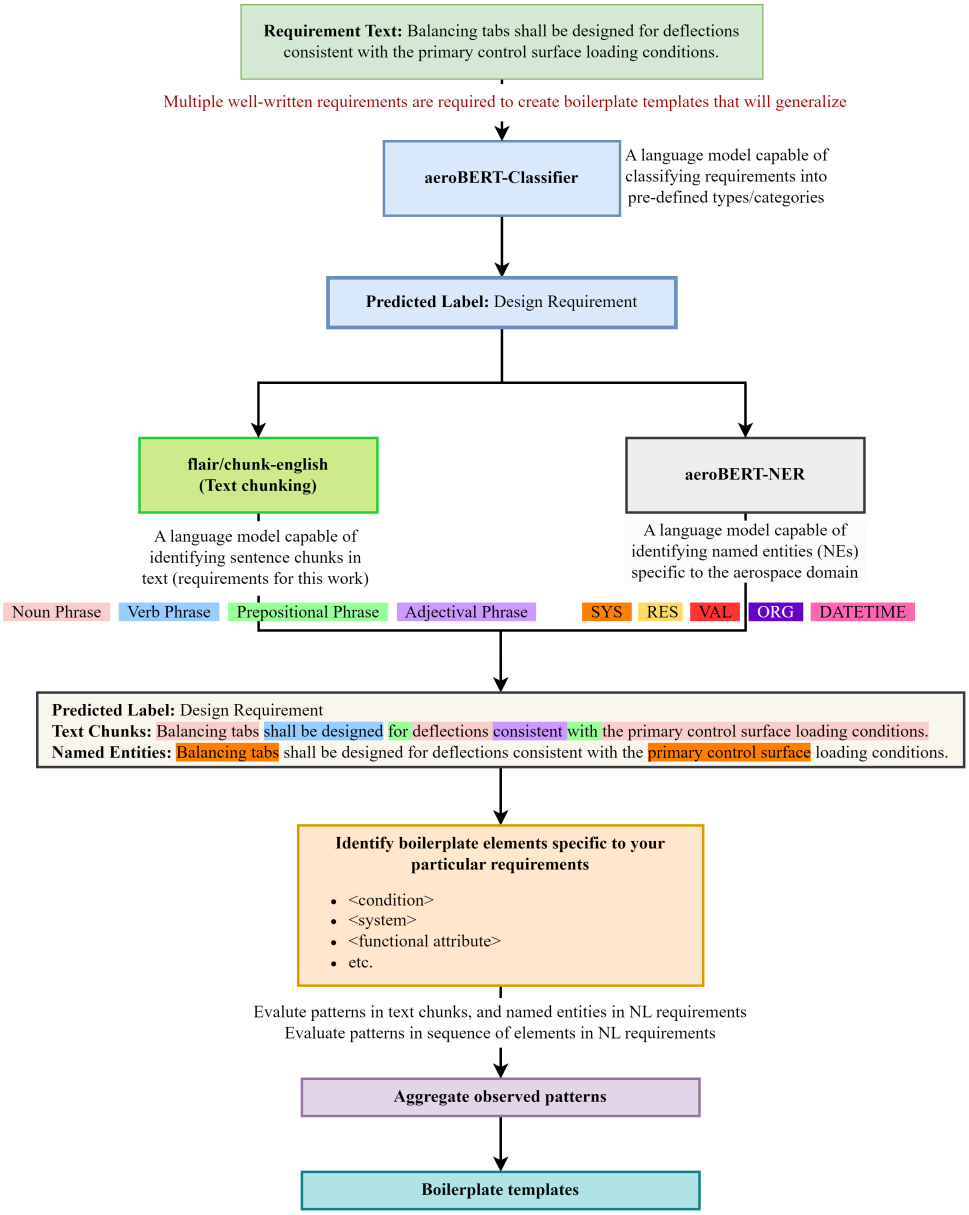


Figure 6. Flowchart showcasing the creation of boilerplate templates using three language models, namely, aeroBERT-Classifier [9], aeroBERT-NER [8], and flair/chunk-english. A zoomed-in version of the figure can be found https://archanatikayatray19.github.io/Practitioners_Guide/boilerplate_flowchart.html and more context can be found in [10].

4. Results

This section discusses findings about pattern identification in requirements using sentence chunks and named entities. This is followed by a two-pronged approach for the standardization of requirements, namely, the creation of a requirement table, and the identification of boilerplate templates. Lastly, a range of identified boilerplate templates and the observed patterns in the text that led to their discovery will be discussed.

4.1. Pattern Identification Using Sentence Chunks and Named Entities

After classifying requirements, flair/chunk-english is used to identify sentence chunks and their order in requirements. The details regarding the textual patterns identified in the design requirements are described below. Patterns observed in functional and performance requirements are included in Appendix A.

4.1.1. Design Requirements

Of the 149 design requirements, 139 started with a noun phrase (NP), 7 started with a prepositional phrase (PP), 2 started with a subordinate clause (SBAR), and only 1 started with a verb phrase (VP). In 106 of the requirements, NPs were followed by a VP or another NP. A detailed sequence of patterns is shown in Figure 7.

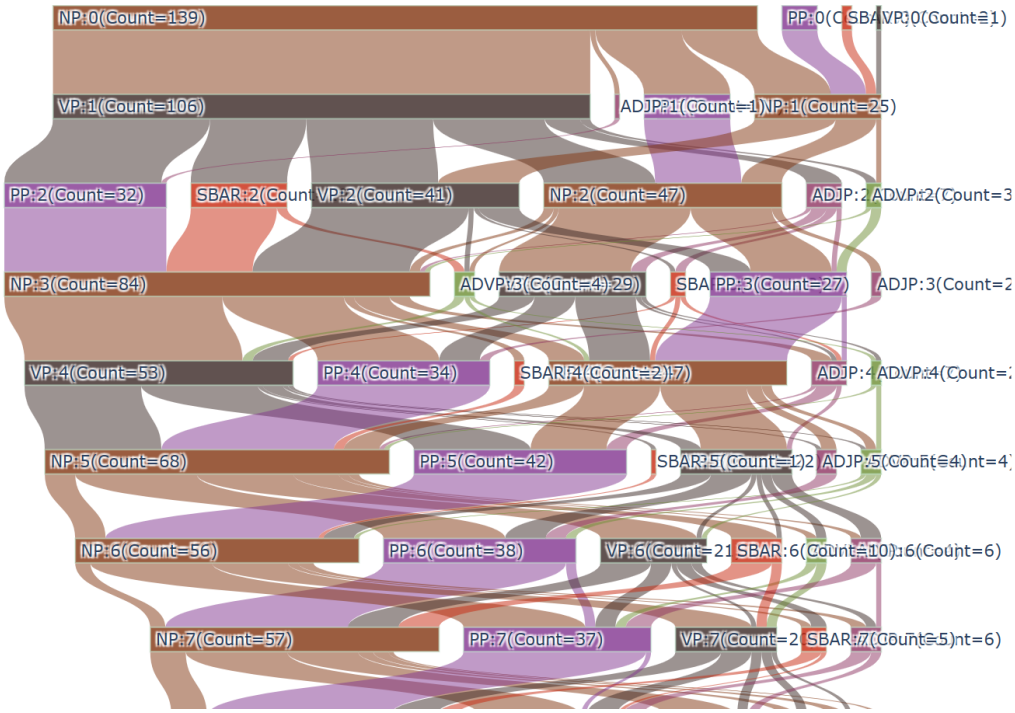


Figure 7. Sankey diagram showing the text chunk patterns in design requirements. A part of the figure is shown due to space constraints; however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Design_pos_chunk_Requirements_Diagram.png [10].

Examples showing design requirements beginning with different types of sentence chunks are shown in Figures 8 and 9.

Examples 1–4 show the design requirements beginning with different types of sentence chunks (PP, SBAR, VP, and NP, respectively), which gives a glimpse into the different ways these requirements can be written and the variation in them. In Example 1, the initial prepositional phrase (PP) appears to specify a particular family of aircraft which are those with retractable landing gear. In all cases, the first NP is often the system name and is followed by VPs.

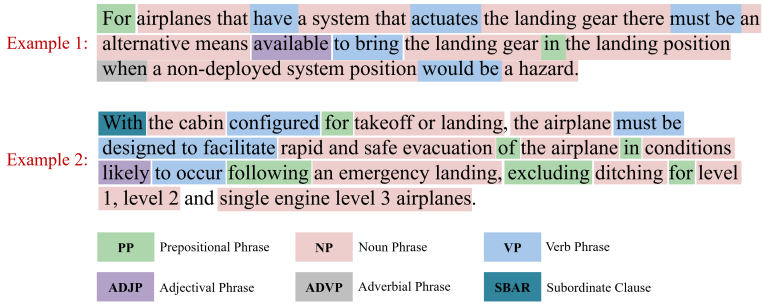


Figure 8. Examples 1 and 2 show a design requirement beginning with a prepositional phrase (PP) and subordinate clause (SBAR), which is uncommon in the requirements dataset used for this work. The uncommon starting sentence chunks (PP, SBAR) are, however, followed by a noun phrase (NP) and verb phrase (VP). Most of the design requirements start with a NP.

