12-2024

Computational Representation, Analysis and Verification of Requirements in Engineering Design and Systems Engineering

Chandan Kumar Sahu
csahu@clemson.edu

Computational Representation, Analysis and Verification of Requirements in Engineering Design and Systems Engineering

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Automotive Engineering

---

by
Chandan Kumar Sahu
December 2024

Accepted by:
Dr. Rahul Rai, Committee Chair
Dr. Gregory Mocko
Dr. Cameron Turner
Dr. Venkat Krovi

Abstract

Systems are developed to satisfy a set of requirements derived from the stakeholders' needs. These requirements define the problem space for which the system is developed as a feasible solution. The system design process begins with eliciting the requirements the system shall meet and concludes with validating whether the created system satisfies those requirements and stakeholders' needs. The requirements engineering (RE) process involves key activities such as elicitation, representation, analysis, documentation, verification, and validation.

Several challenges in RE impede system development. Notably, the creative nature of the elicitation process, proprietary concerns, the iterative nature of the requirement definition process and the difficulty in acquiring well-formed sets of requirements limit the quality and comprehensiveness of elicited requirements. Requirements, predominantly elicited in natural language (NL), suffer from the inherent ambiguity and imprecision of NL. This imprecision complicates the representation and analysis of requirements, making it essential to correct ill-formed requirements as well-formed ones. The lack of consensus among RE practitioners regarding the essential characteristics of a requirement further hinders the verification process. There is currently no standardized metric to compare the quality of requirements. Moreover, the rich but varied semantics of NL complicate the task of providing a formal structure to requirements without human intervention. A formal structure is crucial for documenting requirements to serve as a binding agreement among the stakeholders and the system.

The research in this dissertation aims to address the aforementioned challenges in RE. We begin by reusing requirements from open-source System Requirement Specification Documents (SyRSDs). A formal method is devised to represent the requirements in each SyRSD as a tree. The leaf nodes represent individual requirements, while branch nodes represent components, functions, and other artifacts outlined in the SyRSD. Edges capture the decomposition of the parent node's role

among its child nodes. The tree reflects the requirement allocation process. We release a requirement dataset to support the deployment of natural language processing (NLP), graph theory, and machine learning (ML) techniques for RE. The dataset has three inter-convertible forms: *ReqList* - a *list* of 12,701 textual *req*uirements; *ReqNet* - a *net*work of *req*uirements with 17,375 nodes; and *ReqSim* - 10933 pairs of *req*uirements annotated with their *sim*ilarity scores.

We devise a graph-theoretic method to estimate the semantic similarity between pairs of requirements in SyRSDs. Then we create *Req-sBERT* by fine-tuning the Sentence-BERT model, modified with a six-layer deep neural network (DNN), on the *ReqSim* dataset to produce requirement embeddings. These embeddings can estimate the semantic similarity between any pairs of requirements irrespective of whether associated SyRSDs are available or not. These embeddings align with human judgment. Using the embeddings, we represent the requirements of a system as a weighted semantic network, where nodes are requirements and edge weights represent their similarity. The embeddings can detect the evolution of requirements from functional to non-functional forms and enable the extraction of a semantically sensible hierarchical tree to document requirements.

Furthermore, we propose a requirements verification framework to evaluate the quality of individual requirements. This framework leverages rule-based NLP techniques to evaluate each requirement against the 42 rules prescribed by the `INCOSE Guide to Writing Requirements` for achieving the nine characteristics of quality specified in the ISO/IEC/IEEE 29148:2018(E) standard. We introduce the well-formed Requirement Quality (wRQ) index, a scalar metric within $[0, 1]$, to facilitate the comparison of requirement quality.

This dissertation focuses on accelerating the RE of a system by improving (1) the requirement elicitation and analysis process by offering an expandable RE dataset in three inter-convertible forms, (2) the representation by offering a tree and a semantic network representation of requirements, (3) the analysis by estimating the semantic similarity among requirements, detecting the evolution of requirements from functional to non-functional forms, and enabling the comparison of quality of requirements, (4) the documentation process by hierarchical clustering and (5) the verification process by developing an NLP pipeline to evaluate the conformance to the ISO/IEC/IEEE 29148:2018 (E) standard.

Dedicated

to

my family and friends

whose belief in my abilities exceeds my own

&

whose happiness in my success surpasses my own

...

Acknowledgments

I embarked upon this research expedition, hoping it would be a hike, if not a walk in the park. What unfolded was a trek for survival, demanding resilience, perseverance, and unwavering determination. Completing this PhD dissertation has been a journey filled with arduous challenges sprinkled with sporadic success. Overall, the PhD journey has been a humbling experience. It would have been impossible to survive without the personal and professional support of many individuals.

First and foremost, I extend my heartfelt thanks to *Dr. Rahul Rai*, my advisor, for accepting to supervise my PhD. His unwavering support, guidance, and encouragement have been instrumental throughout this endeavor. Dr. Rai not only provided me with invaluable opportunities but also granted me the freedom and autonomy to explore and delve into my research without fear. His mentorship has been transformative, and I am immensely grateful for his dedication and belief in my abilities. I also thank the Army Research Center for funding my research work through the VIPR-GS center at Clemson University and Dr. Rai for facilitating. This work was supported by the Automotive Research Center (ARC), a US Army Center of Excellence for modeling and simulation of ground vehicles, under Cooperative Agreement W56HZV-19-2-0001 with the US Army DEVCOM Ground Vehicle Systems Center (GVSC).

I am indebted to *Dr. Gregory Mocko*, *Dr. Cameron Turner*, and *Dr. Venkat Krovi* for their invaluable contributions as members of my research committee. Their insightful feedback and constructive criticism have greatly enriched the quality of my work, and I am grateful for their guidance and support. Thank you *Dr. Gregory Mocko* for reviewing my dissertation. I am really thankful to *Dr. Margaret Wiecek* for her guidance and for going out of her way to accommodate my requests. I learned a lot from her. Her contributions greatly enriched my work.

To my dear friends *Raj*, *Vinayak*, and *Sai*, your support and companionship have been my lifeline throughout this journey. I am deeply grateful to you. I would also like to express my

gratitude to *Zhibo* and *Dustin* for not only being friends but also for assuming the role of mentors when needed. Your guidance and wisdom have been invaluable, and I am fortunate to have had your support. *Mohan*, you have been a friend akin to family and a mentor whose advice I have cherished deeply. I extend my appreciation to all the members of the lab for their camaraderie, which has made the research environment enjoyable.

To *Chris and Kathy*, and *Krishna and Vasudha*, thank you for making me a part of your family. I am indebted to you for your kindness. I cannot convey my gratitude with words. *Ksheera and Rudhra*, thank you for offering the essential respite from work and for brightening my days with your presence.

Lastly, I am indebted to *Deepak* and *Swarup* for their help. Without their help, I would have failed miserably at fulfilling the responsibilities of being a son to my parents and a brother to my siblings. Their timely help saved me from my worries and offered me the peace of mind to focus on my research.

I stand on the shoulders of my family, and for that, I am deeply grateful. Thank you for being there, even when I couldn't be. To me, expressing gratitude to my family feels like thanking my own self — I am them, and they are me. No words can truly capture the depth of my appreciation; anything I say will always fall short.

This acknowledgment would be incomplete without extending a sincere apology to each of my friends and relatives whose calls I failed to answer, messages I neglected to respond to, and meet-ups I regrettably could not attend. I accept full responsibility for growing apart, and I deeply apologize for every inconvenience or disappointment my actions have caused.

- Chandan

Finished work is better than the best unfinished work!!!

Table of Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Dedication . . . . . **iv**

Acknowledgments . . . . . **v**

List of Tables . . . . . **x**

List of Figures . . . . . **xi**

1 Introduction **1**
1.1 Engineering Design 1
1.2 Systems Engineering 3
1.3 Requirements in Engineering Design and Systems Engineering 5
1.4 What is a Requirement? 7
1.5 Notional Requirements of a System 9
1.6 Requirements Engineering 14
1.7 Research Questions and Contributions 17
1.8 How to Read the Dissertation? 23

2 Literature Review **24**
2.1 Requirement Datasets 24
2.2 Representation of Requirements 25
2.3 Semantics of Requirements 27
2.4 Requirement Verification 28

3 *ReqTree*: Requirement Allocation from the *Tree* Structure of *Req*uirements in Requirement Specifications **30**
3.1 Introduction 30
3.2 Literature Review 33
3.3 Methodology 35
3.4 Proof: Requirement Graph is a Tree 40
3.5 Results and Discussion 44
3.6 Conclusion 50

4 ReqList, ReqNet and ReqSim: Datasets of Requirements **52**
4.1 Introduction 52
4.2 Data sources 54
4.3 Methodology 57
4.4 Results 64

4.5 Discussion . . . 67
4.6 Conclusion . . . 72

5 A Semantic Network of Requirements from the Embeddings of Req-sBERT . . 73
5.1 Introduction . . . 73
5.2 Literature Review . . . 75
5.3 Methodology . . . 77
5.4 Results and Discussion . . . 84
5.5 Conclusion . . . 93

6 Requirement Verification: Evaluating the Quality of Requirements for ISO 29148:2018 . . . 95
6.1 Introduction . . . 95
6.2 Example Case of Requirement Verification . . . 97
6.3 Literature Review . . . 98
6.4 Industrial Standards . . . 99
6.5 Requirement Quality Framework . . . 100
6.6 Well-Formed Requirement Quality Index . . . 107
6.7 Results and Discussion . . . 108
6.8 Conclusion . . . 111

7 Research Impact . . . 112
7.1 Requirement Allocation from the *Tree* of *Req*uirements . . . 112
7.2 ReqList, ReqNet and ReqSim: Datasets of Requirements . . . 113
7.3 Semantic Network of Requirements . . . 113
7.4 Requirement Verification: Evaluating the Quality of Requirements . . . 114
7.5 Broader Impact on Engineering Design and Systems Engineering . . . 115

8 Future Work . . . 117
8.1 Semantics of Requirements . . . 117
8.2 Change Propagation of Requirements . . . 118
8.3 Verification of Requirements . . . 119

9 Conclusion . . . 120

10 Knowledge Dissemination . . . 123
10.1 Journals . . . 123
10.2 Conferences . . . 124
10.3 Theses . . . 124

Appendices . . . 125
A INCOSE Rule-to-Characteristics Mapping Matrix for the characteristics of individual requirements annotated with the type of implementation . . . 126

Bibliography . . . 129

List of Tables

1.1 Research Questions (RQ) and Research Contributions (RC) 19

3.1 Decomposition criteria of the requirement allocation process in various RSDs 48

4.1 Pairs of requirements with their similarity score 71

5.1 Comparison of performance of different models on the test dataset. 85
5.2 Highly similar requirement pairs 88
5.3 Least similar requirement Pairs 89
5.4 Requirements in the major cluster. Note: Refer Table 5.2 for the descriptions of the requirements 5-6, 13-14, 27-28, 47 and 50-53 which are also in the cluster. 90
5.5 Evolution of similarity from FRs to NFRs 91

6.1 Quality of an example requirement and rewriting it for improvement 97
6.2 Characteristics of a well formed requirement [166, 165] 100
6.3 Patterns for requirements 105
6.4 Well-formed requirement quality of a few sample requirements including their imperfect characteristics and the rules violated by the requirements 109

1 Rules to characteristics (RC) mapping matrix [137]. The *characteristics* of individual requirements are necessary (C1), appropriate (C2), unambiguous (C3), complete (C4), singular (C5), feasible (C6), verifiable (C7), correct (C8), and conforming (C9). The *type* of implementation is split into token-based (T), span-based (s), syntactic structure-based (Sy), sentence-based (S) and others (O) 126

List of Figures

1.1 A notional requirement depicting the capability, condition and constraint 8
1.2 A notional set of requirements of a vehicle system 10
1.3 Requirement allocation 11
1.4 Graph representation of a set of requirements 12
1.5 Prioritization of requirements as an example of requirement analysis 12
1.6 Documentation, verification and validation of requirements 14
1.7 Challenges in requirement engineering activities and our research contributions . . . 15

3.1 The outline of a requirement specification prescribed by ISO/IEC/IEEE 29148:2018 (E) [166] modified by a preprocessed excerpt from the system requirement specification document of the New York City's Connected Vehicle Pilot Deployment (NYC CVPD) program [59] 32
3.2 (a) The path created from the requirement R_i with clause number $c_i = c_1^i \cdot c_2^i \cdot ... c_{r_i-1}^i \cdot c_{r_i}^i \cdot c_{r_i+1}^i ... c_{k_i}^i$. (b) Creation of *ReqTree* from the clause numbers 37
3.3 The tree structure extracted from the RSD of the mobile surveillance system 46
3.4 The tree structure extracted from the RSD of New York City's Connected Vehicle Pilot Deployment 47

4.1 Computational framework to create *ReqList*, *ReqTree*, *ReqNet* and *ReqSim* from SyRSDs 54
4.2 Distribution of requirements in *ReqList*s (Shows requirements from 68 RSDs only. Excluded 18 RSDs with < 40 requirements for legibility.) 56
4.3 Illustration of the tree creation process using a set of sample requirements. 58
4.4 Comparison of unweighted distance from 3.2.1 to 1.2.1.1.1 (within the a *ReqTree*) with 3.2.1 to 1 (across *ReqTree*s) in a notional *ReqTree* 60
4.5 *ReqNet* - A network of requirements 65
4.6 Histogram plot of fine-grained distances in the bottom-up and top-down approaches 66
4.7 Visualizing *ReqNet* as a radial tree 69

5.1 Neural language model 77
5.2 The architecture of *Req-sBERT* 79
5.3 Bidirectional self-attention mechanism: Moving the succeeding tokens to the left of the *masked* token *permutes* (as in MPNet) the order of the tokens while still being bidirectional. 81
5.4 The semantic network of requirements of the Deep Orange 13 program with the node locations from t-SNE 86
5.5 The variation of semantic similarity as the number of semantically unrelated keywords in a sequence of tokens increases (in the DO13 requirements) 87
5.6 A hierarchical structure of the requirements to orginize them in a system requirement specification document 93

6.1 The NLP pipeline to verify the requirements by evaluating for the INCOSE Rules and determining the characteristics 101

6.2 The part-of-speech tags and the structure of a sentence in natural language applied to the requirement 'The security system shall detect firearms with 95% accuracy.' . . 102
6.3 The tree structure of the requirement 'The security system shall detect firearms with 95% accuracy.'. The coarse PoS tags and the fine PoS tags of the tokens and the dependencies among the tokens are also depicted. 103
6.4 The tree structure of a state-driven requirement from Table 6.3 showing that the condition is connected to the root verb with an 'adverbial clause modifier'. 106

Chapter 1

Introduction

1.1 Engineering Design

Engineering seeks to improve human life by addressing pressing challenges through innovative solutions. Engineers identify problems and develop products, systems, or services that provide practical resolutions. Design is a critical aspect of engineering, involving the conceptual creation of products, systems, or software [31, 125, 98]. It integrates multiple domains to optimize objectives while accommodating conflicting constraints. For instance, the design of a medical device must consider patient safety and hygiene, operational precision, power supply, and ergonomic requirements [13]. This interdisciplinary approach ensures that the product meets technical, economic, and user-centric goals.

New functions, features, cost, resource limitations, improved ergonomics, longevity, sustainability drive the design of systems and products. The design process spans the entire lifecycle of a product, beginning with market forces and ending with the product's useful life. This lifecycle transforms raw materials into economically valuable products while addressing market and stakeholder needs [37]. Product planning is the initial stage, involving market surveys and stakeholder consultations to define whether a new product will be developed from scratch or adapted from existing ones. The planning phase establishes the requirements the product must satisfy, including constraints related to production and technology. Design processes are categorized into three main types: original design, adaptive design and variant design [125]. *Original designs* incorporates new solution principles, either through innovative combinations of existing methods or the development

of new technological principles. *Adaptive designs* modifies established solutions to meet new requirements, focusing on geometric, production, and material considerations. *Variant design* varies the size and arrangement of components within predefined design limits.

1.1.1 Phases of Design

The design process remains consistent irrespective of the type of design. The design process involves *planning* to identify the requirements, *conceptualization* to search for solution principles, *embodying* to determine the layout, shapes and materials of all components, and *detailing* to finalize the production and operational details [125].

- **Product planning:**

  The product planning step focuses on the identification of the tasks that needs to be accomplished to develop the product. This step analyses the market needs and organizational capabilities, searches and selects product ideas, formulates product proposals, clarifies the tasks and lists the set of requirements. In essence, the product planning step collects information about the requirements to be fulfilled by the product and the applicable constraints. The subsequent phases, conceptual design and embodiment design, are based on the list of requirements produced by the planning phase.

- **Conceptual design:**

  The conceptual design phase develops solutions based on the requirements identified during planning. It abstracts essential problems, maps them to functions, identifies feasible working principles, and combines them into a conceptual solution. Preliminary evaluations of materials, shapes, structures, and technologies help narrow down viable principles, resulting in a set of variants. Variants failing to meet requirements are eliminated, while the remaining are assessed against defined criteria. Conceptual solutions are often represented as circuit diagrams, function structures, or flow charts. This phase is critical, as flaws in solution principles are difficult to address in later stages like embodiment or detail design.

- **Embodiment design:**

  The embodiment design phase transforms the concept into a complete structure encompassing the function, strength and economic viability. This phase results in a layout in line with

the technical and economic criteria. Each variant result in a layout which is evaluated comparatively with other layouts The layouts result in a higher information level with which the embodiment design phase is reiterated.

- **Detail design:**

  The detailed design phase determines the part arrangement, forms, dimensions, materials, manufacturing details, costs, production documents. The missing details identified in this phase calls for revisiting the previous phases to fill in the missing details.

The four stages of design ensure that a product meets its requirements, but success also depends on entering the right market at the right price and time. Achieving this requires efficient task allocation, scheduling with tools like Gantt charts, and strict cost control to ensure profitability. Value analysis further optimizes the system by breaking functions into subfunctions, assigning them to components, minimizing costs, and eliminating redundancies. Organizations create value by addressing problems through products, systems, or services, balancing strategies like cost leadership via standardization and performance differentiation through niche offerings and specialized development [37]. □

The system design begins with the collection of system-related information through market and stakeholder analysis. This process, essentially a problem analysis, concludes with a set of requirements that the product or system must satisfy. The goal of this step is to acquire a clear, formal definition of the problem and the system's requirements [98, 76]. These requirements establish the criteria for evaluating solution variants to identify the optimum solution. In systems engineering, these steps are referred to as the lifecycle phases of the system.

1.2 Systems Engineering

The `ISO/IEC/IEEE 15288:2023 Systems and software engineering - System life cycle processes` defines a system as an arrangement of parts or elements that together exhibit a stated behavior [2]. A system may also be referred to as a product or service, depending on its nature. The complete system includes everything needed for its creation and operatio. A system element is a discrete part of the system that can be implemented to fulfill specified requirements. Systems engineering is a transdisciplinary and integrative approach that enables the successful realization,

use, and retirement of engineered systems by applying systems principles, concepts, and scientific, technological, and management methods [2, 77]. Systems engineers transform stakeholders' needs, expectations, and constraints into solutions by conceiving, designing, and developing appropriate products.

1.2.1 Lifecycle of a System

The lifecycle of a system, product, or service encompasses its evolution from conception to retirement [2]. A lifecycle model provides a framework of processes and activities concerned with the system's lifecycle, organized into stages that span conception, development, production, utilization, support, and retirement [76].

- **Concept:**

  The concept stage identifies the need for a new mission or business capability and analyzes market conditions, economics, technological readiness, and organizational capabilities. It uses surveys and comparative analyses to define the problem space, characterize the solution space, and determine mission/business requirements and stakeholder needs. Cost, performance, and schedule are estimated, and risks are assessed. Outputs include stakeholder requirements, baseline system requirements, preliminary concept artifacts, design solutions, feasibility assessments, and architectural solutions. Decisions made at this critical stage shape future possibilities, which become increasingly difficult to change later.

- **Development:**

  The development stage defines a system that meets stakeholder needs and requirements while being producible, usable, supportable, and eventually retireable. The goal of this stage is to develop a baseline system along with its architecture, design, and documentation. It produces prototypes and plans for integration, verification, validation, risk management, personnel training, cost, and scheduling.

- **Production:**

  The production stage transforms the development stage's baseline into the actual system. At this stage, the system, product, or service becomes a reality and is prepared for subsequent stages.

- **Utilization:**

  The utilization stage begins with the system's deployment in its intended environment. During this stage, the product is continually modified to address defects, enhance capabilities, and extend its service life.

- **Support:**

  The support stage overlaps with utilization and involves activities that detect defects, deficiencies, and failure modes. These activities enable modifications to reduce costs, improve performance, and extend the system's service life. The support stage ends when the system reaches the end of its useful life.

- **Retirement:**

  In the retirement stage, the system and its services are decommissioned and removed from operation. This stage focuses on the disposal, recycling, or reuse of the system or its constituent elements.

1.3 Requirements in Engineering Design and Systems Engineering

The requirements define the problem space for which the product, system or service is developed as a solution. The requirements are the focus of different domains depending on the nature of the product, system or service under consideration. The engineering design domain focuses on the role of requirements in product development. The systems engineering domain focuses on the role of requirements in system development. The software engineering domain focuses on the requirements for developing software services and products/systems. However, the convergence between the domains is so much that NSF has the same program dedicated to two of them, `Engineering Design and Systems Engineering (EDSE)`[1]. With the dominance of intangible software products and services, the focus of the `software engineering` domain on requirements outweighs the focus on engineering design and systems engineering. The role of requirements in all these domains created a domain for requirements itself, `requirements engineering`.

[1] `https://new.nsf.gov/funding/opportunities/edse-engineering-design-systems-engineering`

In engineering design, requirements provide a clear direction for product development, defining the constraints and objectives that guide the design process. They ensure that designs meet user needs, technical specifications, and regulatory standards. The planning and task clarification phase collects information about the requirements of the product and applicable constraints. The planning phase results in a list of requirements that serve as the basis of the subsequent phases. The conceptual design phase determines the solution principle by abstracting essential problems, establishing function structures, identifying working principles and finally integrating those principles to create the concept of the solution.

The systems engineering uses a V-model to meet the stakeholders' needs by creating the final product [139]. It transforms the stakeholders' needs into stakeholders' requirements. The high-level requirements of the stakeholders are recursively decomposed into system requirements, subsystem and component requirements with lower levels of abstraction. The requirements at the lowest levels can be allocated to functions, which are, in turn, allocated to systems for which a buy/build/code/reuse decision can be made. Then, these simpler systems and components are recursively assembled and integrated to create the final system, which will be delivered to the stakeholders.

In engineering design, requirements are the result of the planning phase and form the basis of the conceptual design phase. In systems engineering, the requirements are the focus of the concept and design stages before the system gets into the development and production stages. Well-structured requirements help designers minimize costly redesigns and errors, ensuring that the final product is both functional and efficient [124]. Requirements also serve as a communication tool between stakeholders, bridging the gap between customer expectations and technical feasibility [173]. In systems engineering, requirements define the expected system behaviors and interactions among subsystems, ensuring that all components work together to achieve the overall system goals. Systems engineering relies on requirements to validate that each stage of development meets predefined standards and that the system functions effectively in real-world environments [107]. Both domains rely on requirements engineering to translate stakeholder needs into viable solutions. The common tasks – such as elicitation, specification, analysis, and validation – ensure that requirements are traceable, clear, and adaptable throughout the project lifecycle [20, 166, 186]. Requirements engineering facilitates the alignment of project goals across both domains, focusing on delivering reliable, cost-effective solutions that satisfy stakeholder demands [90, 38]. Ultimately, both engineering design

and systems engineering share the objective of ensuring that requirements drive the development of products and systems that meet performance and operational goals.

The requirements of a design define the problem space for which the system is developed as a solution [179, 7, 18, 91, 146]. A well-defined set of requirements reduces the risks of developing sub-optimal systems or systems with unnecessary or over-specified requirements [73, 185]. As systems are developed to satisfy stakeholders' needs, requirements formalize these needs and serve as a binding agreement among all stakeholders. The requirement definition process transforms the informal needs into formal design input requirements that define the system architecture, baseline the requirements, and flow down the requirements into a design solution [186]. Poorly defined or managed requirements increase the risk of system failure, schedule delays, and budget overruns [187, 168]. Studies show that errors originating in the requirements phase account for more than 50% of all system defects, with the cost to fix these defects growing exponentially if addressed in later phases such as design, code, or operations [16, 159]. The 2023 report from the Office of Inspector General concerning NASA's 'Top Management and Performance Challenges' underscored the agency's difficulties in overseeing its contractors, attributing the challenges primarily to inadequately defined requirements [168]. High-quality requirements are essential to prevent scope creep, ensure clear communication, and secure stakeholder involvement [169]. Systems designed for poorly elicited requirements are at the risk of pursuing unnecessary, ambiguous, or non-verifiable requirements, which can compromise their success. Therefore, well-formed and well-managed requirements are fundamental to achieving a successful system design that satisfies stakeholders' needs while controlling costs and timelines [158, 140].

1.4 What is a Requirement?

The *ISO/IEC/IEEE 29148:2018 [166] - Systems and software engineering — Life cycle processes — Requirements engineering*[2] [166] defines a requirement as a statement which translates or expresses a *need* and its associated *constraints* and *conditions*'. There are three key components in this definition. The *capability* is a function or a feature of the system that meets one or more needs of the stakeholders. The standard defines a *condition* as 'a measurable qualitative or quantitative attribute that is stipulated for a requirement, and that indicates a circumstance or event under

[2]The keyword 'standard' used anywhere in the document refers to the ISO/IEC/IEEE 29148:2018- Systems and software engineering — Life cycle processes — Requirements engineering standard.

which a requirement applies' (§3.1.6). A *constraint*, on the other hand, is 'an externally imposed limitation on the system, its design, implementation, or the process' (§3.1.7). In essence, the system is developed to achieve a capability under specific conditions and externally imposed constraints. An example illustrating these components is shown in Figure 1.1.

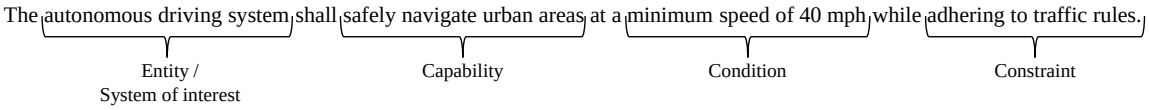


Figure 1.1: A notional requirement depicting the capability, condition and constraint

In this context, the entity or system of interest (SoI) refers to the system, subsystem, or component being developed at a specific level of abstraction. Requirements are defined as well-formed textual "shall" statements that clearly communicate in structured natural language what an entity must do to fulfill the intent of the needs from which they are derived [137]. While the needs reflect the stakeholders' viewpoints, the requirements are system-oriented statements. The standard also provides the following clarifying notes regarding requirements:

1. Requirements exist at different levels in the system structure.

   The requirement definition process creates a hierarchy of the requirements. The levels correspond to the hierarchical positions of the requirements in the requirement definition process and the requirement specifications (RS).

2. A requirement is an expression of one or more particular needs in a very specific, precise and unambiguous manner.

   This note emphasizes two aspects.

   First, as an expression of needs, a requirement shall trace back to the stakeholders' needs from which it originates. This aligns with the requirement validation process, which ensures that the right requirements are written. *Writing the right requirements* ensures that a system that satisfies its requirements meets the stakeholder's needs from which the requirements are derived.

   Second, the requirement must be expressed in a precise and unambiguous manner, which relates to the requirement verification process—ensuring that the requirements are written

right. *Writing the requirements right* ensures that the written requirements are free of defects and are of good quality.

3. A requirement always relates to a system, software, service, or other item of interest.
   This means that each requirement must correspond to a specific system of interest (SoI). Writing the requirements at the right level ensures that they align with the relevant SoI and supports the system requirements definition process.

In summary, it is essential to write the right requirements in the right way at the right levels of system abstraction. Part of our work focuses on writing the requirements right at the right levels. We do not validate the requirements to evaluate if the right requirement is written.

1.5 Notional Requirements of a System

Let us consider the notional requirements of a system, specifically an automotive vehicle [134, 83]. The stakeholders have the need to commute to work. The requirement engineer envisions a system, which is a vehicle. Therefore, a basic requirement of the system is `the vehicle shall drive`. This is a functional requirement. Functional requirements specify a function or a task that the system shall accomplish. A refined version of this requirement is `the vehicle shall drive at a speed of 90 mph`. This requirement is a non-functional requirement. Non-functional requirements include a qualitative attribute. In essence, functional requirements specify WHAT the system shall accomplish, whereas non-functional requirements specify HOW well the specified task shall be accomplished. Functional requirements can typically be evaluated with a binary outcome (satisfied or not), whereas non-functional requirements allow for qualitative or quantitative assessment of the system's performance.

Consider a few more requirements for the vehicle. Now that the stakeholders are able to commute, they need to bring the vehicle to rest at their will. The stakeholders need to stop the vehicle at will, leading to the requirement `the vehicle must be brought to rest within 8 seconds`. Additionally, there is a limit to the force a human can exert, resulting in `the force to be exerted by the driver to stop the vehicle cannot exceed 120 N`. The need for visibility while driving at night leads to the requirement `two white lights shall be symmetrically mounted on the front above 22 inches from the ground but less than 54 inches`. The driver shall indicate his intent to make a turn while driving. We derive one more requirement as `the luminous`

`intensity of the turn signal lamps must be more than 0.3 candela`. We can continue to write more requirements for the system by identifying the needs of the stakeholders and deriving requirements from the needs. More than one requirement can be derived from each need. Figure 1.2 illustrates the requirements discussed.

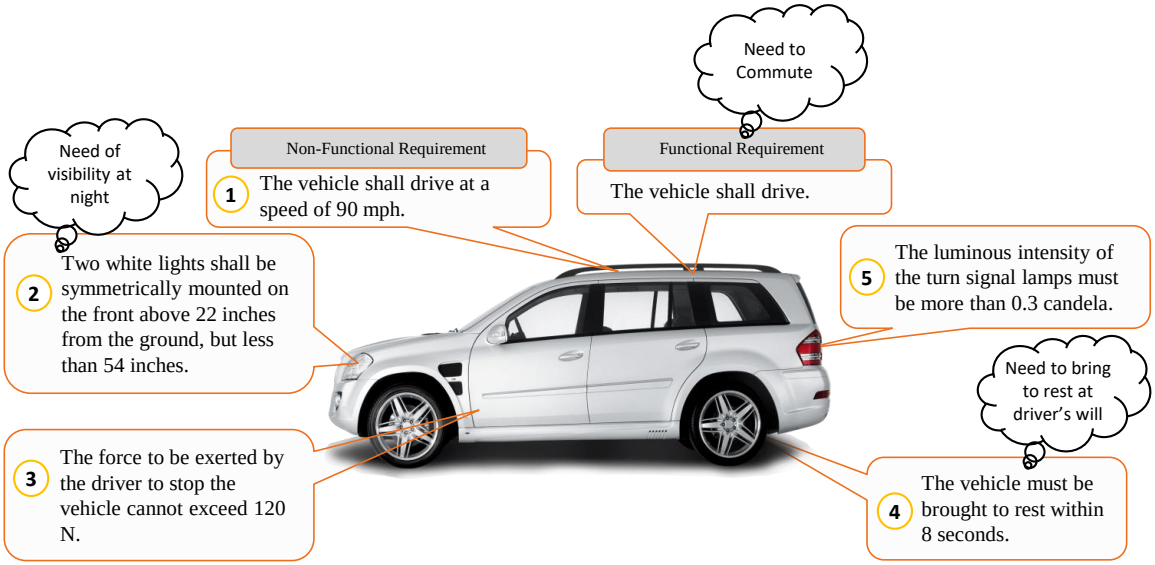


Figure 1.2: A notional set of requirements of a vehicle system

Upon closer inspection of the requirements in Figure 1.2, we observe that requirement ① is a vehicle-level requirement, while requirements ② and ⑤ pertain to the lighting subsystem, and requirements ③ and ④ relate to the brake subsystem. The brake and light systems are subsystems for our consideration because we focus on the whole vehicle as a system. We are thinking at the vehicle's abstraction level. The lighting subsystem will become a system for a supplier who supplies the light assembly to us. We can decompose the vehicle into simpler subsystems, such as the light and brake subsystems, to allocate the requirements to those subsystems. Equipping the vehicle with the brake sub-system will meet ③ and ④, and the light sub-system will meet ② and ⑤. The system will meet its requirements and needs as its subsystems satisfy the requirements allocated to them. Listing the requirements under the system hierarchy offers a hierarchical structure to the requirements. Figure 1.3 depicts the requirement allocation process.

We need to model the relationships among the requirements to represent the requirements for analysis. For instance, the vehicle can be brought to rest faster if a stronger force is applied to the brake pedals. In other words, requirement ③ and ④ are related to each other and to the brake

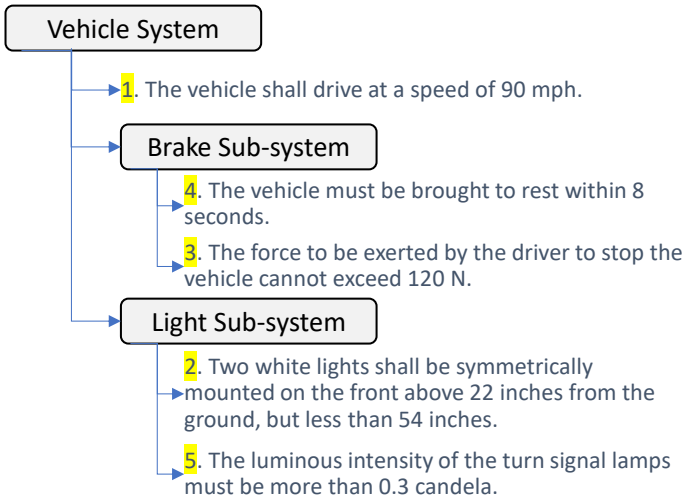


Figure 1.3: Requirement allocation

system. Similarly, requirement ② and ⑤ are related to the lighting system. Nonetheless, ② and ⑤ can be further differentiated by decomposing them into the headlight and direction indicator light sub-sub-systems. All of these requirements are part of the vehicle and, thus, relate to the vehicle-level requirements. Figure 1.4 represents the requirements as a network (Graph). The bold links between the requirements ② and ⑤ and the requirements ③ and ④ reflect stronger relationships than the lighter (gray) links. Alternately, the vehicle system can be decomposed into the brake and lighting sub-systems. And ① can be linked to the vehicle system; ② and ⑤ to the lighting sub-system; and ③ and ④ to the brake sub-system. This decomposition of the system into sub-systems and linking the requirements to the decomposed system creates a hierarchy as in Figure 1.3. This hierarchical organization of requirements refers to writing the requirements at the right levels.

