

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

[Archana Tikayat Ray](#)*, Bjorn F Cole, [Olivia J Pinon Fischer](#)*, Anirudh Prabhakara Bhat, [Ryan T White](#), [Dimitri N Mavris](#)

Posted Date: 18 May 2023

doi: 10.20944/preprints202305.1325.v1

Keywords: Requirements Engineering; Natural Language Processing; NLP; BERT; Requirements boilerplates; Model-Based Systems Engineering; MBSE; Requirements table; Large Language Models (LLMs); Transformer based language models



Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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, Georgia, 30332, USA; dimitri.mavris@aerospace.gatech.edu

² Lockheed Martin Space, Littleton, Colorado, 80127, USA; bjorn.f.cole@lmco.com

³ Amazon, Toronto, Canada, M5H 3T9; anipbhat@gmail.com

⁴ NEural Transmission Laboratory, Department of Mathematical Sciences, Florida Institute of Technology, Melbourne, Florida, 32901, USA; rwhite@fit.edu

* Correspondence: atr@gatech.edu (A.T.R.); olivia.pinon@asdl.gatech.edu (OJ.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 towards 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 that hinder 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 semi-machine-readable requirements. This 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 semi-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 is 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 were utilized. Two of these models were fine-tuned versions of Bidirectional Encoder Representations from Transformers (BERT), specifically *aeroBERT-NER* and *aeroBERT-Classifier*, which were trained on annotated aerospace corpora. Another LLM, called *flair/chunk-english*, was utilized to identify sentence chunks present in NL requirements. All three language models were utilized together to achieve the standardization of requirements. To demonstrate the effectiveness of the methodologies, requirements from Parts 23 and 25 of Title 14 Code of Federal Regulations (CFRs) were employed, and a total of two, five, and three boilerplate templates were identified for design, functional, and performance requirements, respectively.

Keywords: requirements engineering; natural language processing; NLP; BERT; requirements boilerplates; model-based systems engineering; MBSE; requirements table; Large Language Models (LLMs); transformer based language models

1. Introduction

This section provides a short introduction to the field of requirements engineering and discusses its importance to the product lifecycle. The challenges associated with natural language (NL) in requirements elicitation and the benefits expected from transitioning towards model-based approaches are also addressed. It is known that the inherent ambiguities of NL requirements significantly impede this paradigm shift. To address this, methods facilitated by NLP for converting NL requirements into semi-machine-readable requirements, including the use of a requirements table and boilerplate

templates (in a data-driven way), are presented. Finally, this section concludes by summarizing the focus and contributions of the research presented herein.

1.1. Requirements Definition and Requirements Engineering

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 NL to ensure that they are comprehensible to diverse stakeholders, including consumers, subcontractors, and equipment manufacturers, with varying degrees of expertise [4].

Requirements engineering is a discipline that involves understanding stakeholder expectations and translating them into technical requirements [5]. Simply put, it entails defining, documenting, and maintaining requirements throughout the engineering lifecycle of a system [6]. The prevailing approach to requirements engineering involves gathering customer input through a series of design studies and interviews, aimed at identifying the most critical customer needs and values for a potential system. These needs are then translated into technical language, comprising functions, performance attributes, and “-ilities” such as availability, maintainability, reliability, serviceability, usability, etc. While formal mapping methods like Quality Function Deployment (QFD) are sometimes used, it is typically a crafts approach honed within the system-developing organizations.

Having well-defined requirements and implementing sound requirements engineering practices is crucial for the effective design, development, and operation of systems, products, and processes. Conversely, mistakes made during the requirement definition phase can have far-reaching implications for downstream tasks like system architecture, design, implementation, inspection, and testing [7]. These issues can have significant engineering and programmatic consequences if not addressed early in the product life cycle [8,9].

1.2. Shift towards Model-based Systems Engineering (MBSE)

The inherent complexity in modern-day systems warrants a holistic approach to their design and development. Model-Based Systems Engineering (MBSE) has emerged as a solution to this need, involving the use of models to support system design processes rather than traditional document-based methods [10]. Models capture requirements and domain knowledge, making them accessible to all stakeholders [11,12]. However, the inherent ambiguities and inconsistencies of NL requirements hinder their direct conversion to models [13]. As creating models manually is both time-consuming and requires specialized expertise, there is a need to transform NL requirements into a semi-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 [14]. The document envisions an informational environment where requirements are interconnected with each other, 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, as depicted in Figure 1.

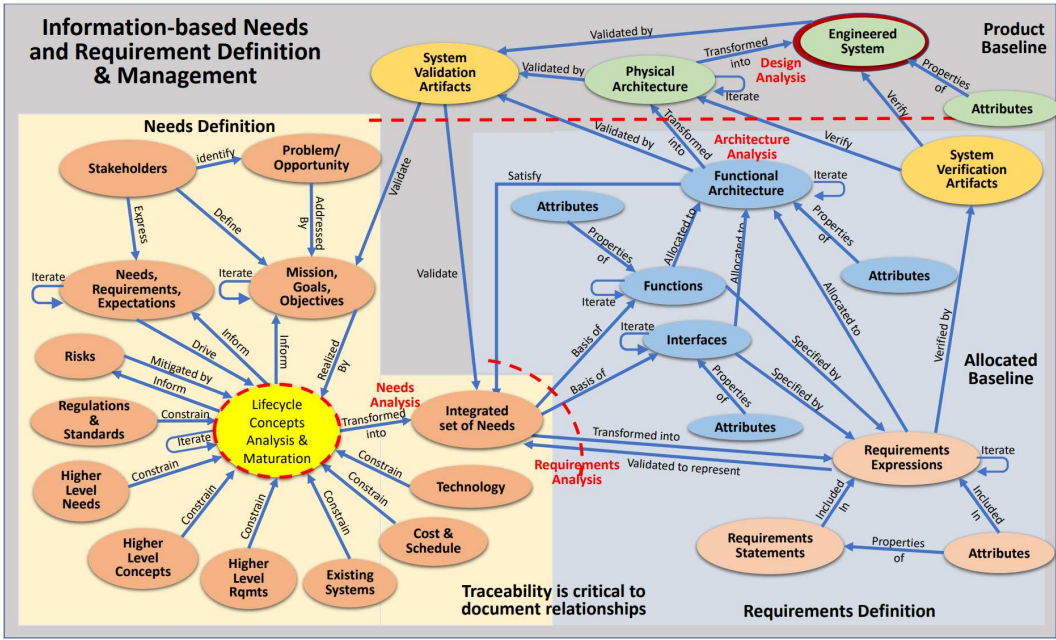


Figure 1. Information-based requirement development and management model [15].

1.3. Semi-Machine-Readable Requirements

As mentioned earlier, utilizing semi-machine-readable requirements (Figure 2) can simplify the incorporation of natural language requirements in a model-based setting. Converting NL requirements to semi-machine-readable requirements can be achieved through the use of requirement tables or boilerplate templates, as discussed below.

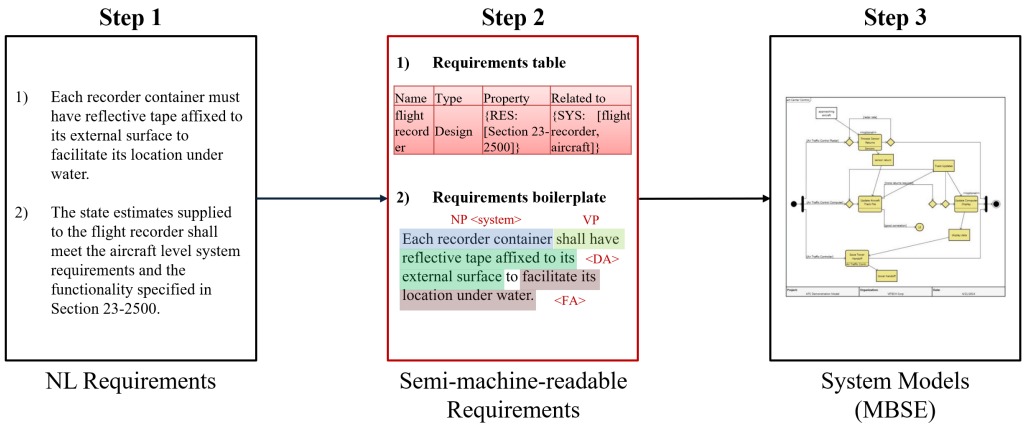


Figure 2. Steps of requirements engineering, starting with gathering requirements from various stakeholders, followed by using NLP techniques to standardize them and lastly converting the standardized requirements into models. The main focus of this work is to convert NL requirements into semi-machine-readable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

1.3.1. Requirement Table

A requirement table is a tabular representation of requirements for a system, product, or process and its associated properties. It typically includes columns for the requirement ID, description, source, priority, verification method, status, and other relevant attributes. The table can be used to manage and track requirements throughout the development lifecycle and ensure that they are adequately fulfilled. It also helps in identifying dependencies and conflicts between requirements, as well as any missing or redundant requirements. The requirements table is an essential tool for requirements engineering

and is widely used in software development, system engineering, and other related fields [16,17]. The creation of a requirement table can be facilitated by extracting useful information (system names, values, etc.) from freeform NL requirements, as shown in Figure 3.

The air-taxi shall have a configuration that can seat five passengers within its passenger cabin.

SYSTEM = air-taxi, passenger cabin
VALUE = five passengers

Figure 3. An illustration of how to convert the contents of a natural language requirement into data objects can be seen in this example. In the given requirement “*The air-taxi shall have a configuration that can seat five passengers within its passenger cabin*”, the *air-taxi* and *passenger cabin* are considered as SYSTEM, while *five passengers* is classified as a VALUE.

1.3.2. Requirement Boilerplates

As stated previously, requirements are articulated in NL because of the universality it offers. Several researchers have proposed the use of templates (molds or patterns) for improving the quality of requirements, reducing ambiguity and inconsistencies [18,19]. *Requirement boilerplates* are pre-defined linguistic patterns [20] that can be applied to NL aerospace requirements to standardize them and eventually allow them to be used for automated analysis. The advantages of boilerplates are that the requirements are still in NL and hence are accessible to all stakeholders while having a well-defined syntactic structure [18] that facilitates their understanding, which takes us closer to semi-machine-readable requirements. The Rupp’s [18,20,21] and EARS boilerplate templates [20,22] are among the most well-known boilerplate templates. Nevertheless, due to their rigid structure, these templates are less flexible and unable to be customized for varying types of requirements across different organizations. Further elaboration on these boilerplates will be provided in subsequent sections of the paper.