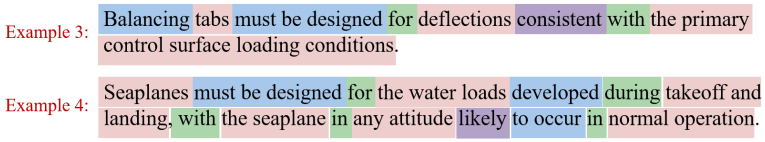


Figure 9. Example 3 shows a design requirement starting with a verb phrase (VP). Example 4 shows the requirement starting with a noun phrase (NP), which was the most commonly observed pattern.

It is important to note that in Example 3 (Figure 9), the term "*Balancing* " is wrongly classified as a VP. The entire term "*Balancing tabs*" should have been identified as an NP instead. This error can be attributed to the fact that an off-the-shelf sentence chunking model (flair/chunk-english) was used and, hence, failed to identify "*Balancing tabs*" as a single NP due to the lack of aerospace domain knowledge. Such discrepancies can be resolved by simultaneously accounting for the named entities identified by aeroBERT-NER for the same requirement. For this example, "*Balancing tabs*" was identified as a system name (SYS) by aeroBERT-NER, which should be a NP by default. In places where the text chunking and NER models disagree, the results from the NER model take precedence since it is fine-tuned on an annotated aerospace corpus and, hence, has more context regarding the aerospace domain. The pattern of named entities found in design requirements is shown in Figure 10. Here, most design requirements begin with a system (SYS) named entity. Moving further along the Sankey diagram, the sequence of different named entities shows greater variability.

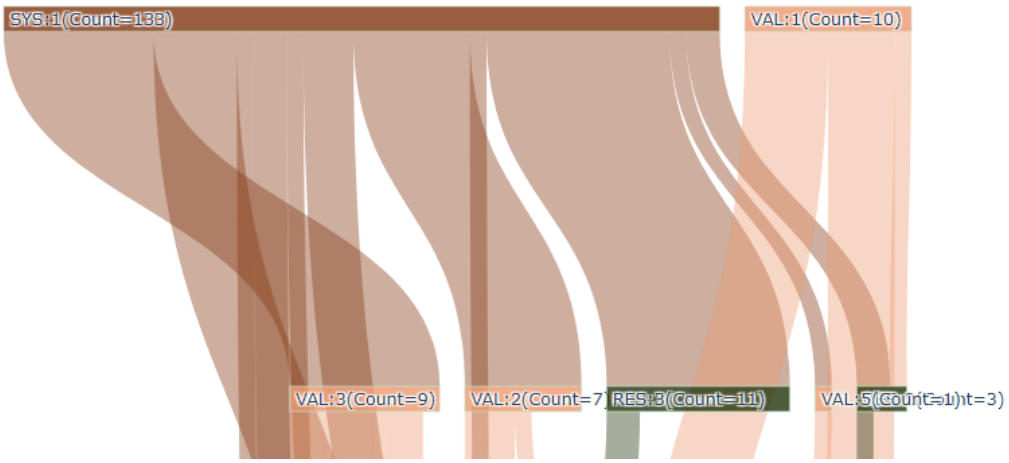


Figure 10. Sankey diagram showing the named entity patterns in design requirements. A part of the figure is shown here due to space constraints; however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Design_NER_Requirements_Diagram.png [10] .

Sankey diagrams can be used to recognize and filter requirements that exhibit dominant linguistic structures. By doing so, the more prominent sequences or patterns can be removed, allowing for greater focus on the less common ones for further investigation to determine if a rewrite is required.

4.1.2. General Patterns Observed in Requirements

After analyzing the three types of requirements, it was discovered that there was a general pattern irrespective of the requirement type, as shown in Figure 11.

The *Body* section of the requirement usually starts with an NP (system name), which (for most cases) contains an SYS-named entity, whereas the beginning of a *Prefix* and *Suffix* is usually marked by an SBAR or PP, namely, 'so that', 'so as', 'unless', 'while', 'if', 'that', etc. Both the *Prefix* and *Suffix* provide more context into a requirement and, thus, are likely to be conditions, exceptions, the state of the system after a function is performed, etc. Usually, the *Suffix* contains various different types of named entities, such as names of resources (RES), values (VAL), other system or sub-system names (SYS) that add more context to the requirement, etc. It is mandatory for a requirement to have a *Body*, while prefixes and suffixes are optional.

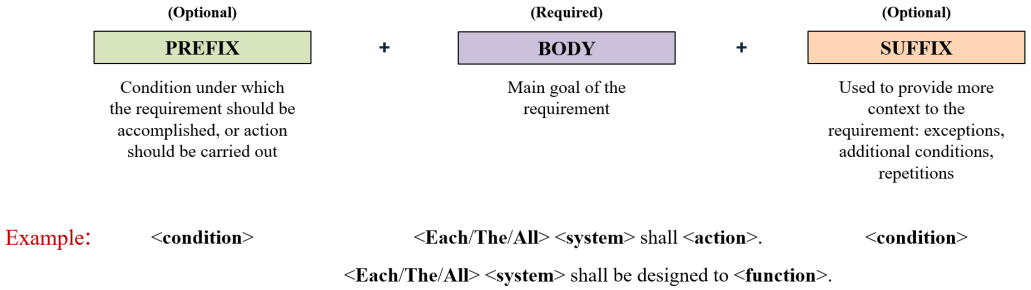


Figure 11. The general textual pattern observed in requirements was <Prefix> + <Body> + <Suffix> out of which Prefix and Suffix are optional and can be used to provide more context about the requirement. The different variations in the requirement Body, Prefix, and Suffix are shown as well [10].

Table 4 presents the various elements of an aerospace requirement, including system, functional attribute, state, condition, and others, along with examples. The presence, absence, and order of these elements distinguish requirements of different types, as well as requirements within the same type, resulting in distinct boilerplate structures. These elements often begin or end with a specific type of sentence chunk and/or contain a particular named entity, which can aid in their identification. Several of the elements (e.g., condition, functional attribute, and state) are excellent candidates to populate not only machine-readable requirements but also architectural models.

Table 4. Different elements of an aerospace requirement. The blue text highlights a specific element of interest in a requirement [10].

Requirement Element	Definition and Example
<condition>	Specifies details about the external circumstance, system configuration, or system activity currently happening for the system while it performs a certain function, etc.; **Example:** with the cabin configured for takeoff or landing, the airplane shall have means of egress, that can be readily located and opened from the inside and outside.
<system>	Name of the system that the requirement is about; **Example:** all pressurized airplanes shall be capable of continued safe flight and landing following a sudden release of cabin pressure.

Table 4. *Cont.*

Requirement Element	Definition and Example
<functional attribute>	The function to be performed by a system or a description of a function that can be performed under some unstated circumstance; **Example:** the insulation on electrical wire and electrical cable shall be self-extinguishing.
<state>	Describes the physical configuration of a system while performing a certain function; **Example:** the airplane shall maintain longitudinal trim without further force upon, or movement of, the primary flight controls.
<design attribute>	Provides additional details regarding a system's design; **Example:** each recorder container shall have reflective tape affixed to its external surface to facilitate its location under water.
<sub-system/system>	Specifies any additional system/sub-system that the *main* system shall include, or shall protect in case of a certain operational condition, etc.; **Example:** each recorder container shall have an underwater locating device.
<resource>	Specifies any resource (such as another part of the Title 14 CFRs, a certain paragraph in the same part, etc.) that the system shall be compliant with; **Example:** the control system shall be designed for pilot forces applied in the same direction, using individual pilot forces not less than 0.75 times those obtained in accordance with Section 25-395.
<context>	Provides additional details about the requirement; **Example:** with the cabin configured for takeoff or landing, the airplane shall have means of egress that can be readily located and opened from the inside and outside.
<user>	Specifies the user of a system, usually a pilot in the case of flight controls, passengers in the case of emergency exits in the cabin, etc.; **Example:** the airplane design shall protect the pilot and flight controls from propellers.
<system attribute>	Some requirements do not start with a *system name* but rather with a certain characteristic of a said system; **Example:** the airplane's available gradient of climb shall not be less than 1.2 percent for a two-engine airplane at each point along the takeoff path, starting from the point at which the airplane reaches 400 feet above the takeoff surface.

4.2. Requirement Table

Table 5 shows a requirement table with five requirements belonging to various types and their corresponding properties. The various columns of the table were populated by extracting information from the original requirement text using different LMs (aeroBERT-NER and aeroBERT-Classifier) that were fine-tuned on an aerospace-specific corpus. This table can be exported as an Excel spreadsheet, which can then be verified by a subject matter expert (SME), and any missing information can be added.

The creation of a requirement table, as described above, is an important step toward the standardization of requirements by aiding the creation of tables and models in SysML. The use of various LMs automates the process, hence reducing the manual labor commonly associated with populating the table. In addition, aeroBERT-NER and aeroBERT-Classifier generalize well and are capable of identifying named entities and classifying requirements despite the noise and variations that can occur in NL requirements. As a result, this methodology for extracting information from NL requirements and storing them in tabular format triumphs in comparison to using a dictionary-based approach, which needs constant updating as the requirements evolve.

Table 5. Requirement table containing columns extracted from NL requirements using language models. This table can be used to aid the creation of SysML requirementTable [10].