The original requirements (such as `the vehicle shall drive at a speed of 90 mph`) are expressed in natural language (NL). NL is the most prevalent mode of expressing requirements. The requirements in the NL form need to be modeled to get transformed into the network form of representation. The modeling involves careful analysis to investigate and comprehend the relations among the requirements. *E.g.*, the semantics of the requirements to relate the requirements ② and ⑤ and the requirements ③ and ④. Thus, understanding and modeling the semantics of requirements is crucial. Understanding the semantics (relating the underlined words) can point us that if the headlight draws more power from the battery, the turn signal lamps may not meet its luminous intensity requirements. Or, if the driver can apply more than 120 N force to the brake

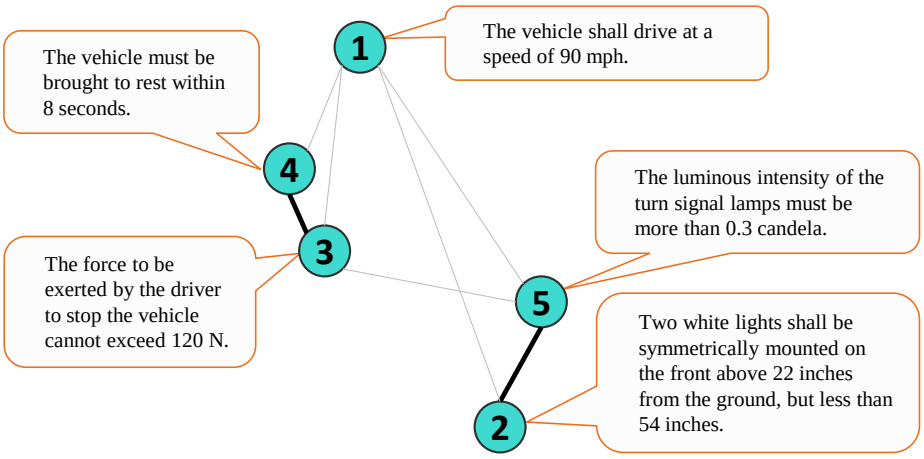


Figure 1.4: Graph representation of a set of requirements

pedal, the vehicle can be brought to rest quicker.

The majority of the analysis of requirements follows representation. For instance, consider the prioritization of requirements. Prioritization is essential to maximize the value of the system while trading off cost. In our set of requirements, ① is of highest priority as it describes a basic functionality of the system. It can be followed by ③ and ④ as they describe the stopping behavior of the vehicle. Failure of the brake system makes the vehicle hazardous. Within the brake requirements, ④ is more important than ③ as bringing the vehicle to rest is more essential than the amount of force the driver applies to the brake pedal. The lighting requirements follow in priority as the vehicle will still be able to drive during daytime. See Figure 1.5.

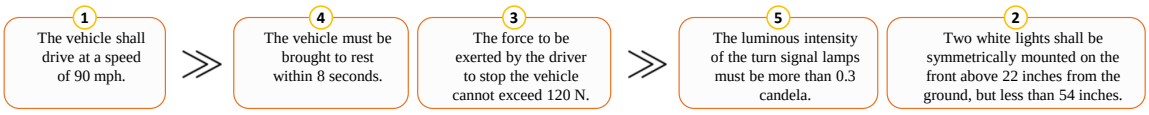


Figure 1.5: Prioritization of requirements as an example of requirement analysis

A few other directions for analysis are as follows:

- Modeling change propagation in requirements:

  Addition, deletion or modification of a requirement induces changes in the system as well as other related requirements. For example, adding a towing capacity of 5000 lb to the vehicle can affect its speed. It can also affect the time it takes to bring the vehicle to rest.

- Requirement allocation:

Decomposing the high-level requirements to low-level requirements is referred to as requirement allocation. For example, vehicle-level system requirements can be decomposed into subsystem-level requirements such as engine requirements, transmission requirements, packaging requirements, lighting requirements, and braking requirements.

- Requirement evolution:

  The requirements evolve from functional to non-functional requirements and then from one non-functional form to another. For instance, the vehicle-level functional requirement `the vehicle shall drive` evolved to its non-functional form `the vehicle shall drive at a speed of 90 mph`. Then, driving speed may vary from 90 mph to 120 or, 45 or 60 mph depending on the technological readiness, cost and safety requirements.

- Traceability:

  Traceability refers to the ability to describe and follow the life of a requirement in both forward and backward directions [63, 44]. In the context of the notional requirements, traceability can be established among the requirements as well as between a system and a requirement. E.g., the link between ② and ⑤ establish requirement-to-requirement traceability. Linking ③ and ④ to the light subsystem establishes system-to-requirement traceability.

Identification of contradictory and redundant requirements is also part of requirement analysis. The tasks that come under the umbrella of requirement analysis are limitless.

The finalized requirements are documented to serve as a binding agreement among the stakeholders and the system. A consistent set of requirements facilitates a consistent development of the system. The requirements are documented in requirement specifications as illustrated in Figure 1.6(a).

The requirements are transformed from the needs of the stakeholders. This transformation entails writing the right requirements in order to ensure that the system developed to satisfy the requirements meets the needs of the stakeholders. This calls for the need-to-requirement traceability at all stages of the requirement evolution. The requirement validation process confirms that the requirements define the right system as intended by the stakeholders [166]. In our example, equipping the vehicle with the light system to meet the lighting requirements will meet the need for night sight. See Figure 1.6(b).

The requirement verification process ensures that the requirements are written right to

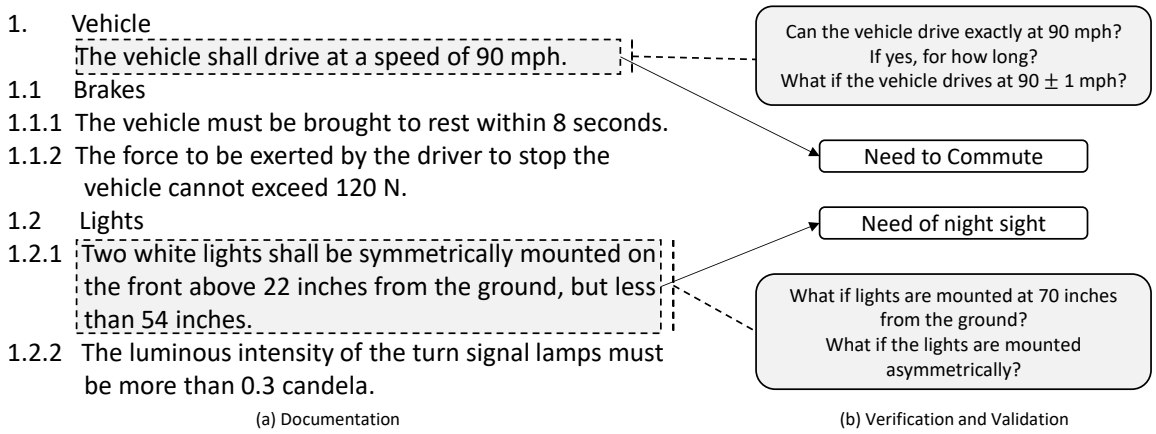


Figure 1.6: Documentation, verification and validation of requirements

eliminate the shortcomings (ambiguity, imprecision and inconsistency) of NL. Verified requirements guarantee that the requirements are achievable by the system. For instance, consider requirement ①, `the vehicle shall drive at a speed of 90 mph`. Is this requirement achievable? Can the vehicle drive exactly at 90 mph? If yes, for how long? How long will the system be tested to evaluate if the vehicle drives at 90 mph or not? What if the vehicle drives at 89.9 mph or 90.1 mph? Will the system fail to meet the requirement if the vehicle drives at 90 ± 0.1 mph? This requirement is not achievable and, thus, not verified. Consider requirement ②. What if the lights are mounted at a height of 70 inches from the ground? The lights will meet part of the requirement, `above 22 inches from the ground`, and violate part of the requirement, `less than 54 inches`. A partial failure failed the complete light installation. Thus, such requirements shall be decomposed into two requirements. ② fails at requirement verification. This is depicted in Figure 1.6(b).

1.6 Requirements Engineering

Requirement engineering (RE) concerns all the activities associated with the requirements of a system throughout the development of a system. RE encompasses the elicitation, modeling and representation, analysis, verification, validation, and Documentation of requirements [166]. Figure 1.7 illustrates the RE process with the associated challenges at each of the activities of RE.

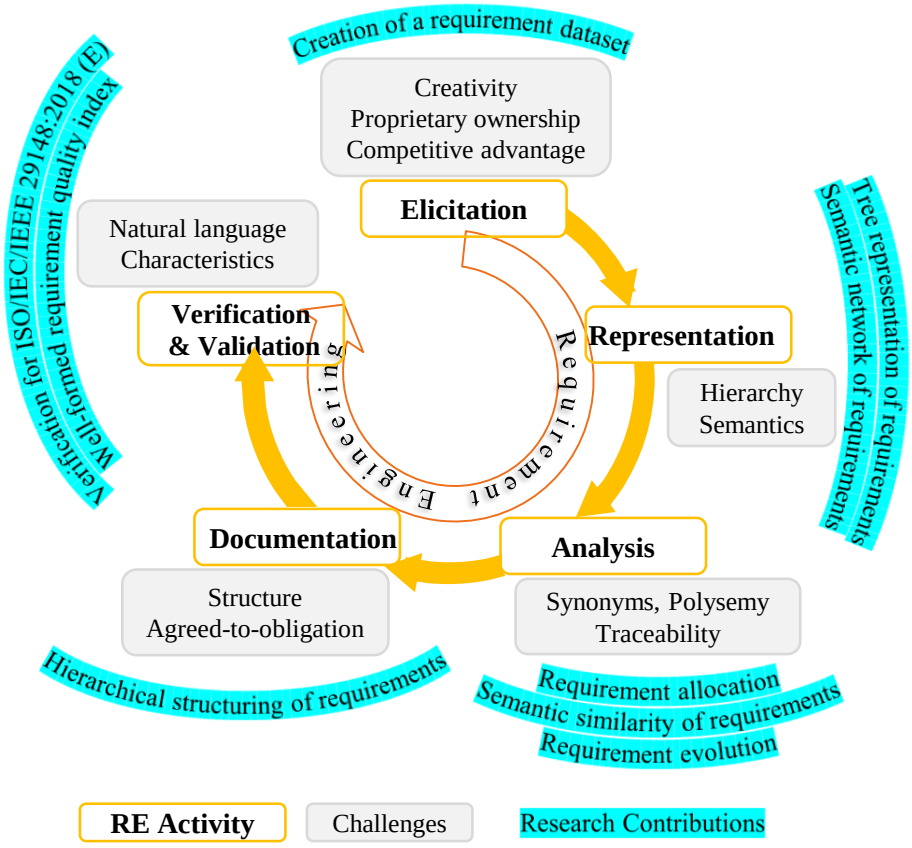


Figure 1.7: Challenges in requirement engineering activities and our research contributions

1.6.1 Elicitation

Requirement elicitation refers to the systematic process of acquiring a set of requirements for the system under consideration. It employs prototyping, structured surveys, hands-on training, brainstorming, document analysis, and comparison with products of the same segment to acquire a set of requirements (§3.1.20 in [166]).

1.6.2 Representation

A representation of requirements refers to a formal structure of an individual and/or a set of requirements with well-defined semantics [106, 85, 84, 196]. The representation aids in requirement analysis.

1.6.3 Analysis

Requirement analysis refers to the process of deploying formal and informal methods to extract value to assist the RE process in accelerating the development of the system. Identification of contradictory and redundant requirements, clustering of requirements, traceability, and allocation are some of the tasks of requirement analysis.

1.6.4 Verification & Validation

Requirement verification refers to the process of confirming that the individual and set of requirements are well-formed. An individual requirement or a set of requirements is well-formed if they meet the characteristics of a good requirement(s) and are well-organized (§3.1.26 in [166]).

Requirement validation refers to the process of ensuring that the individual and set of requirements define the right system as intended by the stakeholders (§3.1.25 in [166]).

1.6.5 Documentation

The final requirements at each stage of the requirement definition process and the system development process are documented in requirement specifications. Requirement specifications are a structured collection of the requirements, functions, performance, design constraints and other attributes for the system and its operational environments and external interfaces (§3.1.32 in [166]). Requirements documented at different levels produce requirement specifications at specific levels, such as `system requirements specification` (SyRS) at the system level, `stakeholder requirements specification` at the stakeholders' level, `software requirements specification` at the software level, `business/mission requirement specification` at the business/mission level.

1.6.6 Traceability

The standard defines requirement traceability as the identification and documentation life of the requirement, both in forward and backward directions ([63, 44], §3.1.23 in [166]). The upward path refers to the derivation of the requirement from the needs or other parent requirements. The downward path refers to the requirement allocation or the flow-down of requirements in a set of requirements.

1.7 Research Questions and Contributions

The research questions and the contributions this dissertation makes while seeking the answers to those questions are listed in Table 1.1. Please refer to Chapter 2 to have a bird-eye-view of the existing work that led to the listed research questions. Figure 1.7 pictorially organizes the contributions with respect to each activity of RE.

1.7.1 Advantages over the State-of-the-Art

This dissertation makes significant leaps in the state-of-the-art of RE. We are the first to show that the requirements in every SyRSD can be represented as a tree. The tree reflects the requirement allocation process. We investigate the criteria of requirement allocation in multiple requirement specifications for validation. Most prior work on requirement allocation focuses on a single criterion. Flanigan and Brouse [56] focused on capacity allocation in systems of systems. Vintr and Holub [178] allocated the availability of sub-systems to upgrade the whole system. Our validation shows that the requirements in the SyRSD of a system do not follow a unique criterion for requirement allocation. Unlike these works, we provide a generalizable methodology that captures diverse allocation criteria, enabling a more holistic and flexible analysis of requirement specifications.

We are the first to propose a dataset in three different forms to deploy modern computing algorithms for RE. Prior requirement datasets are quantitatively smaller, often consisting of about a hundred requirements [29, 70, 71]. The only large requirement dataset is the PURE dataset [53], which is again limited in utility due to the extensive preprocessing required for the requirement specifications available in this dataset. Our dataset is preprocessed to provide a list of requirements and the document structure in pure text form, saving preprocessing effort. While the goals and utility of the PURE dataset are not concrete, the three forms of our dataset are ready for deploying NLP, machine learning, and graph-theoretic investigations to study change propagation, semantics, requirement allocation, and quality of requirements. The formal method underlying our dataset supports expanding it by including additional SyRSDs. This preprocessed dataset provides an unprecedented foundation for applying advanced computational methods, eliminating significant barriers seen in earlier datasets like PURE.

Most past work in RE focuses on word-level semantics of requirements [50] to classify whether a sentence is a requirement (to extract requirements [68, 191]) or categorize requirements

into functional and non-functional requirements and other categories [30, 3]. We devise a method to estimate the sentence-level similarity of requirements. Our graph-theoretic method can estimate the semantic similarity between a pair of requirements whose SyRSDs are available. The surrogate machine learning model, Req-sBERT, can estimate the semantic similarity between any pair of requirements even if their parent SyRSDs are not available. We are among the few in RE to focus on sentence-level semantics of requirements. A few other studies that focus on sentence-level semantics are Mullis et al. [120], Walsh and Andrade [180], and Alhoshan et al. [9]. Nonetheless, each of these works focuses on search and retrieval, semantic similarity, and classification of requirements. While these tasks relate to RE, they are machine-learning-specific tasks. In our work, we show that our embeddings and semantic network can capture the evolution of requirements from functional to non-functional forms. Hierarchical clustering of the requirement embeddings will help in the documentation of requirements. This ability to track requirement evolution and structure them meaningfully offers capabilities beyond what is currently achievable in requirement classification.

We are the first to holistically evaluate the quality of requirements. Our requirement verification pipeline focuses on 42 rules. This is in clear contrast with prior work, which predominantly focuses on just a few criteria. Lamar et al. [96] used parts of speech, sentence structure, and grammar. Conformance to requirement patterns is investigated in [12] and in commercial RE tools such as RQA [172], Rimay [177], and QVscribe [82]. Our 42 rules (see Table 1 in Appendix A) provide one of the most comprehensive assessments of requirement quality, on par with commercial RE tools. Moreover, we are the first to transform the rule evaluations to conform to the nine characteristics specified by the ISO/IEC/IEEE 29148:2018 standard. Lastly, our proposed well-formed requirement quality index offers a qualitative metric to compare the quality of requirements. We are the first to enable a quantitative comparison of the quality of requirements. By aligning quality assessments with international standards and offering a quantitative metric, we set a new benchmark for requirement quality evaluation that bridges the gap between academic research and practical industry tools.

Table 1.1: Research Questions (RQ) and Research Contributions (RC)

ID	Research Question	ID	Research Contribution	Section	RE Tasks
RQ1.1	How to extract the hierarchy of the requirements in RSDs?	RC1.1.1	We devise a formal method to create a network of requirements from the requirements of RSDs. Our process uses the clause numbers corresponding to each requirement in the RSD.	§3.3	Representation, Traceability
		RC1.1.2	We mathematically prove that the graph is a tree using the principle of mathematical induction.	§3.4	
RQ1.2	What is the significance of the hierarchy of requirements in RSDs?	RC1.2	We investigated 86 RSDs and extracted their requirement trees. We conclude that the trees reflect the requirement allocation process.	§3.5	Requirement allocation
RQ1.3	What is the criterion of requirement allocation in a system? Is the criterion unique?	RC1.3	Each RSD uses one or more criteria, including system decomposition, functional decomposition, scenario decomposition, state decomposition, decomposition of -ilities, or decomposition based on other semantic or logical criteria. At each vertex where a decomposition is made, a choice is made for one of the criteria for allocating the requirements.	§3.5	Requirement allocation
RQ2.1	How to compute the semantic similarity between pairs of requirements?	RC2.1	We formulate a graph-theoretic method to compute the semantic similarity between pairs of requirements from the SyRSDs.	§4.3	Requirement analysis
RQ2.2		RC2.2	We release three different forms of data to accelerate requirement analysis.	§4.5	Requirement Elicitation
		RC2.2.1	***ReqList***s: lists comprising of 12701 requirements in pure text form acquired from 86 SyRSDs. They only contain the requirements and corresponding clause numbers to retain their structure. Their document structure supplements them as the metadata. *ReqList*s are ready to use NLP and unsupervised machine learning (ML) techniques without further data preprocessing.	§4.2.2, §4.5.1	
		RC2.2.2	***ReqNet***: a large network of requirements consisting of 17375 nodes to facilitate graph-theoretic investigations for RE. It assembles the *ReqTree*s of 86 distinct systems.	§4.4.1 & §4.5.3	

...continued on next page

Table 1.1 – continued from previous page

ID	Research Question	ID	Research Contribution	Section	RE Tasks
		RC2.2.3	***ReqSim***: a dataset consisting of 10933 pairs of requirements annotated with their similarity scores. The similarity scores align with human knowledge [104]. *ReqSim*, the first dataset that estimates the pairwise similarity among requirements, enables sentence-level supervised learning tasks.	§4.4.2 & §4.5.4	
RQ3.1	How to semantically represent the requirements of a system?	RC3.1	We represent the requirements as a semantic network to effectively capture the semantics at the requirement level. Each requirement becomes a node in the semantic network, and the links correspond to the cosine similarity between the requirements' embeddings. We achieve this by finetuning a modified pre-trained MPNet (Masked and Permutated Network) [156] model using the Siamese architecture of Sentence-BERT (sBERT) [135]. We refer to our model as *ReqsBERT* (**Req**uirement **S**entence **B**idirectional **E**ncoder **R**epresentations from **T**ransformers). Our modified model includes a six-layered deep neural network (DNN) with 768 neurons in each layer following the MPNet model. The pretraining retains the rich structure and semantics of NL, and the finetuning captures the RE-specific aspects. To the best of our knowledge, we are the first to focus on sentence-level semantics in engineering design.	§5.3	Representation, Requirement-to-requirement traceability

...continued on next page

Table 1.1 – continued from previous page

ID	Research Question	ID	Research Contribution	Section	RE Tasks
RQ3.2	What value can be extracted from the semantic representation?	RC3.2	We investigate the requirements of the Deep Orange 13[3] project and show that the semantic network is coherent with the semantics of requirements. Similar requirements share a higher cosine similarity score than dissimilar ones. The similarity reduces as the number of semantically unrelated keywords increases. We illustrate the utility of our semantic network in two RE-specific applications.	§5.4.2 & §5.4.4.1	Requirement analysis
		RC3.2.1	We show that a non-functional requirement (NFR) shares a higher similarity with its other NFR forms than its functional form. We show that the change in the similarity score as a requirement evolves from being a functional requirement (FR) to an NFR, which is sharper than that from one variant of NFR to another.	§5.4.4.2	Requirement analysis
		RC3.2.2	Our semantic network can aid in extracting a hierarchy of requirements to document the requirements in a requirement specification document. The requirement hierarchy reflects the hierarchy showcased by different concepts of the project.	§5.4.4.3	Requirement analysis, Documentation
RQ4.1	How to assess the qualities of requirements?	RC4.1	We create a requirement quality framework to evaluate the qualities of requirements. Our framework holistically evaluates the requirements using 42 NLP rules focusing on the token-level, span-level, syntactic structure-level and sentence-level attributes of the requirement.	§6.5	Requirement verification

...continued on next page

[3] https://cuicardeeporange.com/project/do13/

Table 1.1 – continued from previous page

ID	Research Question	ID	Research Contribution	Section	RE Tasks
RQ4.2	How can the qualities of a pair of requirements be compared?	RC4.2	We propose a quantitative metric, *well-formed requirement quality* (wRQ) index. The metric incorporates the results of the 42 rules and the nine characteristics. The metric results in a score between 0 to 1 to compare the qualities of requirements. To the best of our knowledge, we are the first to offer a quantitative metric to evaluate the qualities of requirements in accordance with ISO/IEC/IEEE 29148:2018 (E) [166].	§6.6	Requirement analysis

1.8 How to Read the Dissertation?

This dissertation is written with a focus on novice researchers as the intended audience. The writing was fueled by my own hardship when I was starting my research in RE in 2021. The research focus spans elicitation, analysis, allocation, documentation, datasets, semantics and verification tasks of RE. The employed techniques balance a blend of formal and informal methods. Graph theory, rule-based natural language processing (NLP) and document analysis are the formal methods used. Deep learning [94, 93, 154, 46, 153, 45], generative artificial intelligence (GenAI) [5], large language models (LLM) [135, 36] are the leveraged informal methods. The chapters do not follow the order of the RE activities depicted in Figure 1.7. Rather, the chapters follow a pedagogical order to ease information assimilation.

Chapter 1 introduces the dissertation by discussing a requirement and the RE activities by considering a notional set of requirements. The chapter lists the research questions and contributions and guides the reader on how to use this dissertation. Chapter 2 reviews the literature to motivate and lay the foundation of each of the RE tasks targeted by this dissertation. Each subsequent chapter, from 3 to 6, includes a brief introduction with a short literature review to position the chosen methodology before diving into the methodology, results, discussion and conclusion. Chapter 7 highlights the impact the dissertation makes. Chapter 8 lists a few directions for future research. Chapter 9 offers the concluding remarks to this dissertation. The dissertation ends with listing the knowledge dissemination efforts in Chapter 10.

Every RE activity is contingent upon a set of requirements. We start by creating a requirement dataset by collecting a set of system requirement specification documents (SyRSD). We represent each SyRSD as a tree in Chapter 3 [145]. Chapter 4 scales up the work of Chapter 3 [146]. Chapter 3 focused on individual SyRSDs and the requirements contained in it. Chapter 4 focuses on 86 SyRSDs and the requirements contained in them. In Chapter 5, we use the results of Chapter 4 to create the *Req-sBERT* model by fine-tuning a modified Sentence-BERT [135] model to adapt the pre-trained model for the RE domain [144]. Chapter 6 develops a rule-based NLP framework to verify requirements [91].

Chapter 2

Literature Review

2.1 Requirement Datasets

All the RE tasks are contingent upon acquiring a set of requirements during the elicitation process. The lack of a RE dataset and a method to create any RE dataset is evident by the plenty of research papers focusing on identifying requirements from source documents by classifying if a particular sentence is a requirement or not [68, 191]. Unlike the abundance of NL corpora, requirement datasets are scarce because of the following reasons. First, as requirements are a product of system ideation, having access to detailed requirements can disclose the system under development. The organization risks losing its competitive edge. Second, it is a challenging task to acquire a set of well-formed requirements with all the characteristics of ISO/IEC/IEEE 29148:2018 [166]. Third, the creative nature of the requirement elicitation process and the ensuing human involvement during elicitation limits the number of elicited requirements. Fourth, having a dataset that is usable for all the tasks of RE is challenging. We need to create multiple variants of the dataset to enable multi-faceted analysis. Fourth, as the requirements vary depending on its domain and application, annotating the requirements for any specific analysis (such as traces, dependence) becomes a resource-intensive task. Thus, most of the existing RE work happened in silos with small requirement datasets. Ref. [52] emphasized the necessity of a good requirement dataset to foster progress in RE.

The most used requirement dataset is the PROMISE non-functional requirements dataset [29]. It contains 625 requirements in the form of a list. Most of the RE work leveraged this dataset.

Notably, most of the requirement datasets contain less than a hundred or just a few hundred (often < 1000) requirements pertaining to a single system (Figure 4.2). The best available open source requirements dataset is the PUblic REquirements (PURE) dataset [53]. It is an aggregation of publicly available SyRSDs from the web in their native forms. While the corpus spans diverse domains, the data is quite restricted in terms of its utility owing to the extensive preprocessing it entails. Chapter 4 offers a requirement dataset in which the data is available in the native form, as well as in a clean list form and a network form, with their semantic similarity scores. Our dataset facilitates multi-faceted analysis as they are variants with the same core set of requirements. Additionally, we offer a formal method to expand the corpus. The dataset is underpinned by the tree structure extractable from the SyRSDs. Our work attempts to address the fourth research direction from the review of semantic networks for engineering design by Han et. al. [69], a comprehensive knowledge graph of requirements with a framework to extend it further.

2.2 Representation of Requirements

Requirements are the foundation of system development, and their effective management is critical for reducing the cognitive load during system design [128]. Traditionally, most representations aim to simplify this process by focusing on a subset of requirements, employing visual structures, and organizing them systematically [105]. Requirements are often grouped by system features, such as in telephone systems with features like local calls, conference calls, and call forwarding [75]. State machines, for instance, first decompose the system into different operational modes (e.g., normal, power-saving, or high-performance) before analyzing requirements [188]. In the simplest form, the elicited requirements are represented as *list*s [41]. The lists need minimal preprocessing of requirements. Thereby, the information content of lists is minimal. Object-oriented representations such as SysML diagrams assign attributes and functions to system objects [97], as seen in healthcare systems where requirements are specified for patients, nurses, and sensors [126]. *Matrices* offer a generic and flexible form of representation [183]. The versatility of matrices can be attributed to their effectiveness in capturing the correspondence between pairs of objects, such as requirements-to-requirements for allocation and traceability, requirements-to-systems/components, and requirements-to-tests for verification and validation. Design structure matrices [28], house of quality modeling schemes [13] are a few instances of matrix-based approaches. *Networks* are another

form of representation [61]. Networks and matrices are interconvertible. These approaches, including the use of functional hierarchies, create structured representations that allow for more efficient organization and analysis of requirements.

Hierarchical representation of requirements is crucial for simplifying system integration and facilitating system development. Traditional models like the WRSPM (World-Requirement-Specification-Program-Machine) model [67], lacking hierarchical structures, face challenges during integration, as noted in system development contexts [185]. Tools like KAOS [175] address this by decomposing requirements into hierarchical structures, forming directed graphs that trace from high-level parent requirements to low-level leaf requirements. This hierarchical decomposition ensures that each requirement is allocated precisely to a system or function, enhancing traceability and decision-making during the design process. The decomposition paths occasionally converge if a single function/system satisfies multiple requirements. However, the requirement decomposition process neither converges nor forms a loop because a loop decomposes a low-level requirement into a high-level requirement [39]. Some of the low-level requirements may satisfy multiple high-level requirements. Such occurrences manifest when two system elements interact. Such requirements can be removed by rewriting the requirements as well-formed requirements by adequately defining each system element's environment [137]. The key takeaway is that the decomposition of requirements creates a hierarchy. Thus, we devise a formal method in Chapter 3 to represent the set of requirements in a SyRSD as a tree.

While lists, matrices, and diagrams are widely used, they often fail to capture the deeper relationships between requirements [183]. Traditional matrix representations capture binary relationships but are insufficient for handling linguistic nuances such as synonyms and word order, leading to sparse and less informative models. The key underlying factor to each of the aforementioned representations is *the extraction of relationships among the requirements*. Rule-based NLP (e.g., TF-IDF [11], term frequency, parts-of-speech (PoS) tagging, latent semantic analysis) is the de facto choice. These approaches rely on the co-occurrence of words in the requirements but are limited in capturing the meaning and order of words, lacking semantic and syntactic depth. Moreover, since these methods use a bag-of-words approach, the length of the resulting vectors matches the length of the vocabulary, resulting in sparse representations. So, we transform the requirements into dense vectors by embedding them in a latent space to capture semantic and syntactic information contained in the requirements. We invoke vector semantics of requirements using these dense

vectors, referred to as *embeddings*. Those embeddings represent the locations of the requirements in a high-dimensional space where similar requirements are located close to each other and vice-versa. Semantic representation of requirements is essential as systems grow more complex, requiring deeper interpretations of natural language. Chapter 5 presents a methodology to get embeddings of the requirements and represents the requirements as a semantic network.

2.3 Semantics of Requirements

The semantics of requirements allows us to understand unknown concepts and requirements by comparing and contrasting the subjectivity and imprecision of requirements expressed in NL. Most of the RE work that focus on the semantics of requirements restrict themselves to word-level semantics instead of sentence-level semantics. They leverage word-level semantics [50] to classify if a sentence is a requirement (to extract requirements [68, 191]); or categorize requirements into functional and non-functional requirements, and other categories [30, 3]. Techniques focusing on sentence-level semantics supplement word-level semantics with other information like syntactic and word-order information [48]. Greedy pairing, optimal matching are a few techniques of acquiring sentence level semantics from word level semantics [152].

However, utilizing word-level semantics for comparing requirement sentences questions the credibility of those methods as they (1) ignore the order of the words in the requirement sentences and (2) fail to capture the contextual information offered by the composition of keywords [54]. Thus, the focus is minimal on other RE tasks such as generating requirements, identifying redundancy and contradiction, semantic traceability, and predicting change [196]. The key impediments include the need for (1) a germane dataset; and (2) sentence level semantics of requirements. Furthermore, vector semantics based on sentence-embeddings are not yet fully adopted for usage in RE. While the sentence-embeddings can be used for inference in RE, their utility is restricted due to the unavailability of a dataset or a method to compute the semantic similarity among requirements to fine-tune the sentence-embedding models.

Chapter 4 proposes a formal method to estimate the similarity between a pair of requirements. Our method of computing the semantic similarity between requirements is inspired from lexical semantics, such as WordNet [115], which enable computing the semantic similarity between words. The underlying idea is to construct a graph of the words and leverage the graph-theoretic

distance to compute the similarity [136, 10]. Such approaches assume that two objects are similar if their distance is short. The distances have been scaled [100, 193] to overcome the fact that the semantic distance between nodes at different levels of the taxonomy are not same. More fine-grained concepts are semantically more similar than high-level concepts. We adopt a hierarchical weighing scheme to incorporate these generalizations to compute the semantic similarity between requirements. Furthermore, we incorporate the document level similarity to consider the highest level of semantic similarity, which is between the SyRSDs. Our method of computing semantic similarity leverages the tree structure of requirements in SyRSDs, thereby harnessing the implicit information stored in the SyRSDs.

2.4 Requirement Verification

The quality of individual requirements has been frequently described by a set of characteristics. The research work to evaluate the characteristics dates back to 1984 when Barry Boehm defined complete, consistent, feasible, and testable as the criteria for verifying and validating a set of requirements [17]. Numerous researchers used one or more characteristics such as necessary, verifiable, implementation-free, complete, attainable, unambiguous, verifiable, minimal, feasible, valid, traceable, consistent, allocated, unique identifier, positively stated, non-redundant [17, 74, 60]. Frameworks such as C3F [186], SMART [42], and VANS [23] were also developed that constitute a specific set of characteristics. The ISO/IEC/IEEE 29148:2018 (E) [166] authoritatively lists fourteen characteristics. A requirement meeting all the essential chatacteristics is a verified requirement.

NLP, boilerplates and ontology are the most sought-after techniques for requirement verification [47]. The importance of evaluating the quality of requirements can be underscored by the burgeoning RE tools. Gea et al. [34] compared the capabilities of 38 commercial RE tools for their capabilities across the elicitation, analysis, specification, modeling, verification and validation, management and traceability activities of RE. These tools primarily rely on NLP techniques to evaluate the quality of requirements. Most of the methods measure the quality of one of the aspects of requirements, such as similarity, ambiguity [4], inconsistency [35], conflict [147]. Aceituna et. al. used a model-based approach to verify requirements for ambiguity and incompleteness [4]. Mokammel et. al. combined diverse requirement quality measures based on lexical, syntactic and semantic approaches in a single software tool [117]. Their quality grade metrics focused on consis-

tency, completeness and expressiveness. Multiple tools were created and endorsed by NASA to craft requirements of good quality [111]. In most of the work, the focus is on classifying if a requirement has characteristics or not. The results are often binary with specific details of the requirement. A holistic view in evaluating the quality of requirements was proposed by Parra et. al. [127]. They used an array of quality metrics such as the number of ambiguous expressions, dependencies, hierarchical levels, verbs in passive voice, words and others. A quantitative measure of the quality of requirements is proposed in [60]. They measured features such as the number of words (for atomicity), the number of ambiguous terms, the number of verbs, and the number of dependencies. Nonetheless, their measures fail to capture the high-level characteristics of requirements. In Chapter 6, we develop a comprehensive rule-based NLP pipeline to verify the individual requirements to the nine characteristics specified by the ISO/IEC/IEEE 29148:2018 (E) [166] standard.

Chapter 3

ReqTree: Requirement Allocation from the *Tree* Structure of *Req*uirements in Requirement Specifications

3.1 Introduction

The design input requirements definition process transforms the stakeholders' needs into design input requirements. The process is essential to define the system architecture, baseline requirements, flow-down of the requirements and implementing a design solution [186]. The requirements are iteratively defined such that each succeeding level offers additional details about the system's design [149]. The process recursively allocates and breaks down the parent system into simpler subsystems relevant to the stakeholders of that level. The decomposition of the complex system into subsystems and components ensures that the knowledge required to design these subsystems becomes manageable for individuals or specialized teams. Prominent modes of decomposition include product breakdown structure (PBS) and functional breakdown structures (FBS) [18, 76]. The decomposition process follows a divide-and-conquer principle, reducing system complexity and

allocating functions, requirements, and performance to simpler subsystems [72]. The decomposition makes the functions of different components of the final product independent of each other to ensure minimal propagation of failure [65]. The decomposition aspires to achieve the independence axiom of functional requirements by creating a hierarchy. The requirement allocation (RA) process ensures that each requirement is fulfilled by one or more system elements. The ISO/IEC/IEEE 29148:2018 (E) [166] standard reinforces the importance of maintaining this hierarchy by emphasizing that requirements must be appropriate to their level, avoiding solution-specific details and unnecessary constraints. The breakdown of the high-level functional requirements into the lowest level of the system hierarchy gives rise to the functional architecture [103]. Thus, the high-level generic requirements are decomposed recursively to a level where the requirements become very specific such that a buy/build/code decision can be made for a system(s) to satisfy the particular requirements [101]. This hierarchical approach allows for the simultaneous development of subsystems before their integration into the parent system, V&V, and aids in debugging and identifying root causes in case of system failures [182].