1.3.3. Leveraging Language Models

Language Models (LMs) can fill in the necessary columns of a requirements table by extracting information from NL requirements. Moreover, if equipped with named entity identification (NER), requirement classification, and sentence chunk identification capabilities, LMs can collaborate to identify linguistic patterns in requirements, facilitating the detection of boilerplate templates. The capabilities offered by LMs make NLP an enabler for achieving the aims of this work.

1.4. Research Focus and Expected Contributions

In summary, the growing complexity of systems has led to a shift towards MBSE for system design and development. However, the challenges posed by the ambiguities and inconsistencies of NL requirements have impeded their integration into model-based environments. To address this issue, this research aims to devise a methodology that leverages LMs to facilitate the conversion of NL requirements into semi-machine-readable formats in a partially automated manner, promoting their use in MBSE environments. The main contributions of this study are:

1. Methodology involving LMs (aeroBERT-NER [23], and aeroBERT-Classifer [24]) to extract information from NL requirements to populate requirement tables.
2. Methodology for identifying boilerplate templates by considering the linguistic patterns extracted from NL requirements by multiple LMs (aeroBERT-Classifer [24], aeroBERT-NER [23], and flair/chunk-english [25]).

The structure of this paper is as follows: Section 2 explores the relevance of NLP in requirements engineering and discusses various LMs that are applicable to the aerospace domain. In addition, it provides background information on sentence chunks, SysML requirement tables, and requirements

boilerplates. Section 3 identifies gaps in the literature and summarizes the research goal. The methodology used for creating requirement tables and boilerplate templates using multiple LMs is presented in Section 4. Section 5 discusses the results, including the development of a requirement table and that of the boilerplate templates developed for each type of requirement considered (design, functional, and performance). Finally, Section 6 summarizes this work, its limitations, and presents avenues for future work.

2. Background

This section is divided into two main subsections. The first subsection presents an overview of the enablers that can be utilized for the standardization of requirements. The second subsection discusses two approaches to standardizing requirements through the use of the enablers described in the first subsection.

2.1. Overview of Enablers

This section explores the importance of NLP techniques for requirements engineering. In particular, it discusses how language models specific to the aerospace engineering domain can be used to convert NL requirements into semi-machine-readable requirements.

2.1.1. Natural Language Processing for Requirements Engineering (NLP4RE)

NL is predominantly used for writing requirements to make them accessible to different stakeholders with varying levels of understanding of requirements engineering processes [26]. While a number of NLP tools (Structured Analysis Design Technique (SADT) and the System Design Methodology (SDM) developed at MIT [27]) have been used to facilitate requirements elicitation and management since the 1970s, the lack of advanced NL technologies and models have prevented them from being used at a wider scale. These tools were also known for not generalizing well to new requirements. Today's NLP capabilities have changed significantly, as illustrated by the availability of many open-source NLP tools and libraries (e.g., Stanford CoreNLP [28], NLTK [29], spaCy [30], etc.), as well as off-the-shelf Transformer-based [31] pre-trained LMs (e.g., BERT [32], BART [33], etc.). Compared to traditional rule-based methods, advanced ML-based techniques offer significant enhancements in terms of generalization, efficacy, as well as cost and time savings [34]. As a result of these attributes, LMs have made significant contributions to the field of Natural Language Processing for Requirements Engineering (NLP4RE). This trend, which has been observed in a recent study by Zhao et al. [35], is anticipated to continue to expand in the future.

However, despite the increase in the use of advanced NLP techniques for requirements engineering, a divide still prevails between NLP4RE research and its industrial penetration [35,36]. This divide can be attributed to the fragmented approach to sharing knowledge pertaining to NLP4RE tools, techniques [37], and the scarcity of datasets. Finally, in contrast to most studies in NLP4RE that are focused on the software engineering domain, this work instead aims to leverage these developments for the benefit of the aerospace domain.

2.1.2. Large Language Models (LLMs) Fine-Tuned for Use in Aerospace Domain

LLMs are at the heart of modern NLP. The advent of neural LMs which leveraged 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 [38]. Innovations such as self-supervision, multi-headed self-attention [31], and improved computational parallelization have paved the way for state-of-the-art LLMs like BERT LM [32], T5 character-level language model [39], and the Generative Pre-trained Transformer (GPT) family [40,41] of auto-regressive LMs. These models are pre-trained on vast corpora of text (such as Book Corpus and English Wikipedia) in a self-supervised manner, and can subsequently be fine-tuned on smaller labeled datasets for various

downstream NLP tasks such as text classification, named entity recognition (NER), part-of-speech (POS) tagging, text chunking, etc.

Training LLMs from the ground up can be prohibitively expensive due to the vast computational resources required. However, pre-trained LLMs from Google, Meta, OpenAI, and others are publicly available on the Hugging Face platform [42], and can be accessed through the *transformers* library.

As previously mentioned, LLMs are frequently trained on general-purpose text corpora (e.g., news articles, 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* [23] and *aeroBERT-Classifier* [24].

- **aeroBERT-NER** [23] 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*.

Table 1. *aeroBERT-NER* is capable of identifying five types of named entities. The BIO tagging scheme was used for annotating the NER dataset [23].

Category	NER Tags	Example
System	B-SYS, I-SYS	nozzle guide vanes, flight recorder, fuel system
Value	B-VAL, I-VAL	5.6 percent, 41000 feet, 3 seconds
Date time	B-DATETIME, I-DATETIME	2017, 2014, Sept. 19,1994
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

- **aeroBERT-Classifier** [24] 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 [43].

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 [44]. This ability to extract relevant information is crucial for ensuring that requirements are standardized, which in turn aids in their integration into an MBSE environment.

2.1.3. Usefulness of Sentence Chunks in Requirements Standardization

In order to standardize requirements, it’s important to identify common linguistic patterns within NL requirements. Achieving this requires a thorough understanding of the syntax used in the language. Techniques such as POS tagging and sentence chunking can aid in this endeavor.

Words in a sentence can be categorized into various POS (or lexical categories). These POS tags can be combined to obtain sentence chunks that are non-overlapping segments [45], as shown in Figure 4. POS tags by themselves can be noisy (since each word has a POS tag associated with it), making them less useful when it comes to identifying patterns in requirements. Sentence chunks, however, are more useful for reorganizing requirements into standardized templates.

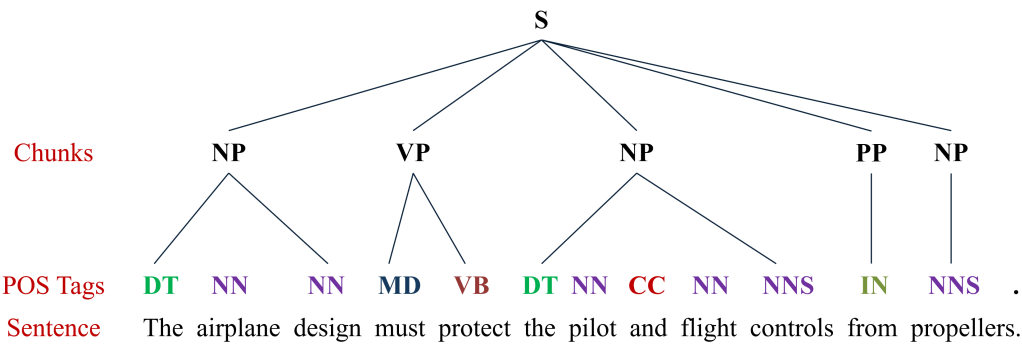


Figure 4. An aerospace requirement along with its POS tags and sentence chunks. Each word has a POS tag associated with it which can then be combined together to obtain a higher-level representation called sentence chunks (NP: Noun Phrase; VP: Verb Phrase; PP: Prepositional Phrase).

Some of the types of sentence chunks are listed in Table 2.

Table 2. A subset of text chunks along with definitions and examples is shown here [25,46]. The blue text highlights the type of text chunk of interest. This is not an exhaustive list.

Sentence Chunk	Definition
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.

Several pre-built English-language LMs are available for tasks like POS tagging and sentence/text chunking [25]. These models can extract relevant text chunks irrespective of the domain and do not need to be fine-tuned specifically for aerospace text.

2.2. Ways to Standardize Requirements

This subsection presents various concepts related to MBSE and the use of requirement tables and boilerplate templates to convert NL requirements into a standardized form, one that is more amenable to model-based approaches.

2.2.1. Requirement Table in SysML

A spreadsheet-like environment in SysML can be used to create a requirement table for capturing requirements. Each requirement is represented by a row in the table, with the columns capturing

related attributes or system model elements. This table can be used to filter requirements and their associated properties and can be easily exported to Microsoft Excel due to its tabular format [16]. In addition, such a table can aid in automated requirements analysis and modeling, resulting in cost and time savings as requirements change over time [17]. A sample SysML requirements table is depicted in Figure 5.
























#	Id	Name	Text	Satisfied By
1	S0.0	 Original Statement	Describe a system for purifying dirty water. - Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger - Boil dirty water is performed by a Boiler. Drain residue is performed by a Drain. The water has properties: vol = 1 liter, density 1 gm/cm ³ , temp 20 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.	
2	S0.1	 Elevation	The water distiller shall be able to operate at least 2 meters vertically above the source of dirty water.	
3	S1.0	 Purify Water	The system shall purify dirty water.	
4	S2.0	 Heat Exchanger	Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger	   
5	S3.0	 Boiler	Boil dirty water is performed by a Boiler.	   
6	S4.0	 Drain	Drain residue is performed by a Drain.	      
7	S5.0	 Water Properties	Water has properties: density 1 gm/cm ³ , temp 20 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.	
8	S5.1	 Water Initial Temp	Water has an initial temp 20 deg C	

Figure 5. A SysML requirement table with four columns, namely, Id, Name, Text, and Satisfied by. It typically contains these four columns by default, however, more columns can be added to capture other properties pertaining to the requirement.

Riesener et al. [47] presented a method for generating a SysML requirements table using a dictionary-based approach. Specifically, domain-specific keywords were added to a dictionary, and these words were extracted when they appeared in mechatronics requirements text. Named entity types such as technical units, related quantities, material properties, and manufacturing processes were of interest in their work. However, creating a dictionary for extracting relevant words is a time-consuming task, and the results may not be applicable to different projects. Furthermore, the dictionary needs to be updated frequently to account for the emergence of new units. Therefore, the use of LMs to extract entities of interest in a more generalizable manner is highly advantageous.

Furthermore, automating the creation of requirement tables can be highly beneficial as the current manual process is prone to human errors and involves repetitive tasks for populating the columns. LMs can be leveraged for this purpose, enabling the automation of the creation of requirement tables and facilitating the development of SysML tables.