Serial No.	Name	Text	Type of Requirement	Property	Related to
1	nozzle guide vanes	All nozzle guide vanes should be weld-repairable without a requirement for strip coating.	Design		{SYS: [nozzle guide vanes]}
2	flight recorder	The state estimates supplied to the flight recorder shall meet the aircraft-level system requirements and the functionality specified in Section 23–2500.	Design	{RES: [Section 23–2500]}	{SYS: [flight recorder, aircraft]}
3	pressurized airplanes	Pressurized airplanes with a maximum operating altitude greater than 41,000 feet must be capable of detecting damage to the pressurized cabin structure before the damage could result in rapid decompression that would result in serious or fatal injuries.	Functional	{VAL: [greater than, 41,000 feet]}	{SYS: [pressurized airplanes, pressurized cabin structure]}
4	fuel system	Each fuel system must be arranged so that any air that is introduced into the system will not result in power interruption for more than 20 s for reciprocating engines.	Performance	{VAL: [20 s]}	{SYS: [fuel system, reciprocating engines]}
5	structure	The structure must be able to support ultimate loads without failure for at least 3 s.	Performance	{VAL: [3 s]}	{SYS: [structure]}

4.3. Boilerplate Templates

The requirements were first classified into various types using the aeroBERT-Classifier. Boilerplate templates for various types of requirements were then determined by utilizing sentence chunks and named entities to detect patterns. To account for the diversity of these requirements, multiple templates were recognized for each type.

Table 6 shows a breakdown of the number of boilerplate templates that were identified for each requirement type and the percentage of requirements the boilerplate templates apply to. Two boilerplates were identified for the design requirements included as part of this effort. Five and three boilerplates were identified for the functional and performance requirements, respectively. A greater variability was observed in the textual patterns occurring in functional requirements, which resulted in a greater number of boilerplates for this particular type. The identified templates are discussed in detail in the following subsections.

Table 6. The table exhibits the outcomes from a coverage analysis performed on the proposed boilerplate templates. The analysis revealed that 67 design, 37 functional, and 26 performance requirements did not align with the boilerplate templates developed during this study.

Requirement Type	Boilerplate Number	Number of Requirements That Fit Boilerplate	Number of Requirements That Do Not Fit Any Boilerplate
Design (149)	1	74 (∼50%)	67 (∼45%)
	2	8 (∼5%)	
Functional (100)	1	20 (20%)	37 (37%)
	2	15 (15%)	
	3	7 (7%)	
	4	15 (15%)	
	5	6 (6%)	
Performance (61)	1	20 (∼33%)	26 (∼42%)
	2	12 (∼20%)	
	3	3 (∼5%)	

4.3.1. Design Requirements

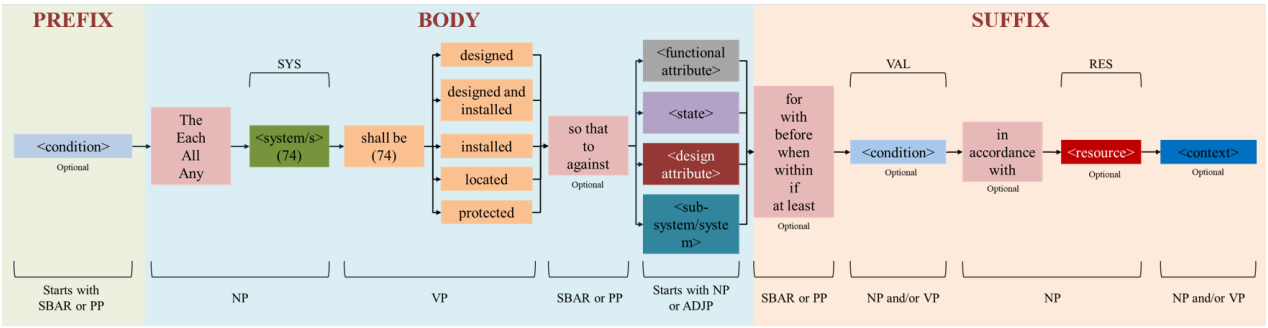
In analyzing the design requirements as they were presented, it was discovered that two separate boilerplate structures were responsible for roughly 55% of the requirements used in the study. These two structures encompass the majority of the requirements. Incorporating additional boilerplate templates would have resulted in overfitting them to only a handful of requirements each. This would have compromised their ability to be applied broadly, reducing their overall generalizability.

The first boilerplate is shown in Figure 12 and focuses on requirements that mandate the way a system should be designed and/or installed, its location, and whether it should protect another system/sub-system from a certain <condition> or <state>. The named entity and sentence chunk tags are displayed above and below the boilerplate structure. Based on these tags, it was observed that a requirement with an <condition> in the beginning usually starts with a PP or SBAR. This is followed by an NP, which contains a <system> name, which can be distinctly identified within the NP by using the named entity tag (SYS). The initial NP is always succeeded by a verb phrase (VP), which contains the term "shall [variations]", e.g., shall be designed, shall be protected, shall have, shall be capable of, shall maintain, etc. These were crucial for the identification of boilerplates since they provided information regarding the action that the <system> performs (to protect, to maintain, etc.).

In the case of Figure 12, the observed "[variations]" were *be designed, be designed and installed, installed, located, and protected*. The VP is followed by a SBAR or PP but this is optional. This is then followed by an NP or ADJP and can contain either a <functional attribute>, <state>, <design attribute>, or a <sub-system/system>. This brings an end to the main *Body* of the requirement. The *Suffix* is optional and can contain additional information, such as operating and environmental conditions, resources, and context.

The second boilerplate for design requirements is shown in Figure 13. This boilerplate accounts for the design requirements that mandate a certain <functional attribute> that a system should have, a <sub-system/system> it should include, and any <design attribute> it should have by design. Similar to the previous boilerplate, the named entities and sentence chunk tags are displayed above and below the structure.

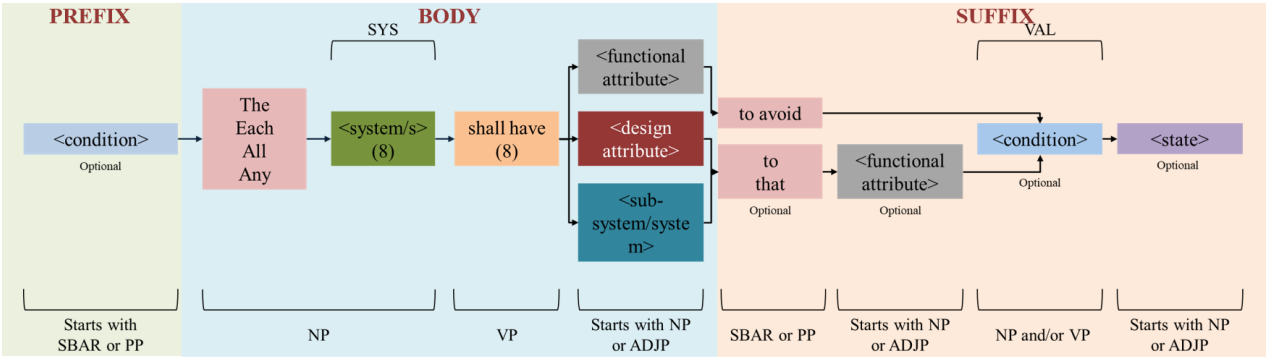
The rest of the design requirements were examined; however, no common patterns were observed in most of them to warrant the creation of boilerplates specific to these requirements. Boilerplates, if created, would have fewer requirements compatible with them, which could have undermined their capacity to be applied more generally. As a result, the overall generalizability of the templates would have been reduced.



The control system shall be designed for pilot forces applied in the same direction, using individual pilot forces not less than 0.75 times those obtained in accordance with Section 25-395.

Any taxi and landing lights shall be designed and installed so they provide sufficient light for night operations.

Figure 12. The schematics of the first boilerplate for **design requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 74 of the 149 design requirements (∼50%) used for this study and is tailored toward requirements that mandate the way a <system> should be designed and/or installed, its location, and whether it should protect another <system/sub-system> given a certain <condition> or <state>. Parts of the NL requirements shown here are matched with their corresponding boilerplate elements via the use of the same color scheme. In addition, the sentence chunks and named entity tags are displayed below and above the boilerplate structure, respectively.



Each part of the airplane shall have adequate provisions for ventilation and drainage.

Each recorder container shall have reflective tape affixed to its external surface to facilitate its location under water.

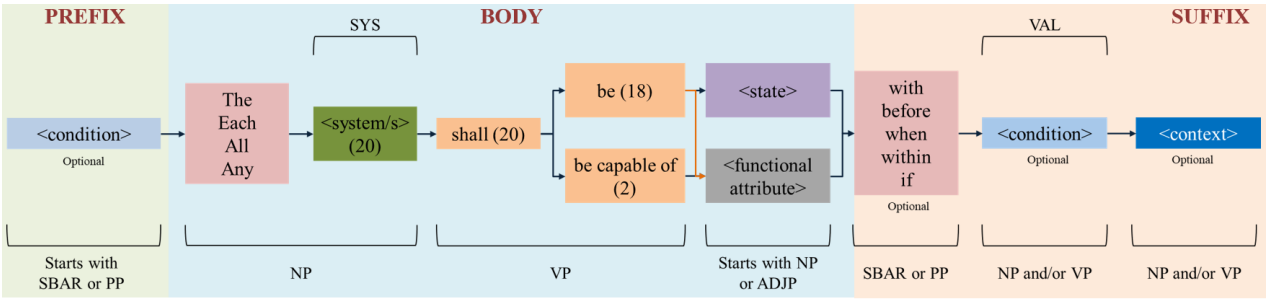
Each recorder container shall have an underwater locating device.