The finalized requirements are documented in RSDs. RSDs include requirements from the system of interest's (SoI) viewpoint. They serve as binding agreements among all the stakeholders of the SoI, starting from the acquirer of the system to the designers and developers of the system [137]. The documents trade between readability and processability [75]. Readability concerns with the structure of the RSDs. Processability concerns the quality and consistency of individuals and sets of requirements to reduce ambiguity and precision. The RSD can include a list of requirements to ease readability without easing processing. On the other hand, the RSD can include object-oriented models, which are easier to process but harder to comprehend for most stakeholders. So, the RSDs incorporate an ordered list of requirements with unique clause numbers for each requirement, which allocate unique positions following the organization and the logical flow of the RSD [75]. A proper structure of the RSD minimizes the number of requirements, eases the understanding of large amounts of information, relates and aggregates similar requirements, and enables easy reuse of requirements across projects. These aspects invoke a hierarchical structure in the RSDs, with multiple levels of sections and subsections [120]. A hierarchical tree structure of requirements is endorsed by the NASA System Engineering Handbook [74] and the INCOSE Needs, Requirements, Verification, Validation Lifecycle Manual [186]. The example outline for a system requirement specification document (SyRSD) prescribed by the ISO/IEC/IEEE 29148:2018 (E) standard [166]

also depicts a hierarchical tree structure (See Figure 3.1). The position of each requirement in the hierarchy offers the primary classification of the requirements. Whalen et al. [185] endorsed a hierarchical organization of requirements in alignment with the architectural decomposition of the system to foster natural notions of traceability and refinement of the system.

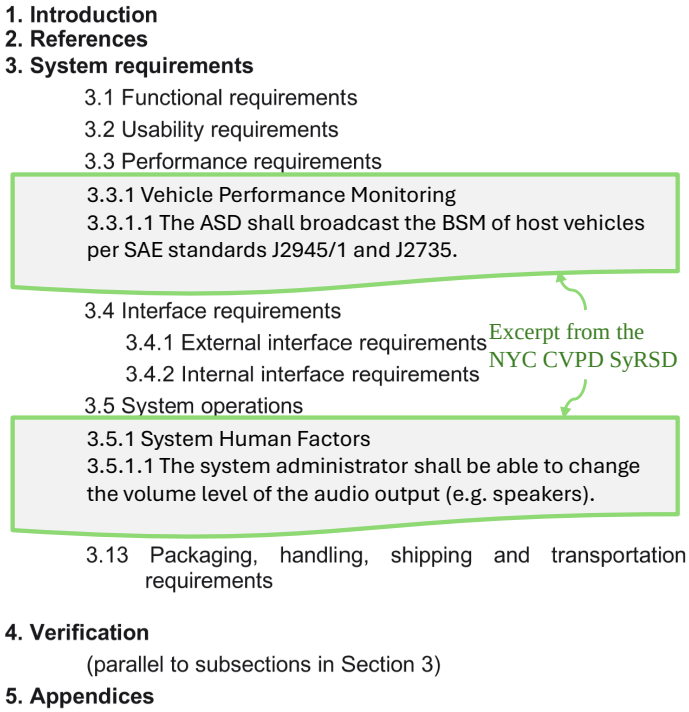


Figure 3.1: The outline of a requirement specification prescribed by ISO/IEC/IEEE 29148:2018 (E) [166] modified by a preprocessed excerpt from the system requirement specification document of the New York City's Connected Vehicle Pilot Deployment (NYC CVPD) program [59]

The RE process starts with high-level requirements and goes through the RA process to get the low-level requirements. All the final requirements are documented in the RSDs. Thus, our work comes under the umbrella of reverse engineering for RE as we start with the RSDs and extract the RA process. Our work is motivated by the need for more data or open-source information about the RA process [52]. The fear of disclosure of the system ideation and subsequent loss of competitive advantage inhibit organizations from sharing the details of their system requirements. Thus, we devise a formal method to extract the RA from openly available RSDs [146]. Our tree creation process is inspired by creating a tree from the table of contents of a document [108, 87]. The RA can offer a better understanding of legacy systems for the maintenance and improvement of their sub-systems. When systems are developed based on legacy systems, combining RE with reverse

engineering will bring the knowledge of the legacy systems to the system under development [18]. The extracted RA can help identify redundant system elements and requirements, which requirement to retain and which requirement to discard or revise [182]. Additionally, the RA can offer information about the system architecture, the requirement to function to system mapping, and the rationale for different design choices.

Essentially, the research in this chapter seeks the answers to the following research questions (RQ):

RQ1 : How to extract the hierarchy of the requirements of a system?

RQ2 : What is the significance of the hierarchy of requirements in a SyRSD?

RQ3 : What is the criterion of requirement allocation in a system? Is the criterion unique?

While answering the research questions, we make the following contributions:

- We devise a formal method to create a graph of requirements from the requirements in the system's SyRSD. Our process uses the clause numbers corresponding to each requirement in the SyRSD.
- We prove that the graph is a tree using the principle of mathematical induction.
- We investigate 86 SyRSDs and extract their requirement trees. We conclude that the trees reflect the requirement allocation process.
- Each system uses one or more criteria, including system decomposition, functional decomposition, scenario decomposition, state decomposition, decomposition of -ilities, or decomposition based on other semantic or logical criteria.

3.2 Literature Review

3.2.1 Hierarchy in Requirements

The product architecture definition process iteratively identifies the requirements, decomposes the requirements to be achieved by functions, decomposes the functions to be accomplished by systems, decomposes the systems into sub-systems, and defines the behaviors, structure and interactions of the sub-systems [89]. It focuses on the organization, functions, relationships and interactions

among the sub-systems. The iterative process terminates upon identifying a set of components with a buy/build/code/reuse decision. The decomposition process simultaneously flows down the system level requirements into the system elements at the succeeding levels of the system architecture for which the requirements will be further defined [186]. The requirements and concepts at one level serve as the constraints for the requirements and concepts of the succeeding level to which they are allocated [72]. The process produces a hierarchy of multiple levels of sub-systems and system elements [186]. This functional analysis ensures that the system satisfies the stakeholders' needs in all lifecycle phases and operational states by systematically identifying and correlating the systems' functions and sub-functions to meet the specified requirements [132].

The hierarchy is prevalent in many systems. The functional requirements and the physical parts of a system were mapped using hierarchical models of the functionality and configuration of the system [92]. Alfaris et al. [8] hierarchically decomposed the design of a sustainable building into a multilevel abstraction and identified the form parameters. A hierarchy in the concepts for elaborating and refining the high-level constraints (e.g., mission success criteria, operability) into low-level parameters (e.g., mass allocation, temperature control) was used by NASA for its Europa mission [78]. The functional and logical decomposition informed their concept hierarchy. The cascading structure of the requirements of a civil aircraft system was hierarchically represented by Li et al.[103]. Lamm and Weilkeins [97] used SysML models to create the functional architecture of a hearing instrument. Wang and Li [182] used an analytical hierarchy process to decompose the system into its components to allocate redundancy. They captured the failure interaction to minimize the failure of complex systems.

3.2.2 Requirement Allocation

Requirement allocation refers to linking a parent requirement with a system element at the next level of physical architecture [186]. It flows down the requirements defined at one level of the physical architecture to the next lower level of the architecture [64, 72]. The allocation can be responsibility, as in accomplishing a task specified by a functional requirement, or a quantity, as in physical and performance attributes. Mass, volume, and dimensions are a few examples of physical quantities [179]. Tolerance, power consumption, precision, and bandwidth are performance attributes. Resource-based allocation allows mapping requirements to functional architecture elements to logical architecture elements to physical architecture elements [57]. RA establishes a

directional relationship from requirements to other architectural elements. The allocation process ensures that the requirements or functionalities at the higher level are implemented and verified by the requirements and functionalities at the lower level. Thus, RA is an optimization problem [195] which can shorten the development period, improve the system's performance and reduce cost. RA allows the simultaneous development of the sub-systems before their integration and interaction to be a part of the parent system. Notably, RA also gives rise to a hierarchy.

Kusiak and Qin [95] used a structural decomposition approach to recursively allocate requirements into finer requirements until a function can achieve the requirement. Miller and Herber [113] split the requirements into different categories and then allocated them to appropriate system levels. They ensured the requirements were exclusive of any level below or above their assigned levels. Flanigan and Brouse [55] developed an allocation process to allocate the requirements of a system of systems (SoS) to individual systems. They focused on assessing the effectiveness of RA. Their method also considered existing systems and requirements during the allocation process. NASA created visualizations to understand the requirement derivation process for the Europa mission [78]. They focused on requirement flow, allocation and comparison of the requirement flow-down with the project decomposition. Li et al. [103] allocated the top-level requirements of a civil aircraft into low-level requirements to constrain systems. They used a model-based functional analysis approach. They investigated the hierarchy of the functional requirements to gain insight into the product breakdown, requirement topology and traceability. They used tree diagrams to show the requirement information architecture involving the requirement flow-down and topology of the aircraft.

3.3 Methodology

We derive our representation of requirements from the unique positions of requirements in RSDs. Every requirement in an RSD is allocated with a unique clause number. The clause number of a requirement is generally referred to as 'requirement number'. Our definition of a clause number is not specific to requirements. The definition also includes the clause numbers allocated to the sections and subsections of the RSDs. The clause numbers refer to requirements during discussions and changes in requirements. We noticed two types of clause numbers in RSDs [171]: sequential numbering (e.g., 2021 - ReqView.pdf) and hierarchical numbering (e.g., 2022 -

MobileSurveillance.pdf) [53]. It is common to find RSDs where the requirements follow a sequential order within the hierarchy of the RSD. The sequential numbering may optionally use a categorical prefix (e.g., 2004 - SGVTF Integration.pdf). The prefix, part of the clause number preceding the terminal separator, indicates the order of the requirement in the document's structure expressed by the sections and subsections. We parse the clause number of each requirement to create a path from the requirement vertex to the document vertex to construct a graph encompassing all the requirements. The process extracts the ancestor vertices (sections and subsections of the document) of the requirement vertex. The path passes through the ancestor vertices to reach the root vertex, the document vertex.

3.3.1 Graph Definition

Consider a requirement $R_i \in \mathcal{D}$, where $i \in [1, N]$. $\mathcal{D}$ represents the list of requirements extracted from the RSD in which the requirement R_i exists. Let N be the total number of requirements in $\mathcal{D}$. The clause number of the requirement R_i is c_i, which is specified in the SyRSD as $c_i = c_1^i \cdot c_2^i \cdots c_{r_i}^i \cdots c_{t_i}^i$ where $c_{r_i}^i$ is the r_i-th token in the clause number c_i. $c_{t_i}^i$ is the terminal token. The tokens are separated by a separator, a period in most RSDs, including our representation. We assume that each requirement has a unique clause number. In case a requirement does not have a clause number or if the clause number is not unique, we allocate a unique clause number based on its location in the RSD. This preprocessing is detailed in §3.5.1. As the requirements can belong to one of the many chapters in $\mathcal{D}$, the clause numbers can be completely distinct from each other. Thus, we prefix the clause number with the document vertex D and redefine the clause number of R_i as $c_i' = D \cdot c_1^i \cdot c_2^i \cdot ... \cdot c_{t_i}^i$. We parse this clause number into $t_i + 1$ elements starting with D as the first one, adding $D \cdot c_1^i$ as the second one, a $D \cdot c_1^i \cdot c_2^i$ as the third one, and so on, and ending with the last $(t_i + 1)$st element $D \cdot c_1^i \cdot ... \cdot c_{t_i}^i$. Hence the clause number implies the set $\{c_i'\} = \{D, D \cdot c_1^i, D \cdot c_1^i \cdot c_2^i, ..., D \cdot c_1^i \cdot ... \cdot c_{t_i}^i\}$ whose elements have been extracted from the clause number. Let $|c_i'|$ denote the number of elements in c_i'. We apply this approach to all the requirements R_i in $\mathcal{D}$, obtain a collection of sets $\{c_i'\}$, and use their elements to construct a graph associated with the entire $\mathcal{D}$.

We define a graph

$$G = (V, E) \tag{3.1}$$

with the set of vertices $V = \{v_j | v_j \in \{c'_i\}, j \leq t_i, i \in [N]\}$, and the set of edges $E = \{(c_j, c_k) \Big| |c_k| - |c_j| = 1, c_j, c_k \in \{c'_i\}\}$. While the set $\{c'_i\}$ contains the vertices extracted from a single requirement R_i with clause number c'_i, the set V contains all the vertices extracted from all the requirements in $\mathcal{D}$. Each vertex in V represents a requirement or one of its parent sections. Each edge connects a requirement/section to its parent section.

For a single requirement R_i, the graph constructed from its clause number $c'_i = D \cdot c^i_1 \cdot c^i_2 \cdot ... \cdot c^i_{t_i}$ has the set of vertices, $V(R_i) = \{c'_i\} = \{D, D \cdot c^i_1, D \cdot c^i_1 \cdot c^i_2, ..., D \cdot c^i_1 \cdot ... \cdot c^i_{t_i}\}$, and the set of edges, $E(R_i) = \{(D, D \cdot c^i_1), (D \cdot c^i_1, D \cdot c^i_1 \cdot c^i_2), ..., (D \cdot c^i_1 \cdot ... \cdot c^i_{t_i - 1}, D \cdot c^i_1 \cdot ... \cdot c^i_{t_i})\}$. See Figure 3.2(a).

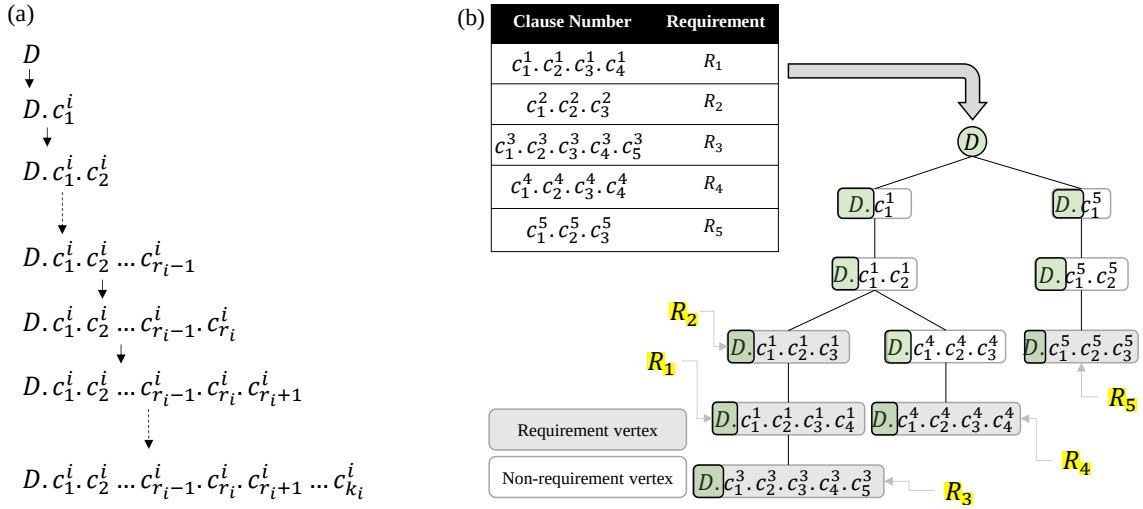


Figure 3.2: (a) The path created from the requirement R_i with clause number $c_i = c^i_1 \cdot c^i_2 \cdot ... c^i_{r_i - 1} \cdot c^i_{r_i} \cdot c^i_{r_i+1} ... c^i_{k_i}$. (b) Creation of *ReqTree* from the clause numbers

Thus given a requirement, every clause number can contribute to multiple vertices and edges in the graph. We refer to the vertex $D \cdot c^i_1 \cdot ... \cdot c^i_{t_i}$ as a *requirement vertex* since it corresponds to the clause number of R_i. All other vertices in $V(R_i)$, with which a requirement is not associated, are non-requirement vertices.

The set of vertices and edges created from $\{c'_i\}$ constitute a path $\mathcal{P}(R_i)$ from D to c'_i consisting of alternating vertices and edges [21]. The path can be written as $\mathcal{P}(R_i) = (D, e^i_1, D.c^i_1, e^i_2, D.c^i_1.c^i_2, \cdots, e^i_{t_i}, D \cdot c^i_1 \cdot c^i_2 \cdot ... \cdot c^i_{t_i})$, where $e^i_j = (D \cdot c^i_1 \cdots c^i_{j-1}, D \cdot c^i_1 \cdots c^i_j)$ is the j-th edge. Thus, G is a union of all the paths created by each requirement R_i in the $\mathcal{D}$, *i.e.*,

$$G = \bigcup_{i=1}^{N} \mathcal{P}(R_i) \ , \ R_i \in \mathcal{D} \tag{3.2}$$

As $\mathcal{P}(R_i)$ connects D to c'_i, the requirement vertices become leaf vertices. The paths created by all the requirements $R_i \in \mathcal{D}$ can intersect by one or more vertices and edges. The paths can potentially create cycles in G by introducing multiple paths between a pair of vertices in G. Thus, we create a graph from the paths by removing the duplicate vertices and edges, and then prove that the graph is a tree.

3.3.2 Example Requirement

Consider the requirement R_i, `The system administrator shall be able to change the volume level of the audio output (e.g., speakers)`, in Figure 3.1. Its clause number c_i is `3.5.1.1`. This clause number is under the subsection `3.5.1: System human Factors`, the section `3.5: System Operations`, and the chapter `3: System Requirements` within $\mathcal{D}$. We redefine the clause number by prefixing it with the document vertex D to obtain $c'_i = D.3.5.1.1$, which creates the set of vertices, $\{c'_i\} = \{D, D.3, D.3.5, D.3.5.1, D.3.5.1.1\}$ and the set of edges, $\{(D, D.3), (D.3, D.3.5), (D.3.5, D.3.5.1), (D.3.5.1, D.3.5.1.1)\}$, constituting a path from D to $D.3.5.1.1$. The vertex $D.3.5.1.1$ is a requirement vertex. The vertices $D.3.5.1$, $D.3.5$, $D.3$ and D are non-requirement vertices as no requirement is associated with them.

3.3.3 Graph Creation

We now present the process of creating the graph and refer the reader to Figure 3.2(b).

We initiate with an empty graph, $G = \phi$, and split the creation process into four cases depending on the nature of the clause number to be added to the graph. They are

- $\{c'_i\} \cap V = \phi$: No part of the clause number is in V.
- $\{c'_i\} \subset V$: The clause number exists in V.
- $\{c'_i\} \cap V \neq \phi$: Part of the clause number is in V.
- $\{c'_i\} \cap V = \{D\}$: Only the document vertex of the clause number is in V.

3.3.3.1 $\{c'_i\} \cap V = \phi$: No part of the clause number is in V

We add requirement R_1 with clause number $c_1 = c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1$. Here, R_1 belongs to the chapter c_1^1 and lies within the section $c_1^1 \cdot c_2^1 \cdot c_3^1$. It becomes $c'_1 = D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1$ upon

prefixing with the document vertex. Here $\{c'_1\} \cap V = \phi$. This case is applicable only for the first vertex that gets added to G as all the requirements are prefixed with D. R_1 creates the vertices $\{D, D \cdot c_1^1, D \cdot c_1^1 \cdot c_2^1, D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1, D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1\}$ where the requirement R_1 corresponds to the vertex $D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1$. Thus, the vertex $D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1$ is a requirement vertex and all other vertices are non-requirement vertices. R_1 creates the edges $\{(D, D \cdot c_1^1)$, $(D \cdot c_1^1, D \cdot c_1^1 \cdot c_2^1)$, $(D \cdot c_1^1 \cdot c_2^1, D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1)$, $(D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1, D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1)\}$ by connecting each pair of vertices according to their order in the clause number. Now, the graph $G = \{\mathcal{P}(R_1)\}$. The edges create a path from the requirement vertex to the document vertex. The path passes through the ancestor vertices of the requirement, where the ancestor vertices correspond to the section/subsections in the RSD. The edges reflect the structure of the document.

3.3.3.2 $\{c'_i\} \subset V$: The clause number exists in V

The same steps are followed for all the requirements in the RSD. At each iteration, the subsequent requirements will use the vertices and edges already existing in the graph. The non-existing vertices and edges shall be added to the graph at each iteration. Consider a requirement R_2 with clause number $c_2 = c_1^2 \cdot c_2^2 \cdot c_3^2$. If $D \cdot c_1^2 \cdot c_2^2 \cdot c_3^2 = D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \Rightarrow \{c'_2\} \subset V$, then we do not have to add any vertex or edge to G to add requirement R_2. R_2 will be added to the vertex $\underbrace{D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1}_{=D \cdot c_1^2 \cdot c_2^2 \cdot c_3^2}$. The vertex $D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1$ will switch from being a non-requirement vertex to a requirement vertex upon addition of R_2. Now, $G = \{\mathcal{P}(R_1), \mathcal{P}(R_2)\}$.

3.3.3.3 $\{c'_i\} \cap V \neq \phi$: Part of the clause number is in V.

Consider another requirement R_3 with a clause number $c_3 = c_1^3 \cdot c_2^3 \cdot c_3^3 \cdot c_4^3 \cdot c_5^3$. It becomes $c'_3 = D \cdot c_1^3 \cdot c_2^3 \cdot c_3^3 \cdot c_4^3 \cdot c_5^3$ upon prefixing with the document vertex. Now, if part of the clause numbers of R_3 is common with V, i.e., $D \cdot c_1^3 \cdot c_2^3 \cdot c_3^3 \cdot c_4^3 = D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1 \Rightarrow \{c'_3\} \cap V \neq \phi$ and $\{c'_3\} \cap V$ extends until a leaf vertex of G. We only need to add the non-existing vertices and edges to the graph to add R_3. R_3 will add the vertex $\underbrace{D \cdot c_1^3 \cdot c_2^3 \cdot c_3^3 \cdot c_4^3}_{=D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1} \cdot c_5^3$ to V and the edge $(\underbrace{D \cdot c_1^1 \cdot c_2^1 \cdot c_3^1 \cdot c_4^1}_{=D \cdot c_1^3 \cdot c_2^3 \cdot c_3^3 \cdot c_4^3}, D \cdot c_1^3 \cdot c_2^3 \cdot c_3^3 \cdot c_4^3 \cdot c_5^3)$ to E. Now, $G = \{\mathcal{P}(R_1), \mathcal{P}(R_2), \mathcal{P}(R_3)\}$.

So far, we have considered requirements on a single path, the path of the requirement R_3 from the document vertex D. Let's consider a requirement R_4 with the clause number $c_4 = c_1^4 \cdot c_2^4 \cdot c_3^4 \cdot c_4^4$. Consider that a small part of the clause number c_4 is in V, i.e., $D \cdot c_1^4 \cdot c_2^4 = D \cdot c_1^1 \cdot c_2^1 \Rightarrow$

$\{c'_4\} \cap V \neq \phi$ and $\{c'_4\} \cap V$ extends until a non-leaf vertex of G. Adding R_4 to G will add the vertices $\{\underbrace{D \cdot c_1^4 \cdot c_2^4}_{=D \cdot c_1^1 \cdot c_2^1} \cdot c_3^4, D \cdot c_1^4 \cdot c_2^4 \cdot c_3^4 \cdot c_4^4\}$ to V and the edges $\{(\underbrace{D \cdot c_1^1 \cdot c_2^1}_{=D \cdot c_1^4 \cdot c_2^4}, D \cdot c_1^4 \cdot c_2^4 \cdot c_3^4), (D \cdot c_1^4 \cdot c_2^4 \cdot c_3^4, D \cdot c_1^4 \cdot c_2^4 \cdot c_3^4 \cdot c_4^4)\}$ to E. The requirement R_4 will be added to the vertex $D \cdot c_1^4 \cdot c_2^4 \cdot c_3^4 \cdot c_4^4$. Now, $G = \cup_{i=1}^{N=4} \mathcal{P}(R_i)$.

3.3.3.4 $\{c'_i\} \cap V = \{D\}$: Only the document vertex of the clause number is in V

Finally let us add a requirement R_5 with clause number $c'_5 = D \cdot c_1^5 \cdot c_2^5 \cdot c_3^5$ where $\{c'_5\} \cap V = \{D\}$. In other words, R_5 shares only the document vertex with the graph G. This happens when the requirement R_5 belongs to a different chapter in the RSD. In such a case, R_5 will create a fresh path from D. R_5 will add the vertices $\{D \cdot c_1^5, D \cdot c_1^5 \cdot c_2^5, D \cdot c_1^5 \cdot c_2^5 \cdot c_3^5\}$ to V and the edges $\{(D, D \cdot c_1^5), (D \cdot c_1^5, D \cdot c_1^5 \cdot c_2^5), (D \cdot c_1^5 \cdot c_2^5, D \cdot c_1^5 \cdot c_2^5 \cdot c_3^5)\}$ to E. $G = \cup_{i=1}^{N=5} \mathcal{P}(R_i)$. □

Algorithm 1 shows a simplified implementation. Steps 6 and 7 take all the four cases of $\{c'_i\} \cap V$ into consideration by removing the duplicate vertices and edges. Please refer to Sahu et al. [146] for a simplified explanation of the graph creation process.

Algorithm 1: Construction of *ReqTree*

```
   Input  : ReqList
   Output: ReqTree
1  for each clause number in ReqList do
2  |  Prefix the clause number with a document vertices;
3  |  Extract vertices and append to a list;
4  |  Extract edges and append to a list;
5  end
6  Extract all the unique vertices;
7  Extract all the unique edges;
8  Create a tree with the set of unique vertices and edges;
```

3.4 Proof: Requirement Graph is a Tree

Consider a requirement $R_i \in \mathcal{D}$, where $i \in [1, N]$. $\mathcal{D}$ represents the list of requirements extracted from a SyRSD in which the requirement R_i exists. N is the total number of requirements present in $\mathcal{D}$. The clause number of the requirement R_i is c_i, which is specified in the SyRSD as $c_i = c_1^i \cdot c_2^i \cdot ... \cdot c_{k_i}^i$. $c_{k_i}^i$ is the k_i-th token in the clause number c^i. The tokens are separated by a separator, which is a period in our representation. We assume that each requirement has a unique clause number. In case if a requirement does not have a clause number or the clause number is not

unique, we allocate a unique clause number based on its location in the SyRSD as discussed in the data preprocessing section (§3.5.1). As the requirements can belong to one of the many chapters in the SyRSD $\mathcal{D}$, the clause numbers can be completely different from each other. Thus, we prefix the clause number with the document ID D and redefine the clause number of R_i as $c'_i = D \cdot c^i_1 \cdot c^i_2 \cdot ... \cdot c^i_{k_i}$. Our network creation process is inspired by creating a tree from the *table of contents* of a document.

We define a network $G(V, E)$ that connects all the requirements from the SyRSD represented by R_i and the SyRSD node D, where V, E represents the set of vertices and edges, respectively. If we consider a single requirement R_i to construct the graph from its clause number $c'_i = D \cdot c^i_1 \cdot c^i_2 \cdot ... \cdot c^i_{k_i}$, the nodes will be the set of $V(R_i) = \{D, D \cdot c^i_1, D \cdot c^i_1 \cdot c^i_2, ..., D \cdot c^i_1 \cdot ... \cdot c^i_{k_i}\}$ and the edges will be the set of $E(R_i) = \{(D, D \cdot c^i_1), (D \cdot c^i_1, D \cdot c^i_1 \cdot c^i_2), ..., (D \cdot c^i_1 \cdot ... \cdot c^i_{k_i - 1}, D \cdot c^i_1 \cdot ... \cdot c^i_{k_i})\}$. In simpler terms, each *separator* in the clause number becomes an edge connecting nodes **A** and **B** where (1) **A** is the node that represents the part of the clause number preceding the *separator*, and (2) **B** is the node that represents the part of the clause consisting of **A** and one token succeeding the separator. We refer the node $D \cdot c^i_1 \cdot ... \cdot c^i_{k_i}$ as a *requirement node* as it corresponds to the clause number of R_i. All other nodes in $V(R_i)$, to which a requirement is not associated with, are non-requirement nodes.

The requirement R_i added $k_i + 1$ nodes and k_i edges. Therefore, $|V(R_i)| = k_i + 1$ and $|E(R_i)| = k_i$. Hence,

$$|V(R_i)| - |E(R_i)| = 1 \tag{3.3}$$

Since the difference between the number of vertices and the number of edges is equal to one, the network formed from one requirement is a tree [122, 40]. The tree is a path connecting the requirement node $D \cdot c^i_1 \cdot ... \cdot c^i_{k_i}$ to the document node D. Thus, the tree can be rooted at the node D.

Consider the addition of a new requirement $R_j \in \mathcal{D}$ with clause number $c_j = c^j_1 \cdot c^j_2 \cdot ... \cdot c^j_{k_j}$, to the graph $G(V, E)$ created using R_i. The redefined clause number is given by $c'_j = D \cdot c^j_1 \cdot c^j_2 \cdot ... \cdot c^j_{k_j}$. While adding a new requirement R_j to the graph G, we must consider the duplicate nodes and duplicate edges arising due to the common parts of the corresponding clauses. We can rewrite $c'_i = D \cdot c^i_1 \cdot ... \cdot c^i_r \cdot ... \cdot c^i_{k_i}$ and $c'_j = D \cdot c^j_1 \cdot ... \cdot c^j_r \cdot ... \cdot c^j_{k_j}$ where the clause numbers are common until the r^{th} term. Alternately, $c'_j = \underbrace{D \cdot c^i_1 \cdot ... \cdot c^i_r}_{\text{common part of the clause}} \cdot c^j_{r+1} \cdot ... \cdot c^j_{k_j}$. Upon adding R_j to G, only the new nodes and edges will be added to G. Therefore, $V(R_i, R_j) = V(R_i) \cup \{D \cdot c^i_1 \cdot ... \cdot c^j_{r+1}, ..., \mathcal{D} \cdot c^i_1 ... \cdot c^j_{k_j}\}$ and $E(R_i, R_j) = E(R_i) \cup \{(D \cdot c^i_1 \cdot ... \cdot c^i_r, D \cdot c^i_1 \cdot ... \cdot c^j_{r+1}), ..., (D \cdot c^i_1 \cdot ... \cdot c^j_{k_j - 1}, D \cdot c^i_1 \cdot ... \cdot c^j_{k_j})\}$. Now,

the number of vertices will be $|V(R_i \cup R_j)| = |V(R_i)| + |V(R_j)| - |V(R_i) \cap V(R_j)|$ and the number of edges will be $|E(R_i \cup R_j)| = |E(R_i)| + |E(R_j)| - |E(R_i) \cap E(R_j)|$. They evaluate to

$$|V(R_i \cup R_j)| = k_i + k_j - r + 1 \tag{3.4a}$$

$$|E(R_i \cup R_j)| = k_i + k_j - r \tag{3.4b}$$

Hence,

$$|V(R_i \cup R_j)| - |E(R_i \cup R_j)| = 1 \tag{3.5}$$

Thus, the tree structure remained intact upon adding one more requirement to the graph. Now, the tree consists of paths to the requirement nodes $D \cdot c_1^i \cdot ... \cdot c_{k_i}^i$ and $c_1^j \cdot c_2^j \cdot ... \cdot c_{k_j}^j$ from the document node D. Thus, the tree is rooted at D.

Consider a set of $n < N$ requirements in the SyRSD $\mathcal{D}$. The clause number c_i of the requirement R_i is rewritten as $c_i' = \underbrace{D \cdot c_1^i \cdot ... \cdot c_r^i}_{\text{common part of the clause}} \quad c_{r+1}^i \cdot ... \cdot c_{k_i}^i$. The number of nodes in the network $G(V, E)$ created by all the requirements is equal to the sum over the difference between the nodes added by a requirement reduced by the nodes created by the common segment of the clause, supplemented by the set of the common nodes to include their occurrence in the network. The sum considers the nodes that appear only once in G, whereas the supplement considers the nodes that occur more than once in G. Thus, the number of nodes in the network is

$$\left| \bigcup_{R_i \in \mathcal{D}}^{i \in [1,n]} V(R_i) \right| = \sum_{i=1}^{n} \Big((k_i + 1) - (r_i + 1) \Big) + |V(\mathcal{D}^c)| \tag{3.6}$$

where $V(\mathcal{D}^c)$ represents the vertices that occur more than once in the network form of the SyRSD. Similarly, the total number of edges in the network created by all the requirements is equal to the sum of the differences between the edges added by a requirement reduced by the edges created by the common segment of the clause, supplemented by the set of common edges to include their occurrence in the network. Mathematically,

$$\left| \bigcup_{R_i \in \mathcal{D}}^{i \in [1,n]} E(R_i) \right| = \sum_{i=1}^{n} \Big(k_i - r_i \Big) + |E(\mathcal{D}^c)| \tag{3.7}$$

We add a new requirement $R_j \in \mathcal{D}$ to $G(V, E)$ and compute the total number of edges and nodes in the modified network. The analogous equations for the new network will be

$$\left| \bigcup_{R_i \in \mathcal{D} \cup \{R_j\}}^{i \in [1,n]} V(R_i) \right| = \sum_{i=1}^{n} \Big(k_i - r_i\Big) + (k_j - r_j) + |V(\mathcal{D}^c)| \tag{3.8}$$

$$\left| \bigcup_{R_i \in \mathcal{D} \cup \{R_j\}}^{i \in [1,n]} E(R_i) \right| = \sum_{i=1}^{n} \Big(k_i - r_i\Big) + (k_j - r_j) + |E(\mathcal{D}^c)| \tag{3.9}$$

It is important to note that the last terms in Equations 3.8 and 3.9 reflect the vertices and edges in the network which occur more than once in $\mathcal{D}^c \cup R_j$. As those common nodes occur in $\mathcal{D}^c$ and R_j, adding R_j does not make a difference to $\mathcal{D}^c$. Thus, the number of common nodes in $\mathcal{D}^c \cup R_j$ equals the number of nodes that occur more than once in $\mathcal{D}^c$.

Upon comparing the difference between Equation 3.8 and 3.9 with the difference between 3.6 and 3.7, we can conclude that

$$\begin{aligned} &\left| \bigcup_{R_i \in \mathcal{D}}^{i \in [1,n]} V(R_i) \right| - \left| \bigcup_{R_i \in \mathcal{D}}^{i \in [1,n]} E(R_i) \right| \\ =&\left| \bigcup_{R_i \in \mathcal{D} \cup \{R_j\}}^{i \in [1,n]} V(R_i) \right| - \left| \bigcup_{R_i \in \mathcal{D} \cup \{R_j\}}^{i \in [1,n]} E(R_i) \right| \\ =&|V(\mathcal{D}^c)| - |E(\mathcal{D}^c)| \end{aligned} \tag{3.10}$$

Thus, the first equality reflects that the difference between the number of vertices and edges in the requirements network remains unchanged upon adding a new requirement from the SyRSD. In other words, the difference between the number of vertices and edges in the network is independent of the number of requirements in the SyRSD. The second equality reflects that the difference between the number of vertices and edges in the two stages of the network remains constant and is equal to the difference between the number of common vertices and common edges in the network. Equations 3.3 and 3.5 show that this difference equals 1. Thus, for any set of requirements in the SyRSD,

$$\left| \bigcup_{R_i \in \mathcal{D}}^{i \in [1,N]} V(R_i) \right| - \left| \bigcup_{R_i \in \mathcal{D}}^{i \in [1,N]} E(R_i) \right| = 1 \tag{3.11}$$

Thus, the difference between the number of vertices and edges in the network created by the requirements from a SyRSD is equal to one. Hence, the network created by the requirements of a SyRSD is a tree [122, 40]. This tree consists of paths from each of the requirement nodes to the

document node D. □

We refer to this *tree* of *req*uirements as **ReqTree**, and denote as T. Each RSD gives rise to a *ReqTree* with the document vertex serving as the root vertex. The *ReqTree* is essentially a collection of paths from the requirement vertices to the root vertex, representing the overall document structure. The requirements listed under sections and subsections of the RSD become the leaf vertices, while the sections and subsections themselves form the branch vertices. The absence of the document vertex would create a forest, with each tree rooted at the chapter vertices. As we parse the clause numbers of requirements, the ancestors of the requirement vertices (such as sections and subsections) are also added to the *ReqTree*, resulting in a structure that often contains more vertices than the number of requirements in the RSD. This is because the branch nodes correspond to the system's subsystems, components, functions, or operational states, as specified in the RSD. Each edge in the *ReqTree* reflects the decomposition of a parent vertex's role into child vertices. The high-level subsystems, functions, and operational states decompose into lower-level ones, illustrating the requirement allocation process. Having presented the graph properties embedded in RSDs, in the next sections we apply these concepts to two real-life settings.