2.2.2. Requirement Boilerplates

To improve the quality and reduce ambiguity in NL requirements, researchers have proposed the use of templates or “requirement boilerplates” in requirements. These pre-defined linguistic patterns

allow for the standardization of requirements using NL. As such, requirement boilerplates have shown to be of importance for requirements specification, which is of particular relevance for the development and certification of safety-critical systems (e.g., CESAR [48], OPENCOS [49], SAREMAN [19]).

Two of the most popular generalized boilerplates for standardizing requirements are Rupp’s [18,20], which is illustrated in Figure 6, and Easy Approach to Requirements Syntax (EARS), which is illustrated in Figure 7.

Two of the most popular generalized boilerplates for standardizing requirements are Rupp’s (Figure 6) which can be divided into six different parts and Easy Approach to Requirements Syntax (EARS) (Figure 7).

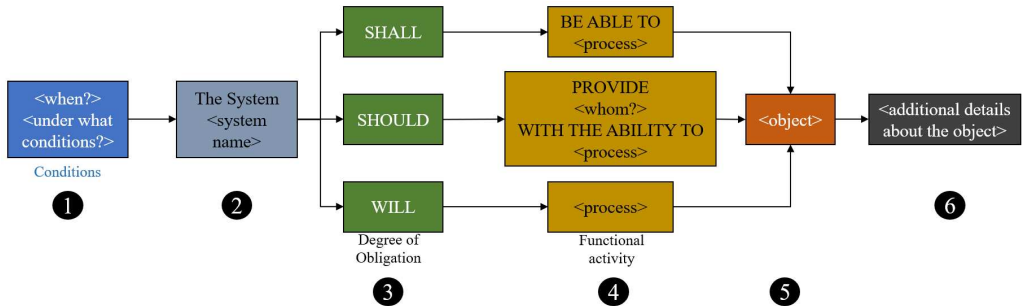


Figure 6. Rupp’s Boilerplate structure, which can be divided into six different parts, namely, (1) Condition, (2) System, (3) Degree of Obligation, (4) Functional activity, (5) Object, (6) Additional details about the object [18,20,21].

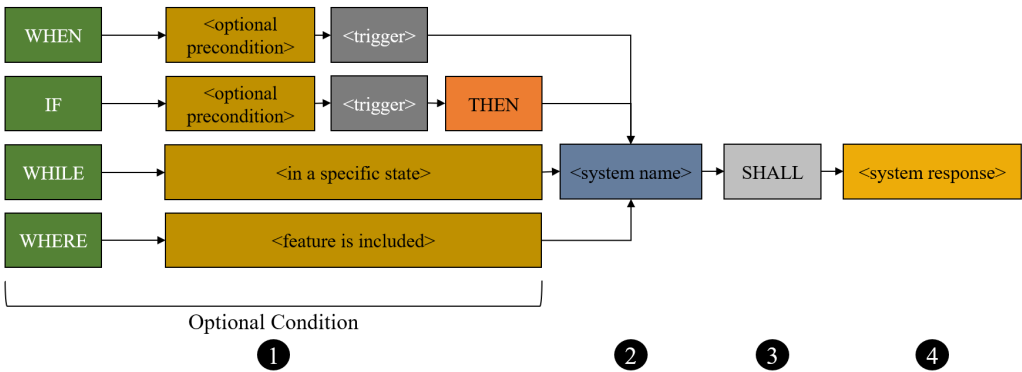


Figure 7. EARS Boilerplate structure, which can be divided into four different parts, namely, (1) Optional condition block, (2) System/subsystem name, (3) Degree of Obligation, (4) Response of the system [20,22].

Despite the benefits offered by requirement boilerplates (Rupp’s and EARS), they can be restrictive when it comes to certain types of functional and non-functional requirements and do not allow for the inclusion of constraints. In some cases, using Rupp’s boilerplate can lead to inconsistency and ambiguity due to the restrictions imposed by the structure leading to the exclusion of ranges of values and bi-conditionals, as well as the lack of reference to external systems that the original system interacts with [18]. Mazo et al. [18] later improved upon Rupp’s template and proposed a boilerplate that addresses the shortcomings of Rupp’s (Figure 8). However, Mazo’s template is quite complex to navigate given the number of degrees of freedom (or moving pieces). Hence, 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.

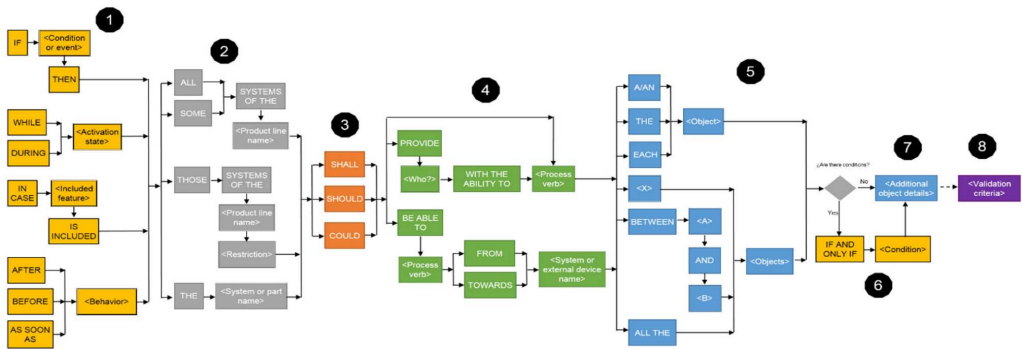


Figure 8. Mazo and Jaramillo template [18].

Conformance to Boilerplate Templates

Arora et al. [50] stress the importance of verifying whether requirements actually conform to a given boilerplate, which is important for quality assurance purposes. In their paper, they check the conformance of requirements to Rupp’s boilerplate using *text chunking* (NPs and VPs). In addition, they suggest that this is a robust method for checking conformance in the absence of a glossary. In another paper by the same authors [20], they develop a conformation check methodology for Rupp’s and EARS boilerplates using the NLP pipeline described in Figure 9.

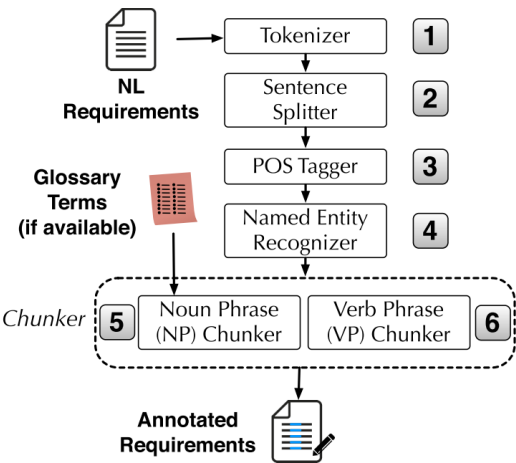


Figure 9. NLP pipeline for text chunking and NER [51].

The Need for Tailored Boilerplate Templates

The development of new boilerplates for various requirements is a positive step, however, progress in this area has been limited. There is *no one-size-fits-all* when it comes to boilerplates as organizations and industries have a variety of different business needs to be expressed in different forms that are most natural. Also due to the connection between requirements and verification, different expressions of technical requirements may naturally align with different modes of demonstration, inspection, or test. For example, computer program usability is focused on executing specific tasks rapidly and accurately. For aircraft, usability is expressed as vehicle handling characteristics in terms of response times or quantities of needed pilot inputs for safe control. Therefore, an agile methodology is necessary to facilitate the creation of boilerplates for different types of requirements. To that end, the present paper proposes a structured, scalable, and replicable approach to the creation of tailored boilerplates. This approach, which is informed by LLMs, addresses many of the issues and challenges with the manual creation of boilerplates, which is often arduous and time-consuming.

3. Research Gaps & Objectives

As mentioned, a divide exists between NLP4RE research and its industrial adaptation, with most of the research in this domain being mostly limited to software engineering. This work instead focuses on the aerospace domain (specifically aerospace requirements) and more specifically the application NLP to analyze and standardize requirements for the purpose of supporting the shift to model-based approaches and MBSE. The two LMs with aerospace-specific domain knowledge (aeroBERT-Classifier [24], and aeroBERT-NER [23]), previously developed by the authors were used in this work to establish a methodology (or pipeline) for standardizing requirements by first being able to classify, followed by extracting named entities of interest. These extracted entities were used to populate a requirements table in an Excel spreadsheet, which is expected to reduce the time and cost involved by helping MBSE practitioners in replicating this table in an MBSE environment, as compared to a completely manual process.

Lastly, a text chunking LM (flair/chunk-english [25]) along with aeroBERT-NER was used to obtain text chunks and named entities. These were used to identify patterns in each type of NL requirements considered. These patterns eventually led to the definition of boilerplates (templates) aimed at facilitating the future standardization of requirements.

Summary

Figure 10 shows the pipeline consisting of different LMs. Merging their outputs enables the conversion of NL requirements into semi-machine-readable (or standardized) requirements. To make it clear how these refactored requirements should be read, boilerplate templates are proposed with a uniform ordering of key *elements* in a given requirement.

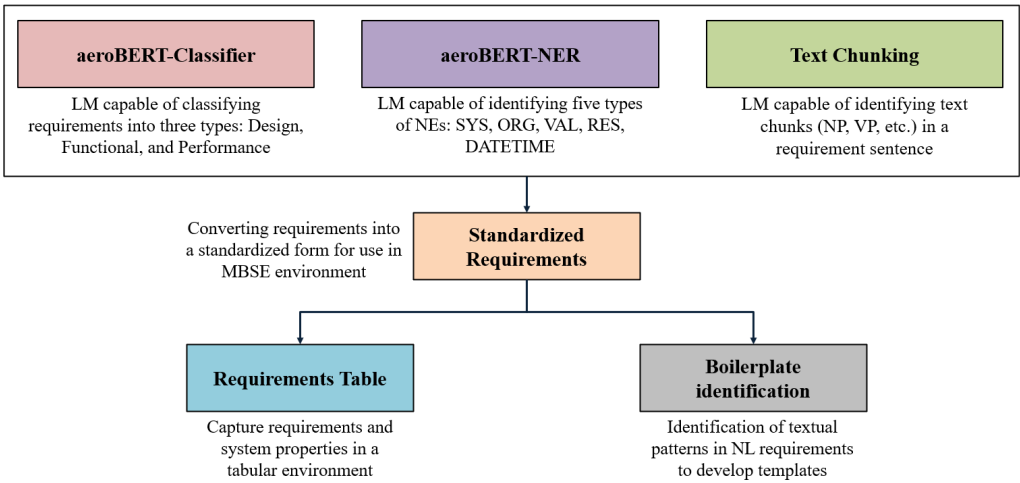


Figure 10. Pipeline for converting NL requirements to standardized requirements using various LMs.