Figure 13. The schematics of the second boilerplate for **design requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 8 of the 149 design requirements (∼5%) used for this study and focuses on requirements that mandate a <functional attribute>, <design attribute>, or the inclusion of a <system/sub-system> by design. Two of the example requirements highlight the <design attribute> element, which emphasizes additional details regarding the system design to facilitate a certain function. The last example shows a requirement where a <sub-system> is to be included in a system by design.

4.3.2. Functional Requirements

In analyzing the NL functional requirements as they appeared in Parts 23 and 25 of Title 14 CFRs, the study identified five separate boilerplate structures that encompassed a total of 63% of the functional requirements. However, as previously mentioned, introducing more boilerplate templates would have led to fitting a smaller number of requirements to these structures, potentially limiting their overall applicability and generalizability.

The first boilerplate is shown in Figure 14. It is tailored toward requirements that describe the capability of a <system> to be in a certain <state> or perform a certain <function>. The example requirements focus on the handling characteristics of the system. The associated sentence chunks and NEs for each of the elements of the boilerplate are also shown.



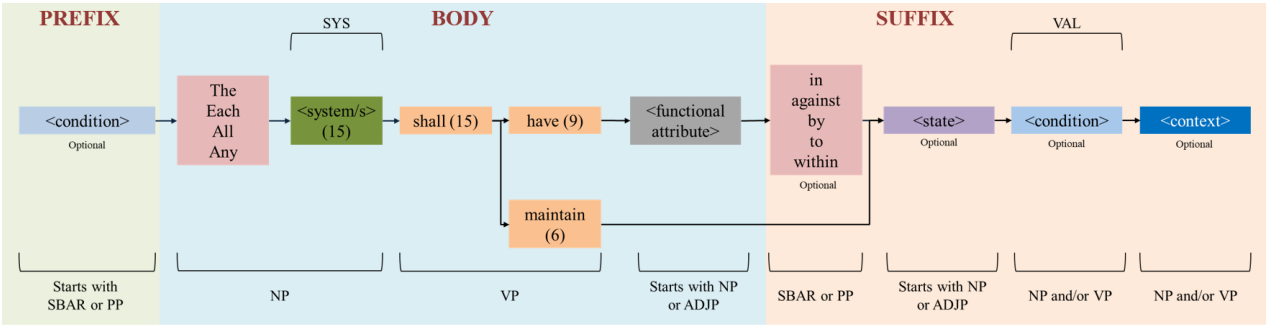
The airplane shall be controllable and maneuverable, without requiring exceptional piloting skill, alertness, or strength, within the operating envelope at all loading conditions for which certification is requested.

The insulation on electrical wire and electrical cable shall be self-extinguishing.

All pressurized airplanes shall be capable of continued safe flight and landing following a sudden release of cabin pressure.

Figure 14. The schematics of the first boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 20 of the 100 functional requirements (20%) used for this study and is tailored toward requirements that describe the capability of a <system> to be in a certain <state> or have a certain <functional attribute>. The first example requirement focuses on the handling characteristics of the system (airplane in this case).

The second boilerplate for functional requirements is shown in Figure 15 and focuses on requirements that require the <system> to have a certain <functional attribute> or maintain a particular <state>. This boilerplate structure accounts for 15% of all the functional requirements.



The airplane shall maintain longitudinal trim without further force upon, or movement of, the primary flight controls.

With the cabin configured for takeoff or landing, the airplane shall have means of egress that can be readily located and opened from the inside and outside.

Figure 15. The schematics of the second boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 15 of the 100 functional requirements (15%) used for this study and is tailored toward requirements that require the <system> to have a certain <functional attribute> or maintain a particular <state>.

Figure 16 shows the third boilerplate for functional requirements and is tailored toward requirements that require the <system> to protect another <sub-system/system> or <user> against a certain <state> or another <sub-system/system>. This boilerplate structure accounts for 7% of all the functional requirements.

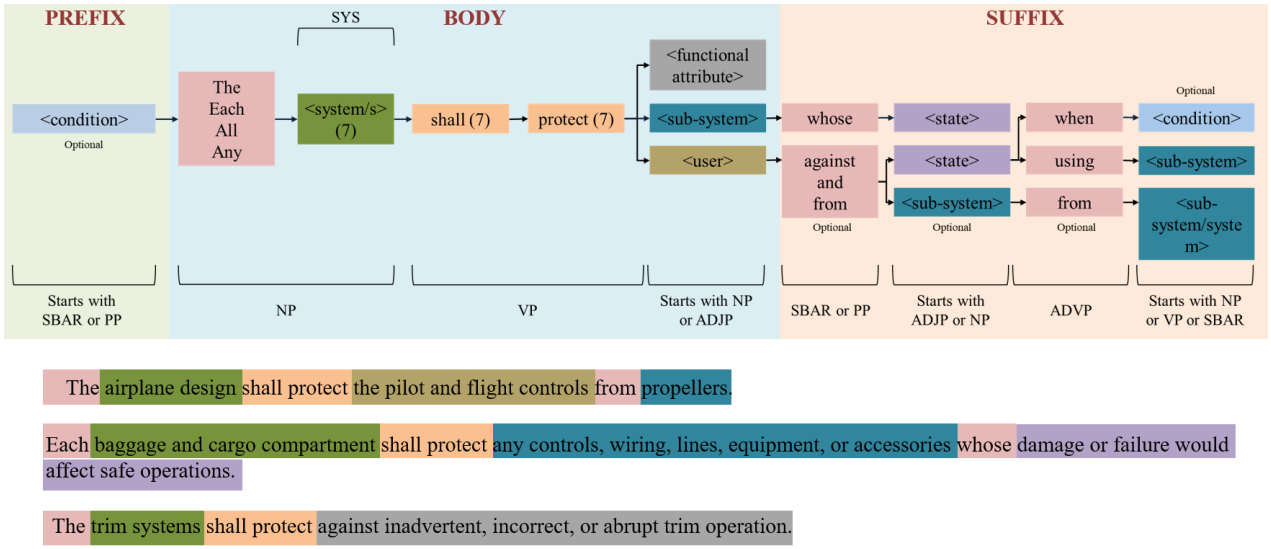


Figure 16. The schematics of the third boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 7 of the 100 functional requirements (7%) used for this study and is tailored toward requirements that require the <system> to protect another <sub-system/system> or <user> against a certain <state> or another <sub-system/system>.

Figure 17 shows the fourth boilerplate for functional requirements and is tailored toward requirements that require the <system> to provide a certain <functional attribute> given a certain <condition>. This boilerplate structure accounts for 15% of all the functional requirements.

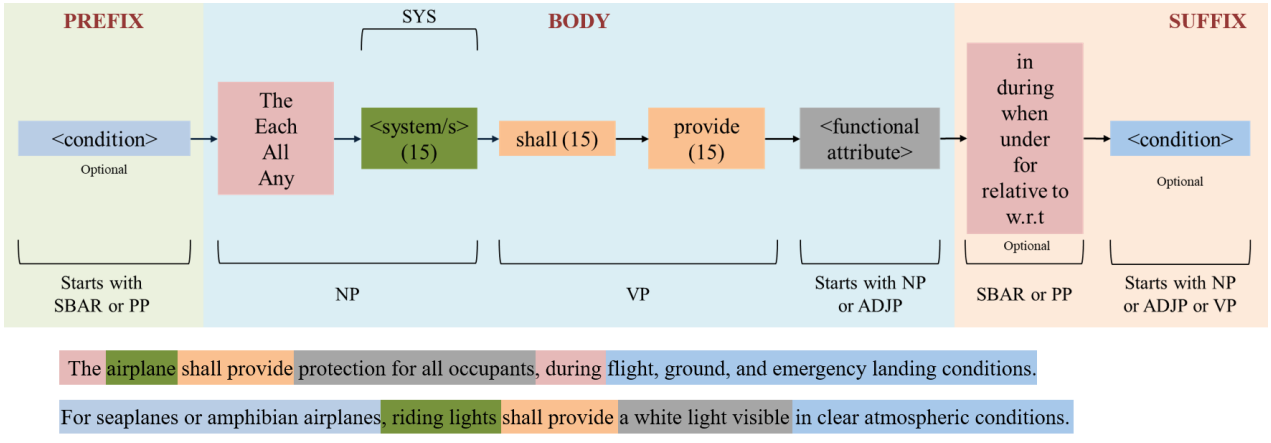


Figure 17. The schematics of the fourth boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 15 of the 100 functional requirements (15%) used for this study and is tailored toward requirements that require the <system> to provide a certain <functional attribute> given a certain <condition>.