3.5 Results and Discussion

3.5.1 Data Preprocessing

The clause numbers of the requirements in RSDs are the foundation of the *ReqTree*. The data preprocessing (1) extracts requirements with their corresponding clause numbers from the RSDs and (2) ensures that every requirement in the RSD has a unique clause number. The requirements in RSDs are located in one or more chapters dedicated to requirements [53]. We adopted a semi-automatic method to extract the requirements from the RSDs. We converted the RSDs from native formats (e.g., PDF, DOC, HTML) into pure text form. Then, we manually extracted the requirements. We only retained the requirement statements and their corresponding clause numbers. We discarded the chapters and sections unrelated to requirements. Headers, footers, tables, flow charts, and images are ignored. To ensure each requirement has a unique clause number, we allocated unique clause numbers to requirements if they lack a clause number. If multiple requirements are grouped under the same clause number, we split them into multiple requirements with unique clause numbers. If two requirements are packed under clause number `3.5`, then we split those requirements

and allocated the clause numbers `3.5.1` and `3.5.2` to them. If two requirements are packed under clause number `3.5`, and clause number `3.5.1` also has a requirement, then we split the requirement and allocate them to clause numbers `3.5.0.1` and `3.5.0.2`. If the multiple requirements (packed under bullet points) cannot be split due to common context (e.g., subject, condition, state), we pack them under the same clause number. We refer the reader to our previous work [146] for a detailed overview of the data preprocessing.

3.5.2 Case 1: Mobile Surveillance System

We extract the tree structure from the RSD of the mobile surveillance system (MSS) [164]. The MSS RSD specifies the requirements for procuring mobile surveillance systems by the International Organization for Migration (IOM). The MSS will support the Republic of Macedonia to manage its southern border to manage the European migration crisis. The MSS RSD has 311 requirements. Its tree structure has 351 nodes, which indicates that the additional 40 nodes correspond to the sections and subsections of the MSS RSD. The tree is depicted in Figure 3.3. It has a radial layout. The requirement nodes are the leaf nodes. They are colored black. The tree is rooted at the document node D. The requirements are specified in five chapters: `Carrier Functional Requirements` (Chapter 3: Vertex $D.3$), `Sensor Suite Requirements` (Chapter 4: Vertex $D.4$), `System Control Requirements` (Chapter 5: Vertex $D.5$), `Communications` (Chapter 6: Vertex $D.6$), `Logistics and Non-functional Requirements` (Chapter 7: Vertex $D.7$). The first level of the hierarchy is created by functional decomposition, where the requirements are classified according to their relevance to mobility, sensing, control and communication functions. The requirements that are not related to these four functions are documented in Chapter 7. The Carrier functional requirements are decomposed into the requirements of the vehicle ($D.3.1$), body ($D.3.2$), engine ($D.3.3$), drive system ($D.3.4$), suspension ($D.3.5$) and brake system ($D.3.6$). This decomposition of the vertex $D.3$ into its child vertices shows that the decomposition is based on the system architecture. Similarly, Chapters 4, 5 and 6 also decompose the system architecture. On the other hand, it is hard to coin a label for the criteria behind the organization of requirements in Chapter 7. The requirements in Chapter 7 pertains to documentation ($D.7.1$), maintenance ($D.7.2$), warranty ($D.7.3$), endurance and reliability ($D.7.4$), training ($D.7.5$), transportability and delivery ($D.7.6$), verification ($D.7.7$) and copyright ($D.7.8$). We label the criteria with a generic label, 'logical decomposition'.

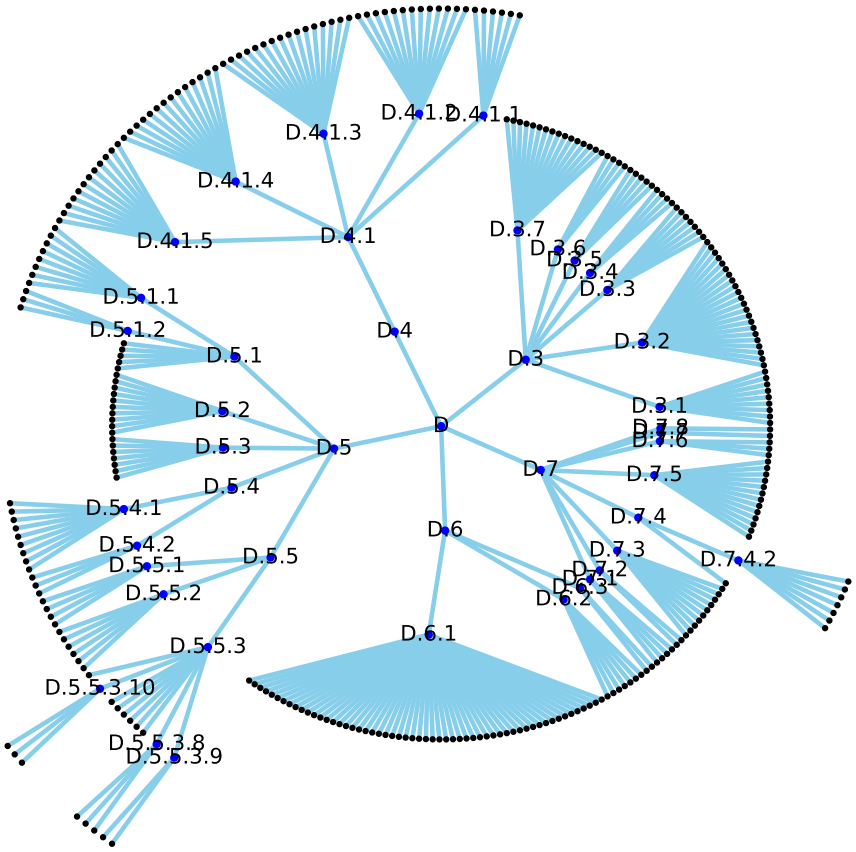


Figure 3.3: The tree structure extracted from the RSD of the mobile surveillance system

3.5.3 Case 2: Connected Vehicle Pilot Deployment Program

Our second case study focuses on the RSD of phase I of New York City's Connected Vehicle Pilot Deployment (NYC-CVPD) program [59]. The tree extracted from the RSD of the NYC-CVPD program is shown in Figure 3.4. It has 456 nodes consisting of 395 requirement nodes and 61 section nodes. The black nodes are requirements. It uses a force-directed layout to reduce clutter. Here, the requirements are documented under two chapters: `System Capabilities, Conditions, and Constraints` (Chapter 3: Vertex $D.3$) and `System Interfaces` (Chapter 4: Vertex $D.4$). Thus, the first level in the hierarchy follows a logical decomposition, where all the system-related requirements are documented under Chapter 3 and all the interface-related requirements under Chapter 4. `Section 3.1: Physical` ($D.3.1$) follows a logical decomposition where it gets decomposed into construction ($D.3.1.1$), durability ($D.3.1.2$), adaptability ($D.3.1.3$) and environmental conditions ($D.3.1.4$). The vertex $D.3.1.1$ gets further decomposed into mechanical and electrical requirements

following a logical decomposition. The vertex $D.3.8.2$ (V2V application) is decomposed into six systems (e.g., `D.3.8.2.1: forward crash warning system`, `D.3.8.2.2: emergency electronic brake light system`) by following a system decomposition approach. The vertex $D.4$ is split into twelve nodes that specify the interface requirements of various components of NYC-CVPD and their external interfaces. □

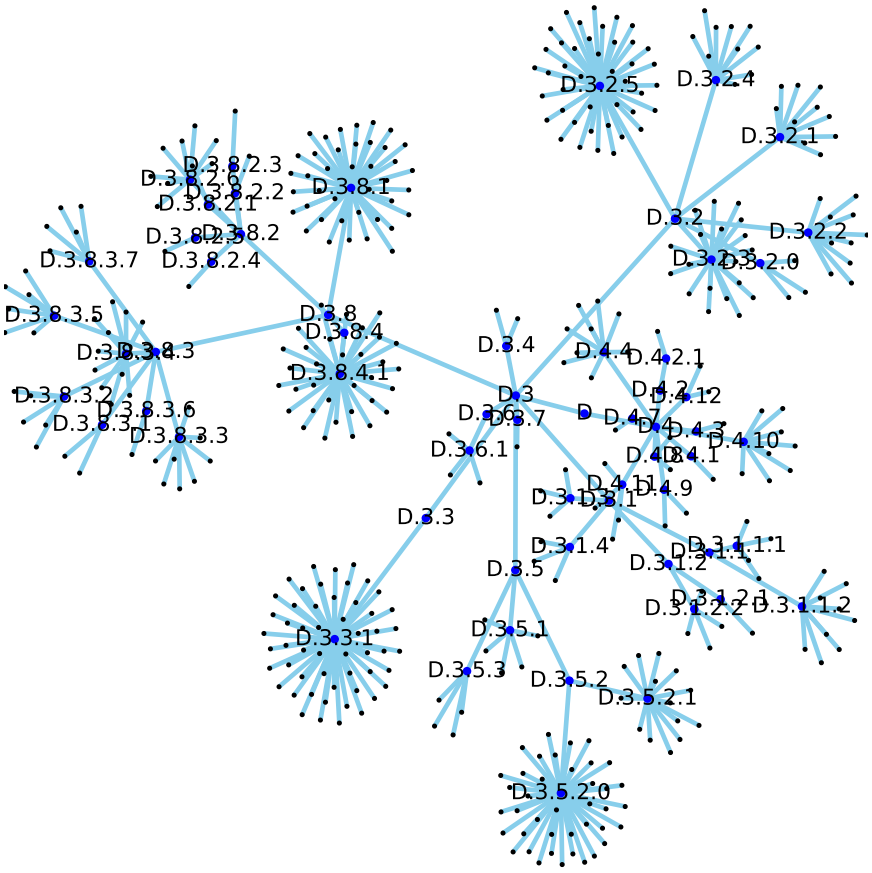


Figure 3.4: The tree structure extracted from the RSD of New York City's Connected Vehicle Pilot Deployment

The MSS RSD used functional decomposition, system decomposition and logical decomposition as its criteria for allocation. The NYC-CVPD RSD used logical decomposition, system decomposition and interface decomposition criteria. These two case studies show that RSDs do not restrict themselves to only one mode of decomposition. Any allocation criterion can be chosen at a vertex during its decomposition.

Table 3.1: Decomposition criteria of the requirement allocation process in various RSDs

Name of RSD	System	Function	Scenario	State	Logical	Interface	Others
1998 - themas		Y					
2002 - evla corr					Y	Y	Ilities
2017 - NISTMfgData		Y			Y		Ilities
2000 - Barrel					Y		
2010 - fishing		Y					
2010 - mashboot					Y		
2006 - stewards		Y			Y	Y	Ilities
2004 - SGVTraffic		Y			Y		
2021 - ReqView		Y					
2001 - ctc network					Y	Y	
2010 - home 1.3	Y				Y		Object (User)
2002 - evla back						Y	Ilities
2003 - pnnl		Y			Y		Ilities
2007 - nde				Y	Y		
0000 - cctns		Y			Y		
1995 - Landsat7	Y						
2005 - phin					Y		
2007 - Wildlife		Y					Stimulus-Response
2005 - clarus high					Y		
2004 - sprat		Y					
2012 - EMR HHS					Y		
2000 - nasa x38		Y			Y		
2004 - rlcs		Y					Ilities
2012 - EMR HL7 IN		Y			Y		
2007 - ertms		Y					
1995 - gemini		Y					Ilities
2005 - pontis		Y			Y		
2004 - grid bgc		Y			Y		Object
2019 - MOSAR	Y				Y		
2009 - peppol	Y	Y			Y		
2018 - DataWarehouse			Y		Y		
2011 - CCHIT		Y					
2022 - MobileSurveillance	Y				Y		
1999 - Defense	Y	Y			Y		
2012 - EMR Pharmacy					Y		
2012 - EMR HL7 DC		Y					
2021 - NYC-CVPD		Y			Y	Y	
2006 - eirene sys 15	Y	Y		Y		Y	
1999 - tcs		Y		Y		Y	Ilities, Resource
2007 - eirene fun 7	Y	Y				Y	

3.5.4 RA in Open-Source RSDs

We have investigated the tree structure of 84 other RSDs[1] [146]. The dataset is quite diverse as it includes RSDs from 11 industrial sectors categorized by the North American Industry Classification System (NAICS) [161]. Our dataset broadly covers entertainment, construction, education, health care, information technology, manufacturing, science, public administration, retail, transportation and utilities sectors. The results of 40 RSDs are listed in Table 3.1 for brevity. We observed that the decomposition criteria could be system decomposition (System), functional decomposition (Function) or decomposition of use-cases (Scenario), states (State), interfaces, objects, ilities (qualITY attributes such as maintainabilITY, durabilITY, reliabilITY, usabilITY) or any other logical (Logical) or semantic criteria. The results reinforce our observation that the tree structure of RSDs reflects the requirement allocation process. The criteria of decomposition in an RSD need not be unique.

3.5.5 Limitations

Hierarchical representation allows sub-systems to be developed simultaneously since leaf node system elements are generally independent. However, this independence can lead to optimization in isolation, where each element is optimized without considering the overall performance of the integrated system. For instance, individual sub-systems might achieve optimal power consumption, but the overall system may not be fully optimized. To mitigate this issue, interactions and interdependencies between system elements should be incorporated into the hierarchy. These interactions appear as cross-references and interface requirements. For example, in the NYC-CVPD, `ReqID: 405.1.10`[2] with clause number `3.5.2.0.27` refers to `ReqID 405.1.9`[3] with clause number `3.5.2.0.26`. The requirement `401.17.1`[4] with clause number `4.1.1` involves the interaction between `DSRC` and `GNSS` systems.

Incorporating these dependencies into the ReqTree introduces additional links that can create loops, disrupting the tree structure. To mitigate this, the requirements can be rewritten using EARS patterns [109] by switching the subject and the object based on the SoI to recover the tree

[1] Available at `https://zenodo.org/doi/10.5281/zenodo.10976096`

[2] If the ASD RF log files exceed the space allocated (Req 405.1.9), then the oldest data shall be written over without damaging newer log files.

[3] The ASD RF log space shall be sufficient to store 7 days of interactions with 3,000 ASDs and 450 RSUs, equivalent to about 2,000 entries per day.

[4] Each DSRC device shall obtain its time and position from the GNSS per the requirements of J2945/1 Section 6.2.

structure. The requirements become conditional based on a state, event, unwanted behavior or an optional feature. This will introduce additional requirements to remove the coupling of system elements. During the design, integration and V&V of interacting requirements, any system element other than the SoI becomes part of the environment. Complex dependencies, where a single system fulfills multiple requirements or multiple systems meet a single requirement, are common in networked systems and systems of systems [39, 55]. While these interdependencies can be managed by refining requirements to lower levels of abstraction, we do not claim the generalizability of our work to them as we did not find any relevant RSD. Lastly, transforming the design process into an optimization problem [179] and adopting an integrative architecture with many-to-many mappings between requirements, functions, and system elements [18] can overcome the limitations of a purely hierarchical approach, resulting in a more comprehensive and effective system design.

3.6 Conclusion

Systems are developed to satisfy the stakeholders' needs. The informal needs of the stakeholders are formalized as requirements based on which the system is developed. The system development process starts with the elicited requirements and terminates upon validating if the created system satisfies the elicited requirements. The requirement definition process involves decomposing complex high-level requirements into simpler low-level requirements, which can be allocated to functions or system elements. The process terminates with requirements allocated to system elements, which can be procured, built, coded or reused. The final set of requirements is documented in requirement specification documents (RSD), which organize the requirements in a hierarchical structure by allocating a unique clause number to each requirement.

We extract the hierarchical tree structure of the requirements from the RSDs. We leverage the clause numbers of the requirements to create a network of the requirements. We mathematically prove that the graph is indeed a tree where the leaf nodes correspond to requirements. The branch nodes correspond to the sections and subsections of the RSD, which reflect how the requirements are decomposed. The links allocate the role of the parent node among the child nodes. This tree reflects the requirement allocation process. We investigate the trees of 86 RSDs and detail the cases of two RSDs: the mobile surveillance system (MSS) and New York City's connected vehicle pilot deployment (NYC-CVPD) program. Our investigation revealed that the RSDs may

use multiple criteria to allocate the requirements. Each branch node chooses one of the criteria from function decomposition, system decomposition, scenario decomposition, state decomposition, or decomposition based on any other logical/semantic criteria.

Recovery of the requirement allocation process from the RSDs can offer insights into the legacy systems for their maintenance, repair, improvement and reverse engineering. Development of new systems based on legacy systems will be easier. Lastly, since organizations treat requirements as proprietary information and do not share them, our work will accelerate the research on requirement allocation by mapping the requirements to corresponding system elements, managing variants in the same product family, and budgeting requirements based on performance, resources and quality attributes. The automated identification of the decomposition criteria at each node and moving to budgeting of requirements from allocation are a few directions for future research.

Chapter 4

ReqList, ReqNet and ReqSim: Datasets of Requirements

4.1 Introduction

All the RE tasks are contingent upon acquiring a set of requirements during the elicitation process. The lack of a RE dataset and a method to create any RE dataset is evident by the plenty of research papers focusing on identifying requirements from source documents by classifying if a particular sentence is a requirement or not [68, 191]. Unlike the abundance of NL corpora, requirement datasets are scarce because of the following reasons. First, as requirements are a product of system ideation, having access to detailed requirements can disclose the system under development. The organization risks losing its competitive edge. Second, it is a challenging task to acquire a set of well-formed requirements with all the characteristics of ISO/IEC/IEEE 29148:2018 [166]. Third, the creative nature of the requirement elicitation process and the ensuing human involvement during elicitation limits the number of elicited requirements. Fourth, having a dataset that is usable for all the tasks of RE is challenging. We need to create multiple variants of the dataset to enable multi-faceted analysis. Fourth, as the requirements vary depending on its domain and application, annotating the requirements for any specific analysis (such as traces, dependence) becomes a resource-intensive task. Thus, most of the existing RE work happened in silos with small requirement datasets. Ref. [52] emphasized the necessity of a good requirement dataset to foster

progress in RE.

The overarching goal of our research is to offer a multipurpose requirement dataset in order to accelerate requirement analysis. As the requirements are elicited in natural language, they call for *natural language processing* (NLP) techniques for analysis [196]. The requirements are recorded in a system requirement specification document (SyRSD). Even though matrix-based representations are widely used to represent requirements [183], a *hierarchical tree structure* was endorsed by the NASA System Engineering Handbook [74] and the INCOSE Needs, Requirements, Verification, Validation Lifecycle Manual [186]. The tree structure opens the portal for deploying *network analysis* techniques. Furthermore, the limited capability of rule-based NLP techniques has plateaued their utility for requirement analysis, especially in RE applications involving the semantics of requirements. *Artificial intelligence* (AI) techniques emerged as a promising solution to capture the *semantics in RE* [190]. Nonetheless, the lack of a pertinent requirement dataset hinders the full-fledged adoption of AI in RE.

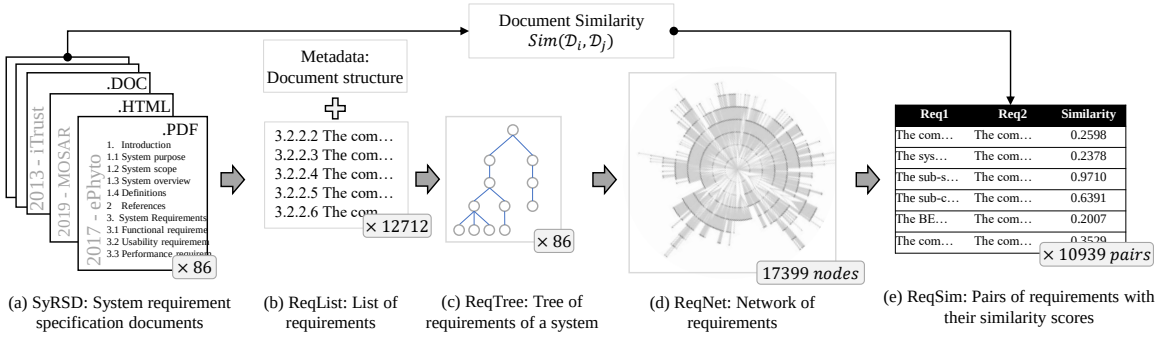
We create the requirement dataset from the SyRSDs. Our dataset is underpinned by the tree structure extractable from the SyRSDs. Our work attempts to address the fourth research direction from the review of semantic networks for engineering design by Han et. al. [69], a comprehensive knowledge graph of requirements with a framework to extend it further. We seek the answers to the following research questions (RQ). RQ1: How to represent the requirements in SyRSDs? Can a tree structure be extracted from the requirements in SyRSDs? RQ2: How to compute the semantic similarity between pairs of requirements? To answer these RQs, we adopt a semi-automatic method [88] to create an open-source[1] [143] requirement dataset by reusing requirements from existing open-source SyRSDs. The process is illustrated in Figure 4.1. While doing so, we make the following contributions:

1. We devise a method to represent the requirements from a SyRSD as a tree. We refer to these requirement trees as *ReqTree*s.

2. We formulate a method to compute the semantic similarity between pairs of requirements from the SyRSDs.

3. We release three different forms of data to accelerate requirement analysis.

[1]The dataset is available at `https://github.com/ChandanKSahu/ReqList_ReqNet_ReqSim` with DOI:10.5281/zenodo.10976096



(a) SyRSD: System requirement specification documents

(b) ReqList: List of requirements

(c) ReqTree: Tree of requirements of a system

(d) ReqNet: Network of requirements

(e) ReqSim: Pairs of requirements with their similarity scores

Figure 4.1: Computational framework to create *ReqList*, *ReqTree*, *ReqNet* and *ReqSim* from SyRSDs

(a) *ReqList*s: lists comprising of 12701 requirements in pure text form acquired from 86 SyRSDs. They only contain the requirements and corresponding clause numbers to retain their structure. Their document structure supplements them as the metadata. *ReqList*s are ready to use NLP and unsupervised machine learning (ML) techniques without further data preprocessing.

(b) *ReqNet*: a large network of requirements consisting of 17375 nodes to facilitate graph-theoretic investigations for RE. It assembles the *ReqTree*s of 86 distinct systems.

(c) *ReqSim*: a dataset consisting of 10933 pairs of requirements annotated with their similarity scores. The similarity scores align with human knowledge [104]. *ReqSim*, the first dataset that estimates the pairwise similarity among requirements, enables sentence-level supervised learning tasks.

4.2 Data sources

4.2.1 Requirement Specifications

Our principle of creating this dataset is based on requirement reuse. In accordance with the principles of systematic literature review [130], we have queried for the keywords (ϕ OR *System* OR *Software*) AND (*Requirement Specification*) OR (*Document*) OR (ϕ OR *PDF* OR *WORD* OR *HTML*). Additional queries include *Requirement Dataset* and *SRS*. The search string in each query consists of multiple keywords. One keyword is selected from the keywords within each parentheses based on the 'OR' condition. The selected keyword is included/excluded from the search string

based on the 'AND/OR' condition preceding the enclosing parentheses. An example query is 'System Requirement Specification Document'. It includes a keyword from the first, second and third parentheses. It excludes the keyword from the fourth parenthesis using the 'OR' condition preceding it. ϕ indicates that the keyword from the parentheses is optional, as in the query 'Requirement Specification Document' which makes the first keyword optional. The queries have been made in Google Scholar[2], Google Search[3], Google Dataset Search[4], GitHub[5], ACM Digital Library[6], IEEE Xplore[7], ScienceDirect[8] and SpringerLink[9]. We excluded the results from the search based on the following criteria. First, the search results that lacked a structure in the SyRSDs are discarded, as the structured clause numbers are crucial to our tree structure. Second, the SyRSDs that focus exclusively on use-case scenarios where the requirement statements omit the subject or object of the sentences are excluded. Third, we excluded the SyRSDs created as part of capstone projects. We included only six SyRSDs created from capstone projects that had a consistent structure for diversity. Based on the number of SyRSDs retained, Google Search produced the best results, followed by Google Dataset Search and Github. While GitHub produced the most results, almost all were capstone projects, which led to their exclusion. Our search results have discovered the PURE dataset [53]. The exclusion criteria rejected 25 out of the 79 documents from the PURE dataset. We added 32 more SyRSDs and created a dataset comprising 86 SyRSDs.

4.2.2 *ReqList*: Preprocessed *List* of *Req*uirements

The requirements are organized logically in the source SyRSDs to ease their interpretation to the reader [41]. Predominantly, the system/software RSDs follow IEEE Std 1233-1998 [162] and IEEE Std 830-1998 [163] guidelines irrespective of the form (such as PDF, HTML, DOC, XLS) of the SyRSDs. SyRSDs graciously use tables, sub-clauses, figures, flow charts, and cross-references to ease interpretation and texts in different forms (bold, italics, underlines) to emphasize. They vary significantly from each other in terms of their structure, *e.g.*, order, and inclusion of header/footer. Over two-thirds of the SyRSDs contained unstructured content [53]. Additionally, the requirement

[2] `https://scholar.google.com/`
[3] `https://www.google.com/advanced_search`
[4] `https://datasetsearch.research.google.com/`
[5] `https://github.com/`
[6] `https://dl.acm.org/search/advanced`
[7] `https://ieeexplore.ieee.org/search/advanced`
[8] `https://www.sciencedirect.com/search/entry`
[9] `https://link.springer.com/advanced-search`

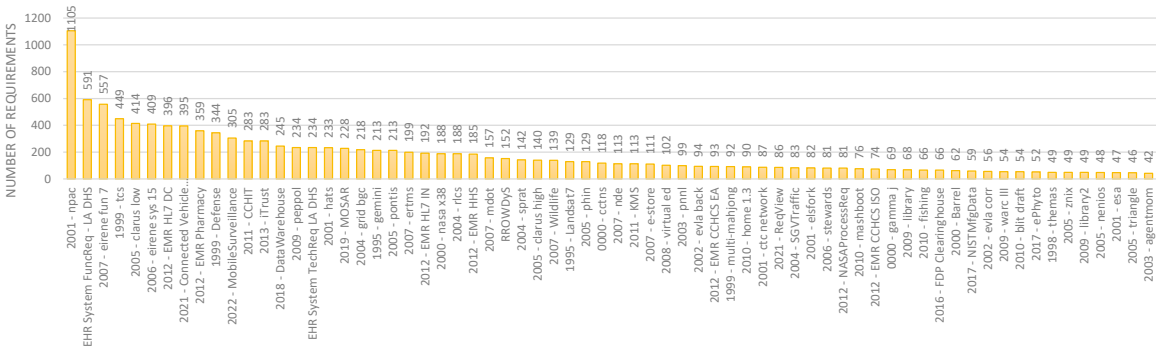


Figure 4.2: Distribution of requirements in *ReqList*s (Shows requirements from 68 RSDs only. Excluded 18 RSDs with < 40 requirements for legibility.)

statements contain modal verbs such as 'has', 'have', 'must', and 'should' in addition to 'shall'. These facets hinder the deployment of any standard NLP technique to extract the requirements. Thus, we adopted a semi-automatic method to extract requirements from these documents [88]. We only used automated text processing techniques to convert the PDF/DOC/ HTML documents into pure text form. We extracted the requirements manually.

The automated conversion to pure text form has a few unique characteristics requiring manual cleaning. It discards raster images. It extracts the text from vector images (*e.g.*, flow charts) and tables by ignoring their structure. We discarded such information completely. We initiated the manual processing by comparing the extracted texts with their source SyRSDs. We removed the titles, table of contents, headers and footers, and other texts which are not requirements. Annotations, such as 'M' for mandatory and 'O' for optional requirements (*e.g.*, 2007 - ertms.pdf, 2014 - Minutia.pdf) or traces (*e.g.,* 2021 - Connected Vehicle Pilot NYC.pdf, 1997 - Modis.pdf), are ignored. Often, the documents contain requirements with sub-clauses and bullet points. If they cannot be split into different requirements due to their common context, we packed the bullet points in a single clause. The conversion from PDFs to texts accrued noise, such as fragmented words, additional line breaks, and conjoined words. The line breaks were removed. Fragmented words were merged, and conjoined words were split using a word processor.

During the manual cleaning of the extracted texts, we retained the structure of the RSDs by keeping the hierarchical order within the documents. As the structure is reflected by the hierarchical ordering (sections, subsections, clause numbers, etc.), we retained the section/clause numbers. In the absence of a unique clause number or if there are multiple requirements in a single clause,

we allocate a unique clause number to each requirement based on its position in the SyRSD. For instance, if clause 2.1 has two requirements, we used 2.1.1 and 2.1.2 to have a unique clause number corresponding to each of those requirement. If clause 2.1 contains two requirements and 2.1.1 also includes a requirement, we used 2.1.0.1 and 2.1.0.2 to allocate unique clause numbers to them. This structure is crucial because it is created by a domain expert who has organized the document logically [108, 87] to ease comprehension.

After preprocessing, we created a clean *list of requirements* from each SyRSD with their corresponding unique clause numbers, preserving the structure of the SyRSD. It is referred to as **ReqList**. Each of the *ReqList* is supplemented with its metadata to capture the contextual information. The metadata file includes the hierarchical structure (*e.g.*, system $\rightarrow$ sub-system $\rightarrow$ component) of the SyRSD. The number of requirements extracted from each source document is shown in Figure 4.2. The 54 SyRSDs from the PURE [53] dataset produced 7310 requirements even though they include 34268 sentences. Our 32 SyRSDs added 5391 requirements. *ReqList* has 12701 requirements from 86 systems.

4.3 Methodology

4.3.1 ReqTree: Tree of Requirements from a SyRSD

We create a requirement tree for each SyRSD. Our network creation process is inspired by creating a tree from the *table of contents* of a document. We illustrate the tree creation process by considering a set of requirements as shown in Figure 4.3. This is a simpler, less formal, explanation of the tree creation process described in §3.3.

Consider a requirement R_1 with clause number $c_1.c_2.c_3.c_4$. Here, R_1 belongs to the chapter c_1 and lies in the sub-sub-section $c_1.c_2.c_3$. We prefix the clause number with the document node making the clause number as $D.c_1.c_2.c_3.c_4$. This clause creates the nodes D, $D.c_1$, $D.c_1.c_2$, $D.c_1.c_2.c_3$ and $D.c_1.c_2.c_3.c_4$ in the tree where the node $D.c_1.c_2.c_3.c_4$ corresponds to R_1. The node $D.c_1.c_2.c_3.c_4$ is a requirement node and others are non-requirement nodes. It creates the edges $(D, D.c_1)$, $(D.c_1, D.c_1.c_2)$, $(D.c_1.c_2, D.c_1.c_2.c_3)$, $(D.c_1.c_2.c_3, D.c_1.c_2.c_3.c_4)$ by connecting each pair of nodes according to their order in the clause $D.c_1.c_2.c_3.c_4$. Essentially, each separator becomes an edge. This process creates a path from the requirement node $D.c_1.c_2.c_3.c_4$ to the document node D where the path passes through the ancestor nodes of the requirement. The path is a tree rooted at

the document node D.

Clause Number	Requirement
$c_1.c_2.c_3.c_4$	R_1
$c_1.c_2.c_3.c_4.c_5$	R_2
$c_1.c_2.c_3$	R_3
$c_1.c_2.c_6.c_7$	R_4
$c_8.c_9.c_{10}$	R_5

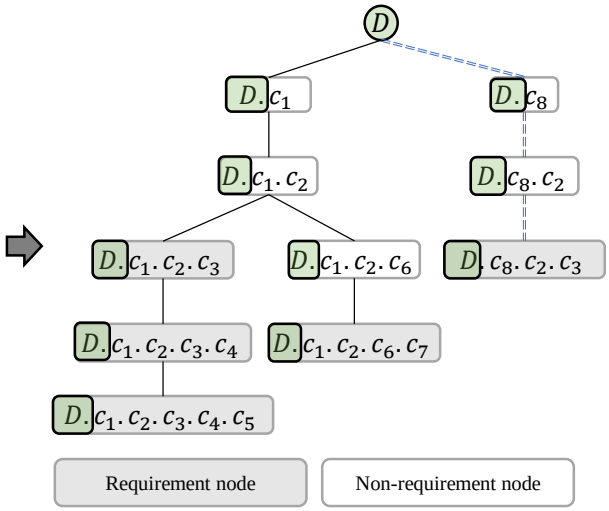


Figure 4.3: Illustration of the tree creation process using a set of sample requirements.

The same method is followed for all the clauses in the extracted list of requirements. The succeeding requirements will use the nodes/edges already appended to the tree, and non-existing nodes/edges will be added to the tree to add new requirements. The requirement R_2 with clause number $c_1.c_2.c_3.c_4.c_5$ will create only one new node $D.c_1.c_2.c_3.c_4.c_5$ and add the requirement R_2 to it. It will create a new edge $(D.c_1.c_2.c_3.c_4, D.c_1.c_2.c_3.c_4.c_5)$ to connect to the tree. The requirements R_3 with clause number $c_1.c_2.c_3$ will not add any new node or edge as the corresponding node is already in the tree. R_3 will be added to the node $D.c_1.c_2.c_3$. If no requirement exists for the clause $c_1.c_2.c_3$ in the extracted list of requirements, the node $D.c_1.c_2.c_3$ will remain as a non-requirement node. The requirement R_4 with clause number $c_1.c_2.c_6.c_7$ will add the nodes $D.c_1.c_2.c_6$ and $D.c_1.c_2.c_6.c_7$ and the edges $(D.c_1.c_2, D.c_1.c_2.c_6)$, $(D.c_1.c_2.c_6, D.c_1.c_2.c_6.c_7)$. Now, the requirement R_5 with clause number $c_8.c_2.c_3$ belongs to chapter c_8 in the document D. Adding R_5 will add the nodes $D.c_8$, $D.c_8.c_2$ and $D.c_8.c_2.c_3$ and the respective edges as shown in Fig. 4.3. This tree, rooted at the document node D, has two subtrees. The first sub-tree is rooted at the node $D.c_1$ corresponding to Chapter c_1 in the SyRSD in which the requirements R_1, R_2, R_3 and R_4 exists. The other sub-tree is rooted at the chapter node $D.c_8$ to which the requirement R_5 belongs. Exclusion of the document node D would create a forest with each tree rooted at the chapter nodes. We refer to this *tree* of *req*uirements as **ReqTree**, and denote as T.