The resulting standardized requirements support the creation of system models using the principles of MBSE. This effort achieves it through the:

1. **Creation of requirements tables:** The proposed requirement table contains columns populated by outputs obtained from aeroBERT-Classifier [24] and aeroBERT-NER [23]. This table can further aid in the creation of model-based (e.g., SysML) requirement objects by automatically extracting relevant words (system names, resources, quantities, etc.) from free-form NL requirements.
2. **Identification and creation of requirements boilerplates:** Distinct requirement types have unique linguistic patterns that set them apart from each other. "Patterns" in this case means the order of types of tags (sentence chunks and NEs) in sequences representing the original requirement.

To ensure the boilerplate templates are tailored to each requirement type in a swift and efficient manner, it is crucial to adopt an agile approach based on dynamically identified syntactic patterns in requirements, which is a more adaptive approach when compared to their rule-based counterparts. This study leverages language models to detect the linguistic patterns in requirements, which in turn aid in creating bespoke boilerplates.

The `aeroBERT-Classifier` is used for requirement classification, `aeroBERT-NER` for identifying named entities relevant to the aerospace domain, `flair/chunk-english` for extracting text chunks present in each requirement type. The extracted named entities and text chunks are analyzed to identify different *elements* present in a requirement sentence, and their presence and order in a requirement are used to determine distinct boilerplate templates. The templates identified might resemble the following formats:

- (a) <Each/The/All> <system/systems> shall <action>.
- (b) <Each/The/All> <system/systems> shall allow <entity> to be <state> for at least <value>.

In this study, well-crafted requirements are used to identify boilerplate templates, which can be utilized by inexperienced engineers to establish uniformity in their requirement compositions from the outset.

4. Methodology

This work uses a collection of 310 requirements (the list of the requirements can be found in [24]) categorized into design, functional, and performance requirements (Figure 3). Since requirements 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) [52]. While these requirements are mostly used for verifying compliance during certification, they have been used in this work to establish a methodology for the conversion of NL requirements into semi-machine-readable requirements.

Table 3. Breakdown of the types of requirements used for this work. Three different types of requirements were analyzed namely, design, functional, and performance.

Requirement type	Count
Design	149
Functional	99
Performance	62
Total	310

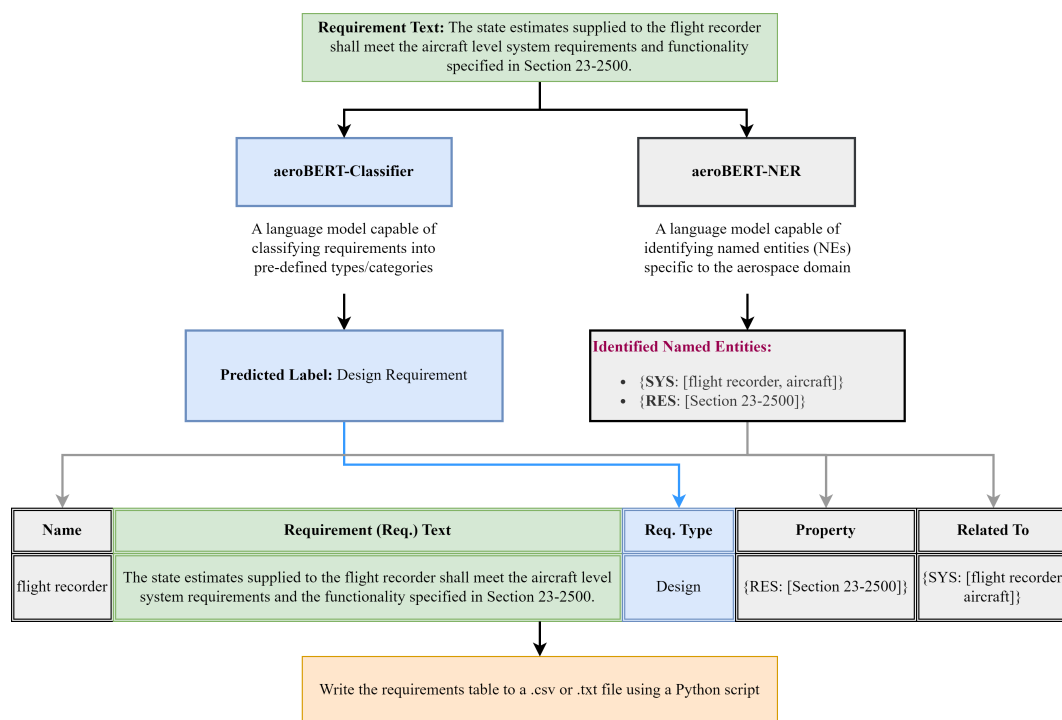
4.1. Creating Requirement Tables

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 4. The *Name* column was populated by the system name (SYS named entity) identified by `aeroBERT-NER` [23]. The *Type of Requirement* column was populated after the requirement is classified as a design, functional, or performance requirement by `aeroBERT-Classifier` [24]. The *Property* column was populated by all the named entities identified by `aeroBERT-NER` [23] (except for SYS) Lastly, the *Related to* column was populated by system names identified by `aeroBERT-NER`.

The flowchart in Figure 11 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.

Table 4. 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 [23]
Text	Original requirement text	Original requirement text
Type of Requirement	Classification of the requirement as design, functional, or performance	aeroBERT-Classifer [24]
Property	Identified named entities belonging to RES, VAL, DATETIME, and ORG categories present in a requirement related to a particular system (SYS)	aeroBERT-NER [23]
Related to	Identified system named entity (SYS) that the requirement properties are associated with	aeroBERT-NER [23]

**Figure 11.** Flowchart showcasing the creation of 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-Classifer [24] and aeroBERT-NER [23] 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 [53].

4.2. Observation and Analysis of Linguistic Patterns in Aerospace Requirements for Boilerplate Creation

This section discusses the process of identifying linguistic patterns in the requirements after they have been processed by flair/chunk-english and aeroBERT-NER. Discovered patterns will be discussed here as well. The next paragraphs, in particular, describe the pattern mining and standardization task for NL requirements analysis.

All the requirements were first classified using aeroBERT-Classifer [24]. This was done to examine if different types of requirements exhibit different textual patterns using sentence chunks (Table 2). flair/chunk-english language model [25] was used for text chunking and aeroBERT-NER [23] was used to identify named entities.

The details regarding the textual patterns identified in requirements belonging to various types are described below.

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. The detailed sequence of patterns is shown in Figure 12.

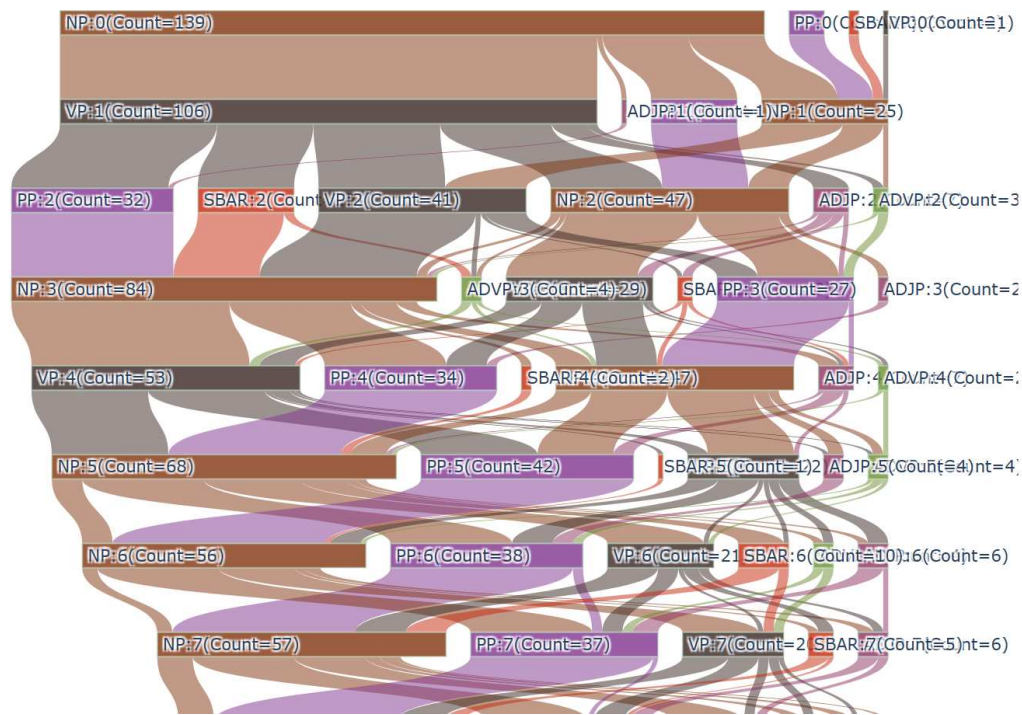


Figure 12. Sankey diagram showing the text chunk patterns in design requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found [here](#) [53].

Examples showing design requirements beginning with different types of sentence chunks are shown in Figures 13 and 14.

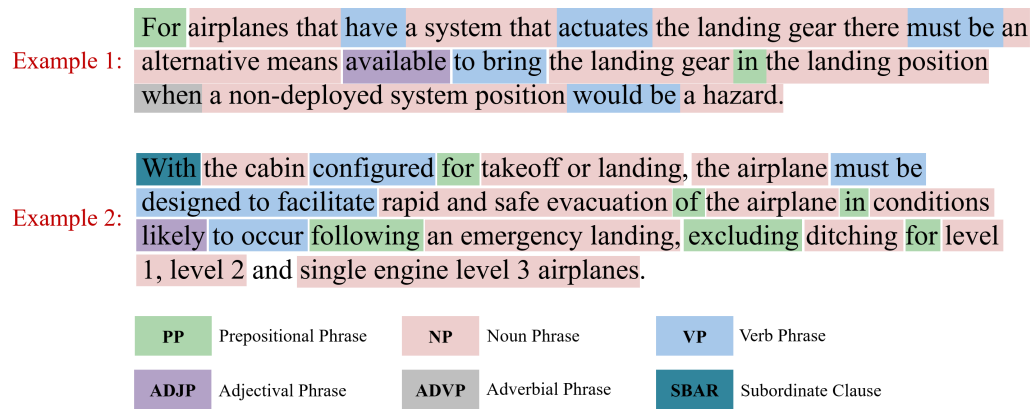


Figure 13. Examples 1 and 2 show a design requirement beginning with a prepositional phrase (PP) and subordinate clause (SBAR) which is uncommon in the requirements dataset used for this work. The uncommon starting sentence chunks (PP, SBAR) are however followed by a noun phrase (NP) and verb phrase (VP). Most of the design requirements start with a NP.