Figure 18 shows the fifth boilerplate for functional requirements and is specifically focused on requirements related to the *cockpit voice recorder* since a total of six requirements in the entire dataset were about this particular system and its <functional attribute> given a certain <condition>. This boilerplate structure accounts for 6% of all the functional requirements. Although it is generally not recommended to have a boilerplate template that is specific to a particular system, in this case, it was deemed acceptable because a significant portion of the requirements pertained to that system, and the dataset used was relatively small.

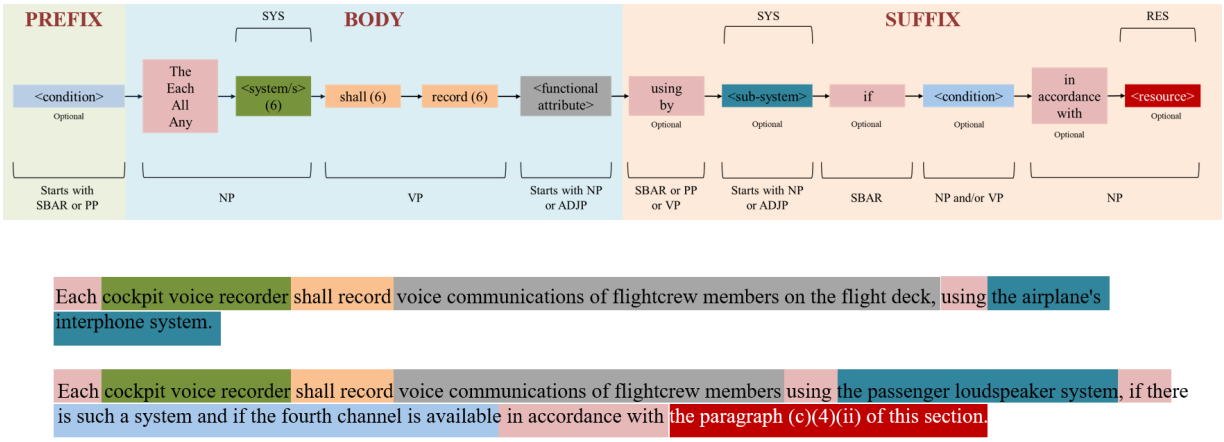


Figure 18. The schematics of the fifth boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 6 of the 100 design requirements (6%) used for this study and is specifically focused on requirements related to the *cockpit voice recorder* since a total of six requirements in the entire dataset were about this particular system and its <functional attribute> given a certain <condition>.

It is worth noting here that the number of templates may be changed somewhat as a matter of taste. If "provide" or "record" are lumped into the functional attribute block, then the five boilerplates become three. However, in that case, some of the nuances about the conditions of production in the third template or the media of recording in the fourth template may be lost. With the tools provided in this paper, such considerations can be discussed since the patterns have been clearly identified.

4.3.3. Performance Requirements

Three distinct boilerplates were identified for performance requirements which accounted for approximately 58% of all the requirements belonging to this type.

The first boilerplate for performance requirements is shown in Figure 19. This particular boilerplate has the element <system attribute>, which is unique as compared to the other boilerplate structures. In addition, this boilerplate caters to the performance requirements specifying a <system> or <system attribute> to satisfy a certain function or <condition>. Roughly 33% of all the performance requirements match this template.

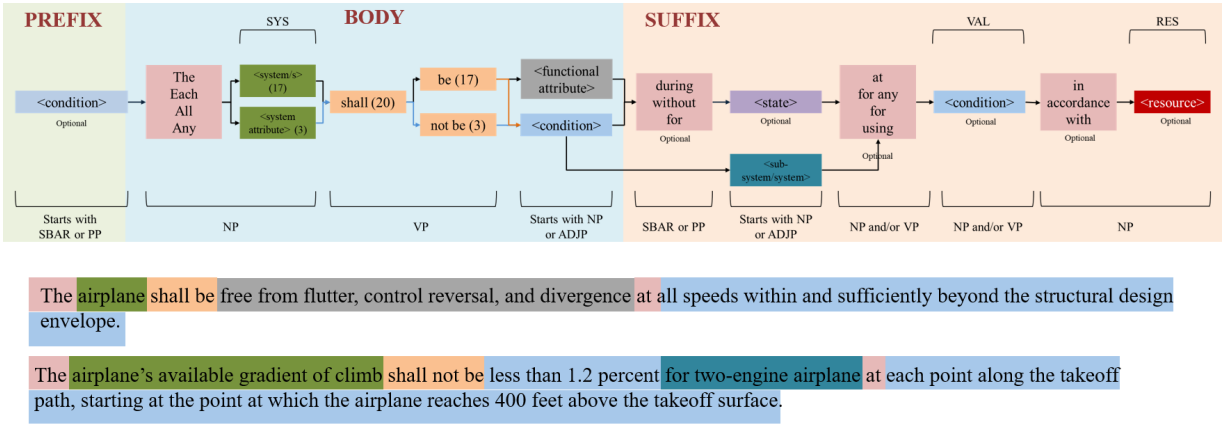
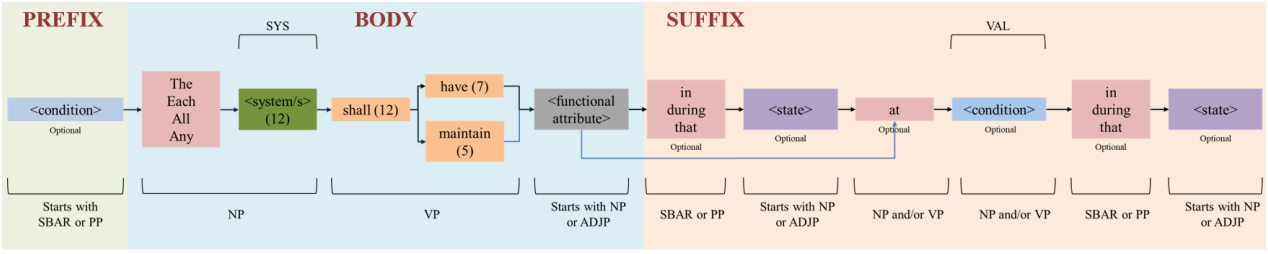


Figure 19. The schematics of the first boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 20 of the 61 performance requirements (~33%) used for this study. This particular boilerplate has the element <system attribute>, which is unique as compared to the other boilerplate structures. In addition, this boilerplate caters to the performance requirements specifying a <system> or <system attribute> to satisfy a certain <condition> or have a certain <functional attribute>.

Figure 20 shows the second boilerplate for performance requirements. This boilerplate accounts for roughly 20% of the requirements used for this study. This boilerplate focuses on performance requirements that specify a <functional attribute> that a <system> should have or maintain given a certain <state> or <condition>.

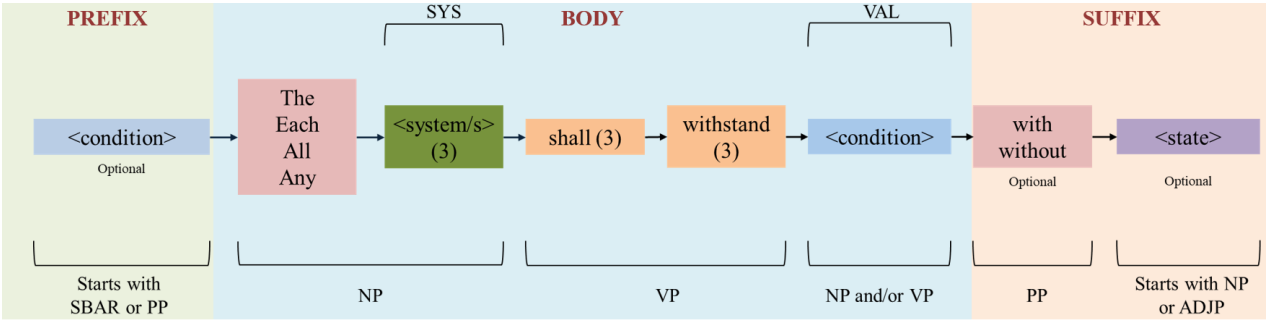


The airplane shall maintain longitudinal, directional, and lateral trim at 1.3 V_{SR1} during climbing flight with the remaining engines at maximum continuous power.

The airplane shall have controllable stall characteristics in straight flight, turning flight, and accelerated turning flight.

Figure 20. The schematics of the second boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 12 of the 61 performance requirements (∼20%) used for this study. This boilerplate focuses on performance requirements that specify a <functional attribute> that a <system> should have or maintain given a certain <state> or <condition>.

Lastly, Figure 21 shows the third boilerplate for performance requirements. This boilerplate accounts for 3 of the 61 performance requirements (about 5%) used for this study and focuses on a <system> being able to *withstand* a certain <condition> with or without ending up in a certain <state>.



Each fuel storage system shall withstand the loads under likely operating conditions without failure.

Figure 21. The schematics of the third boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 3 of the 61 performance requirements (∼5%) used for this study and focuses on a <system> being able to *withstand* and certain <condition> with or without ending up in a certain <state>.