4.3.2 ReqNet: Network of Requirements from multiple SyRSDs

The distance between a pair of requirements in the *ReqTree*, T, allows us to compare any pair of requirements from the same SyRSD. To compare a pair of requirements spanning across two different systems, we need to connect those systems. We achieve this by joining the trees. We define a global node (G) as the root node to connect the *ReqTree*s of any two systems. The underlying idea of connecting the trees is that all the systems are components of a global system. This idea is inspired by computing the distance between disconnected synsets in WordNet [114], where WordNet uses a simulated root node to compute the distance which is otherwise infinity [62]. Moreover, we connect *ReqTree*s from multiple SyRSDs to the global root node to form a huge requirement network to compare any pair of requirements. We refer to this requirement network as **ReqNet**, denoted as $\mathcal{N}$. *ReqNet* is an unweighted undirected tree. Mathematically, the *ReqNet* can be expressed as

$$\mathcal{N} = \bigcup_{i=0}^{\mathbf{N}} \{G\} \cup T_i \tag{4.1}$$

where $\mathbf{N}$ represents the number of SyRSDs.

4.3.3 Weighted Tree

The similarity between a pair of requirements can be computed from the distance between the corresponding nodes in *ReqNet*. The distance in *ReqNet* reflects the number of edges to be hopped to reach one requirement from another. The number of edges to be traversed is a coarse measure of distance. It has a significant shortcoming. The unweighted distance ignores the difference between comparing a pair of requirements within a *ReqTree* and across a pair of *ReqTree*s (See Figure 4.4). It lays equal importance to both cases. Conceptually, requirements within the same *ReqTree* are more similar than those across different *ReqTree*s as the latter pertain to different systems whereas the former is from the same system. The latter should produce a larger distance. Thus, we convert the unweighted *ReqTree*s to weighted *ReqTree*s.

The unweighted *ReqTree* can be converted to a weighted *ReqTree* by leveraging the requirement allocation process [186, 56, 78]. We adopt two methods to acquire the weighted trees. Our first consideration is that 'all the non-reducible requirements are allocated with equal importance.' The leaf nodes are the non-reducible nodes in the *ReqTree*. Thus, we allocate a unit weight to each of the edges connecting the leaf nodes of the *ReqTree* and propagate the weights until the

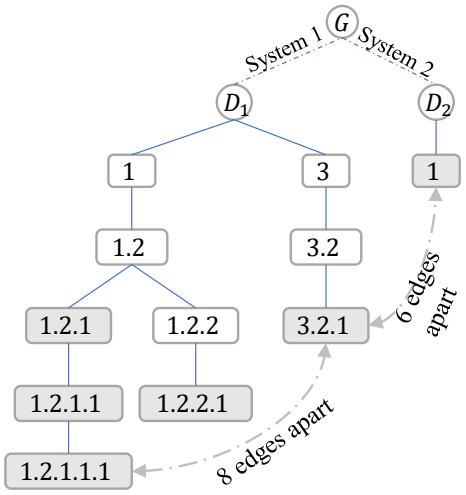


Figure 4.4: Comparison of unweighted distance from 3.2.1 to 1.2.1.1.1 (within the a *ReqTree*) with 3.2.1 to 1 (across *ReqTrees*) in a notional *ReqTree*

root node. This consideration adopts a bottom-up mode of weight update. The bottom-up allocation mimics the system design behaviour where the system is designed based on existing or reusable components. Reverse engineering [51], component analysis and interface analysis methods give rise to a bottom-up requirement allocation. The second consideration is that 'all the requirements allocated from the same parent requirement are allocated with equal importance.' We start by allocating a unit weight to the edges connected to the root node of the *ReqTree*. Then, we distribute the weight equally among its child nodes until we reach the leaf nodes. The second consideration translates to a top-down mode of weight update. The top-down approach mimics the system design behaviour where the system is designed using functional analysis or structural analysis. The higher-level requirements/systems/functionalities are decomposed into a set of low-level requirements/systems/functionalities that satisfy the parent node upon composition. The weight update starts by considering the unweighted undirected *ReqTree* (UUT) as an unweighted directed *ReqTree* (UDT) and ends as a weighted undirected *ReqTree* (WUT). Mathematically, the weight update process starts with nodes with in-degree zero and terminates with nodes with out-degree equal to zero during execution in either mode.

4.3.3.1 Bottom-Up Mode

We temporarily consider the UUT as a UDT where the edges are directed away from the leaf nodes *towards* the root node. The weight update process starts at the leaf nodes in the bottom-up

UDT. The in-degree of the leaf nodes is 0, and the out-degree of the leaf nodes is 1. A unit weight is allocated to each edge outgoing from the leaf nodes as all the leaf nodes are equally important. These are the same edges that are incoming to the parents of the leaf nodes. The in-degree of these branch nodes is ≥ 1, whereas their out-degree equals one. Then, we update the weight of the only outgoing edge from the parent nodes with the sum of the weights of the incoming edges. The process continues, and terminates upon reaching the root node, whose out-degree is zero. The weight update process can be mathematically written as

$$w_{bu}(v^{out}) = \sum_{v \in T} w_{bu}(v^{in}) \tag{4.2}$$

where, w represents the weight; v^{out} and v^{in} represent the edge outgoing and incoming to the vertex v in the *ReqTree*, T. The subscript *bu* denotes the bottom-up mode of weight update. The bottom-up mode of weight update translates to an *additive* mode of weight update.

4.3.3.2 Top-Down Mode

Alternately, in the top-down mode, we consider a UDT where the edges are directed *away from* the root node towards the leaf nodes. The weight update starts at the root node, whose in-degree is zero. We allocate a unit weight to each edge outgoing from the root node. The root node's child nodes (or branch nodes) have an in-degree of 1 but an out-degree of ≥ 1. As all the child nodes of a node are equally important, the weight of the incoming edge of the node is equally distributed among its outgoing edges. The process terminates upon reaching the leaf nodes, whose out-degree is zero. Mathematically, the weight update process can be written as

$$w_{td}(v^{out}) = \frac{w_{td}(v^{in})}{|\Gamma(v^{out})|}, v \in T \tag{4.3}$$

where, w represents the weight; v^{out} and v^{in} represent the edge outgoing and incoming to the vertex v in the *ReqTree* T. $|\Gamma(v^{out})|$ denotes the number of edges outgoing from the node v. The subscript *td* denotes the top-down approach of weight update. The top-down mode of weight update translates to a *divisive* mode of weight update.

4.3.3.3 Weight Normalization

The weighted trees suffer from exploding (in the additive mode) or vanishing (in the divisive mode) weights. The weights become system-dependent and become too large or too small in the case of large trees. An adverse consequence of these exploding/vanishing weights is that any comparison between the requirements of the two systems will be affected by the number of requirements listed for each of them. So, we normalize the weights of the *ReqTree* to make all the systems equally important. The weights of the edges are scaled by double the weight of the tree, as shown below:

$$w_{u,v} = \frac{w(u,v)}{2\sum_{u,v \in T} w(u,v)} \tag{4.4}$$

where $w(u,v)$ denotes the weight of an edge connecting a pair of nodes u and v in the *ReqTree* T. Normalization with double the tree's total weight slashed the maximum distance of a requirement from the global root to 0.5. Thus, the distance between a pair of nodes is ≤ 1. The weighted trees are denoted as T^w. Algorithm 2 summarizes the bottom-up mode of weight update with weight normalization.

Algorithm 2: Conversion of an unweighted *ReqTree* into a weighted *ReqTree* using the bottom-up approach

Input : *ReqTree*
Output: Weighted *ReqTree*
Extract adjacency matrix of *ReqTree*;
Identify the leaf nodes;
Consider the leaf nodes as the initial set of *child nodes*;
while *child nodes* $\neq$ *root node* **do**
    Find *parent node*s of the *child node*s;
    for each *parent node* **do**
        In the adjacency matrix, $w^{out}_{parent} \leftarrow \sum w^{in}_{parent}$
    end
    child nodes $\leftarrow$ *parent nodes*;
end
Normalize the weights of the adjacency matrix;
Construct a weighted tree from the updated adjacency matrix;

4.3.3.4 Weighted ReqNet

The weighted *ReqTrees* of all the RSDs are connected to the global root node to construct the weighted *ReqNet*. Mathematically,

$$\mathcal{N}^w = \bigcup_{i=0}^{\mathbf{N}} \{G\} \cup T_i^w \tag{4.5}$$

where $\mathcal{N}^w$ is the weighted *ReqNet*. The weighted *ReqNet* offers a fine-grained measure of the distance between pairs of requirements following the practices of requirement allocation. It aligns better with human judgment. The weighted distances overcome the shortcomings (See §4.3.3) of unweighted distances as the weights accumulate towards the root node. Since the path between two requirements across two systems passes through the root node, the weighted distances become higher. The distance between a pair of requirement nodes within a *ReqTree* is low as their connecting path does not pass through the root node. This is in coherence with human knowledge that a pair of requirements from the same system are more similar (from the same *ReqTree*) than a pair from different systems (from different *ReqTree*s).

4.3.4 ReqSim: Similarity of Requirements

Our method of computing the semantic similarity rely on the assumption that the requirements are semantically organized in the SyRSDs. The foundation of this assumption is that the creators of the SyRSDs organize the requirements based on their similarity. In other words, the creators aggregate similar requirements and segregate dissimilar requirements. The *ReqTree* captures this organization of requirements. Second, the distance between a pair of requirement nodes in the tree reflects their semantic similarity [136, 10].

We leverage *ReqNet*, $\mathcal{N}^w$, to compute the similarity between pairs of requirements. As ReqNet has three variants, we will have three different distances between a pair of requirements. And, correspondingly, three different similarity scores: a coarse-grained similarity, an additive fine-grained similarity, and a divisive fine-grained similarity. We can construct the *ReqNet* to compute the distance between any pair of nodes and the similarity. We use the path similarity [181, 62], the reciprocal of the distance incremented by one, to compute the similarity between a pair of requirements. The path similarity did not consider the similarity across the source SyRSDs where the pair of requirements exists. Thus, we incorporate the document level similarity, defined as the

similarity between a pair of SyRSDs, into the path similarity. The similarity between a pair of requirements R_i and R_j is defined as

$$\underset{\mathcal{N}}{Sim}(R_i, R_j) = \frac{Sim(\mathcal{D}_i, \mathcal{D}_j)}{1 + \underset{\mathcal{N}}{Dist}(R_i, R_j)} \tag{4.6}$$

where $R_i \in \mathcal{D}_i$ and $R_j \in \mathcal{D}_j$.

4.4 Results

4.4.1 ReqNet: Network of Requirements

The 86 *ReqTree*s created from the 86 SyRSDs are assembled as per §4.3.2 to create the *ReqNet*. Figure 4.5 shows the *ReqNet* with 17375 nodes. It has 12701 requirement nodes and 4674 non-requirement nodes.

4.4.2 ReqSim: Similarity of Requirements

The *ReqNet*, a union of all the *ReqTree*s, is a huge network with heavy resource demand. We adopt specific properties of trees to ease the computation of $Dist(R_i, R_j)$. The path that connects a pair of nodes in a tree is unique. In the case of *ReqNet*, a tree, the unique path connecting a pair of requirements connects the corresponding *ReqTree*s. The path passes through the global node, G, if the pair of requirements pertain to different *ReqTree*s. The path is a subset of the union of the paths from the selected requirements to the global node, G. So, we avoid using the complete *ReqNet* while computing $Dist(R_i, R_j)$ by leveraging the path of the requirements from the global node. The distance between a pair of requirements can be computed if their paths from the root node and the weights of the connecting edges are known. We extract the weights of the edges along the path that connects each requirement node to the global node. Then, the requirements are randomly paired, and the graph-theoretic distances between the pairs are calculated. We compute the additive distance as well as the divisive distance. Then, we factor in the document similarity to compute the semantic similarity between the pair of requirements. This is illustrated in Algorithm 3.

To estimate the similarities across different SyRSDs, we computed the embeddings of the SyRSDs. The 512 token limits of most transformer-based models [176] deem them unsuitable for embedding SyRSDs as the number of tokens in a SyRSD far exceeds 512 tokens. Thus, we embed the

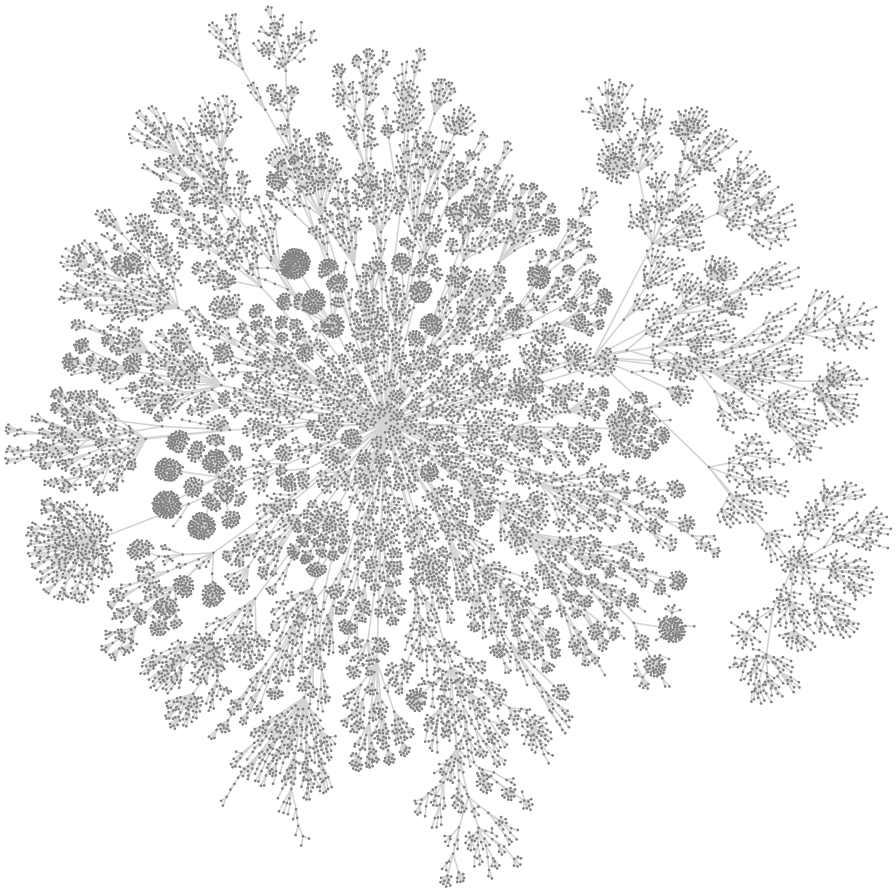


Figure 4.5: *ReqNet* - A network of requirements

*ReqList*s and their corresponding metadata using the Longformer-Encoder-Decoder (LED) model [14], which has a token limit 16384. The document similarity, $Sim(\mathcal{D}_i, \mathcal{D}_j) \in [0.3623, 1]$. The document similarity is the highest when the two documents are identical.

Random pairing of the requirements produces a distribution of the weighted distances where the data is biased towards dissimilar requirements against similar requirements. See Fig. 4.6. The reason being that the highly similar requirements are at the tail end of the distribution. Moreover, if a reference SyRSD has ten requirements and the dataset has 100 requirements, there is a 90% probability that a requirement will be paired with a dissimilar requirement in case of random pairing. So, we additionally paired the requirements within a document with each other (Step 7 in Algorithm 3) to include more similar pairs. 12701 requirements from *ReqList*s produced 6350 pairs. We created an additional 4583 pairs by pairing the requirements within SyRSDs. We set a maximum limit of 100

Algorithm 3: Construction of *ReqSim*

Input : *ReqLists* of all systems
Output: ReqSim

```
for ReqList of each SyRSD do
    Construct the ReqTree (See Algorithm 1);
    Convert the unweighted ReqTree to a weighted one (See Algorithm 2);
    for each clause number in a ReqList do
        Find and store the weights of the edges connecting to the global root node;
    end
    Pair the requirements within the system;
end
Assemble the ReqLists of all the systems;
Shuffle and pair the requirements of the assembled list;
Merge the two kinds of requirement pairs;
for each requirement pair do
    Compute the graph-theoretic distance between each pair;
    Estimate the similarity between the requirement pairs while factoring the document
     similarity (Equation 4.6) ;
end
```

pairs from any document to prevent any system from dominating the highly similar pairs. Overall, we have 10933 requirement pairs with their semantic similarity scores.

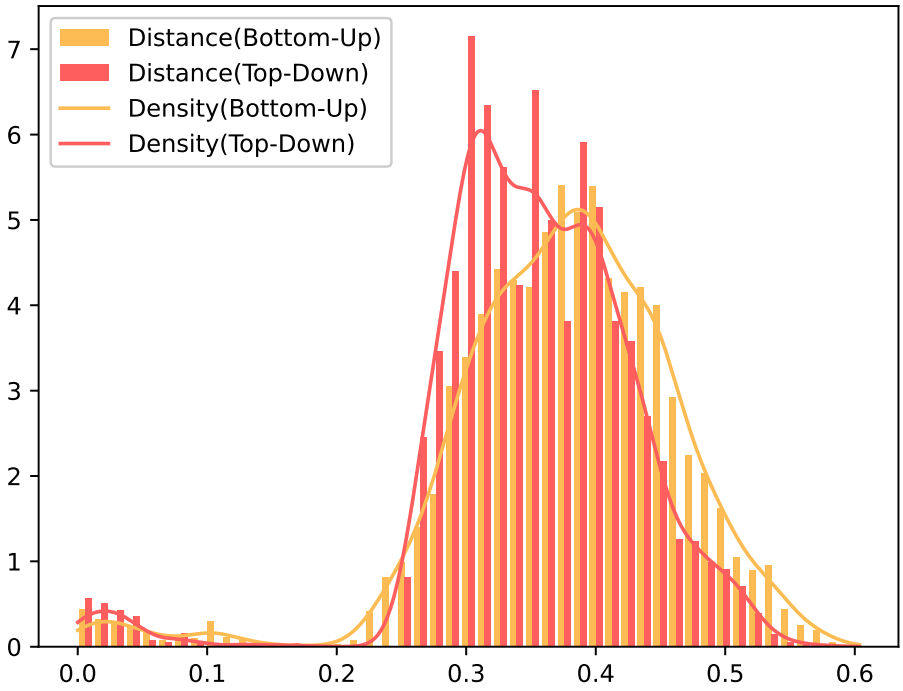


Figure 4.6: Histogram plot of fine-grained distances in the bottom-up and top-down approaches

4.5 Discussion

4.5.1 *ReqList*

ReqList, being a list of requirements of a system, is cleaner and closer to the source SyRSDs. Most RE applications that can be practiced on SyRSDs can be practiced on *ReqList*s. The primary one is NLP for RE [99, 196]. We quote a few ideas for potential applications. First, the *ReqList*s can serve as the ground truth benchmark for research works that focus on identifying and extracting requirements from source SyRSDs [151, 155]. Second, the requirements can be related and traced among each other using clustering techniques and word frequency-based techniques. Moreover, system-to-requirement level traces can be established by deploying NLP techniques on the *ReqList*s and their corresponding document structure. Third, the requirements can be evaluated for complying with linguistic rules for compliance to ISO/IEC/IEEE 29148:2018(E) [166]. Fourth, if NLP techniques deployed on the *ReqList* discover its *ReqTree*, it will eliminate the need for manual structuring of the requirements for documentation. Lastly, *ReqList*s are well-suited for deploying unsupervised or self-supervised learning techniques. For instance, the requirements in *ReqList* can help in prompting generative AI tools to generate requirements.

4.5.2 *ReqTree*

An effect of the weighted *ReqTree*s is that the parent nodes (one degree of separation) accumulate more weight as the number of children increases. Alternately, the similarity between a requirement and its child requirements reduces as the number of child requirements increases. The effect of weight accumulation gets further amplified as we compare a requirement with its grandparents and other higher ancestors (≥ 2 degrees of separation). This aligns with requirement allocation, where the correlation between parent and child requirements reduces as the parent requirement is decomposed into more child requirements. Due to weight accumulation, a requirement shares more similarities with its siblings than its grandparent requirement, even though both are two edges apart.

We want to emphasize that weighted trees are essential for computing the semantic similarity. Our weighing methods consider either the leaf requirements to be of equal importance or the child requirements of the same parent node to be of equal importance. In reality, all the requirements may not be equally important. Thus, a weighted requirement allocation process will eliminate the need of our weighing process while being more close to reality. Furthermore, a different hierarchy in

the SyRSD will affect the weighted trees.

4.5.3 *ReqNet*

ReqNet, being a graph, opens up the portal to deploy graph-theoretic and network analysis techniques. Moreover, tree-specific algorithms can also be exploited. Notably, traceability, detecting change in requirements, and analyzing the propagation of its impact are a few applications of network analysis techniques in the domain of RE.

ReqNet revealed a few interesting characteristics about the structure of the SyRSDs. First, the depth of *ReqNet* is low. The depth of a tree is defined as the average unweighted distance of all the nodes from the root node. The average unweighted distance of a requirement node from the document node is 3.8384. This results from the document hierarchy, usually three layers deep: section, sub-section and sub-sub-section. The requirements typically lie within the sub-sub-section. They account for the ≈ 4 degrees of average separation of a requirement node from the document node. Beyond those three layers, the authors of the SyRSDs use bullet points, sub-clauses, and enumerated lists. Their presence adds a few more layers in *ReqNet*. Nonetheless, these deeper layers (> 6 edges from the global node, G) are a bit of noise than the norm. A consequence of the low-depth nature of *ReqNet* is that *ReqNet* portrays small-world network behavior. The average unweighted distance between any pair of nodes is 9.5619. The links connect the global node $\xrightarrow{\text{to}}$ the document node $\xrightarrow{\text{to}}$ section heading $\xrightarrow{\text{to}}$ sub-section heading $\xrightarrow{\text{to}}$ sub-sub-section heading $\xrightarrow{\text{to}}$ requirement. Five edges for each requirement in the pair approximate to 9.5619.

4.5.4 *ReqSim*

We notice that the similarity, $Sim_{bu} \in [0.21456, 0.99955]$ and $Sim_{td} \in [0.20400, 0.99999]$ where the subscripts bu and td correspond to bottom-up and top-down modes respectively. This is a result of the normalized distances which are $Dist_{bu} \in [0.00045, 0.83705]$ and $Dist_{td} \in [4.33e-06, 0.83838]$. The pairs of requirements with the maximum (Max) and minimum (Min) semantic similarity in the bottom-up and the top-down approaches are shown in Table 4.1. It is evident from the table that the highest similarity is exhibited by requirements from the same SyRSD. They are separated by one or two edges, reflecting that such pairs share a parent-child, sibling, or grandparent-child relationship in the *ReqNet*. This behavior aligns with human judgment as the

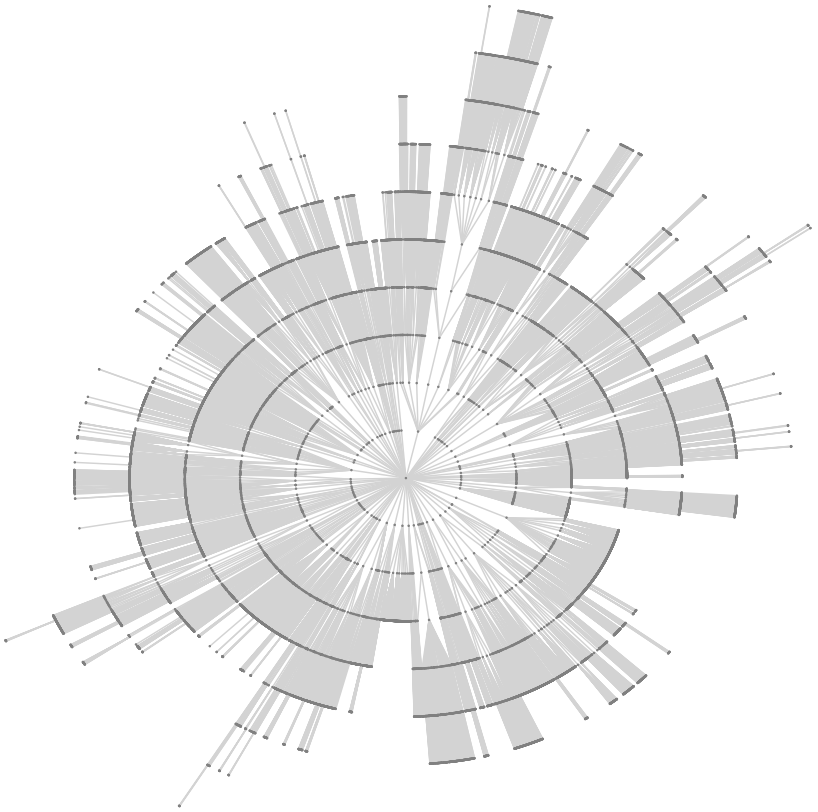


Figure 4.7: Visualizing *ReqNet* as a radial tree

child requirement nodes inherit several properties from their parent requirement nodes. Another attribute of requirement pairs with high similarity scores is that such requirements are leaf nodes or lie far from the root node where the weights do not accumulate.

Additionally we observe that the least similarity is exhibited by a pair of requirements from distinct SyRSDs. Their lowest similarity score arises from their low document similarity score, even though their weighted distance is moderate. Moreover, the most distant nodes (separated by 18 edges) exhibit moderate similarity due to weight normalization and relatively high document similarity scores. Thus, the similarity scores of *ReqSim* align with human interpretation. We can conclude that *ReqNet* and *ReqSim* capture human knowledge. So, *ReqSim* can push the boundaries of existing NLP and ML approaches from word-level to sentence-level semantics.

It is important to emphasize that we assume that the requirements are semantically organized in the SyRSDs. While the *ReqList* and the unweighted *ReqNet* datasets do not have any known shortcomings, we specify two shortcomings of the *ReqSim* dataset. The shortcomings emerge

from the similarity scores, which are derived from the tree structure created from the clause numbers. First, the tree structure disregards the textual information contained in the requirements corresponding to the clause numbers. A pair of textually identical requirements from two different SyRSDs (E.g., the requirement for the service life of the motor from the SyRSD of a DC motor and that from the SyRSD of an AC motor) may not produce the expected high similarity score. Their similarity score will depend on the similarity of the SyRSDs and the locations of the requirements in the SyRSDs. Second, criticism stems from the weighing scheme. Both the weighing schemes, top-down and bottom-up, allocate uniform weights to each child node of a parent node. They consider each child node requirement with the same parent node requirement to be equally important. This naive approach lacks a pragmatic sense. For instance, the internal combustion engine and the electric motor in a hybrid vehicle may not produce the same amount of power to propel the vehicle. Thus, all the child nodes with the same parent node are not equally important. Additionally, as a parent requirement is decomposed into child requirements, the sum of the child requirements shall be equal to the parent requirement in an ideal case. However, the sum of the child nodes is not always equal to the parent node [186].

Table 4.1: Pairs of requirements with their similarity score

Requirement (R_i)	Document ($\mathcal{D}_i$)	Requirement (R_j)	Document ($\mathcal{D}_j$)	Sim $(\mathcal{D}_i,\mathcal{D}_j)$	$Dist$ (R_i,R_j)	$Dist_{bu}$ (R_i,R_j)	Sim_{bu} (R_i,R_j)	$Dist_{td}$ (R_i,R_j)	Sim_{td} (R_i,R_j)
The user shall be prompted to enter a text string describing a string pattern.	2001 - hats	The HATS-GUI shall provide the user the capability to search the text display for text substrings. The search criteria are described in Appendix F.	2001 - hats	1.0 (Max)	1 (Min)	0.0004	0.9995 (Max)	9e-6	0.9999
The Clarus program shall determine how it will provide data and information to contributors.	2005 - clarus low	The Clarus program shall determine the need for bilateral Clarus Data Sharing Agreements with countries, agencies, states, and regions.	2005 - clarus low	1.0	2	0.0007	0.9993	4e-6	0.9999 (Max)
The Development Team shall label every scheduled [SYSTEM NAME] build in the source control system.	2010 - blit draft	All user sessions involving financial transactions or sensitive data are encrypted using SSL/HTTPS.	EHR System TechReq LA DHS	0.2918 (Min)	8	0.5061	0.1934 (Min)	0.5196	0.1920
Configuration stores are secured from unauthorized access and tampering.	EHR System TechReq LA DHS	The system shall continue to depend upon and use a single SQL Server 2008 database.	2010 - blit draft	0.2918 (Min)	8	0.4939	0.1953	0.5212	0.1918 (Min)
The TCS shall provide override of payload automated as well as preprogrammed inputs.	1999 - tcs	NPAC SMS shall notify the Old and New Service Provider when a Subscription Version is set to conflict upon Subscription Version modification.	2001 - npac	0.7538	18 (Max)	0.4372	0.5244	0.4003	0.5383

4.6 Conclusion

Our dataset has three forms with the same core set of requirements. First, we release *ReqList*, a list of 12701 requirements from 86 distinct systems to facilitate natural language processing and unsupervised machine learning algorithms (such as prompting for generative artificial intelligence). They can help in requirement identification and extraction tasks, creating requirement-to-requirement and requirement-to-system traces, and characterizing well-formed requirements using rule-based NLP, among many others. Each of these *ReqList* is supported by their metadata consisting of the structure of the SyRSDs and the corresponding *ReqTree*s. Second, we create *ReqNet*, a large network of requirements consisting of 17375 nodes, to support graph-theoretic explorations. *ReqNet* can help in traceability analysis, identification of changes in requirements, and the study of cascading changes in requirements. *ReqNet* portrays small-world characteristics. Third, we create *ReqSim*, a dataset containing 10933 pairs of requirements and their similarity scores, to gear towards sentence-level semantics from word-level semantics. The semantic similarity scores align with human judgment as they are created from the structure of the SyRSDs, which are created by humans. Thus, they will serve as the gold standard for supervised learning tasks. The datasets can be expanded further by incorporating SyRSDs of new systems while following the proposed formal method. Our method considers all the systems equally important and the requirements with the same parent node equally important. We lay higher importance on requirements that are higher in the hierarchy of the document structure. Our dataset will accelerate the deployment of natural language processing, graph theory, and unsupervised and supervised learning techniques in requirement engineering.

Chapter 5

A Semantic Network of Requirements from the Embeddings of Req-sBERT

5.1 Introduction

The stakeholders elicit requirements in natural language (NL). Requirements, being an instance of NL, inherits the power of expression, communicability and accessibility of NL along with the impreciseness, ambiguity and inconsistencies of NL [186, 83]. The semantics of requirements get further complicated due to the various ways in which the same requirement can be elicited, variations in the order of the words in the requirement, evolution of requirements, presence of synonyms, polysemy and homonyms. A suitable representation shall precisely capture the semantics of the design requirements amongst the ambiguity and impreciseness of requirements to enable analysis. We represent requirements as a semantic network following the endorsement by the survey of Han et al. [69] for the use of semantic networks in engineering design to serve as knowledge bases. The methods of most of the prior work, such as WordNet [49], ConceptNet [157] or TechNet [148], split a sentence into multiple concepts based on constituent keywords or phrases. They lose the contextual information offered by the order and composition of multiple concepts. Thus, we use contextual embeddings to embed the requirements as whole sentences. Many researchers have also highlighted

the limited capacity of generic domain-agnostic knowledge bases for engineering tasks that need fine-grained inference or are solution-oriented [27]. Thus, the semantic networks need to be tailored for the application to derive any value from it. Our semantic network falls beyond their two categories of semantic networks, trained on non-engineering data sources and trained on technical data sources in an unsupervised fashion, as we fine-tune our model in a supervised way. Our research work is an attempt at the third research direction of Han et al. [69], the use of transformer-based models to capture semantic relations for engineering design. We go beyond conventional generic semantic reasoning tasks, such as search and retrieval, to RE-specific tasks.

In essence, our research work seeks the answers to the following research questions.

1. How to semantically represent the requirements of a system?
2. What value can be extracted from the semantic representation?

We make the following contributions while answering the above research questions:

1. We represent the requirements as a semantic network to effectively capture the semantics at the requirement level. Each requirement becomes a node in the semantic network, and the links correspond to the cosine similarity between the requirements' embeddings. We achieve this by finetuning a modified pre-trained MPNet (Masked and Permutated Network) [156] model using the Siamese architecture of Sentence-BERT (sBERT) [135]. We refer to our model as *Req-sBERT* (**Req**uirement **S**entence **B**idirectional **E**ncoder **R**epresentations from **T**ransformers). Our modified model includes a six-layered deep neural network (DNN) with 768 neurons in each layer following the MPNet model. The pretraining retains the rich structure and semantics of NL, and the finetuning captures the RE-specific aspects. To the best of our knowledge, we are the first to focus on sentence-level semantics in engineering design.
2. We are the first to finetune *Req-sBERT*, a language model, in a supervised fashion. We finetune the DNN on the *ReqSim*[1] dataset [146] consisting of 10000 requirement pairs with their cosine similarity scores for adopting the model for engineering design. During finetuning, we harmonize the attention-based learning mechanism (based on a scaled dot product) [176] with the evaluation metric, mean-squared error, based on the cosine similarity between pairs of requirements.

[1] `https://github.com/ChandanKSahu/ReqList_ReqNet_ReqSim/tree/main`

3. We investigate the requirements of the Deep Orange 13[2] project and show that the semantic network is coherent with the semantics of requirements. Similar requirements share a higher cosine similarity score than dissimilar ones. The similarity reduces as the number of semantically unrelated keywords increases. We illustrate the utility of our semantic network in two RE-specific applications:

   (a) We show that a non-functional requirement (NFR) shares a higher similarity with its other NFR forms than its functional form. We show that the change in the similarity score as a requirement evolves from being a functional requirement (FR) to an NFR is sharper than that from one variant of NFR to another.

   (b) Our semantic network can aid in extracting a hierarchy of requirements to document the requirements in a requirement specification document. The requirement hierarchy reflects the hierarchy showcased by different concepts of the project.

5.2 Literature Review

5.2.1 Contextual Embeddings of Requirements

Representing the words as vectors to capture meaning was started for sentiment analysis where words are considered as stimuli and represented in a three-dimensional semantic space on the criteria of valence, arousal and dominance [123]. These three-dimensional representations have exploded into more than a hundred-dimensional representations. These vector representations, such as Word2Vec [112] and GloVe [129], referred to as embeddings, have portrayed vector addition/subtraction operations consistent with semantic sense. The principle underlying vector semantics is the distributional hypothesis, which states that the meaning of the words can be inferred from the context in which the words appear. Change in the semantics of a word is related to the change in its context [81]. Nonetheless, static embeddings (like Word2Vec [112] and GloVe [129]) cannot handle polysemy and the contextual variations as they generate a unique embedding for each word. For instance, static embeddings of the keyword *book* will mean the same in the requirements 'the system shall *book* the nearest cab' and 'the library shall issue *book*s to an authenticated user'. Moreover, the relationship between the keywords 'library' and 'book' in the latter is ignored, which

[2] `https://cuicardeeporange.com/project/do13/`

can offer additional context to the keyword *book*. Static embeddings cannot understand the semantics of the composition of words, such as phrasal verbs. Thus, we adopt BERT-based [36] contextual embeddings where the embeddings of the same keyword will vary depending on its context.

There are several research works that leveraged BERT models for engineering design. Mullis et al. [120] classified requirements into functional and non-functional requirements using BERT. BERT-based embeddings were used to screen evidence for federal grants for agriculture, nutrition and resilience applications [43]. Essentially, the authors classified documents as evidence or not based on the word embeddings. Ref. [194] used BERT to identify a correlation between the sentiment of the user reviews and the descriptions of the products in an online footwear store. The correlation was used to inform the designers of the desirability of the product's concept and attributes. The semantic network created by Cheligeer et al. [26] for the braking system of an aircraft used the static embeddings produced by Word2Vec [112], which are prone to failure if the context changes between different occurrences of the same word. Each of these research works focuses on word-level semantics, whereas requirements need sentence-level semantics.