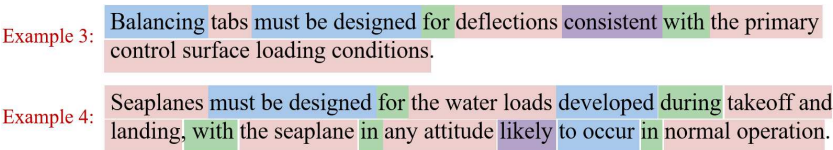


Figure 14. Example 3 shows a design requirement starting with a verb phrase (VP). Example 4 shows the requirement starting with a noun phrase (NP) which was the most commonly observed pattern.

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. For example 1, the initial prepositional phrase (PP) appears to specify a particular family of aircraft which are those with retractable landing gear. This provides a feel of a sector-wide policy statement rather than a limitation to a particular component or system. In all cases, the first NP is often the system name and followed by VPs.

It is important to note that in example 3 (Figure 14), 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 it 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.

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.

Functional Requirements

Of the 100 requirements classified as functional, 84 started with a 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 15.

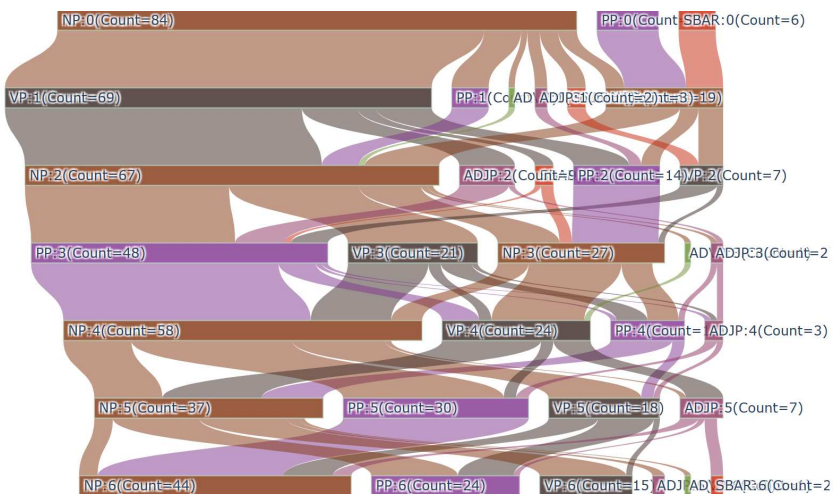


Figure 15. Sankey diagram showing the text chunk patterns in functional requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found https://github.com/archanatikayatray19/Sankey_diagram_Requirements/blob/main/Functional_pos_chunk_Requirements_Diagram.png [53].

Functional requirements used for this work start with an NP, PP, or SBAR. Figure 16 shows examples of functional requirements beginning with these three types of sentence chunks.

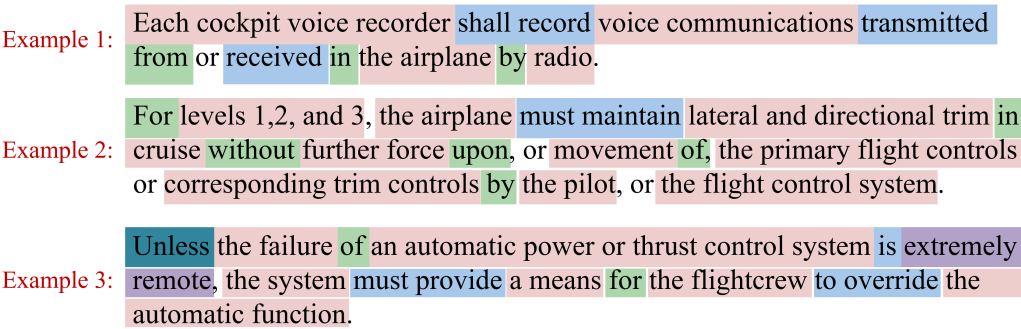


Figure 16. Examples 1, 2, and 3 show functional requirements starting with a NP, PP, and SBAR. Most of the functional requirements start with a NP, however.

The functional requirements beginning with an NP have system names in the beginning (Example 1 of Figure 16). However, this is not the case for requirements that start with a condition, as shown in example 3.

Performance Requirements

Of the 61 requirements classified as performance requirements, 53 started with a NP and 8 started with a PP. The majority of the NPs are followed by a VP for 39 requirements. The detailed sequence of patterns is shown in Figure 17.

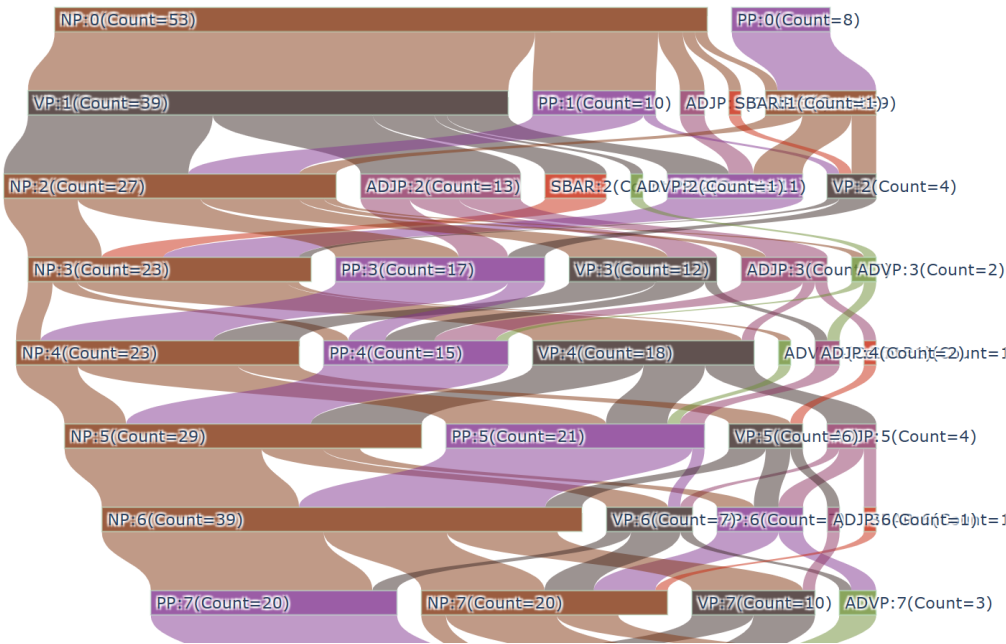


Figure 17. Sankey diagram showing the text chunk patterns in performance requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found [here](#) [53].

Examples of performance requirements starting with a NP and PP are shown in Figure 18. The requirement starting with a 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.

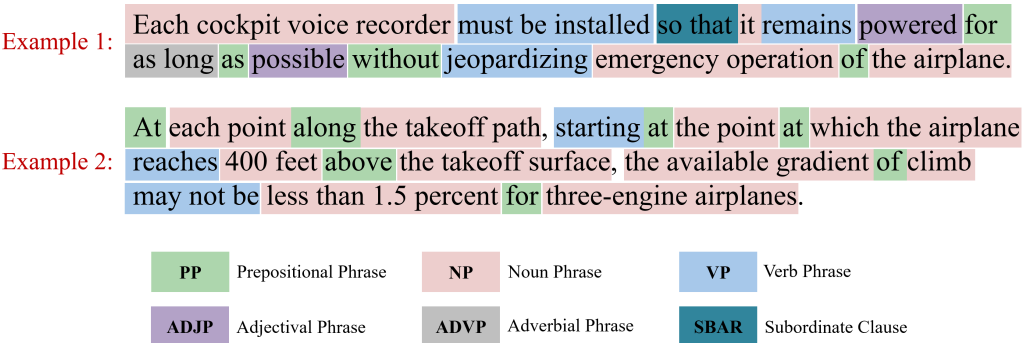


Figure 18. Examples 1 and 2 show performance requirements starting with NP and PP respectively.

In example 2 (Figure 18), 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.

4.2.1. 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 19.

The *Body* section of the requirement usually starts with a NP (system name), which (for most cases) contains an SYS named entity, whereas the beginning of a *Prefix* and *Suffix* is usually marked by a 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.

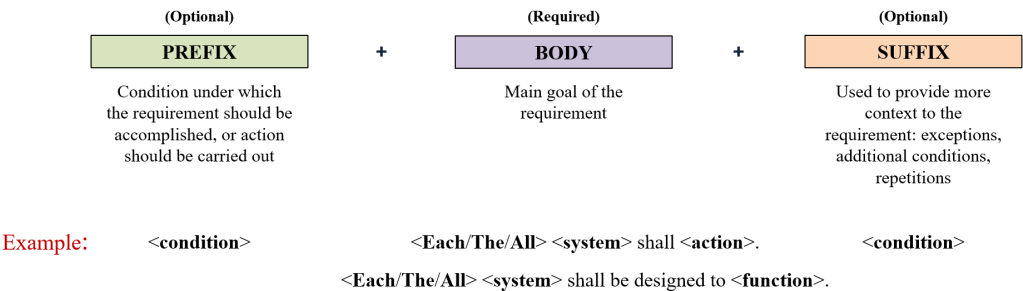


Figure 19. The general textual pattern observed in requirements was <Prefix> + <Body> + <Suffix> out of which Prefix and Suffix are optional and can be used to provide more context about the requirement. The different variations of the requirement Body, Prefix, and Suffix are shown as well [53].

Table 5 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.

Below, the methodology employed to identify patterns in requirements for the purpose of creating boilerplates is discussed, including the details regarding how this observed general pattern (Figure 19) was tailored to different types of requirements.

Table 5. Different elements of an aerospace requirement. The blue text highlights a specific element of interest in a requirement [53].

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.
<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 case flight controls, passengers in 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 certain characteristic of a said system; Example: The airplane's available gradient of climb shall not be less than 1.2 percent for two-engine airplane at each point along the takeoff path, starting at the point at which the airplane reaches 400 feet above the takeoff surface.