In summary, this work identified two boilerplate structures for design requirements, five for functional requirements, and three for performance requirements. These structures were established by observing patterns in sentence chunks and named entities from the requirements dataset. In addition, the chunking exercise led to the definition of multiple elements that can lead to further understanding of the requirements. These elements were also the building blocks for passing on to other system models or means of analyzing architectures. A body of relevant conditions and states for the system were also highlighted and made ready for extraction in these boilerplates. Table 6 illustrates a detailed breakdown of the coverage of the boilerplate templates developed in this study.

5. Conclusions Future Work

Two language models, aeroBERT-NER and aeroBERT-Classifier, fine-tuned on annotated aerospace corpora, were used to demonstrate a methodology for creating a requirements table. This table contains five columns and is intended to assist with the creation of a requirements table in SysML. Additional columns can be added to the table if needed, but this may require developing new language models to extract the necessary data. Furthermore, aeroBERT-NER and aeroBERT-Classifier can be extended by adding new named entities or requirement types to extract other relevant information.

Boilerplate templates for various types of requirements were identified using the aeroBERT models for classification and NER as well as an off-the-shelf sentence chunking model (flair/chunk-english). To account for variations within requirements, multiple boilerplate templates were obtained. Two, five, and three boilerplate templates were identified for the design, functional, and performance requirements (used for this work), respectively.

The use of these templates, particularly by inexperienced engineers working with requirements, will ensure that requirements are written in a standardized form from the beginning. In doing so, this work democratizes a methodology for the identification of boilerplates given a set of requirements, which can vary from one company or industry to another.

Boilerplates can be utilized to create new requirements that follow the established structure or to assess the conformity of NL requirements with the identified boilerplates. These activities are valuable for standardizing requirements on a larger scale and at a faster pace. Subject matter experts (SMEs) should review the identified boilerplates to ensure their accuracy and consistency.

The boilerplates were identified specifically for certification requirements outlined in Parts 23 and 25 of Title 14 CFRs; while these boilerplate templates may not directly correspond to the proprietary system requirements used by aerospace companies, the methodology presented in this paper remains applicableand reproducible.

Future work could focus on exploring the creation of a data pipeline that incorporates different LMs to automatically or semi-automatically translate NL requirements into system models. This method would likely involve multiple rounds of refinement and necessitate input from experienced MBSE practitioners. In addition to models in SysML, more formal models, such as linear temporal logic, satisfiability modulo theories, and so on, are worthy of consideration for downstream processing. Furthermore, it is possible to carry out post-processing of the results generated by the LLMs. This process entails the development and utilization of dictionaries that encompass different system names alongside their aliases.

Exploring the use of generative LMs, like T5, the GPT family, etc., may also prove beneficial in rewriting free-form NL requirements. These models could be trained on a dataset containing NL requirements and their corresponding rewritten versions that adhere to industry standards. With this training, the model may be capable of generating well-written requirements based on the input NL requirements. Although untested, this approach presents an interesting research direction to explore.

6. Other Details

The models leveraged in this study to identify boilerplates were aeroBERT-NER [8], aeroBERT-Classifier [9], and flair/chunk-english [11]; while the specifics regarding the training of these models are not discussed in this paper, Figure 22 below is included to support practitioners in the creation of the aforementioned models.

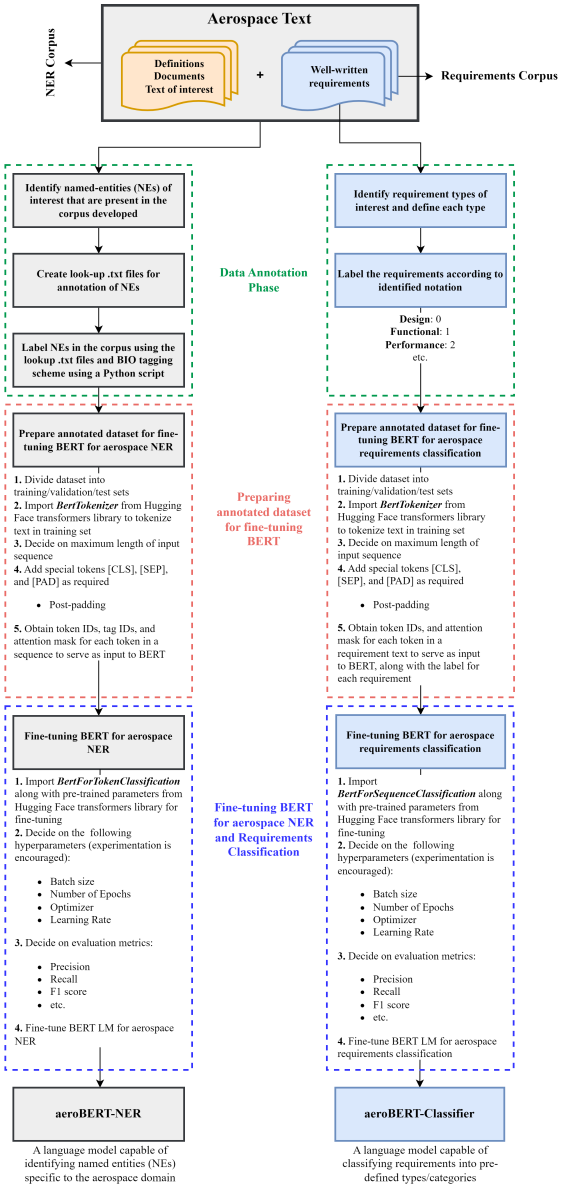


Figure 22. Practitioner's Guide to creation of aeroBERT-NER and aeroBERT-Classifier. A zoomed-in version of this figure can be found https://archanatikayatray19.github.io/Practitioners_Guide/ [10].

Author Contributions: A.T.R.: conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing—original draft preparation, and writing—review and editing; B.F.C.: conceptualization and writing—review and editing; O.J.P.F.: conceptualization and writing—review and editing; A.P.B.: data curation, formal analysis, investigation, methodology, software, validation, visualization, and writing—review and editing; R.T.W.: writing—review and editing; D.N.M.: writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The annotated aerospace requirements dataset can be found on the Hugging Face platform (URL: https://huggingface.co/datasets/archanatikayatray/aeroBERT-classification, accessed on 10 May 2023). The annotated aerospace NER dataset can be found on the Hugging Face platform (URL: https://huggingface.co/datasets/archanatikayatray/aeroBERT-NER, accessed on 10 May 2023).

Acknowledgments: The authors express their gratitude to Evan Harrison for creating the requirement table included in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BERT	Bidirectional Encoder Representations from Transformers
CFR	Code of Federal Regulations
FAA	Federal Aviation Administration
FAR	Federal Aviation Regulations
GPT	Generated Pre-trained Transformer
INCOSE	International Council on Systems Engineering
LM	Language Model
LLM	Large Language Model
MBSE	Model-Based Systems Engineering
NE	Named Entity
NER	Named Entity Recognition
NL	Natural Language
NLP	Natural Language Processing
NLP4RE	Natural Language Processing for Requirements Engineering
ORG	Organization (Entity label)
RE	Requirements Engineering
RES	Resource (Entity label)
SME	Subject Matter Expert
SYS	System (Entity label)
SysML	Systems Modeling Language

Appendix A

Appendix A.1. Functional Requirements

Of the 100 requirements classified as functional, 84 started with an NP, 10 started with a PP, and 6 started with a SBAR. The majority of the NPs are followed by a VP, which occurred in 69 requirements. The detailed sequence of patterns is shown in Figure A1.

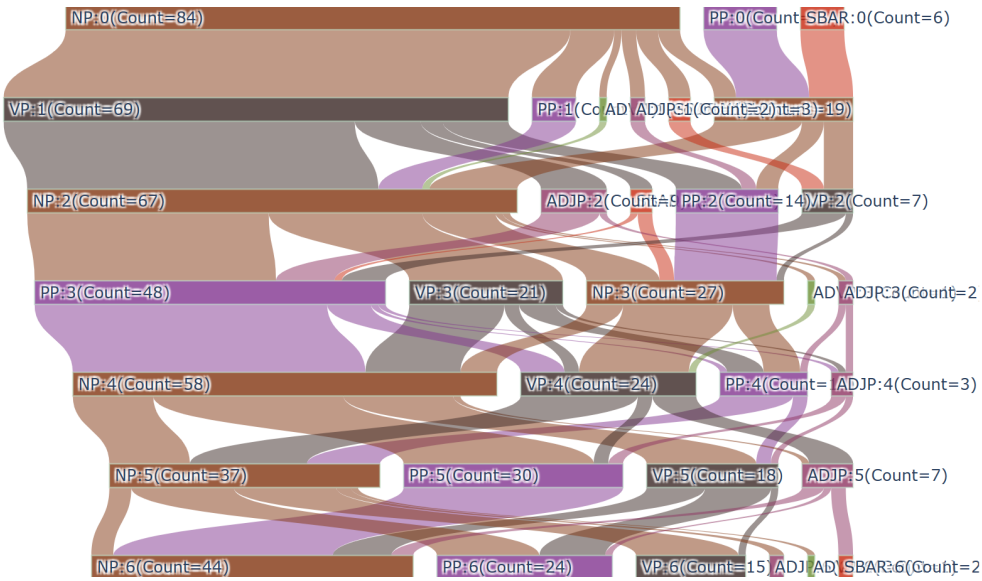


Figure A1. Sankey diagram showing the text chunk patterns in functional requirements. A part of the figure is shown due to space constraints; however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Functional_pos_chunk_Requirements_Diagram.png [10].