5.2.2 Learning Embeddings

The idea of embeddings originates from embedding words. The method we adopted to embed sentences adapts the word embedding machinery to embed sentences. Numerical representation of words started by tokenizing (unique identity for each word) words as in skip-gram models [26], commonly referred to as word2vec [112]. They evolved into feed-forward neural networks (NN) that accept the n-preceding keywords and predict the probability distribution over possible succeeding words (e.g., the vocabulary). These NNs are referred to as neural language models (NLM). Transformer-based [176] neural language models have become the de facto choice for NLP because of their long-range dependency, accuracy, generalizability and parallelization.

In particular, we adopt and modify a BERT-based [36] model with the sBERT [135] architecture to learn the embedding of sentences. A few prior research used sBERT for requirements engineering. The semantic similarity between the requirements was computed by embedding the requirements [32]. Alhoshan et al. [9] used sBERT to classify requirements by computing the similarity between the embedded requirements and the embedded classification labels. Other BERT-based models that use word embedding for RE include BERT4RE [6], which was finetuned on 1055 functional requirement sentences. RE-BERT [33] extracted requirements from app reviews. Their

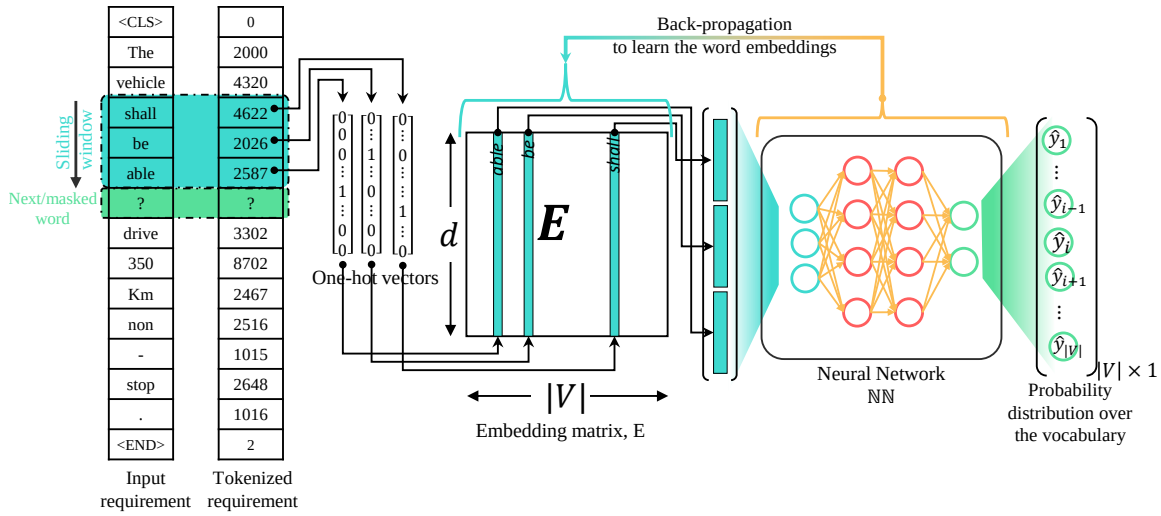


Figure 5.1: Neural language model

work directly translates to classifying if a sentence is a requirement or not. A key shortcoming of all prior research is that they (1) primarily focused on the classification of requirements and (2) finetuned on small requirements datasets. If the model was finetuned in a self-supervised way, they did not differentiate between requirements and non-requirement sentences [32]. Thus, we finetune a modified sBERT architecture on the *ReqSim* dataset with 10000 requirement pairs in a supervised fashion.

5.3 Methodology

Our method involves three parts: (1) a neural language model based on transformers to learn the contextual embeddings of requirements (§5.3.1), (2) fine-tuning the *Req-sBERT* model on the *ReqSim* dataset (§5.3.2), and (3) a cosine similarity operation to construct the semantic network (§5.3.3).

5.3.1 Neural Language Model

Neural language models (NLM) transform the symbolic representation (=words) of requirements into embeddings (numeric representation of words). Our NLM predicts the next word or masked word (as in fill-in-the-blank tasks). Operationally, NLMs rely on a classification task where it allocate probabilities to the words from the vocabulary, reflecting their suitability for being the

next/masked word. The resulting embeddings reflect the contextual meaning of the words in the vocabulary.

Consider a requirement $R = (w_1, w_2, ..., w_n)$ where w_i represents the words. The requirement is first tokenized as $R = (x_1, x_2, ..., x_n)$ where x_i are tokens. Our tokens are unique numbers allocated to each word in the vocabulary. The tokenization also produces a binary mask reflecting the usability of the tokens. The tokenization process adds special tokens to each requirement: $< CLS >$ to indicate classification and $< END >$ to indicate the end of the input. Each of the tokens is expressed as an embedding, e. The embeddings of all the tokens in the vocabulary (V) are stacked in an embedding matrix, E. Considering the number of words in the vocabulary to be $|V|$ and the length of the embedding vectors to be d, the size of E is $d \times |V|$. As per sBERT [135], we initialize E with pre-trained MPNet [156] embeddings. We update E during training to learn the embeddings. The embedding of a word e_i can be retrieved by the product of E and its token x_i as in

$$e_i = Ex_i \tag{5.1}$$

While predicting the next/masked word using its preceding words, we retrieve the embeddings of the preceding words and concatenate them to be the input to the NLM. We pass the concatenated embedding (e) into the neural network ($\mathbb{NN}$) to predict the probability distribution of the words ($\hat{w}$) in the vocabulary to be the next word. The probability of a requirement, a sequence of words $(w_1, w_2, ..., w_n)$, is given by

$$P(w_1, ...w_n) = \Pi_{i=1}^{N} P(w_i | w_1, w_2, ..., w_{i-1}) \tag{5.2}$$

where i is the context window. During training, cross-entropy loss ($\mathcal{L}_{CE} = -\log\, p(\hat{w}|w)$) is used to optimize the parameters of the NLM. The cross-entropy loss becomes equal to the negative log-

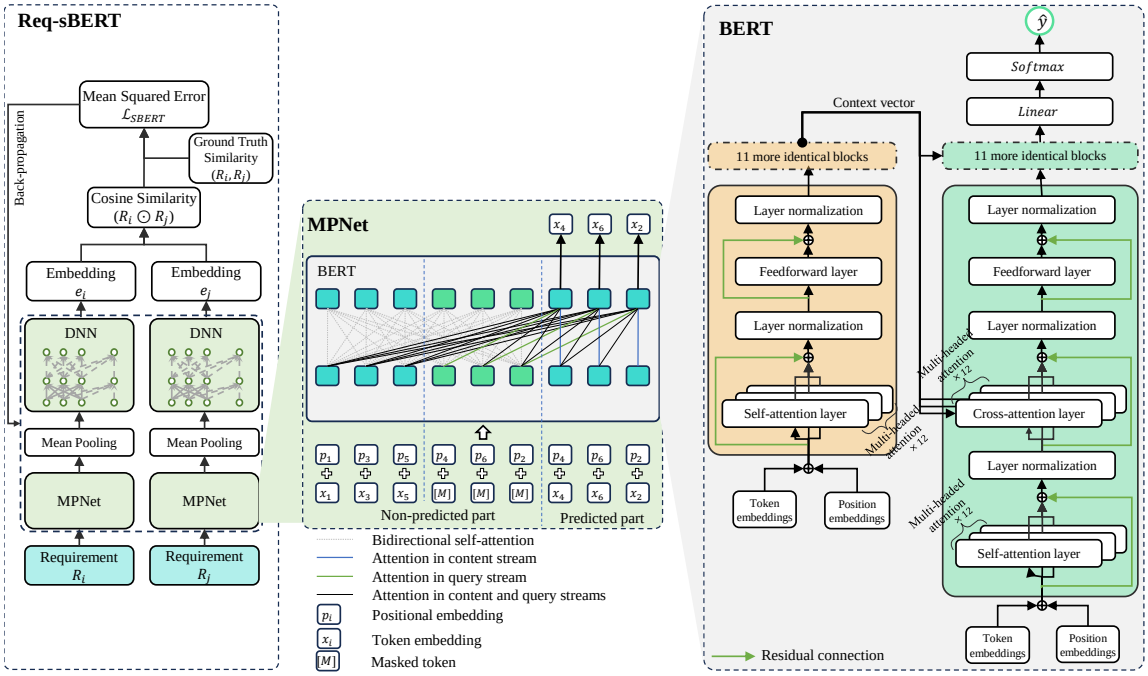


Figure 5.2: The architecture of *Req-sBERT*

likelihood loss by considering a hard classification. Thus,

$$e = [e_1 : e_{i-1}] \tag{5.3a}$$

$$z = \mathbb{NN}(We + b) \tag{5.3b}$$

$$\hat{w} = \text{softmax}(z) \tag{5.3c}$$

$$\mathcal{L}_{CE}(\hat{w}, w) = -\sum_{i=1}^{|V|} w_i \log \hat{w}_i = -\log(\hat{w}_i) \tag{5.3d}$$

E eliminated the need to learn distinct weight matrices to map the representations of the words to the first layer of the NN [133]. The weight matrix of the final layer, which feeds into the softmax function, is similar to the embedding matrix E. So, we do an average pooling over the word embeddings from the last layer of the NLM to compute the sentence embeddings. The trained model offers (1) a learned NLM and (2) learned embeddings that capture the semantics of the words more effectively than their initialization. Figure 5.1 illustrates the idea of our NLM.

5.3.1.1 Neural Network Model Architecture

Our NN block of the NLM consists of the MPNet [156] model powered by a BERT core [36] and a DNN packed together in a siamese structure following the sBERT architecture [135] as illustrated in Figure 5.2. We refer to the model as *Req-sBERT*. The Siamese structure accepts pairs of requirements for comparison. We chose MPNet because of its highest average performance in sentence embeddings (69.57) and second highest performance in semantic search (57.60) over multiple datasets [135]. BERT is almost identical to Transformers [176] in terms of the architecture except for the bidirectional attention, the number of layers and the inputs. MPNet differs from BERT in terms of the execution of the attention mechanism. In *Req-sBERT*, the terminal DNN block further processes the mean-pooled output of the MPNet model for learning requirements. Unlike the original sBERT implementation, our *Req-sBERT* maximizes the cosine similarity over the *ReqSim* dataset while fine-tuning.

Our BERT core consists of 12 transformer blocks, each comprising of 12 self-attention layers, residual connections, normalizing layers and feed-forward layers (as shown in the BERT block in Figure 5.2). The token embeddings and position embeddings of the requirements are fed to MPNet. The attention mechanism, which is based on the dot product, extracts the relationship between each pair of tokens in the requirement R. The attention mechanism is based on a scaled dot product given by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5.4}$$

where the tokens in the requirement play three different roles: *query* (Q), *key* (K), *value* (V). The query is the target token, which is the current focus of attention. Keys are the tokens with which the query is compared. The value is used to compute the output for the query. Each unmasked token in the requirement serves the dual purpose of being a key and value. We use bidirectional attention, as in BERT, where the query is compared with both preceding and succeeding tokens (See Figure 5.3). So, our model masks a token and tries to predict it using the keys. Similar to the original BERT model, the dot product is scaled by $\sqrt{d_k}$ to bring back the variance of the dot product from d_k to 1, and the softmax normalization scales the product to $[0, 1]$. The scaled dot product is multiplied with the *value*s to result in an embedding representation for each token in R. The multiheaded attention captures multiple relations among the words in the requirement R, such as syntactic, semantic, discourse and other relations (e.g., grammar) [176].

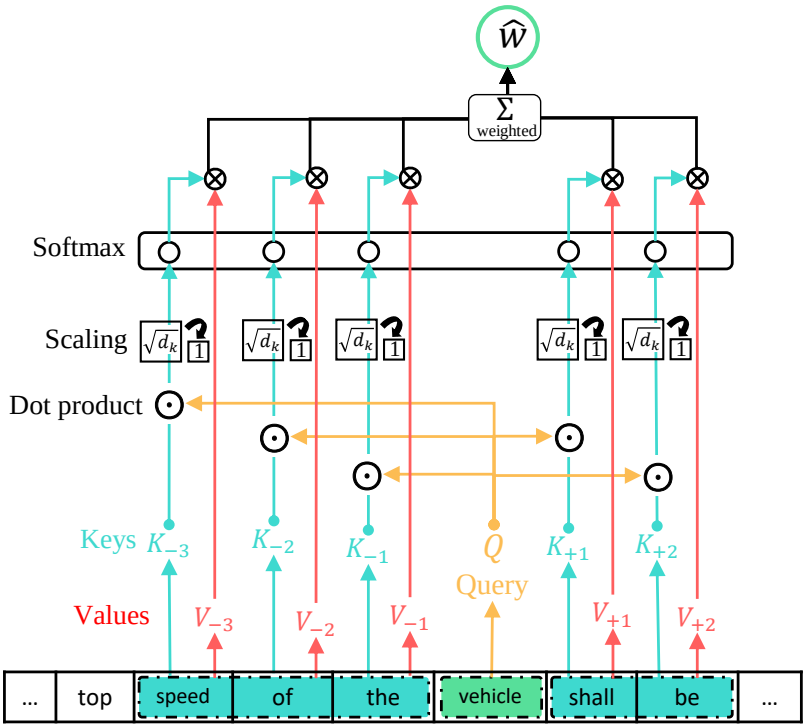


Figure 5.3: Bidirectional self-attention mechanism: Moving the succeeding tokens to the left of the *masked* token *permutes* (as in MPNet) the order of the tokens while still being bidirectional.

The MPNet [156] model uses masked and permutated language modeling to compute the embeddings of the requirements. MPNet randomly *permutes* the tokens of the requirement and *masks* a subset of the tokens at the tail end of the input requirement. The original unmasked tokens in the masked subset are appended to the input requirement. MPNet (1) improved the dependency in the predicted output tokens and (2) made the pretraining and fine-tuning behavior consistent. The output of the last layer of MPNet undergoes mean-pooling before passing into the DNN. Our best-performing DNN is a plain six-layered feed-forward network with 768 neurons in each layer. It includes batch normalization, L2 regularization and dropout regularization. The choice of 768 neurons in each layer stems from the 768-dimensional embeddings produced by most BERT-based language models. The output of the last layer of the DNN serves as the requirement embeddings.

Our data instances are triplets, $(R_i, R_j, Sim(R_i, R_j))$, consisting of pairs of requirements and their similarity scores. Each requirement from the triplet is passed into the *Req-sBERT* model to estimate the embeddings. The similarity score serves as the ground truth. From Fig. 5.1, we can notice that every NLM needs an embedding matrix. The Siamese structure enables parameter

sharing and model sharing. Thus, we use the same *Req-sBERT* model and the embedding matrix at the cost of increased time of computation. The requirements take turns to pass into the *Req-sBERT* model to compute the 768-dimensional embeddings. We estimate the similarity between the requirements from the dot product of their embeddings. Then, we use the mean squared error loss between the estimated similarity and the ground truth to train the *Req-sBERT* model. The loss function is defined as

$$\mathcal{L}_{\text{Req-sBERT}} = \frac{1}{N} \sum_{i,j \leq N} \Big(R_i \odot R_j - Sim(R_i, R_j) \Big)^2 \tag{5.5}$$

where $R_i \odot R_j$ is the cosine similarity between the embeddings of a pair of requirements predicted by the NLM and $Sim(R_i, R_j)$ is the ground truth cosine similarity score.

5.3.2 Finetuning

We use transfer learning by using a pre-trained sBERT model powered by MPNet. MPNet is pre-trained on 160GB of text data consisting of 1.17 billion sentences from multiple data sources, including Reddit, Semantic Scholar Open Research Corpus, Stack Exchange and WikiAnswers [192, 135]. The sBERT model is further trained on the SNLI (Stanford Natural Language Inference) corpus [19] and the broader multi-genre NLI [189] corpus. The SNLI includes 570k sentence pairs, and the multi-genre NLI contains 433k sentence pairs annotated with three labels to indicate the relationship among pairs of sentences. Our *Req-sBERT* differs from sBERT in terms of the labels where we use the scalar values of cosine similarity as the labels, whereas the sBERT model uses one of the three labels from (entailment, contradiction, and neutral). The sBERT model still lacks the semantics of requirements owing to the absence of requirements in the training dataset.

Thus, we finetune our *Req-sBERT* model on the '*ReqSim*: Similarity of Requirements' dataset [146] to embed the requirements and predict their similarity scores as per the Equation 5.5. The data instances in *ReqSim* include triplets consisting of pairs of requirements and their similarity scores computed using the tree structure of system requirement specification documents (SyRSD). *ReqSim* dataset includes 10933 triplets. In particular, we use the *ReqSim* dataset as our test dataset and resample requirement pairs from the '*ReqList*: List of Requirements' dataset to train and evaluate our model. As *ReqList* contains 12701 requirements, it is possible to construct $\approx$ 160 million requirements pairs. We consider 10% of *ReqList* as our evaluation dataset. Thus,

we have 1271 requirements for evaluation and 11430 requirements for training. We construct 10000 requirement pairs for training and 1000 requirements for evaluation. As we construct 10000 pairs from 11430 requirements, the requirements get repeated. We ensure that each data instance (triplet) is unique. Splitting *ReqList* prior to constructing the requirement pairs ensures that no requirement is shared between the train and evaluation datasets. Following the recommendations of Sahu et al. [146], we incorporate a similar fraction of ≈ 0.5 in our datasets for training and evaluation. We randomly pair the requirements within the same SyRSD to get similar requirement pairs. Our training dataset has 4818 pairs from the same SyRSDs, and the remaining 5182 requirement pairs are randomly paired. The evaluation dataset has 466 pairs where both the requirements belong to the same SyRSD. We use a threshold of 100 pairs at most from any SyRSD to prevent over-representation by any SyRSD.

5.3.3 Semantic Network Creation

Even though the 768-dimensional embeddings capture the semantics, they are not amenable to interpretation. Even for simpler tasks like data visualization, we have to reduce the dimension without losing much information contained in the high-dimensional nonlinear space. Thus, we transform the requirements into a semantic network of requirements for maximal information retention. As the embeddings of the requirements maximize the cosine similarity, we create the adjacency matrix of the semantic network based on the cosine similarity of the embeddings as defined in

$$Sim(R_i, R_j) = \frac{\vec{R_i} \cdot \vec{R_j}}{|\vec{R_i}||\vec{R_j}|} \tag{5.6}$$

The cosine similarity is a good choice as it results in a high similarity score if a pair of vectors are located close to each other. It results in a zero score (high dissimilarity) if the vectors are orthogonal. The semantic network's nodes represent the requirements, and the connecting edges represent the similarity across the pair of requirements. While networks, in general, do not pay any heed to the locations of the nodes, we store the embeddings of the requirements in the nodes to avoid information loss.

5.4 Results and Discussion

5.4.1 Fine-Tuning Performance

We tokenize the requirements using the MPNet sub-word tokenizer with 30527 BPE (Byte-Pair Encoding) codes [36]. We limit the requirements to a length of 384 tokens. That translates to approximately 288 words per requirement, which is sufficient for most requirements. If a requirement has less than 384 tokens, the requirement is padded with the pad token (<pad>) to the right. Alternately, longer requirements are truncated to 384 tokens.

The *Req-sBERT* model is trained in three stages. In stage one, we use the pretrained model for inference to compute the embeddings, which become the inputs to the DNN. In stage two, we pre-train the DNN to modify its weights from random initializations to bring them close to requirement embeddings. In stage three, we fine-tune the *Req-sBERT* model as a whole. Stages two saved the time of fine-tuning *Req-sBERT* as the DNN does not have random weights anymore. Stage one saved the time of stage two as computing the embeddings using sBERT in each epoch is time-consuming. Our best-performing fine-tuned model used 32 requirements in each batch and the Adam [86] optimizer with weight decay. Our Adam has $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and $\epsilon = 1e-8$ with a weight decay of 0.001. During training, we dynamically reduced our learning rate if the training loss plateaued. The model showed the best validation loss at a learning rate of $4.5e-7$. Each epoch during fine-tuning took ≈ 314 seconds on an NVIDIA A100 GPU with 80GB memory. Our fine-tuned *Req-sBERT* has a training loss of 0.00094 and an evaluation loss of 0.00696. It has a test loss of 0.00336 on the original *ReqSim* [146] dataset. We compared the performance of our finalized *Req-sBERT* model with sBERT models supplemented with a terminal 6-layered DNN (Res6) and a 10-layered DNN (Res10), with each of them having residual connections and layer normalization. The choice of the models for comparison is driven by the enhanced performance in the original transformer model [176] due to the use of residual connections and layer normalization. The results are summarized in Table 5.1 based on MSE as well as mean-absolute error (MAE), mean absolute percentage error (MAPE) and Pearson correlation (CORR) metrics.

Table 5.1: Comparison of performance of different models on the test dataset.

Model	MSE	MAE	MAPE	CORR
Req-sBERT (Best)	**0.0034**	**0.0306**	**0.0617**	**0.8823**
Req-sBERT (Res6)	0.0051	0.0373	0.0752	0.8430
Req-sBERT (Res10)	0.0085	0.0509	0.1032	0.7606
Sentence-BERT (Source)	0.1147	0.3085	0.5975	0.1844

5.4.2 Model Deployment

We deploy the *Req-sBERT* model on the requirements of the Deep Orange 13[3] (DO13) project. The project is dedicated to developing a non-combat autonomy-enabled off-road vehicle that can optionally be remote-controlled. The requirements of the vehicle system span two mission scenarios: 'cold weather disaster relief' to support relief efforts in blizzards or similar cold hazards and 'urban reconnaissance' to search and navigate through debris in narrow passages while avoiding obstacles to support rescue teams and locate survivors. The requirements were elicited by graduate-level student automotive engineers, project leaders, and faculty members working on the project. The team has elicited a list of 54 requirements. We investigate the utility of the embeddings acquired from the *Req-sBERT* model and the semantic network created from the requirements.

5.4.3 Semantic Network and Embeddings

We embed the requirements of the DO13 program into a 768-dimensional space. Then, we use cosine similarity to construct the semantic network. We visualize these high dimensional embeddings in 2D using t-SNE [174], a non-linear dimensionality reduction technique. See the locations of the nodes in Figure 5.4. Our t-SNE converts the pairwise cosine similarities into conditional probabilities between pairs of requirements. The similarity reflects the conditional probability $p_{j|i}$ that the embedding of a requirement e_i will pick e_j as its neighbor. We minimize the KL divergence (mismatch between the distribution of the data in the reduced and original space) of all the requirements while using t-SNE. Our KL-divergence is $5.04e-8$. The semantic network is shown in Figure 5.4 where the nodes represent the requirements and the links reflect their pairwise similarity. As the similarity of a requirement with itself is 1, the main diagonal of the adjacency matrix creates self-loops in the network. So, we made the diagonal elements of the adjacency matrix equal to zero for visualization. Darker links reflect high similarity and vice-versa. The lighter links from node 42

[3] `https://cuicardeeporange.com/project/do13/`

reflect that it is less similar to other requirements in DO13. The locations of the nodes in Figure 5.4 reflect the rough similarity among the embeddings in a 2-dimensional space with the caveat of t-SNE. The caveat is the information loss due to the non-linear dimensionality reduction and the convoluted interpretation [184].

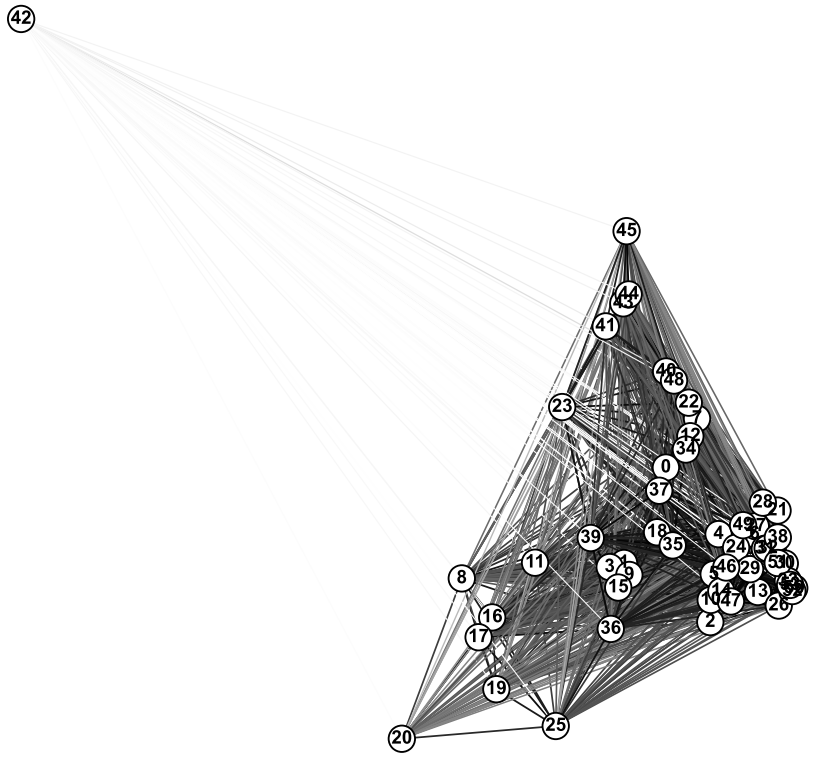


Figure 5.4: The semantic network of requirements of the Deep Orange 13 program with the node locations from t-SNE

5.4.4 Inference

5.4.4.1 Semantic Similarity among the Requirements

We evaluate the semantic network and the embeddings by comparing them with their coherence with human judgement. Before starting inference, we need to revisit the basis of the similarity score allocated to any pair of requirements. A human annotator of pairwise similarity evaluates pairs of requirements and scores them based on the following hierarchy: ordered set of identical keywords $\geq$ identical terms $\geq$ semantically similar words (*i.e.,* book and novel) $\geq$ semantically related words (*i.e.,* tea and cup) $\geq$ semantically unrelated words (*i.e.,* cup and computer) [160, 22]. A word can

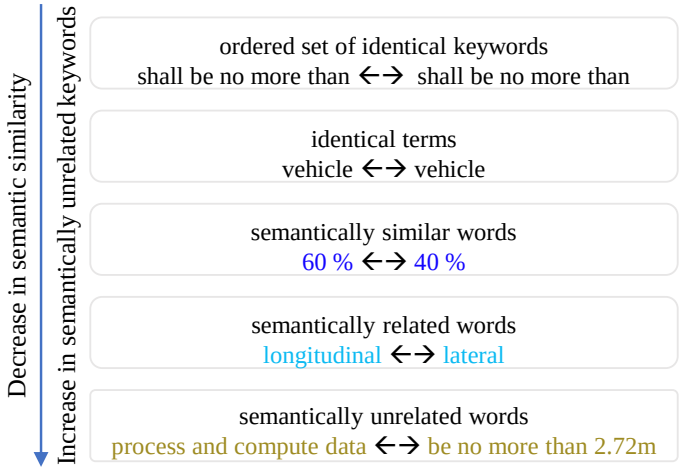


Figure 5.5: The variation of semantic similarity as the number of semantically unrelated keywords in a sequence of tokens increases (in the DO13 requirements)

be replaced by another semantically similar word without significantly altering the meaning of the requirement. Semantically related words occur in a similar context whereas semantically unrelated words lack a common context. The variation of semantic similarity in the context of DO13 is illustrated in Figure 5.5. We need to be mindful that a pair of requirements can contain one or more of these semantic features. For instance, they can include identical, semantically similar as well as semantically unrelated sequence of tokens. The evaluation of semantics gets further complicated by the limited capability of humans in the quantitative and fine-grained evaluation of semantic similarity [80].

A set of highly similar requirement pairs are listed in Table 5.2. A common observation is that the highly similar requirements have lots of keywords in common. They share common phrases. The requirement pairs (52, 50), (28, 27), (51, 50), (52, 51), (6, 5) and (33, 14) differ from each other by very few keywords. The non-identical keywords are semantically similar or related. The differing keywords vary from being semantically similar (e.g., dimensions such as length, width and height in R50, R51 and R52) to semantically related (e.g., lateral and longitudinal in R27 and R28) to semantically unrelated (e.g., snow and mud in R5 and R6) as the similarity score decreases. The numeric values can be modified without significant changes in the semantics of a requirement. The similarity score of 1 was produced by comparing a requirement with itself due to the comparison between a pair of requirements with an orderly set of identical keywords. The highly similar pairs

Table 5.2: Highly similar requirement pairs

Similarity	ID1	ID2	Requirement 1	Requirement 2
0.9984	52	50	The vehicle's height shall be no more than 2.6m.	The vehicle's width shall be no more than 2.72m.
0.9982	53	13	The vehicle's mass shall be no more than 4500 kg.	The vehicle shall carry up to 600 kg cargo.
0.9976	28	27	The vehicle shall drive on a maximum of 60% longitudinal grade.	The vehicle shall drive on a maximum of 40% lateral grade.
0.9974	51	50	The vehicle's length shall be no more than 11.89m.	The vehicle's width shall be no more than 2.72m.
0.9971	52	51	The vehicle's height shall be no more than 2.6m.	The vehicle's length shall be no more than 11.89m.
0.9969	6	5	The vehicle shall operate in up to 0.5 m mud.	The vehicle shall operate in up to 0.5 m snow.
0.9963	47	5	The vehicle shall operate between -42 to 49 degrees Celsius.	The vehicle shall operate in up to 0.5 m snow.
0.9962	33	14	The vehicle shall ford at least 0.914m water.	The vehicle shall ford up to 1 m high water.
0.9950	47	31	The vehicle shall operate between -42 to 49 degrees Celsius.	The vehicle shall operate in a variety of temperatures.

revealed some surprisingly useful information. The pair (47, 5) show that the model related snow in R5 with a low temperature of -42 degree Celsius in R47. Similarly, the pair (47,31) shows that the temperature range of -42 to 49 degrees Celsius in R47 was related to the phrase a variety of temperatures from R31. Furthermore, the high similarity between the pair (33, 14) makes one of the requirements redundant. We noticed that the similarity score decreases as the proportion of non-identical keywords in a pair increases. These highly similar pairs are located close to each other in Figure 5.4.

The requirement pairs with the least similarity scores are listed in Table 5.3. It is evident that dissimilar words dominate these pairs of requirements. The requirement R42 exhibits the least similarity ($\in [0.5773, 0.6724]$) with all the other 53 requirements. It is evident from the location of node 42 in Figure 5.4, which is far away from all other nodes. The last similarity of 0.5773 is exhibited by the pair (42, 50). The requirements R42 and R50 are semantically completely unrelated and do not share any keyword other than the auxiliary verb 'shall.' The highest similarity exhibited by node 42 is with node 40 because of the subject, 'the autonomous system.' Nonetheless, their contexts are semantically unrelated. A few more requirement pairs with subsequent similarity scores are shown in Table 5.3. It is evident that they are semantically unrelated. While requirements R33 and R47 pertain to the powertrain system, they are compared with requirements unrelated to power

Table 5.3: Least similar requirement Pairs

Similarity	ID1	ID2	Requirement 1	Requirement 2
0.5773	42	50	The autonomous system shall process and compute data.	The vehicle's width shall be no more than 2.72m.
0.6724	42	40	The autonomous system shall process and compute data.	The autonomous system shall perceive obstacles and topography in a 3D map.
0.7404	45	20	The powertrain system shall generate at least 200 kW power while stationary.	The vehicle shall maintain communication with the rescue team.
0.7662	45	19	The powertrain system shall generate at least 200 kW power while stationary.	The vehicle shall deploy drones.
0.7718	44	20	The powertrain system shall generate at least 40 kW power continuously.	The vehicle shall maintain communication with the rescue team.

or the powertrain system. Their only commonality is being the requirements of a vehicle. The highly dissimilar pairs are located far from each other in Figure 5.4.

A revisit to Fig. 5.4 reveals a few clusters. Small clusters include the pair (17, 16) with a similarity of 0.9845. R17 states that 'The vehicle shall detect objects of up to 100m away in rubble.' and R16 states that 'The vehicle shall identify people in rubble of up to 20 m far.' R16 and R17 are semantically similar. The other cluster is (43, 44, 45). They specify the requirements of the powertrain system. The similarity score of this cluster lies $\in [0.9795, 0.9911]$. The position of R45 (described in Table 5.3), being relatively far from R43 and R44 in Figure 5.4, does not corroborate with the similarity score due to the information corruption by t-SNE [184]. Contrarily, R41 is closer to (43, 44) than R45 due to t-SNE. Including R41 in this cluster reduces the minimum similarity to 0.9483 from 0.9795 as R41 specifies an autonomy requirement. The cluster (15, 1, 3, 9) specifies requirements related to offroad reconnaissance. The similarity of this cluster lies $\in [0.9823, 0.9907]$. The requirements in the largest cluster are listed in Table 5.4. A closer look at the requirements in the cluster relates all of them to the vehicle's performance in off-road conditions. The similarity score of this cluster lies $\in [0.9799, 0.9992]$. This reinforces that our network reflects semantic similarity.

5.4.4.2 Evolution of requirements

Requirements evolve continually throughout the product development process. The most prevalent change in requirements is the evolution of requirements from functional requirements (FR)

Table 5.4: Requirements in the major cluster. Note: Refer Table 5.2 for the descriptions of the requirements 5-6, 13-14, 27-28, 47 and 50-53 which are also in the cluster.

ID	Requirement
2	The vehicle shall carry/deliver critical supplies (MREs, fuel, blankets) up to 600 kg.
4	The vehicle shall operate on icy roads.
10	The vehicle shall travel 175 km distance (one-way) in a maximum of 2.5 hours.
21	The vehicle shall be transportable.
24	The vehicle shall provide auxiliary power.
26	The vehicle shall have at least 0.4 m ground clearance.
29	The vehicle shall have a minimum of 17.9 m/s top speed.
30	The vehicle shall have at least 20 m/s speed at 6-inch washboard.
31	The vehicle shall operate in a variety of temperatures.
32	The vehicle shall drive 350 km non-stop.
38	The vehicle's dimensions and mass shall allow its transportation.
46	The vehicle shall accelerate from 0 to 13 m/s in no more than 19 s.
49	The vehicle shall operate in silent mode for at least 50 km.

to non-functional requirements (NFR), and then from one NFR to another NFR due to the change in the measurable response [91]. FRs specify *what* task the system shall accomplish without specifying any attributes of quality or a measurable performance. NFRs specify *how well* the specified task needs to be accomplished. For instance, R46 in Table 5.4 is an NFR as it has a measurable response of '0 to 13 m/s in no more than 19 s'. Its functional form is 'The vehicle shall accelerate.' which is devoid of the measurable performance attribute. The validation of the system with the functional form produces a binary output without clearly specifying how well/poorly the system performs with respect to the requirement. On the other hand, validation of the system with R46 clearly highlights the extent to which the system meets the target speed or target time. The requirement R46 can be modified suitably for the next iteration of system development. Table 5.5 shows the similarities between the functional form and various non-functional forms of R46. The NFRs compared with R46 in the 2^{nd} and 4^{th} columns of Table 5.5 differ from R46 only in terms of the measurable response of the requirement as specified in the column titles [4].

The similarity matrix shown in Table 5.5 is a symmetric matrix as it is a pairwise comparison matrix. The values on the main diagonal are equal to one due to the comparison of a requirement with itself. The **bold values** in the first row shows the comparison of the FR with the NFRs. The *italicized values* show the comparison between the NFRs. It is evident from the similarity matrix

[4] **FR**:The vehicle shall accelerate.
5 s: The vehicle shall accelerate from 0 to 13 m/s in no more than *5 s*.
R46: The vehicle shall accelerate from 0 to 13 m/s in no more than 19 s.
10 m/s: The vehicle shall accelerate from 0 to *10 m/s* in no more than 19 s.