Methodology for the Identification of Boilerplate Templates

The requirements are first classified using the `aeroBERT-Classifier`. The patterns in the sentence chunk and NE tags are then examined for each of the requirements and the identified patterns are used for boilerplate template creation. Based on these tags, it was observed that a requirement with a `<condition>` in the beginning usually starts with a PP or SBAR. Requirements with a condition, in the beginning, were however rare in the dataset used for this work. The initial `<condition>` is almost always followed by a 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 VP which contains the term “shall [variations]”. These “[variations]” (shall be designed, shall be protected, shall have, shall be capable of, shall maintain, etc.) were crucial for the identification of different types of boilerplates since they provided information regarding the action that the `<system>` should be performing (to protect, to maintain, etc.). The VP is followed by a SBAR or PP but this is optional. The SBAR/PP is succeeded by a NP or ADJP and can contain either a `<functional attribute>`, `<state>`, `<design attribute>`, `<user>`, or a `<sub-system/system>`, depending on the type/sub-type of a requirement. This usually 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 (can contain VAL named entity), resources (RES), and context.

The process of categorizing requirements, chunking, performing NER, and identifying requirement elements is iterated for all the requirements. The elements are recognized by combining information obtained from the sentence chunker and `aeroBERT-NER`. The requirements are subsequently aggregated into groups based on the sequence/order of identified elements to form boilerplate structures. Optional elements are included in the structures to cover any variations that may occur, allowing more requirements to be accommodated under a single boilerplate.

The flowchart in Figure 20 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, patterns observed across multiple requirements need to be identified.

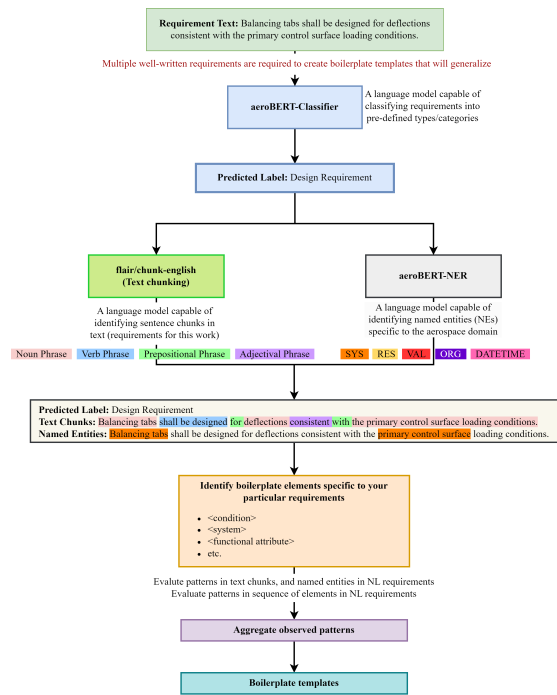


Figure 20. Flowchart showcasing the creation of boilerplate templates using three language models, namely, `aeroBERT-Classifier` [24], `aeroBERT-NER` [23], and `flair/chunk-english`. A zoomed-in version of the figure can be found [here](#) and more context can be found in [53].

5. Results

A two-pronged approach was employed for the standardization of requirements, namely, the creation of a requirement table, and the identification of boilerplates. This section discusses the results obtained upon applying the aforementioned methodology to NL requirements to standardize them.

5.1. Requirement Table

Table 6 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-Classifer) 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.

Table 6. Requirement table containing columns extracted from NL requirements using language models. This table can be used to aid the creation of SysML requirement table [53].

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 to 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: 23-2500}} [Section	{SYS: [flight recorder, aircraft]}
3	pressurized airplanes	Pressurized airplanes with maximum operating altitude greater than 41000 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, 41000 feet]}	{SYS: [pressurized airplanes, pressurized cabin structure]}
4	fuel system	Each fuel system must be arranged so that any air which is introduced into the system will not result in power interruption for more than 20 seconds for reciprocating engines.	Performance	{VAL: [20 seconds]}	{SYS: [fuel system, reciprocating engines]}
5	structure	The structure must be able to support ultimate loads without failure for at least 3 seconds.	Performance	{VAL: [3 seconds]}	{SYS: [structure]}

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 and limits the manual way of populating the table. In addition, aeroBERT-NER, and aeroBERT-Classifer generalize well and are capable of identifying named entities and classifying requirements despite the noise and variations that can occur in NL requirements. 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.

5.2. Boilerplate Templates

The requirements were first classified into various types using the aeroBERT-Classifer. 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 7 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 7. Summary of boilerplate template identification task. Two, five, and three boilerplate templates were identified for design, functional, and performance requirements that were used for this study.

Requirement type	Count	Boilerplate Count	% of requirements covered
Design	149	2	~55%
Functional	100	5	63%
Performance	61	3	~58%

5.2.1. 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 21 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 a 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 21, the observed “[variations]” were *be designed*, *be designed and installed*, *installed*, *located*, and *protected*, respectively. The VP is followed by a SBAR or PP but this is optional. This is then followed by a 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.

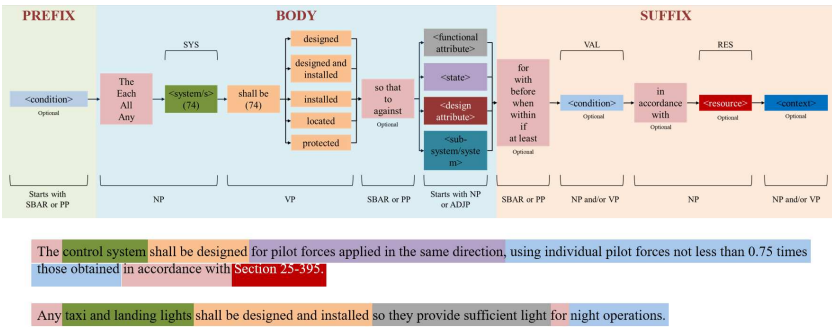


Figure 21. The schematics of the first boilerplate for **design requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 74 of the 149 design requirements (~50%) used for this study and is tailored toward requirements that mandate the way a <system> 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.

The second boilerplate for design requirements is shown in Figure 22. 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.

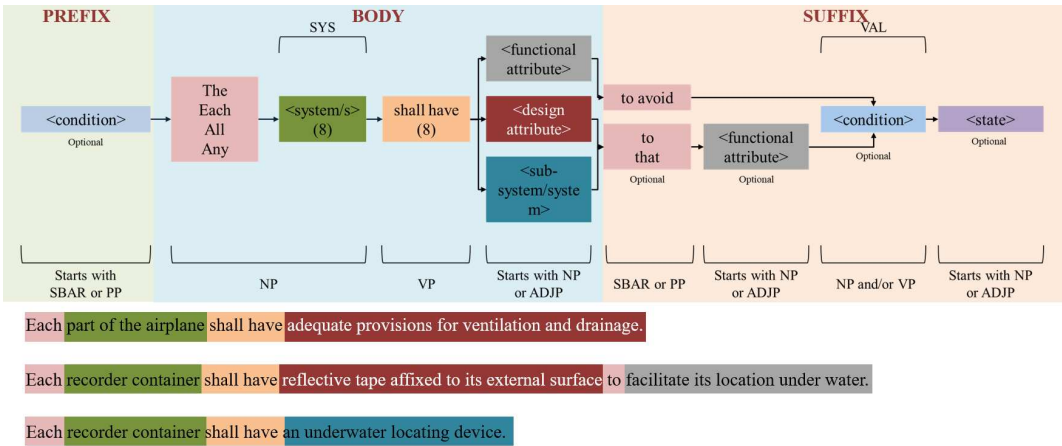


Figure 22. The schematics of the second boilerplate for **design requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 8 of the 149 design requirements (~5%) used for this study and focuses on requirements that mandate a <functional attribute>, <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.

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.

5.2.2. 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 23. 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.

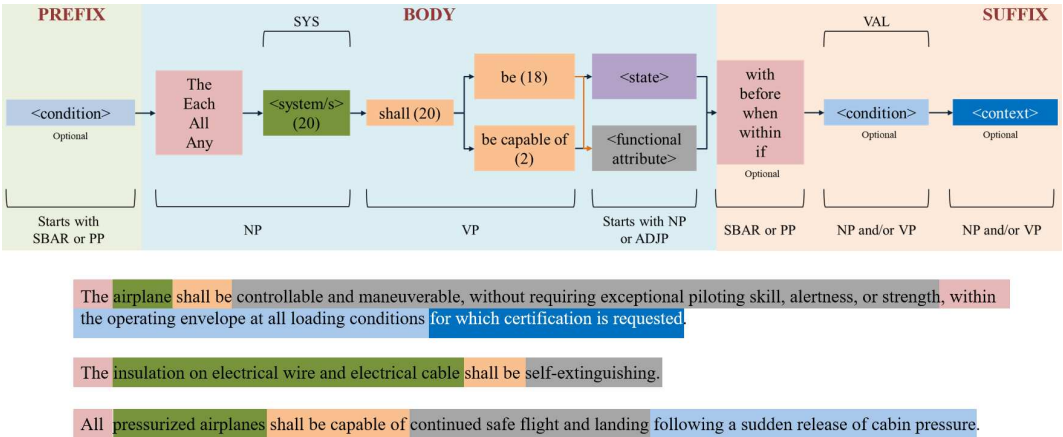


Figure 23. The schematics of the first boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 20 of the 100 functional requirements (20%) used for this study and is tailored toward requirements that describe the capability 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 24 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.

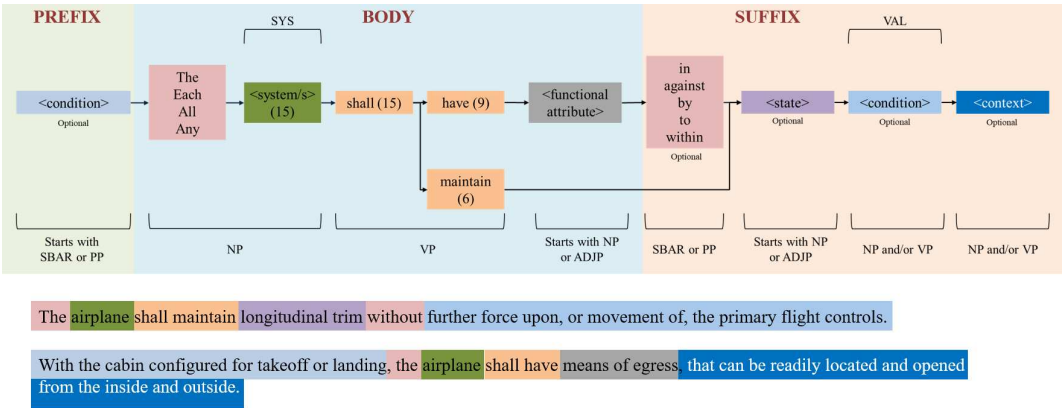


Figure 24. The schematics of the second boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 15 of the 100 functional requirements (15%) used for this study and is tailored toward requirements that require the <system> to have a certain <functional attribute> or maintain a particular <state>.