Functional requirements used for this work start with an NP, PP, or SBAR. Figure A2 shows examples of functional requirements beginning with these three types of sentence chunks.

Example 1: Each cockpit voice recorder shall record voice communications transmitted from or received in the airplane by radio.

Example 2: For levels 1,2, and 3, the airplane must maintain lateral and directional trim in cruise without further force upon, or movement of, the primary flight controls or corresponding trim controls by the pilot, or the flight control system.

Example 3: Unless the failure of an automatic power or thrust control system is extremely remote, the system must provide a means for the flightcrew to override the automatic function.

Figure A2. Examples 1, 2, and 3 show functional requirements starting with an NP, PP, and SBAR. Most of the functional requirements start with an NP, however.

The functional requirements beginning with an NP have system names in the beginning (example 1 of Figure A2). However, this is not the case for requirements that start with a condition, as shown in example 3.

Figure A3 shows the patterns of named entities for functional requirements where a majority of the requirements start with a system name (SYS), and only seven requirements start with a value (VAL) named entity. The requirements starting with a VAL might contain conditions in the beginning of these requirements.

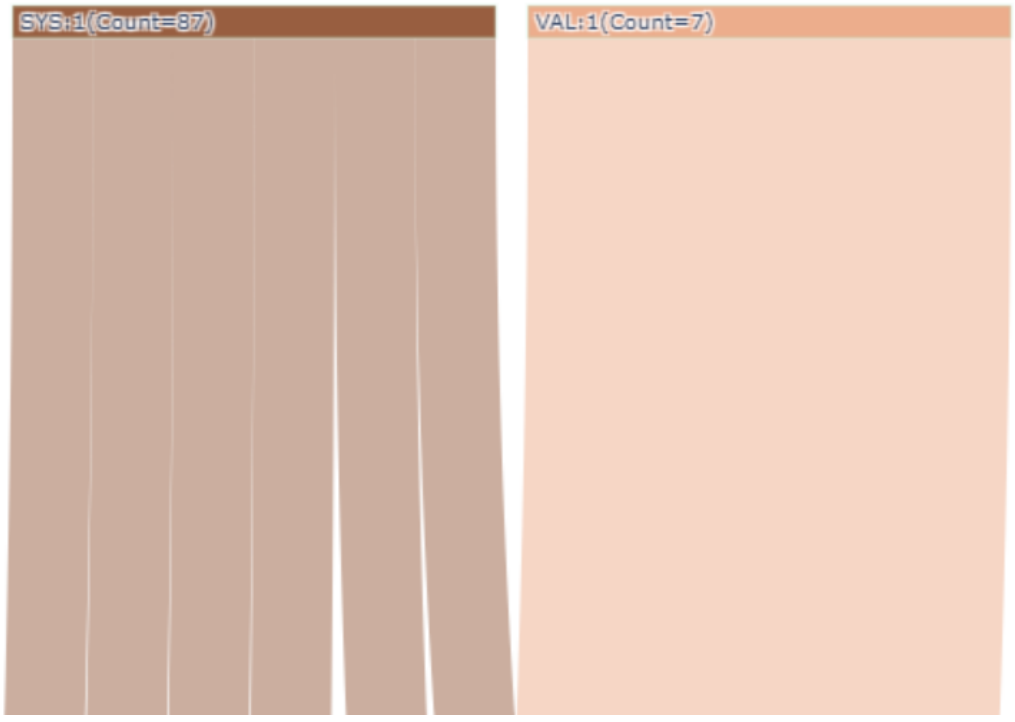


Figure A3. Sankey diagram showing the named entity patterns in functional requirements. A part of the figure is shown due to space constraints; however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Functional_NER_Requirements_Diagram.png [10].

Appendix A.2. Performance Requirements

Of the 61 requirements classified as performance requirements, 53 started with an NP and 8 started with a PP. The majority of the NPs are followed by a VP for 39 requirements. A detailed sequence of patterns is shown in Figure A4.

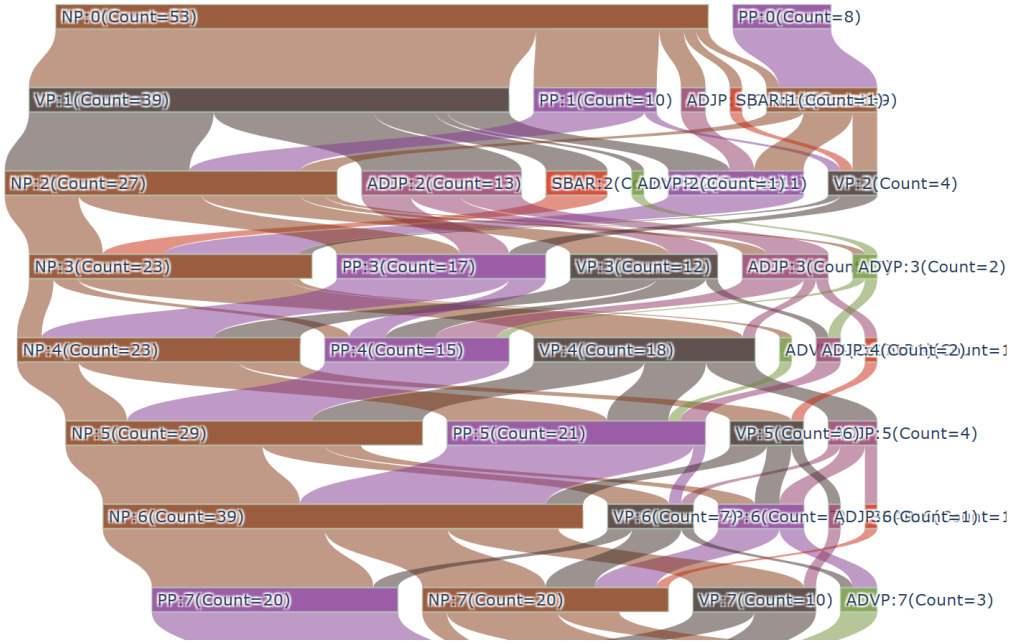


Figure A4. Sankey diagram showing the text chunk patterns in performance requirements. A part of the figure is shown due to space constraints; however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Performance_pos_chunk_Requirements_Diagram.png [10].

Examples of performance requirements starting with an NP and PP are shown in Figure A5. The requirement starting with an NP usually starts with a system name, which is in line with the trends seen in the design and functional requirements, whereas the requirements starting with a PP usually have a condition in the beginning.

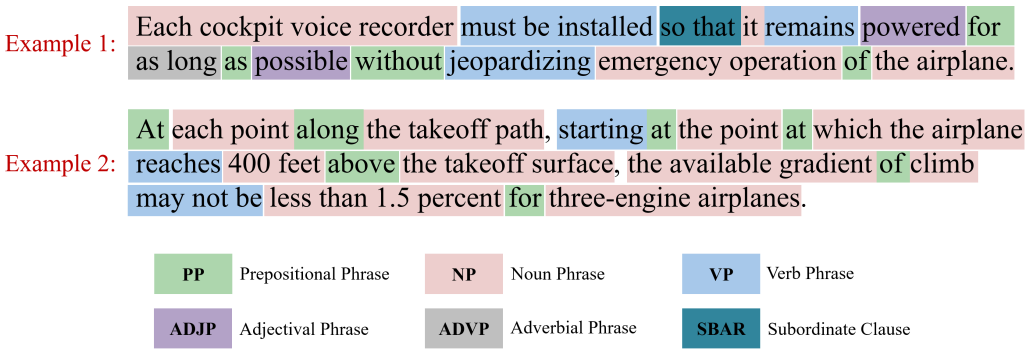


Figure A5. Examples 1 and 2 show performance requirements starting with NP and PP, respectively.

In example 2 (Figure A5), quantities such as "400 feet" and "1.5 percent" are tagged as NP; however, there is no way to distinguish between the different types of NPs (NPs containing cardinal numbers vs. not). Using aeroBERT-NER in conjunction with flair/chunk-english helps clarify different types of entities beyond their text chunk tags, which is helpful for ordering entities in a requirement text. The same idea applies to resources (RES, for example, Section 25–395) as well.

Figure A6 shows the named entity patterns observed in performance requirements. Just like design and functional requirements, the majority of performance requirements

start with a system (SYS) named entity. More variation in the sequence of named entities can be observed when one moves down the Sankey diagram.