Table 5.5: Evolution of similarity from FRs to NFRs

	FR	5 s	R46	10 m/s
FR	1	**0.9749**	**0.9722**	**0.9729**
5 s		1	*0.9993*	*0.9989*
R46			1	*0.9994*
10 m/s				1

that the similarity between a FR and an NFR is less than that between a pair of NFRs. A similar behavior was exhibited by other (FR, NFR) and (NFR, NFR) pairs during our experimentation. We noticed that there is a sharp decline in the similarity from 1 as a FR changes to an NFR. Then, as the NFR changes from one NFR to another, the similarity score increases and saturates. This behavior is due to the high textual similarity among the NFRs compared to the similarity between an FR and an NFR. The FR and NFR forms of R46 differ by the string 'from 0 to 13 m/s in no more than 19 s', which consists of more than one token. Meanwhile, the NFR forms of R46 differ from each other by only one token: '5 s' instead of '19 s' or '10 m/s' instead of '13 m/s'.

5.4.4.3 Hierarchy among Requirements

We illustrate that a hierarchical semantic structure can be extracted from our semantic network. The extracted hierarchical tree structure can serve as an aid to organize the requirements as in a system requirement specification document. The organization of requirements in a SyRSD renders direct implementation of methods related to minimal spanning trees and their variants (e.g., minimum routing cost spanning trees and optimal communication spanning trees) unsuitable. The reason is the presence of nodes other than the requirement nodes in a SyRSD. The additional nodes serve as the sections/subsections in the document that corresponds to a function, use-case state or goal of the system. In a SyRSD, the requirements are leaf nodes [146]. The branch node and the root node are made up of section and sub-section headings. Our method consists of two steps: agglomerative hierarchical clustering followed by weight-based removal of selected intersection nodes.

We start by partitioning the set of requirements into singleton nodes and iteratively merge the singleton nodes into clusters until we find a single cluster comprising of all the requirement nodes [121]. Our implementation follows a greedy approach to minimize the dissimilarities between different clusters. If A and B are two clusters of requirements, we minimize the dissimilarity between

the clusters $\min_{R_i \in A, R_j \in B} d(R_i, R_j)$ where the dissimilarity $d(R_i, R_j)$ is given by

$$d(R_i, R_j) = 1 - Sim(R_i, R_j) \tag{5.7}$$

where $d(R_i, R_j)$ is the dissimilarity between the requirements $R_i \in A$ and $R_j \in B$. At each iteration, we greedily merge a cluster C with one of the two clusters A and B based on the minimum distance $\min(d(A, C), d(B, C))$ using a new intersection node. The hierarchical clustering of n requirement nodes produces a hierarchical tree consisting of $2n - 1$ nodes where the additional $n - 1$ nodes are intersection nodes. Then, we eliminate selected intersection nodes if the distance between the intersection node and its parent node does not exceed the minimum distance between the intersection nodes and its child nodes by a threshold, t. We connect the child nodes of the intersection nodes to the parent node of the intersection node upon eliminating the intersection node. Mathematically, for an intersection node $i \in \mathcal{I}$ with parent node $p(i)$ and child nodes $c(i) \in C$, if $d(i, p(i)) \leq t \times \min\ d(i, C(i))$, then eliminate the intersection node i and connect the child nodes $c(i) \in C$ with the parent node $p(i)$. We update the weight of the edges connecting the child nodes to the parent node with the weight of the edge connecting the intersection node with its parent node, $w(c(i), p(i)) \leftarrow w(i, p(i))$. $t = 1.7$ in our case. The final hierarchy of requirements is shown in Figure 5.6.

It can be seen from Figure 5.6 that the intersection node 91 corresponds to the powertrain system as its child nodes 43[5], 44 and 45 specify the requirements related to the powertrain system. Similarly, the six child nodes of the intersection node 105 elicit requirements related to reconnaissance. The themes of the requirements pertain to perception, localization, communication and search operations. Additionally, the tree structure makes it clear that R41 is less similar to the requirements R43, R44 and R45 even though Figure 5.4 implied otherwise due to the partial information corruption by t-SNE [184]. The node 105 flows down to node 99. The intersection node 99 relates to offroad mobility as its child requirements nodes relate to remote locations, driving in narrow passages and unstructured terrains. The node 99 flows down to node 85 and subsequently to node 68. Node 68 includes general mobility requirements related to dimensions, cargo capacity, gradability and range. Thus, it is evident that intersection nodes relate to specific themes. Recognizing these themes can aid in organizing the requirements in a semantic hierarchy.

[5]**R43**: The powertrain system shall generate at least 4 kW auxiliary power in for 10 hours in silent operation. See Table 5.3 for descriptions of requirements 44 and 45.

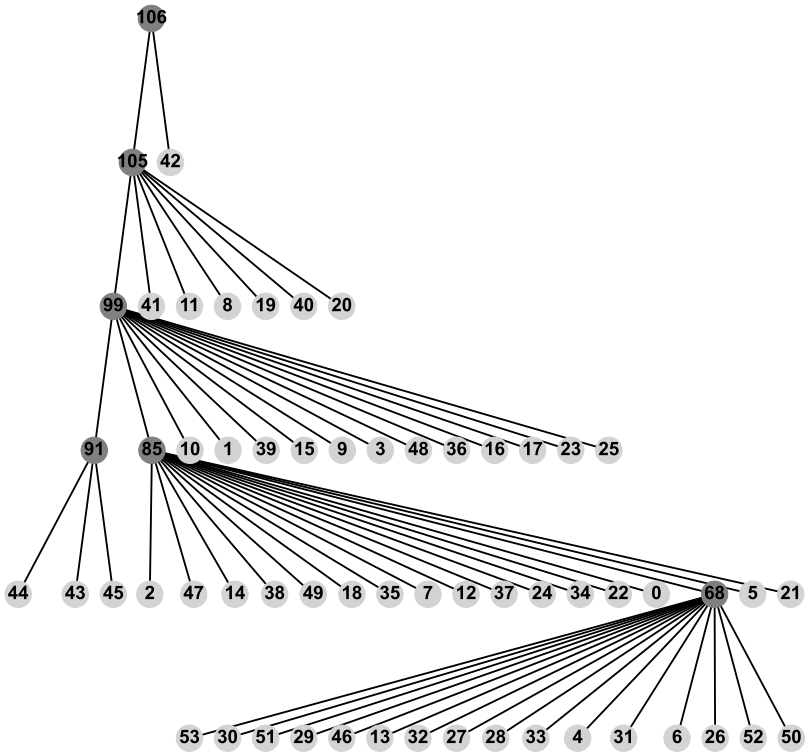


Figure 5.6: A hierarchical structure of the requirements to orginize them in a system requirement specification document

5.5 Conclusion

Products are designed to satisfy the requirements derived from the stakeholders' needs. Poor understanding and management of the requirements risk the failure of the system under development in addition to delays in schedules and overshoot of the budget and resources. As natural language is the de facto choice of eliciting requirements, requirements enjoy the freedom of expression and ease of accessibility and communication. Simultaneously, requirements suffer from the ambiguity, impreciseness and inconsistencies of natural language. The presence of synonyms, polysemy, homonyms, and a diversity of sentences to communicate the same concepts further complicates the management of requirements. That necessitates comprehension of the semantics of requirements for effective requirement engineering.

We develop a neural language model, *Req-sBERT*, based on the pre-trained MPNet model supplemented with a six-layered deep neural network within the Siamese architecture of the Sentence-BERT model. We finetune *Req-sBERT* on 10000 requirement pairs from the *ReqSim* dataset to

compare requirements pairs in order to understand the sentence-level semantics of requirements. This is a clear deviation from most prior work that relies on word-level semantics and self-supervised learning, where they do not compare requirements. We represent the requirements as a semantic network based on the cosine similarity between pairs of requirements.

We deploy the *Req-sBERT* model on the 54 requirements of the 'Deep Orange 13' program that develops an off-road autonomous vehicle with the twin goals of cold weather disaster relief and urban reconnaissance. The results showed semantics coherent with human judgment. We observed that the similarity between a pair of requirements follows the order - ordered set of identical keywords $\geq$ identical terms $\geq$ semantically similar words (*i.e.,* book and novel) $\geq$ semantically related words (*i.e.,* tea and cup) $\geq$ semantically unrelated words (*i.e.,* cup and computer) - based on the words present in the requirements. Continued investigation revealed that *Req-sBERT* captures the evolution of requirements. It allocated a lower similarity score between the functional and non-functional forms of a requirement than that between various non-functional forms of a requirement. Furthermore, we showed that a hierarchical tree structure can be extracted from the semantic network. The hierarchical structure can aid in documenting requirements.

In the future, we will extend our work to create a complete hierarchy, such as in system requirement specification documents, and name the intersection nodes to automate the documentation of requirements completely. Exploring the utility of *Req-sBERT* for tracing requirements among each other and other system artifacts and investigating change propagation are a few more possible avenues for future research.

Chapter 6

Requirement Verification: Evaluating the Quality of Requirements for ISO 29148:2018

6.1 Introduction

The requirements are *formally transformed* from the needs of stakeholders or higher-level parent requirements, constituting an *agreed-to-obligation* [138, 137]. This obligates the system to perform specific functions under defined conditions and constraints, aiming to fulfill stakeholders' needs. The system, upon materialization, is verified with the specified requirements to evaluate if the designed system will meet the stakeholder's needs. The *formal transformation* is essentially a requirement analysis process that ensures that the system is developed for the right purpose, to realize the need. The formal transformation of requirements necessitates that they be necessary, appropriate, singular, correct, and conforming. Concurrently, the *agreed-to-obligation* between stakeholders and the system ensures consensus on the requirements, stipulating that they must be unambiguous, complete, feasible, and verifiable [138, 137]. In the absence of these characteristics, a requirement fails to embody a fair agreement between all parties. Therefore, a requirement is considered well-formed when it embodies these essential attributes, which fundamentally constitute the "requirements of requirements".

The stakeholders elicit requirements in natural language (NL) due to its ergonomic nature. However, the freedom of expression in NL is also accompanied by inherent ambiguity and an unstructured nature. The presence of synonyms, homonyms, and context-dependent nature make NL a double-edged sword to elicit requirements [137, 150]. Thus, the ISO/IEC/IEEE 29148:2018 [166] states that the requirements shall be expressed in a very *specific, precise and unambiguous* manner to communicate the intent of the needs. Furthermore, consistent language across all the requirements eases the interpretation of requirements and deployment of formal methods on requirements. In light of the comprehensive array of characteristics that define the "requirements of requirements," it becomes imperative to delve deeper into understanding their implications and operationalization within the context of system development. This exploration naturally gives rise to pivotal inquiries, underscoring the essence of our investigation. The key underlying research questions (RQ) boil down to the following:

- **RQ1**: How to assess the qualities of requirements?
- **RQ2**: How do you compare the qualities of a pair of requirements?

To address these research questions, our paper makes the following key contributions:

- We create a requirement quality framework to evaluate the qualities of requirements. Our framework holistically evaluates the requirements using 42 NLP rules focusing on the token-level, span-level, syntactic structure-level and sentence-level attributes of the requirement.
- We propose a quantitative metric, *well-formed requirement quality* (wRQ) index. The metric incorporates the results of the 42 rules and the nine characteristics. The metric results in a score between 0 to 1 to compare the qualities of requirements. To the best of our knowledge, we are the first to offer a quantitative metric to evaluate the qualities of requirements in accordance with ISO/IEC/IEEE 29148:2018 (E).

Our research falls within the domain of requirement analysis, with a particular emphasis on verifying the quality of requirements through the application of Natural Language Processing (NLP) techniques. This approach is distinct from requirement validation, which assesses whether a requirement effectively communicates the intent of its need or parent requirement. In our work, we focus on verifying the well-formed nature of individual requirements by examining whether they exhibit the essential characteristics of a well-formed requirement, through a comprehensive evaluation based on a predefined set of rules.

6.2 Example Case of Requirement Verification

Let us consider the requirement `the autonomous driving system shall safely navigate urban areas at a speed of 40 mph while adhering to traffic rules` to evaluate its quality. While this requirement makes sense and seem practicable and feasible, this requirement is hard to implement, achieve and verify. This requirement is acceptable as a stakeholder requirement or as a mission requirement. However, this requirement is not acceptable as a system requirement. This requirement is necessary for the system. It seems feasible. However, this requirement is not complete. It is not clear what the elcitor meant by the terms `safely`, `urban areas` and `traffic rules`. Lack of clarity over these terms makes the requirement non-verifiable. What does the elicitor meant by `safely`? Will the system be acceptable if it doesn't get its occupants killed? Or, will the system be acceptable if gets into accident but its occupants are unscathed? It not clear what the elicitor meant by `safely`. Similarly, the definition, population density, width of the roads are not clear in the context of `urban areas`. It is not clear which traffic rules are being referred to by this requirement. This requirement needs a glossary that explicitly defines these keywords to be a well-formed (good) requirement. Furthermore, can the system drive exactly at 40 mph? If yes, for how long? What if the system drives at 40 ± 1 mph, will the system fail to satisfy the requirement? These ambiguous artifacts prohibit the requirements from being verifiable.

Let use rewrite the requirement as `3.5.1: The autonomous driving system shall navigate at a speed of 40±2 mph while avoiding obstacles`. The requirement is written for the `autonomous driving system`, making it level-appropriate. The system can drive within the speed range of 40 ± 2 while avoiding obstacles. The keyword `urban areas`, `traffic rules` and `safely` are removed. The requirement has improved, more likely to be verified. However, how to quantify the improvement in the rewritten requirement? These questions drive the research in this Chapter.

Table 6.1: Quality of an example requirement and rewriting it for improvement

Original requirement		Rewritten requirement
The autonomous driving system shall safely navigate urban areas at a speed of 40 mph while adhering to traffic rules.	3.5 3.5.1 3.5.1.1	Autonomous system Autonomous driving system The autonomous driving system shall navigate at a speed of 40±2 mph while avoiding obstacles.

6.3 Literature Review

6.3.1 NLP in RE

Requirements are an instance of NL. That drives the dominance of NLP in RE. The need for dedicated training and access tools for non-textual forms of requirements further advocates for NLP in RE [110, 137]. The systematic mapping study by Zhao et. al. is an excellent resource for exploring the full spectrum of applications of NLP in RE [196]. NLP has been used to detect, extract, model, classify, trace, search and retrieval of requirements. Part-of-Speech (PoS) tagging, tokenization, term extraction, stemming, stop-word removal, and parsing are some of the most commonly used NLP techniques in RE. Our work also uses some of them. The capability of NLP techniques for lexical, syntactic, semantic and pragmatic analysis of requirements was explored by Lash et. al. [99]. They proposed mapping requirements to a formal language to identify the subject, verb, object, and additional clauses in the requirement. The formal language gives rise to patterns for requirements. A formal syntax based on parts-of-speech, sentence structure and grammar was proposed by Lamar C. [96]. Structured requirements have been used to create ontologies as well [102]. The needs of customers were extracted from online product reviews using an NLP-based aspect-based sentiment analysis approach [116].

The use of structured text has been widely endorsed to overcome the shortfalls of eliciting requirements in NL. Chahadi and Herbert recommended using consistent terms while creating a semantic network from product requirements [24]. They mentioned that homonyms and synonyms should be properly incorporated into the NLP system. The work of Ref [110] is one of the widely used formalizations of RE using patterns to mitigate ambiguity, vagueness, complexity, omission, duplication, wordiness, untestability and inappropriate implementation. They proposed simple syntactic rules of thumb for using 'while' for state-driven, 'when' for event-driven, 'where' for optional features and 'if-then' statements for unwanted behaviors. Usage of their patterns reduced vagueness, wordiness and lexical ambiguity. A small segment of our work, focusing on sentence-level patterns, is inspired by their patterns. The work of Fuentes et. al. is one of two works that are closest to our work [58]. Their work is based on the 2013 version of the INCOSE rules. In contrast, our work is based on the 2023 version, which has significant revisions. They did not offer a single metric to assess the overall individual quality of requirements. The other work is by the QVscribe tool from QRA Corp [170]. However, their implementation leave about half of the rules to be assessed by

other tools and RE professionals.

6.4 Industrial Standards

6.4.1 ISO 29148 Guidelines

We establish a formal framework to evaluate the quality of the *requirement construct* following the guidelines of ISO/IEEE/IEC 29148:2018 (E) [166]. It states that a requirement is well-formed if the requirement (1) is qualified by measurable conditions, (2) is bounded by constraints, (3) can be verified, (5) defines the performance of the system when used under a specific condition by a specific user; and finally, (5) meets a need of the stakeholder. For instance, the requirement 'the diameter of the shaft shall be 2 mm' fails to be a well-formed requirement as a dimension of exactly 2 mm is hard to achieve. The inclusion of a tolerance, such as 2 ± 0.1 mm, makes the requirement achievable. Some of the guidelines for the requirements to be well-formed are very specific and clear. Examples include (1) the use of 'shall'; (2) avoiding the use of 'is', 'are', 'will', 'must', or 'should'; (3) avoiding the use of negation, such as 'shall not'; (4) avoiding 'shall be able to'. They can be directly implemented by natural language processing (NLP) techniques. Nonetheless, evaluating requirements only for these factors does not ensure the requirement to be well-formed. Each individual requirement shall have the following nine characteristics to be well-formed: Necessary (C1), Appropriate (C2), Unambiguous (C3), Complete (C4), Singular (C5), Feasible (C6), Verifiable (C7), Correct (C8), and Conforming (C9) [166]. The characteristics are briefly explained in Table 6.2. The characteristics C1, C2, C5, C8, and C9 pertain to the *formal transformation* of the sources (stakeholders' needs or parent requirements) to acquire the requirements. The characteristics C3, C4, C6, and C7 pertain to the *agreed-to-obligation* between the stakeholders and the system.

6.4.2 INCOSE Guidelines

While ISO/IEEE/IEC 29148:2018 (E) [166] specifies the characteristics of a well-formed requirement, it does not specify how to achieve each of those characteristics, nor does it establish objective rules [66]. For instance, the questions 'how to ensure that the requirement shall be interpretable in only one way (unambiguous)' or 'how to ensure that the requirement statement does not need any additional information (completeness)' still remain unanswered. The guidelines of the

Table 6.2: Characteristics of a well formed requirement [166, 165]

Characteristics	ID	Description
Necessary	C1	Shall define an essential capability, constraint or quality factor. Its exclusion shall deem the system deficient.
Appropriate	C2	The details of the requirement shall be appropriate to the level of abstraction of the requirement.
Unambiguous	C3	The requirement shall be interpretable in only one way.
Complete	C4	Shall define the necessary capability/constraint to meet the need without needing any additional information.
Singular	C5	The requirement shall state a single capability, characteristic, constraint or quality factor.
Feasible	C6	The requirement shall be realizable within the system's constraints (e.g., technology, cost, schedule).
Verifiable	C7	The requirement's realization by the system shall be verifiable. The requirement shall be measurable.
Correct	C8	The requirement shall accurately represent the need from which it was transformed.
Conforming	C9	The requirement shall conform to an approved style of writing or a standard requirement template.

Requirements Working Group of the International Council on Systems Engineering (INCOSE) go a level deeper and specify how to achieve each of those characteristics without being specific to any implementation details [137]. We do not discuss the specifics of the rules for brevity. We discuss our implementation of the INCOSE rules in §6.5. Table 1 in Appendix A summarizes the mapping between the characteristics of well-formed requirements with the INCOSE rules and offers a bird's-eye-view of our implementation.

6.5 Requirement Quality Framework

Our requirement quality framework is based on the premise that every requirement is an instance of natural language. All the rules of NL are applicable to requirements in addition to the requirements of requirements. Thus, we analyze the requirements at the atomic level of NL, the tokens. Tokens refer to words, symbols, punctuation, or numbers in the context of our work. Then, we evaluate the requirements at the span level. *Span* refers to an ordered set of tokens. Often, tokens depend on each other even though they are not in order. So, such dependencies can be analyzed using the syntactic structure of requirements. Lastly, we analyze the requirements at the sentence level. A few things to note are as follows: First, we consider each requirement to consist of a single sentence. In case a requirement consists of multiple sentences, we consider only the first sentence for

analysis. Second, two rules (**R29** and **R42**) are not applicable to the characteristics of individual requirements. Our NLP pipeline to verify the requirements is depicted in Figure 6.1.

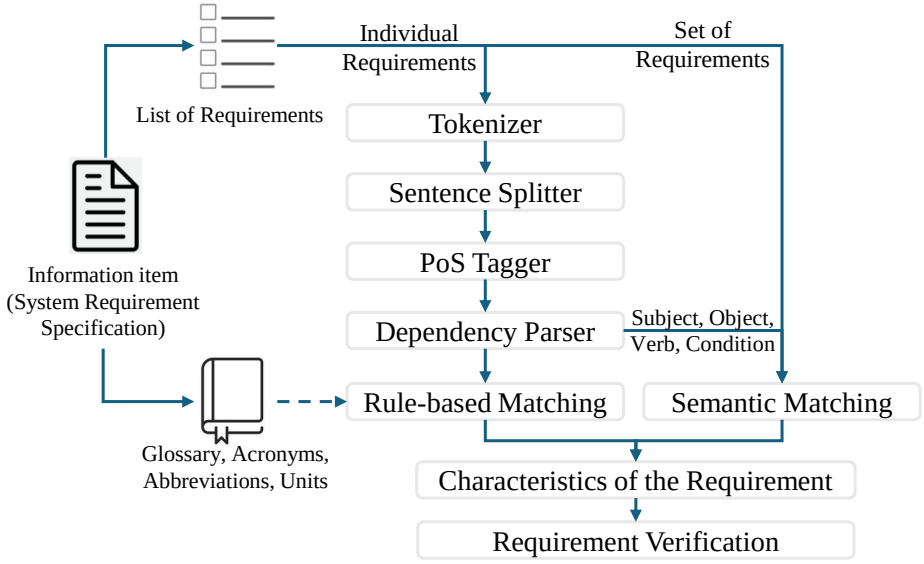


Figure 6.1: The NLP pipeline to verify the requirements by evaluating for the INCOSE Rules and determining the characteristics

6.5.1 Token Based

We implemented many of the rules using tokens or the attributes of the tokens. We attribute each token with both coarse-grained[1] PoS tags as well as their fine-grained PoS tags [131]. There are 20 coarse-grained PoS tags and 56 fine-grained tags in our implementation. The coarse PoS tags of an example requirement are shown in Fig. 6.2. Rule **R5** recommends the usage of the definite article 'the' over the usage of indefinite articles 'a' and 'an'. We check for the coarse-grained tag 'DET' to detect determiners and then verify whether the token is the same as 'the'. See the coarse PoS tag of the first token 'The' in Fig. 6.2. Similarly, **R7** prescribes to avoid vague terms. Vague terms compromise the verifiability of requirements. For example, it is hard to verify whether the requirement 'the mouse shall move smoothly' or 'the system shall display the relevant information' is correct. How much is smooth and what is relevant is hard to verify. We complemented two methods to implement **R7**. First, we used a pattern to detect adverbs. The coarse PoS tag of adverbs is 'adverb'. Then, we use a token library to detect words such as 'relevant'. It includes a list of tokens

[1] `https://universaldependencies.org/u/pos/`

such as ‘several’, ‘allowable’, ‘several’, ‘common’, ‘flexible’, ‘typical’ and ‘sufficient’. We detected pronouns (**R24**) using a mixed approach like **R7**.

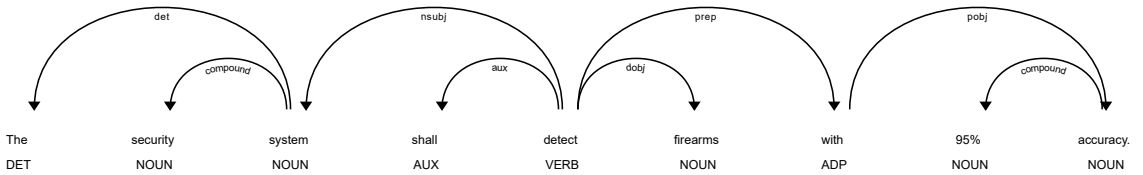


Figure 6.2: The part-of-speech tags and the structure of a sentence in natural language applied to the requirement ‘The security system shall detect firearms with 95% accuracy.’

Rule **R14** advises to use correct punctuation. Moreover, the presence of more punctuation in a requirement makes it more ambiguous. Thus, we use two token-based rules to assess the requirements. We inspect the coarse PoS tags to check if the token is a punctuation. Then, we check for the finer PoS tags to see if the token is a period, a comma, a colon or a hyphen. If there are more than two punctuations in a sentence, we deem the requirement to violate **R14**. The terminal period and a comma within the requirement are permissible without complicating the requirement.

6.5.2 Span Based

This class of implementation focuses on the spans to assess the requirements for the rules. E.g., **R8**, which recommends avoiding the inclusion of escape clauses. We match the span of the requirement with escape clauses such as ‘as little as possible’, ‘if necessary’ and ‘as applicable’. Rule **R9** prescribes avoiding open-ended clauses such as ‘and so on’ and ‘including but not limited to.’ Rule **R20** prohibits the inclusion of phrases indicating reason/intent/purpose in requirements. We include example phrases such as ‘... to ...’, ‘so that’, ‘thus allowing’ and ‘in order to’. We supplemented the list of phrases with additional phrases during our implementation. **R26** recommends against the use of unachievable absolutes such as ‘100%’, ‘every’, ‘all’, ‘never’ and ‘always’. They make the requirements non-verifiable. We detect such phrases along with additional phrases that play a similar role, such as ‘without exception’, ‘at all times’, ‘without interruption’. We use both token-based and span-based features to implement **R26**.

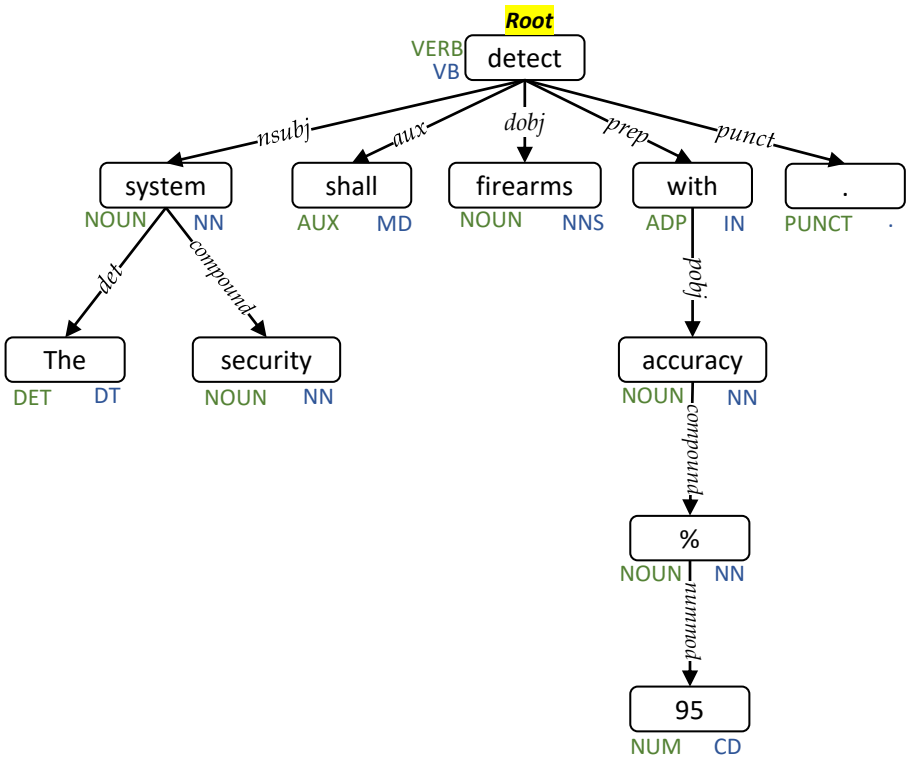


Figure 6.3: The tree structure of the requirement 'The security system shall detect firearms with 95% accuracy.'. The coarse PoS tags and the fine PoS tags of the tokens and the dependencies among the tokens are also depicted.

6.5.3 Syntactic Structure-based

If the order of the tokens is altered, the span-based patterns become ineffective. Patterns based on the syntactic structure of the requirements come in handy in such cases where we exploit the connection between various tokens irrespective of their order. We represent the syntactic structure of the requirements using a tree structure. Every token in the requirement becomes a node in the tree. The syntactic relations between pairs of tokens become the links in the tree. The tree is rooted at the verb token. Other tokens (such as subject, object, period, and preposition) and subtrees (from phrases, such as conditions, constraint and capabilities of requirements) are connected to the main tree or other subtrees. The nodes are attributed with their coarse-grained and fine-grained PoS tags from the Penn Treebank [131]. The tree structure for an example requirement is depicted in Fig. 6.3. Consider **R6**, which recommends the use of common units of measure. The units of measure are nouns, and they succeed numeric values. Units such as 'm/s' and 'V/mm' are a composition of units of measure. We use the dependency of the token to classify it as a unit. If a token's dependency is

'numeric modifier', we label it as a unit. Then, we use a bank of units for each measuring system (such as Metric or Imperial) to check if the unit is consistent with one of the measuring systems. While **R6** needs the link between a pair of tokens, some rules call for scrutinizing the dependencies of multiple tokens. We consider the case of **R2**, use of active voice, to illustrate this. The onus of satisfying a requirement is with the subject of the sentence. The active voice clearly identifies the entity responsible for making the action. A span-based implementation to detect the presence of 'shall be' by generalizing from requirements like 'The firearms shall be detected by the security system.' labels requirements such as 'The length of the firearm shall be within 1m.' as passive voice. The latter is in active voice. In our implementation, we check for the dependencies of the tokens connected to the root verb. If the tokens with 'nominal subject (passive)', 'auxiliary (passive)' dependencies are connected to the root node, we classify the requirement to be in passive voice.

Furthermore, we combine token-based and tree-based rules as per necessity and effectiveness. We implement **R10**, which prescribes avoiding superfluous infinitives. We inspect the tokens and check if their fine-grained PoS tag is 'infinitival to'. If yes, we check for the coarse-grained PoS tag on its head to verify if it is a verb. The requirement contains superfluous infinitives if both conditions are evaluated to be true. We inspect the subtrees in the tree structure of the requirement to verify if a separate clause is used for each condition or qualification (for **R11**).

6.5.4 Sentence Based

We implement a pattern-based approach to evaluate the structure of requirements. These patterns are essentially a series of building blocks (pattern slots) to construct a well-formed requirement. A requirement shall consist of a *capability*, a *condition* and a *constraint*. The basic pattern for a requirement is 'The <entity> shall <do what> <how well> <under what conditions>.' For instance, 'The autonomous driving system shall $\underbrace{\text{safely navigate urban areas}}_{\text{capability}}$ at $\underbrace{\text{a minimum speed of 40 mph}}_{\text{condition}}$ while $\underbrace{\text{adhering to traffic rules}}_{\text{constraint}}$'. This pattern ensures that the requirement is complete and the system is verifiable with the requirement. We implement five patterns in our framework to assess if the requirement is a structured statement (**R1**). The patterns are listed in Table 6.3. We focus on multiple aspects of the requirements to evaluate if a requirement follows one of the patterns. First, we construct the tree structure of the requirements. Adherence to the basic pattern is evaluated by identifying the dependencies of the root verb. The primary subject and primary object of a sentence

Table 6.3: Patterns for requirements

Type	Pattern	Example
Basic	The <SoI> shall <action> <measurable response>.	The security system shall detect firearms with 95% accuracy.
Event-driven	When <trigger>, the <SoI> shall <system response>.	When the sensors detect unauthorized access, the security system shall detect firearms with 95% accuracy.
State-driven	While <precondition)>, the <SoI> shall <system response>.	While the security system is in 'alert' mode, it shall detect firearms with 95% accuracy.
Unwanted behavior	If <trigger>, then the <SoI> shall <system response>.	If a person carries a concealed firearm, the security system shall detect firearms with 95% accuracy.
Optional Feature	Where <feature is included>, the <SoI> shall <system response>.	Where the advanced threat detection module is enabled, the security system shall detect firearms with 95% accuracy.
Complex	More than one of the above cases are applicable.	While the security system is in 'alert' mode, if motion sensors detect unauthorized access into the premises, the security system shall detect firearms with 95% accuracy.

have the 'nominal subject' and 'direct object' dependencies, respectively. The 'measurable response' is a phrase that becomes a sub-tree in the sentence tree. See Fig. 6.3 and 6.4 for the phrase 'with 95% accuracy'. The subtree is attached to one of the primary components of the sentence, the subject, the verb or the object. The subtree is attached to the primary tree with a 'prepositional modifier' or a 'clausal modifier of noun (adjectival clause)' or an 'adverbial clause modifier'. For every requirement, we conduct a minimum check for the presence of the primary subject, verb and object. Then, we check for the presence of the sub-tree. We conclude that a requirement follows the basic pattern if both conditions are satisfied. We want to emphasize that the presence of only ['subject', 'verb', 'object'] still makes it a requirement. It is a functional requirement such as 'The security system shall detect firearms.' Such a requirement does not include a measurable outcome as in how often the security system shall detect or miss detecting firearms so that the system becomes verifiable. Inclusion of a measurable outcome, as in 'The security system shall detect firearms with 95% accuracy' with the preposition 'with' makes the system verifiable. Furthermore, as per the standard INCOSE [137] and EARS [110] guidelines, we classify the requirement as 'event-driven' or 'state-driven' or 'unwanted-behavior' or 'optional feature' based on the presence of the keywords 'when' or 'while' or 'if' or 'where' respectively (See Table 6.3). See Fig. 6.4 illustrating that the

condition in a state-driven requirement is connected to the root verb using an 'adverbial clause modifier.' These patterns place the identifying keywords at the beginning of the sentences without explicitly restricting the locations of these keywords. We do not restrict the location of the keywords in our implementation. Lastly, we classify the requirement as a complex requirement if the requirement satisfies more than one criterion.

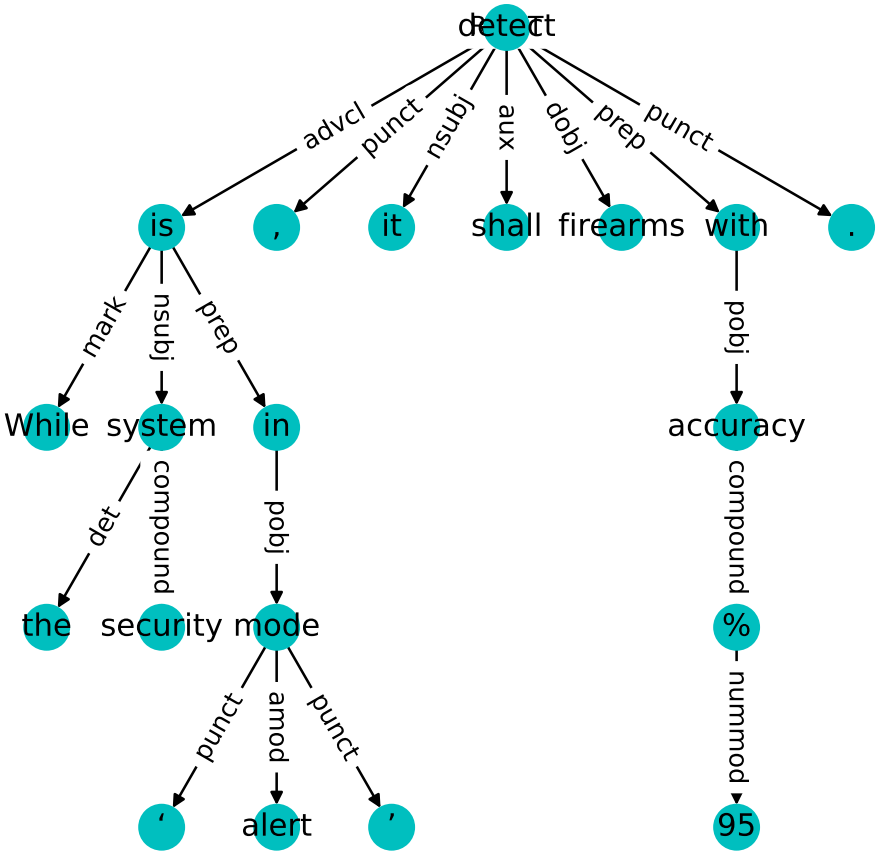


Figure 6.4: The tree structure of a state-driven requirement from Table 6.3 showing that the condition is connected to the root verb with an 'adverbial clause modifier'.