Figure 25 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.

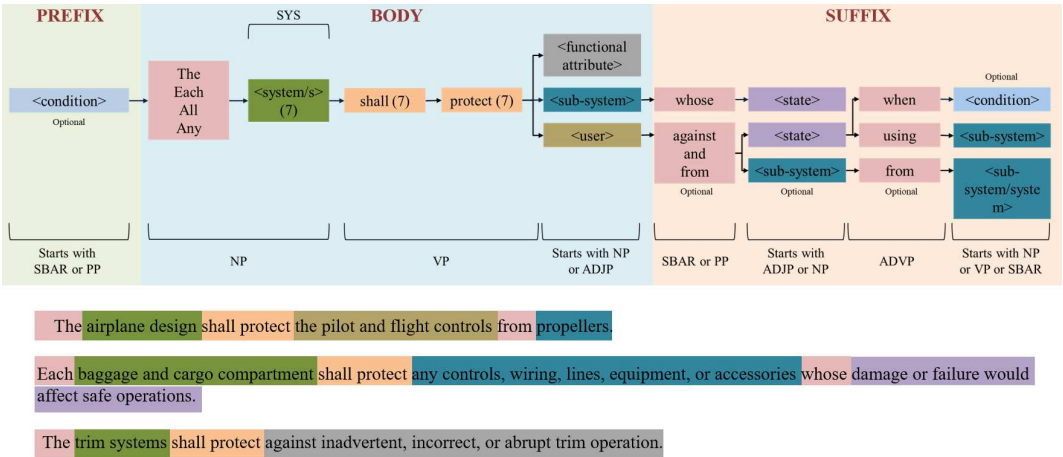


Figure 25. The schematics of the third boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 7 of the 100 functional requirements (7%) used for this study and is tailored toward requirements that require the <system> to protect another <sub-system/system> or <user> against a certain <state> or another <sub-system/system>.

Figure 26 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.

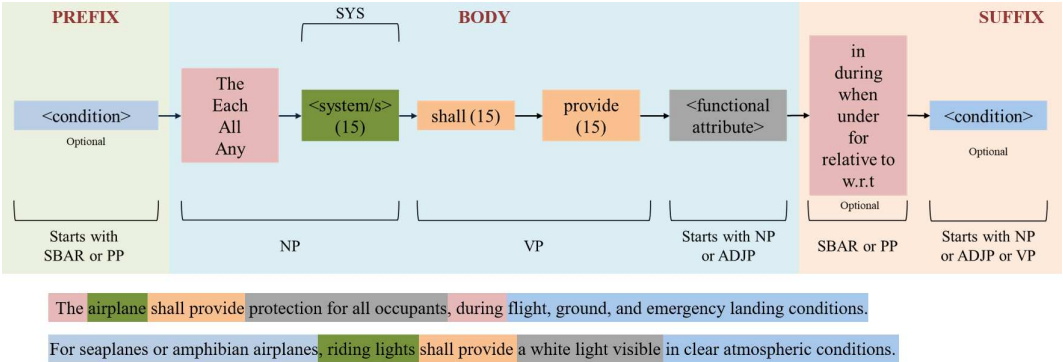


Figure 26. The schematics of the fourth boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 15 of the 100 functional requirements (15%) used for this study and is tailored toward requirements that require the <system> to provide a certain <functional attribute> given a certain <condition>.

Figure 27 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.

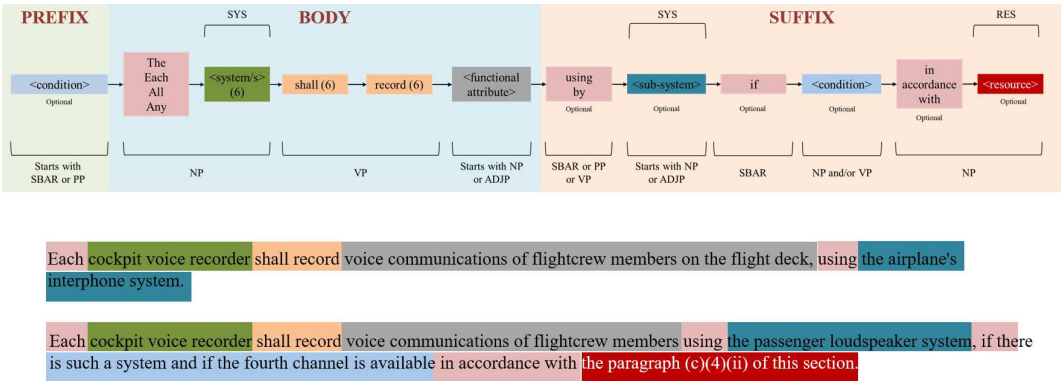


Figure 27. The schematics of the fifth boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 6 of the 100 design requirements (6%) used for this study and is specifically focused on requirements related to the *cockpit 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. But 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.

5.2.3. 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 28. 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.

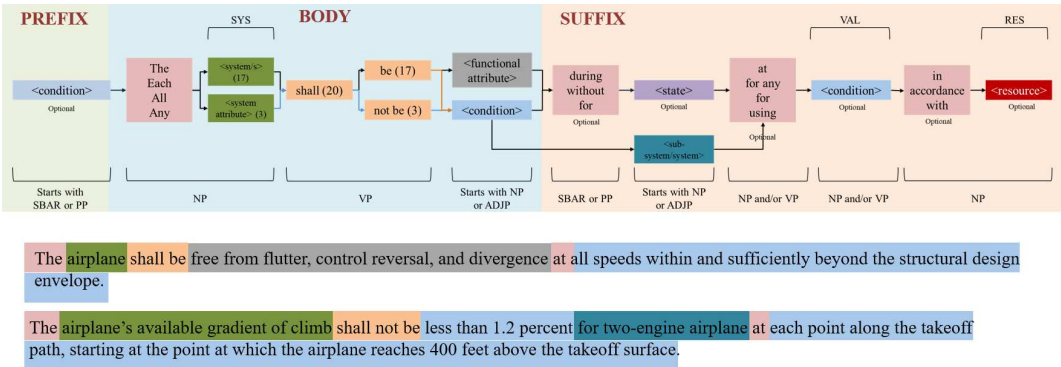


Figure 28. The schematics of the first boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 20 of the 61 performance requirements (~33%) used for this study. This particular boilerplate has the element <system attribute> 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 29 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>.

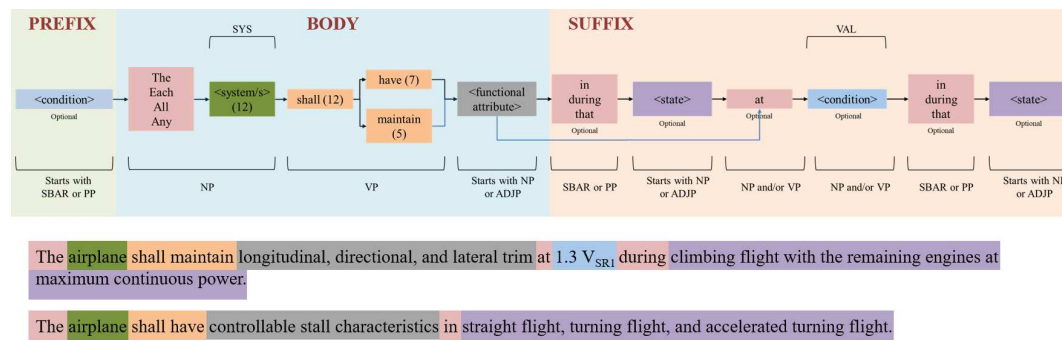


Figure 29. The schematics of the second boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 12 of the 61 performance requirements (~20%) used for this study. This boilerplate focuses on performance requirements that specify a <functional attribute> that a <system> should have or maintain given a certain <state> or <condition>.

Lastly, Figure 30 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>.

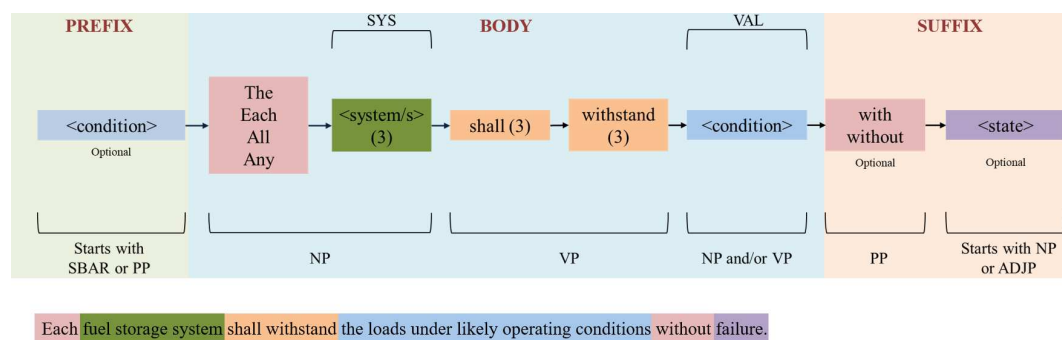


Figure 30. The schematics of the third boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 3 of the 61 performance requirements (~5%) used for this study and focuses on a <system> being able to *withstand* and certain <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. Table 8 illustrates a detailed breakdown of the coverage of the boilerplate templates developed in this study.

Table 8. This table presents the results of a coverage analysis on the boilerplate templates that were proposed. According to the analysis, 67 design, 37 functional, and 26 performance requirements did not conform to the boilerplate templates that were developed as part of this work.

Requirement type	Boilerplate #	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%)	

6. 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 improved 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. As such they are expected to contribute to the adoption of MBSE effectively. 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 applicable.

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.

Exploring the use of generative LMs like T5, the GPT family, etc., may 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.

7. Other Details

The models leveraged in this study to identify boilerplates were `aeroBERT-NER` [23], `aeroBERT-Classifier` [24], and `flair/chunk-english` [25]. While the specifics regarding the training of these models are not discussed in this paper, Figure 31 below is included to support practitioners in the creation of the aforementioned models.