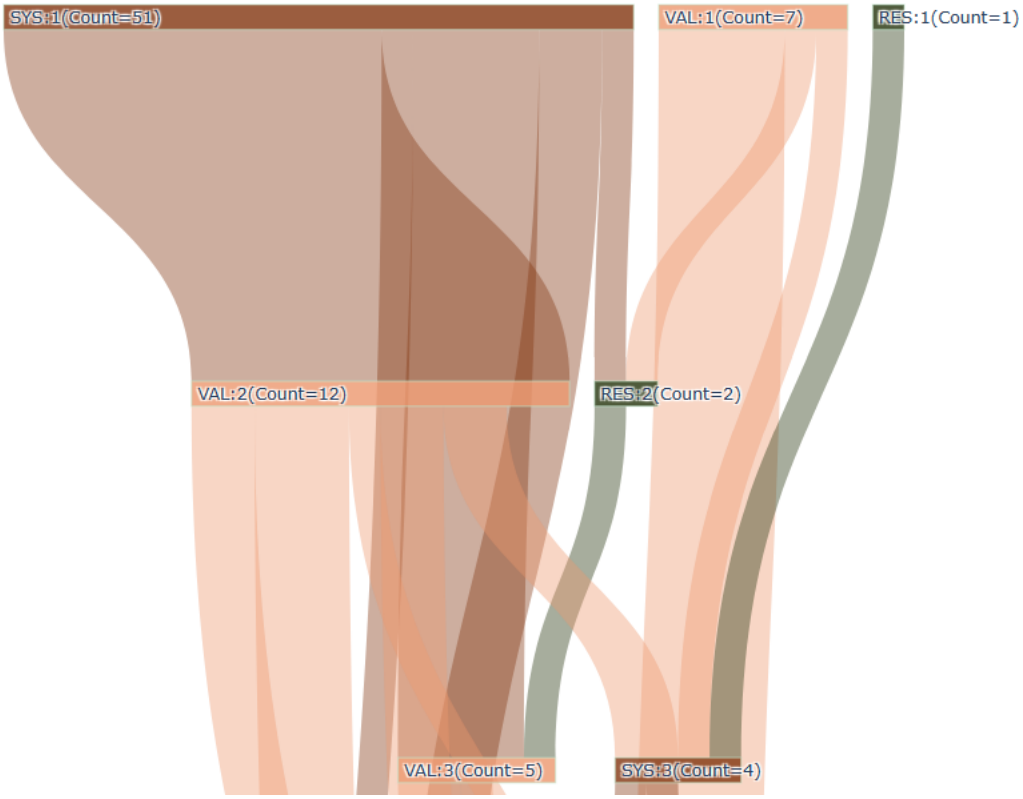


Figure A6. Sankey diagram showing the named entity patterns in performance requirements. A part of the figure is shown due to space constraints; however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Performance_NER_Requirements_Diagram.png [10].

References

1. INCOSE . *Guide to the Systems Engineering Body of Knowledge*; BKCASE Editorial Board; INCOSE: San Diego, CA, USA , 2020; p. 945.
2. INCOSE. INCOSE INFRASTRUCTURE WORKING GROUP Charter. pp. 3–5. Available online: https://www.incose.org/docs/default-source/wgcharters/infrastructure.pdf?sfvrsn=9e0eb2c6_18 (accessed 10 January 2023).
3. NASA. Appendix C: How to Write a Good Requirement. pp. 115–119. Available online: https://www.nasa.gov/seh/appendix-c-how-to-write-a-good-requirement (accessed on 5 January 2023).
4. Regnell, B.; Svensson, R.B.; Wnuk, K. Can we beat the complexity of very large-scale requirements engineering? In Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality, Montpellier, France, 16–17 June 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 123–128.
5. Firesmith, D. Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *J. Object Technol.* **2007**, *6*, 17–33. [CrossRef]
6. Haskins, B.; Stecklein, J.; Dick, B.; Moroney, G.; Lovell, R.; Dabney, J. 8.4. 2 error cost escalation through the project life cycle. *INCOSE Int. Symp.* **2004**, *14*, 1723–1737. [CrossRef]
7. Bell, T.E.; Thayer, T.A. Software requirements: Are they really a problem? In Proceedings of the Proceedings of the 2nd International Conference on Software Engineering, San Francisco, CA, USA, 13–15 October 1976; pp. 61–68.
8. Tikayat Ray, A.; Pinon Fischer, O.J.; Mavris, D.N.; White, R.T.; Cole, B.F. aeroBERT-NER: Named-Entity Recognition for Aerospace Requirements Engineering using BERT. In Proceedings of the AIAA SCITECH 2023 Forum, National Harbor, MD, USA, 23–27 January 2023. [CrossRef]
9. Tikayat Ray, A.; Cole, B.F.; Pinon Fischer, O.J.; White, R.T.; Mavris, D.N. aeroBERT-Classifier: Classification of Aerospace Requirements Using BERT. *Aerospace* **2023**, *10*, 279 . [CrossRef]
10. Tikayat Ray, A. Standardization of Engineering Requirements Using Large Language Models. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2023. [CrossRef]
11. Akbik, A.; Blythe, D.; Vollgraf, R. Contextual String Embeddings for Sequence Labeling. In Proceedings of the COLING 2018, 27th International Conference on Computational Linguistics, Santa Fe, NM, USA, 20–26 August 2018; pp. 1638–1649.

12. Estefan, J.A. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* **2007**, *25*, 1–12.
13. Jacobson, L.; Booch, J.R.G. *The Unified Modeling Language Reference Manual*; Addison-Wesley: Boston, MA, USA , 2021.
14. Ballard, M.; Peak, R.; Cimtalay, S.; Mavris, D.N. Bidirectional Text-to-Model Element Requirement Transformation. In Proccedings of the IEEE Aerospace Conference, Big Sky, MT, USA, 7–14 March 2020; pp. 1–14.
15. Lemazurier, L.; Chapurlat, V.; Grossetête, A. An MBSE approach to pass from requirements to functional architecture. *IFAC-PapersOnLine* **2017**, *50*, 7260–7265. [CrossRef]
16. INCOSE. *Needs, Requirements, Verification, Validation Lifecycle Manual*; BKCASE Editorial Board; INCOSE: San Diego, CA, USA, 2022; p. 457.
17. Requirement Table. Available online: https://docs.nomagic.com/display/SYSMLP182/Requirement+Table (accessed on 10 February 2023).
18. Modeling Requirements with SysML. Available online: https://re-magazine.ireb.org/articles/modeling-requirements-with-sysml (accessed on 10 February 2023).
19. Rupp, C. *Requirements-Engineering und-Management: Professionelle, Iterative Anforderungsanalyse für die Praxis*; Hanser Verlag: Munich, Germany, 2007.
20. Vallejo, P.; Mazo, R.; Jaramillo, C.; Medina, J.M. Towards a new template for the specification of requirements in semi-structured natural language. *J. Softw. Eng. Res. Dev.* **2020**, *8*, 3. [CrossRef]
21. Mavin, A.; Wilkinson, P.; Harwood, A.; Novak, M. Easy approach to requirements syntax (EARS). In Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, 31 August–4 September 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 317–322.
22. Ferrari, A.; Dell'Orletta, F.; Esuli, A.; Gervasi, V.; Gnesi, S. Natural Language Requirements Processing: A 4D Vision. *IEEE Softw.* **2017**, *34*, 28–35. [CrossRef]
23. Abbott, R.J.; Moorhead, D. Software requirements and specifications: A survey of needs and languages. *J. Syst. Softw.* **1981**, *2*, 297–316. [CrossRef]
24. Dalpiaz, F.; Ferrari, A.; Franch, X.; Palomares, C. Natural language processing for requirements engineering: The best is yet to come. *IEEE Softw.* **2018**, *35*, 115–119. [CrossRef]
25. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J.; Ajagbe, M.A.; Chioasca, E.V.; Batista-Navarro, R.T. Natural language processing for requirements engineering: A systematic mapping study. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–41. [CrossRef]
26. Bengio, Y.; Ducharme, R.; Vincent, P. A Neural Probabilistic Language Model. In *Advances in Neural Information Processing Systems*; Leen, T., Dietterich, T., Tresp, V., Eds.; MIT Press: Cambridge, MA, USA, 2000; Volume 13.
27. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30* .
28. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:cs.CL/1810.04805.
29. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* **2020**, *21*, 1–67.
30. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
31. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; Curran Associates, Inc.: New York, NY, USA, 2020; Volume 33, pp. 1877–1901.
32. Hugging Face. Available online: https://huggingface.co/ (accessed on 10 January 2023).
33. Dalpiaz, F.; Dell'Anna, D.; Aydemir, F.B.; Çevikol, S. Requirements Classification with Interpretable Machine Learning and Dependency Parsing. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju, Republic of Korea, 23–27 September 2019; pp. 142–152. [CrossRef]
34. Sonbol, R.; Rebdawi, G.; Ghneim, N. The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review. *IEEE Access* **2022**, *10*, 62811–62830. [CrossRef]
35. Riesener, M.; Dölle, C.; Becker, A.; Gorbatcheva, S.; Rebentisch, E.; Schuh, G. Application of natural language processing for systematic requirement management in model-based systems engineering. *INCOSE Int. Symp.* **2021**, *31*, 806–815. [CrossRef]
36. Arora, C.; Sabetzadeh, M.; Briand, L.; Zimmer, F.; Gnaga, R. Automatic checking of conformance to requirement boilerplates via text chunking: An industrial case study. In Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, USA, 10–11 October 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 35–44.
37. Federal Aviation Administration (FAA) . *Title 14 Code of Federal Regulations*; FAR: Washington, DC, USA, 2023 .
38. Jurafsky, D.; Martin, J.H. *Speech and Language Processing (Draft)*; Stanford University: Stanford, CA, USA, 2021.
39. Syntax. Available online: https://webspace.ship.edu/cgboer/syntax.html (accessed on 21 February 2023).