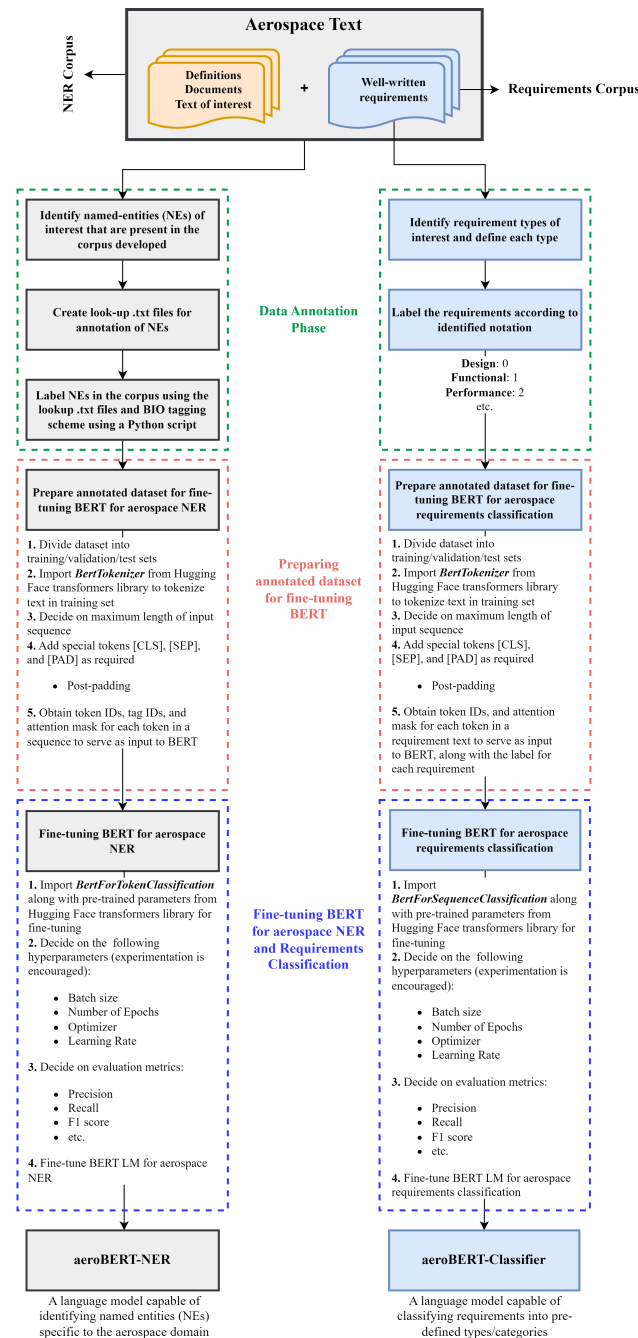


Figure 31. Practitioner's Guide to creation of aeroBERT-NER and aeroBERT-Classifier. A zoomed-in version of this figure can be found https://archanatikayatray19.github.io/Practitioners_Guide/ [53].

Author Contributions:

- **Archana Tikayat Ray:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing—original draft preparation, Writing—review and editing
- **Bjorn F. Cole:** Conceptualization, Writing—review and editing
- **Olivia J. Pinon Fischer:** Conceptualization, Writing—review and editing
- **Anirudh Prabhakara Bhat:** Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing—review and editing
- **Ryan T. White:** Writing—review and editing
- **Dimitri N. Mavris:** Writing—review and editing

Data Availability Statement:

- The annotated aerospace requirements dataset can be found on the Hugging Face platform. URL: <https://huggingface.co/datasets/archanatikayatray/aeroBERT-classification>.
- The annotated aerospace NER dataset can be found on the Hugging Face platform. URL: <https://huggingface.co/datasets/archanatikayatray/aeroBERT-NER>.

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

References

1. Guide to the Systems Engineering Body of Knowledge; BKCASE Editorial Board, INCOSE, 2020; p. 945.
2. INCOSE. In cose Infrastructure Working Group Charter; pp. 3–5.
3. NASA. Appendix C: How to Write a Good Requirement; pp. 115–119.
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; Springer, 2008; pp. 123–128.
5. NASA. 2.1 The Common Technical Processes and the SE Engine. *J. Object Technol.* 4.
6. Nuseibeh, B.; Easterbrook, S. Requirements Engineering: A Roadmap. In Proceedings of the Conference on The Future of Software Engineering; Association for Computing Machinery: New York, NY, USA, 2000; ICSE '00; pp. 35–46. <https://doi.org/10.1145/336512.336523>.
7. Firesmith, D. Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *J. Object Technol.* 2007, 6, 17–33.
8. Haskins, B.; Stecklein, J.; Dick, B.; Moroney, G.; Lovell, R.; Dabney, J. 8.4. 2 error cost escalation through the project life cycle. In Proceedings of the INCOSE International Symposium; Wiley Online Library, 2004; Volume 14, pp. 1723–1737.
9. Bell, T.E.; Thayer, T.A. Software requirements: Are they really a problem? In Proceedings of the 2nd International Conference on Software Engineering, 1976; pp. 61–68.
10. Estefan, J.A.; others. Survey of model-based systems engineering (MBSE) methodologies. *In cose MBSE Focus Group* 2007, 25, 1–12.
11. Jacobson, L.; Booch, J.R.G. The unified modeling language reference manual. 2021.
12. Ballard, M.; Peak, R.; Cimentalay, S.; Mavris, D.N. Bidirectional Text-to-Model Element Requirement Transformation. *IEEE Aerosp. Conf.* 2020, 1–14.
13. Lemazurier, L.; Chapurlat, V.; Grossetête, A. An MBSE approach to pass from requirements to functional architecture. *IFAC-PapersOnLine* 2017, 50, 7260–7265.
14. Needs, Requirements, Verification, Validation Lifecycle Manual; BKCASE Editorial Board, INCOSE, 2022; p. 457.
15. Wheatcraft, L.; Ryan, M.; Llorens, J.; Dick, J. The Need for an Information-based Approach for Requirement Development and Management. In Proceedings of the INCOSE International Symposium; Wiley Online Library, 2019; Volume 29, pp. 1140–1157.
16. Requirement Table. Available online: <https://docs.nomagic.com/display/SYSMLP182/Requirement+Table>. (accessed on 10 February 2023).

17. Modeling Requirements with SysML. Available online: <https://re-magazine.ireb.org/articles/modeling-requirements-with-sysml>. (accessed on 10 February 2023).
18. 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*, 1–3.
19. Arora, C.; Sabetzadeh, M.; Briand, L.; Zimmer, F. Automated checking of conformance to requirements templates using natural language processing. *IEEE Trans. Softw. Eng.* **2015**, *41*, 944–968.
20. Arora, C.; Sabetzadeh, M.; Briand, L.C.; Zimmer, F. Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns. In Proceedings of the 2014 IEEE 4th International Workshop on Requirements Patterns (RePa) 2014; pp. 1–8.
21. Rupp, C. *Requirements-Engineering und-Management: Professionelle, iterative Anforderungsanalyse für die Praxis*; Hanser Verlag, 2007.
22. 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, 2009; pp. 317–322.
23. Ray, A.T.; 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. <https://doi.org/10.2514/6.2023-2583>.
24. Ray, A.T.; Cole, B.F.; Pinon-Fischer, O.J.; White, R.T.; Mavris, D.N., aeroBERT-Classifer: Classification of Aerospace Requirements using BERT. *Preprints*. <https://doi.org/10.20944/preprints202302.0077.v1>.
25. Akbik, A.; Blythe, D.; Vollgraf, R. Contextual String Embeddings for Sequence Labeling. In Proceedings of the COLING 2018, 27th International Conference on Computational Linguistics, 2018; pp. 1638–1649.
26. Ferrari, A.; Dell’Orletta, F.; Esuli, A.; Gervasi, V.; Gnesi, S. Natural Language Requirements Processing: A 4D Vision. *IEEE Softw.* **2017**, *34*, 28–35.
27. Abbott, R.J.; Moorhead, D. Software requirements and specifications: A survey of needs and languages. *J. Syst. Softw.* **1981**, *2*, 297–316.
28. Manning, C.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S.; McClosky, D. The Stanford CoreNLP Natural Language Processing Toolkit. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations; Association for Computational Linguistics: Baltimore, Maryland, 2014; pp. 55–60. <https://doi.org/10.3115/v1/P14-5010>.
29. Natural Language Toolkit. Available online: <https://www.nltk.org/> (accessed on 10 January 2023).
30. spaCy. Available online: <https://spacy.io/> (accessed on 10 January 2023).
31. 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*.
32. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**. arXiv:1810.04805.
33. Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; Zettlemoyer, L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *CoRR* **2019**, *abs/1910.13461*.
34. Sebastiani, F. Machine Learning in Automated Text Categorization. *ACM Comput. Surv.* **2002**, *34*, 1–47. <https://doi.org/10.1145/505282.505283>.
35. 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.
36. 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.
37. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J. Classification of Natural Language Processing Techniques for Requirements Engineering, 2022. <https://doi.org/10.48550/ARXIV.2204.04282>.
38. 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, 2000; Volume 13.
39. 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.
40. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; others. Language models are unsupervised multitask learners. *OpenAI blog* **2019**, *1*, 9.

41. 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., 2020; Volume 33, pp. 1877–1901.
42. Hugging Face. Available online: <https://huggingface.co/> (accessed on 10 January 2023).
43. 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), 2019; pp. 142–152. <https://doi.org/10.1109/RE.2019.00025>.
44. 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. <https://doi.org/10.1109/access.2022.3182372>.
45. Jurafsky, D.; Martin, J.H. Speech and language processing (draft). 2021.
46. Syntax. <https://webpace.ship.edu/cgboer/syntax.html>. (accessed on 21 February 2023).
47. Riesener, M.; Dölle, C.; Becker, A.; Gorbacheva, S.; Rebertsch, E.; Schuh, G. Application of natural language processing for systematic requirement management in model-based systems engineering. *INCOSE International Symposium* **2021**, *31*, 806–815. <https://doi.org/10.1002/j.2334-5837.2021.00871.x>. Available online: <https://incose.onlinelibrary.wiley.com/doi/pdf/10.1002/j.2334-5837.2021.00871.x>.
48. Rajan, A.; Wahl, T. *CESAR: Cost-efficient methods and processes for safety-relevant embedded systems*; Number 978-3709113868, Springer, 2013.
49. Ruiz, A.; Sabetzadeh, M.; Panaroni, P.; others. Challenges for an open and evolutionary approach to safety assurance and certification of safety-critical systems. In Proceedings of the 2011 First International Workshop on Software Certification. IEEE, 2011, pp. 1–6.
50. 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. IEEE, 2013, pp. 35–44.
51. Arora, C.; Sabetzadeh, M.; Briand, L.; Zimmer, F. Automated extraction and clustering of requirements glossary terms. *IEEE Trans. Softw. Eng.* **2016**, *43*, 918–945.
52. FAA. Title 14 Code of Federal Regulations; FAA: 2023.
53. Tikayat Ray, A. *Standardization of Engineering Requirements Using Large Language Models*; Georgia Institute of Technology: 2023.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.