6.5.5 Others

We have implemented rules for which we need information beyond just the requirement itself. For instance, **R4** advises to define all the terms in the requirements with a glossary. This glossary is specific to a set of requirements, as in a requirement specification document or the common terminology of a project. Our framework accepts this glossary and verifies if the requirement contains

one or more terms from the glossary. Similarly, **R12** advises for using correct grammar, and **R13** advises for using correct spelling. We use the LanguageTool[2] for checking for grammar and spelling. It uses 6000+ rules to check for English grammar and spelling [1].

6.6 Well-Formed Requirement Quality Index

We evaluate every requirement for each of the 42 rules. During implementation, we produce binary results: 1 if the requirement satisfies a rule and 0 otherwise. Evaluation of each requirement produces a binary vector of length 42, one dimension for each rule. There are $[2, 3, 30, 16, 8, 2, 25, 11, 10]$ rules applicable for the characteristics [C1, C2, C3, C4, C5, C6, C7, C8, C9] respectively. See the last nine columns (grey color) in Table 1 in Appendix A where 'O' indicates that a rule is applicable to a characteristic. An empty cell indicates that the rule is not applicable for specific characteristics. The *rule-to-characteristics* (RC) mapping matrix for individual requirements is a 42×9 matrix. In case of a list of l requirements, the *normalized characteristics matrix* C^N can be computed as

$$C^N_{l\times 9} = \frac{RR_{l\times 42} \times RC_{42\times 9}}{\mathbf{c}^w_{1\times 9}} \tag{6.1}$$

where RR is the *rule evaluation matrix* for l requirements, and C^N is the *normalized characteristics matrix* for l requirements. The subscripts specify the size of the matrices and vectors. $\mathbf{c}^w$ normalizes the characteristics matrix. $\mathbf{c}^w$ is the characteristic vector of a well-formed requirement, a requirement that satisfies all the rules, *i.e.*, $\mathbf{c}^w = [2, 3, 30, 16, 8, 2, 25, 11, 10]$. The normalization scales the value corresponding to each characteristic in C^N to 1. A requirement that does not satisfy all the rules will have its elements in its characteristic vector less than 1. Substituting $l = 1$ in Eq. 6.1 produces the equation to compute the normalized characteristic vector for a single requirement. The normalized characteristic vector of a well-formed individual requirement is equal to the unit vector of length 9, one for each characteristic. We quantify the well-formed nature of a requirement by the *well-formed requirement quality* (wRQ) index, which is the cosine similarity between the normalized characteristic vector of an individual requirement and that of a well-formed requirement. The well-formed requirement quality index for l requirements is given by

$$wRQ^i_{l\times 1} = C^N_{l\times 9} \odot \mathbf{I}_{1\times 9} \tag{6.2}$$

[2] `https://github.com/jxmorris12/language_tool_python`

where the $\odot$ represents the cosine similarity between the two vectors. $\mathbf{I}$ is the normalized characteristics vector of a well-formed requirement. Since it satisfies all the rules, its normalized characteristic vector becomes a unit vector upon normalization. wRQ^i is the well-formed requirement quality index of l individual requirements. The superscript i represents individual requirements. Finally, we define the individual well-formed requirement quality of a list of requirements (wRQ^l) as the average of the well-formed requirement quality index of each individual requirement. It is defined as

$$wRQ^l = \frac{1}{l}\sum_{i=1}^{l} wRQ^i \tag{6.3}$$

The well-formed requirement quality index is useful for comparing the quality of lists of requirements, such as requirement specification documents. $wRQ \in [0, 1]$, i.e., the well-formed requirement quality index, lies between 0 and 1. A well-formed individual requirement has $wRQ^i = 1$, whereas a requirement that does not satisfy any rule will have $wRQ^i = 0$.

6.7 Results and Discussion

6.7.1 Quality of an Individual Requirement

We illustrate our requirement quality framework by evaluating the individual requirement: 'The autonomous driving system shall safely navigate urban areas at a minimum speed of 40 mph while adhering to traffic rules.' which is listed as *Req1* in Table 6.4. It follows the ISO/IEC/IEEE 29148:2018 definition of a requirement to include a condition and a constraint. Nonetheless, evaluating this requirement with the fourty-two INCOSE rules (See Table 1 in Appendix A) highlights the violation of four rules: **R5** (includes an indefinite article 'a'), **R7** (includes the vague terms 'minimum' and 'safely'), **R33** (did not include a range of values for the speed), **R34** (includes the keyword 'minimum' for a measurable performance). Evaluating this requirement with the rules produce the 42-dimensional rule evaluation vector: $RR_{1\times 42} = [1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]^T$. It has zeros at the $5^{th}, 7^{th}, 33^{rd}$ and the 34^{th} positions indicating that the requirement has violated the rules: **R5**, **R7**, **R33** and **R34**. Multiplying this with the $RC_{42\times 9}$ matrix produces the characteristics vector $C_{1\times 9} = [2, 3, 26, 13, 8, 1, 21, 10, 10]$. The indices of the red-colored values correspond to the imperfect characteristics: unambiguous (C3), complete (C4), feasible (C6), verifiable (C7) and correct (C8). Upon dividing the characteristics

vector C by the well-formed characteristic vector $\mathbf{c}^w$, we acquire the normalized characteristic vector $C^N = [1, 1, 0.8667, 0.8125, 1, 0.5000, 0.8400, 0.9091, 1]$. The values in the normalized characteristic vector which are not equal to 1 represent the imperfect characteristics, and vice-versa. Now, the wRQ index of *Req1* is 0.985353 which is its cosine similarity with the normalized characteristic vector of a well-formed requirement ($=\mathbf{I}_{1\times 9}$, a nine-dimensional unit vector).

6.7.2 Comparison of Requirements

Table 6.4: Well-formed requirement quality of a few sample requirements including their imperfect characteristics and the rules violated by the requirements

ID	Requirement	wRQ	Imperfect Characteristics	Violated Rules
*Req*1	The autonomous driving system shall safely navigate urban areas at a minimum speed of 40 mph while adhering to traffic rules.	0.985353	C3, C4, C6, C7, C8	R5, R7, R33, R34
*Req*2	The firearms shall be detected by the security system.	0.986713	C2, C3, C4, C7, C8, C9	R1, R2
*Req*3	The vehicle shall be able to operate in a variety of temperatures.	0.999425	C3, C4, C7	R5, R7
*Req*4	The ASD shall process all radio messages at a minimum rate of 10 Hz.	0.985360	C3, C4, C6, C7, C8	R5, R32, R33, R34

We show the effectiveness of our requirement quality framework and wRQ by comparatively evaluating few more individual requirements. See Table 6.4. Upon revisiting *Req1*, we notice that the benchmark for the target system is not clear. The specified minimum speed of '40 mph' can be achieved by an autonomous vehicle with a top speed of 60 mph or 80 mph or a vehicle with an average speed below 40 mph which can reach a top speed beyond 40 mph. Moreover, the term 'safely' is hard to define and makes the requirement impossible to validate without the proper definition of 'safe'. Since the requirements are elicited from the perspective of the system, it is not clear (1) if the keyword 'safe' in the requirement refers only to the safety of the vehicle or also includes the safety of its occupants; (2) if the vehicle shall be deemed safe if the vehicle experienced an accident without injuring any of its occupants; (3) if an autonomous vehicle that faced an accident without injuring any of its occupants is equally safe as a vehicle which avoided all accidents. *Req1* can be rewritten as 'The autonomous driving system shall navigate urban areas at the speed of 40 ± 5 mph while adhering to traffic rules.'. The rewritten form does not violate any rules, satisfies all the characteristics, and has a wRQ index equal to one. The rewritten form is a well-formed requirement.

It is shown as *Req2* in Table 6.4. Additionally, *Req1* needs the definition of 'traffic rules' to elicit the expected behaviour of the autonomous vehicle when specific traffic rules are applicable. *Req1* shall be decomposed into child requirements such as 'when a red signal is detected, the autonomous driving system shall stop within 30 ± 5 seconds'.

The requirement *Req*3 in Table 6.4 violates four rules: **R1** as it does not match any of the specified patterns due to lack of a measurable performance, **R2** as it is in passive voice, **R20** for including the purpose phrase 'for the safety of the employees', and **R26** for including the unachievable keyword 'always'. As these four rules altogether affect all the nine characteristics, *Req3* fails to achieve any of the nine characteristics even though its $wRQ = 0.756098$. Nonetheless since only four rules are violated, it is relatively easier to rewrite *Req3* (for instance, 'The security system shall detect firearms with $95 \pm 2\%$ accuracy.') to make it a well-formed requirement.

Now, we consider a requirement *Req*4 from the Deep Orange 13[3] project developed at Clemson University. The project developed a non-combat autonomy-enabled off-road vehicle. *Req*4 violated **R5** (used the indefinite article 'a') and **R7** (has a vague term 'variety'). Specifying the operating conditions as 'a variety of temperatures' makes the requirement ambiguous, incomplete and non-verifiable. Thus, the characteristics C3, C4, C7 are imperfect. Specifying a range of temperatures as '0-40 F' or '80-100 F' would have demanded the vehicle to operate in cold or hot deserts. Our last example, *Req*5, is from the system requirement specification of New York City's connected vehicle pilot deployment program [167]. It failed to satisfy **R5** (presence of the indefinite article 'a'), **R32** (includes 'all' for universal qualification), **R33** (did not use a range of values for '10 Hz') and **R34** (includes the keyword 'minimum' for measurable performance). Consequently, it did not meet the unambiguous (C3), complete (C4), feasible (C6), verifiable (C7) and correct (C8) characteristics.

We want to emphasize the differences between the requirements listed in Table 6.4. The requirements *Req*1 and *Req*3, being random samples, do not have an accompanying glossary (**R4**), a hierarchy (for **R3** to use the subject or verb appropriate to the entity), or a heading (**R25**). These rules are not applicable to *Req*1 and *Req*2. Such rules are evaluated to be true. On the other hand, requirements drawn from system requirement specifications, as *Req*4 from [167], have an accompanying glossary, acronyms, hierarchy and other additional details. More rules are applicable. The additional rules are evaluated accordingly. E.g., *Req*4 has an acronym ASD, an acronym for

[3] `https://cuicardeeporange.com/project/do13/`

aftermarket safety device. Thus, **R37**, using consistent acronyms, is applicable and evaluate to be true. Furthermore, **R23** recommends a supporting diagram or model. This rule fails if the requirement refers to an object and the object does not accompany the requirement. Since none of the example requirements refer to a supporting object, we evaluate them to satisfy the rules.

6.8 Conclusion

We introduce an innovative framework for quantitatively assessing the quality of system requirements. Leveraging advanced rule-based NLP techniques in accordance with ISO/IEC/IEEE 29148:2018 (E) and INCOSE guidelines, we propose the well-formed requirement quality (wRQ) index as a metric for objectively evaluating the quality of individual system requirements. The comprehensive framework outlined herein underscores the critical characteristics that requirements must embody to be deemed well-formed. Those characteristics are necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct, and conforming. We include practical examples and a detailed discussion on the implementation of the wRQ. The wRQ emerges as a pivotal tool for professionals in requirements engineering, offering a precise measure to gauge and enhance the quality of system requirement specifications. In the future, we will extend our framework to accommodate broader and more complex requirement scenarios, such as recursion in sentence-based patterns. Furthermore, we will upgrade our framework from assessing the rules to correcting requirements to satisfy the rules. Our work contributes significantly to the field of requirements engineering and sets a new benchmark for the systematic improvement of requirement quality, ultimately facilitating the development of more reliable, efficient, and successful systems.

Chapter 7

Research Impact

The importance of the research presented in this dissertation lies in its comprehensive exploration of system requirements, which form the backbone of any engineering design or systems engineering process. These works address fundamental challenges in requirements engineering, including hierarchical organization, dataset creation, semantic representation, and quality assessment, and offer applications ranging from system architecture design to automated quality assurance. The details are as follows.

7.1 Requirement Allocation from the *Tree* of *Req*uirements

Chapter 3 emphasizes the importance of understanding the hierarchical structure of SyRSDs. Requirements are often organized hierarchically to ensure traceability from high-level stakeholder needs to detailed, actionable tasks. This hierarchical structure helps in understanding system architecture and supports decomposition, which is crucial for modular system design.

By proving that requirements organized with clause numbers can be represented as a tree structure, this research contributes to formalizing requirement decomposition. The ability to convert these requirements into ReqTrees and weighted trees allows for a quantitative evaluation of requirement distribution across systems and subsystems. This formalism aids in analyzing legacy systems, improving traceability, and optimizing system integration. The applications include

- Reverse Engineering: Facilitates the understanding and documentation of legacy systems.

- System Design: Supports decomposition criteria, such as functional or scenario-based approaches, helping engineers align system design with stakeholder needs.
- Requirements Management: Assists in identifying gaps, redundancies, or over-specified requirements during system development.

7.2 ReqList, ReqNet and ReqSim: Datasets of Requirements

Chapter 4 tackles the challenge of data scarcity in requirements engineering by introducing ReqList, ReqNet, and ReqSim, datasets that enable researchers to apply advanced computational techniques to system requirements. Requirements engineering has traditionally lacked standardized datasets, making it difficult to validate new tools and methodologies.

ReqList captures hierarchical relationships within requirements, ensuring that datasets mirror real-world SyRSDs. ReqNet offers a graph-theoretic representation of requirements, enabling researchers to explore connections between them, such as dependencies and traceability links. ReqSim addresses the growing interest in semantic analysis by providing a dataset annotated with human-labeled semantic similarity scores for requirement pairs. The applications include

- Machine Learning in Requirements Engineering: Enables the application of NLP, graph theory, and supervised learning techniques to improve requirements analysis.
- Requirement Traceability: Supports the development of tools for automated traceability across complex systems.
- Semantic Analysis: Facilitates tasks such as duplicate detection, similarity analysis, and requirement clustering.

By standardizing datasets, this research sets the stage for collaborative development and benchmarking in requirements engineering, encouraging the adoption of computational techniques.

7.3 Semantic Network of Requirements

Chapter 5 advances the field of semantic representation by introducing Req-sBERT, a fine-tuned version of Sentence-BERT specifically trained on requirements. Semantic understanding is

critical in requirements engineering because ambiguous or inconsistent language can lead to project delays or failures.

Req-sBERT represents requirements as contextual embeddings, capturing their meaning in high-dimensional space. These embeddings are then used to create semantic networks, where nodes are requirements, and edges are defined by cosine similarity. This network provides a visual and computational means to analyze relationships, such as semantic similarity and clustering.

The research is validated using a real-world example (Deep Orange 13), demonstrating that the model can track the evolution of requirements over time, such as functional to non-functional transitions. The semantic network facilitates traceability and hierarchical organization, enhancing the ability to manage complex systems. The applications include

- Requirement Clustering: Groups similar requirements for easier management and analysis.
- Traceability: Tracks requirement evolution, ensuring alignment between stakeholder needs and system capabilities.
- Documentation: Hierarchical clustering of the embeddings can assist in the documentation of requirements

This research enables semantic understanding and enhances the reliability of requirement management systems.

7.4 Requirement Verification: Evaluating the Quality of Requirements

Chapter 6 introduces a framework for assessing the quality of requirements, addressing the critical issue of poorly specified requirements that often lead to project failures. The framework adheres to ISO/IEC/IEEE 29148:2018 standards and employs Natural Language Processing (NLP) to evaluate the nine essential characteristics of well-formed requirements: necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct, and conforming.

The framework's innovation lies in the Well-Formed Requirement Quality (wRQ) Index, a quantitative metric that aggregates compliance with these characteristics into a single scalar value. This metric facilitates the objective evaluation of individual requirements, enabling both qualitative

and quantitative comparison. Practical examples demonstrate how the defects in requirements make the requirement vague or incomplete. Such requirements can be rewritten to meet quality standards, improving the reliability of engineering systems. The applications of the requirement verification pipeline include

- Automated Quality Assurance: Provides a systematic approach to identify and rectify issues in requirement specifications.
- Standard Compliance: Ensures that requirements meet international standards, reducing ambiguities and misinterpretations.
- Benchmarking and Comparison: Enables organizations to evaluate and compare the quality of requirement documents systematically.

This research contributes to reducing errors in the requirements phase, where the cost of addressing issues increases exponentially in later stages of the system lifecycle.

7.5 Broader Impact on Engineering Design and Systems Engineering

The research discussed in the dissertation addresses critical challenges in requirements engineering and has far-reaching implications for the domains of engineering design and systems engineering. These fields, which lie at the heart of innovative problem-solving, rely heavily on high-quality requirements to guide the development of systems and products that are efficient, reliable, and tailored to stakeholder needs. By formalizing requirements, providing datasets, leveraging semantic representations, and ensuring quality assessment, this research paves the way for transformative impacts in both domains.

The integration of hierarchical structuring, computational datasets, semantic analysis, and quality assessment forms a cohesive research agenda addressing the lifecycle of system requirements. By tackling challenges from hierarchical organization and dataset creation to semantic representation and quality evaluation, this research creates a comprehensive toolkit for advancing requirements engineering. These advancements improve traceability, reduce ambiguities, and ensure that requirements effectively guide the development of efficient, reliable, and stakeholder-aligned systems and

products. Setting a new standard for precision and adaptability in modern engineering practices, the research enhances innovation and mitigates risks across engineering design and systems engineering. In essence, the research offers

- Improved Formalism: By structuring requirements hierarchically and semantically, the research formalizes the traditionally unstructured nature of requirement documents.
- Data-Driven Insights: The datasets enable scalable computational approaches, opening doors for AI and machine learning applications in requirements engineering.
- Semantic Understanding: Semantic networks ensure that the intent and relationships of requirements are preserved and clarified, reducing errors.
- Quality Assurance: The wRQ Index quantifies requirement quality, providing actionable feedback for improving specifications.

Chapter 8

Future Work

8.1 Semantics of Requirements

It is important to emphasize that we assume in Chapter 4 that the requirements are semantically organized in the SyRSDs. While the *ReqList* and the unweighted *ReqNet* datasets do not have any known shortcomings, we specify two shortcomings of the *ReqSim* dataset and directions to overcome these limitations. The shortcomings emerge from the similarity scores, which are derived from the tree structure created from the clause numbers. First, the tree structure disregards the textual information contained in the requirements corresponding to the clause numbers. A pair of textually identical requirements from two different SyRSDs (E.g., the requirement for the service life of the motor from the SyRSD of a DC motor and that from the SyRSD of an AC motor) may not produce the expected high similarity score. Their similarity score will depend on the similarity of the SyRSDs and the locations of the requirements in the SyRSDs. Incorporating the textual information into the graph using embeddings or other bag-of-words-based methods can improve the effectiveness of the *ReqSim* dataset.

The second criticism stems from the weighing scheme. Both the weighing schemes, top-down and bottom-up, allocate uniform weights to each child node of a parent node. They consider each child node requirement with the same parent node requirement to be equally important. This naive approach lacks a pragmatic sense. For instance, the internal combustion engine and the electric motor in a hybrid vehicle may not produce the same amount of power to propel the vehicle. Thus, all the child nodes with the same parent node are not equally important. Additionally, as a parent

requirement is decomposed into child requirements, the sum of the child requirements shall be equal to the parent requirement in an ideal case. However, the sum of the child nodes is not always equal to the parent node [186]. While this is definitely a shortcoming, investigating further into automated requirement allocation and the semantics of the requirements can potentially help overcome this shortcoming.

8.2 Change Propagation of Requirements

We can represent the requirements in the form of a *multiplex network* [122, 15] that incorporates the requirement tree and the semantic network of a system. Each requirement becomes a node in the multiplex network. Each kind of relationship among the requirements becomes a network, which becomes a layer in the multiplex network. Each of these layers and their interactions can reveal different kinds of interactions and impacts among the requirements. They complement each other in harnessing the documented knowledge of the requirement engineers and the undocumented contextual meaning of the requirements. While the semantic network directly connects each pair of requirements, the requirement tree connects the requirements through the subsystems or components of the system [134]. Changes in the requirements propagate directly to another requirement in the semantic network, whereas they propagate through the interconnecting subsystems in the requirement tree. The analysis for change management consists of three factors: the trigger to the change, the affected requirements and a change propagation model [118]. A change in the requirements is triggered by addition or deletion or modification[1] of one or more requirements [119, 79].

A change in any of the requirements propagates to other requirements in the *ReqTree* through the interconnected subsystems of the CPS. An addition (or deletion) of a requirement in the SyRSD adds (or removes) a clause number from the SyRSD. Subsequently, a node and the corresponding link are added (or removed) from the *ReqTree*. The modification of a requirement does not alter the nodes or links. Contrary to the *ReqTree*, which is an unweighted network in its original form, the semantic network is a weighted network. A change in one requirement directly propagates to other requirements based on the weights of the connected links. Requirements connected to the trigger by links with higher weights get affected more than those connected by weaker links.

[1] A modified requirement can be expressed as the deletion of the original requirement and addition of the modified requirement.

8.3 Verification of Requirements

The limitation of our work in Chapter 6 to verify requirements stems from the limitation of rule-based NLP techniques in capturing the diversity and the rich semantics of NL. The patterns were created by generalizing a set of instances of requirements. While the patterns generalize well, exceptions do exist. For instance, **R16** prescribes to avoid the use of 'not'. The INCOSE GtWR [137] cites two reasons for **R16**. First, the inclusion of the word 'not' in a requirement implies a 'never' behavior for the system. Such a behavior is hard to verify in a finite time. Second, there are a large number of actions the system should not take. INCOSE GtWR permits the usage of 'not' in requirements such as `the system shall not be in red color`. The usage of 'not' is permitted if the requirement can be unambiguously verified. It is hard to list all such cases and develop a pattern for them. Second, the implementations of the rules that need semantic understanding are not fool-proof. Examples include **R29**, **R30**, **R41** and **R42**. These requirements need machine learning-based language models to be more effective. Employing Large Language Models (LLMs), such as ChatGPT [5], in the NLP pipeline can come in handy in incorporating the semantics of requirements to overcome these limitations.

Chapter 9

Conclusion

Systems are developed to satisfy the needs of the stakeholders. The stakeholders' needs are formally transformed into requirements that serve as an agreed-to-obligation among the stakeholders and the system itself. The system development process begins with eliciting the system's requirements and terminates upon validating the system against the requirements and the needs. The requirements are elicited, modeled and represented, analyzed, verified, validated and documented as part of the systems engineering process. These activities constitute requirements engineering (RE). Each stage of RE faces unique challenges that impact the development of the system. Poorly defined, specified, or managed requirements risk system failure, sub-optimal performance, schedule delays, and budget overruns.

The system design process starts with eliciting a set of requirements for the system. The elicitation process includes surveying stakeholders, conducting hands-on training, comparing similar products, and investigating applicable regulations, among other activities. The elicited requirements are generally saved as proprietary information to protect competitive advantage by preventing the disclosure of the product's features and details. However, the creative nature of the elicitation process limits the comprehensiveness of requirements, while their proprietary nature restricts open sourcing. To address these limitations, we collected a set of open-source requirement specifications. We processed and released the largest requirements dataset to date to accelerate the application of modern computing techniques in RE. Our dataset has three inter-convertible forms: *ReqList*, *ReqNet*, and *ReqSim*. *ReqList* consists of a *list* of 12701 *req*uirements, ready for deploying natural language processing (NLP) and unsupervised machine learning techniques. *ReqNet* consists of a *net*work of

*req*uirements with 17375 nodes, supporting graph-theoretic and network analysis techniques for RE. *ReqSim* consists of 10933 pairs of *req*uirements annotated with *sem*antic similarity scores aligned with human judgment, facilitating supervised learning for RE. Our dataset is expandable using the formal method we developed to create the dataset.

The stakeholders' needs are transformed into requirements. These requirements have a higher abstraction. These high-level requirements are then decomposed recursively as part of the requirement definition process to yield a set of low-level requirements with reduced complexity for which a buy/build/code/reuse decision can be made. This decomposition process establishes a hierarchy. We represent a system's requirements as a tree where the requirements are the leaf nodes. The root node represents the system itself, as described in the system requirement specification document (SyRSD). The branch nodes represent other artifacts corresponding to the sections and subsections of the SyRSD. The tree reflects the requirement allocation process, which decomposes the role of each node among its child nodes. The criteria of requirement allocation are not unique to a system. At each node of the tree, where a decomposition occurs, one among many possible criteria for allocating requirements is chosen. The criteria for requirement allocation include system decomposition, functional decomposition, decomposition of use-cases, states, interfaces, objects, ilities (qualITY attributes such as maintainabilITY, durabilITY, reliabilITY, usabilITY) or any other logical (Logical) or semantic criteria.

The requirements, being an instance of NL, are blessed as well as cursed. While they are easier to elicit, the rich but diverse semantics of NL prevent the requirements from a consistent analysis and inference. Human intervention is inevitable to extract the relationships among the requirements and represent and analyze the requirements. Thus, we develop a graph-theoretic approach to estimate the similarity between pairs of requirements. The similarity scores are coherent with the human judgment of the semantics of the requirements. Our semantic similarity is underpinned by the tree structure of requirements. The tree structure is limited to the requirements listed in SyRSDs. To overcome this, we create *Req-sBERT*, a large language model for embedding requirements, from which the semantics can be inferred. *Req-sBERT* is developed by fine-tuning the MPNet model within the Siamese architecture of the Sentence-BERT model, supplemented with a 6-layered deep neural network. Sentence-BERT's pretraining retains the rich semantics of NL while fine-tuning on the *ReqSim* dataset adapts the model specifically for RE. When applied to requirements of the Deep Orange 14 program, *Req-sBERT* demonstrated semantics consistent with that of humans. We

represent these requirements as a weighted semantic network, where nodes represent requirements and edge weights represent semantic similarity. The semantic similarity scores can be used to detect the evolution of requirements from a functional requirement to a non-functional requirement, and a hierarchical tree structure can be extracted from the semantic network to organize the requirements as in a SyRSD.

The requirements, being instances of NL, lack formalism at both the individual level and set level. The lack of formalism fails the requirements to serve as an agreed-to-obligation among the system's stakeholders. Poorly specified requirements risk the system aspiring to meet unnecessary, non-verifiable, incomplete, ambiguous, and solution-specific requirements, thereby compromising the developmental integrity and potential success of the system. Thus, we develop a requirement verification framework. Our framework scrutinizes the words, phrases, sentences, grammar and semantics of individual requirements. We scrutinize the requirements for the 42 rules prescribed by the 'INCOSE Guide to Writing Requirements in order to achieve the nine characteristics of well-formed requirements listed in the 'ISO/IEC/IEEE 29148:2018 - Systems and software engineering — Life cycle processes — Requirements engineering' standard. We evaluate if the individual requirements are necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct and conforming. Furthermore, we propose a metric, the well-formed Requirement Quality (wRQ) Index, to quantitatively evaluate the quality of a requirement. wRQ enables the comparison of the qualities of different requirements and of the same requirement at different stages of its evolution.

In summary, this dissertation improves the RE process across multiple dimensions. It enhances the requirement elicitation process by offering a RE dataset in three inter-convertible forms and provides a formal method to expand the dataset. It represents the requirements as both a tree, reflecting the requirement allocation process, and a semantic network that captures the semantics of the requirements to support the representation of requirements. This dissertation investigates the criteria of requirement allocation within a system, the evolution of requirements, and automated documentation of requirements as part of requirement analysis. Then, it proposes a requirement verification framework to assess the quality of requirements and introduces the wRQ metric to compare the requirement quality. Overall, this dissertation contributes to the holistic improvement of the RE process in engineering design and systems engineering.

Chapter 10

Knowledge Dissemination

10.1 Journals

- Rahul Rai and **Chandan K. Sahu**. Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. IEEE Access, 8:71050–71073, 2020, DOI:10.1109/ACCESS.2020.2987324 [134]

- **Chandan K. Sahu**, Crystal Young and Rahul Rai. Artificial intelligence (ai) in augmented reality (ar)-assisted manufacturing applications: a review. International Journal of Production Research, 59(16):4903–4959, 2021. DOI:10.1080/00207543.2020.1859636 [25]

- Dhruv Patel, **Chandan K. Sahu** and Rahul Rai. Security in modern manufacturing systems: integrating blockchain in artificial intelligence-assisted manufacturing. International Journal of Production Research, 62(3):1041–1071, 2024. DOI:10.1080/00207543.2023.2262050 [37]

- Zhibo Zhang, **Chandan K. Sahu**, Shubhendu Kumar Singh, Rahul Rai, Zhuo Yang and Yan Lu. Machine learning based prediction of melt pool morphology in a laser-based powder bed fusion additive manufacturing process. International Journal of Production Research, 62(5):1803–1817, 2024. DOI:10.1080/00207543.2023.2201860 [197]

- **Chandan K. Sahu**, Rahul Rai, Margaret Wiecek, and David Gorsich. ReqNet and ReqSim: A Network and Semantic Similarity Dataset of Requirements from the Tree Structure of System Requirement Specifications. Journal of Computing and Information Science in Engineering,

pages 1–15, 06 2024.DOI:10.1115/1.4065786 [146]

- **Chandan K. Sahu**, R. Rai, M. P. Castanier, and D. Gorsich, "A Semantic Network of Requirements from the Contextual Embeddings of Req-sBERT," Under review at ASME Journal of Mechanical Design. [144]

- **Chandan K. Sahu**, R. Rai and M. Wiecek, "Requirement Allocation from the Tree Structure of Requirement Specification Documents," Under review at ASME Journal of Mechanical Design. [145]

- **Chandan K. Sahu**, Vinayak Khade, Mohan S. R. Elapolu, Rahul Rai, Matthew P. Castanier, and David Gorsich, "ReqCity: A City-based Visualization for the Verification of System Requirements to Conform to the ISO/IEC/IEEE 29148:2018 (E) Standard," Under review at IEEE Transactions on Software Engineering. [141]

10.2 Conferences

- **Chandan K. Sahu**, Vedant Rai, and Vinit Roshan. Well-formed quality of system requirements for verifying to ISO 29148-2018: A natural language processing (NLP) based framework and quantitative metric. In International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (Accepted for publication). American Society of Mechanical Engineers, 2024. [91]

10.3 Theses

- **Chandan K. Sahu**, Boon of hybrid approaches over the bane of model based and machine learning approaches in modelling cyber-physical systems, Master's Thesis, State University of New York at Buffalo [142]

Appendices

Appendix A INCOSE Rule-to-Characteristics Mapping Matrix for the characteristics of individual requirements annotated with the type of implementation

Table 1: Rules to characteristics (RC) mapping matrix [137]. The *characteristics* of individual requirements are necessary (C1), appropriate (C2), unambiguous (C3), complete (C4), singular (C5), feasible (C6), verifiable (C7), correct (C8), and conforming (C9). The *type* of implementation is split into token-based (T), span-based (s), syntactic structure-based (Sy), sentence-based (S) and others (O)

	Rule	Description	Type	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1	Use Structured Statements	Conform to one of the agreed patterns	S			O	O			O	O	O
R2	Use Active Voice	With the responsible entity as the subject	Sy		O	O	O			O		
R3	Appropriate Subject-Verb	Use subject and verb appropriate to the entity	T		O	O				O		
R4	Use Defined Terms	Define all the terms in an associated glossary	T			O				O		
R5	Use Definite Articles	Use 'the' instead of 'a' or 'an'	T			O				O		
R6	Common Units of Measure	Consistent units with common measurement system	T			O	O			O	O	
R7	Avoid vague Terms	E.g., 'some', 'relevant', 'efficient', 'typical'	T			O	O			O		
R8	Avoid Escape Clauses	E.g.,'if necessary', 'as appropriate', 'as required'	s			O				O		
R9	Avoid Open-ended Clauses	E.g., 'including but not limited to', 'and so on'	s			O	O	O		O		
R10	Avoid Superfluous infinitives	E.g., 'to be able to', 'to allow', 'to enable'	s			O				O		
R11	Separate Clauses	Use a separate clause for each condition/qualification	Sy			O	O			O	O	
R12	Correct Grammar	Use grammatically correct sentences	O			O				O	O	O
R13	Correct Spelling	Use correct spelling	O			O				O		
R14	Correct Punctuation	Use correct and fewer punctuations	T			O					O	
R15	Logical Expressions	Use a defined convention for logical expressions	Sy			O				O		

...continued on next page

Table 1 – continued from previous page

	Rule	Description	Type	C1	C2	C3	C4	C5	C6	C7	C8	C9
R16	Avoid the Use of 'not'	Avoid negation in requirements	T			O				O	O	
R17	Avoid the Oblique Symbol	Do not use the oblique ('/') symbol	T			O				O		
R18	Single-thought Sentence	Single sentence qualified by relevant sub-clauses	Sy			O		O		O		O
R19	Avoid Combinators	E.g., 'and', 'then', 'or', 'but', 'whether', 'whether'	T			O		O				
R20	Avoid Purpose Phrases	Indicates intent/reason, e.g., '... because ...', '... to ...'	s	O				O				
R21	Avoid Parentheses and Brackets	They contain subordinate text.	T					O				
R22	Enumerate Sets Explicitly	Instead of using a group noun to name the set	T			O		O				
R23	Supporting Diagram, Model	Refer to relevant figure/diagram/model	T			O	O	O				
R24	Avoid Pronouns	Avoid personal and indefinite pronouns	T			O	O			O		
R25	Avoid Reliance on Headings	Avoid relying on headings to get context	T				O					
R26	Avoid Unachievable Absolutes	E.g., '100%', 'all', 'always', 'never'	T						O	O	O	
R27	Use Explicit Conditions	State conditions' applicability explicitly	Sy				O			O	O	
R28	Use Multiple Conditions	Multiple conditions for single action is better	Sy			O				O		
R29	Classify Requirements	According to the aspects of the problem or system	NA									
R30	Unique Expression	Express each requirement once and only once	S	O								O
R31	Be Solution Free	Avoid stating implementation in a requirement	Sy		O							
R32	Universal Qualification	Use 'each' instead of 'all', 'any' or 'both'	T			O				O	O	
R33	Use a Range of Values	Define each quantity with a range of values	Sy			O	O		O	O	O	
R34	Measurable Performance	Provide specific measurable performance targets	T			O	O			O		

...continued on next page

Table 1 – continued from previous page

	Rule	Description	Type	C1	C2	C3	C4	C5	C6	C7	C8	C9
R35	Explicit Temporal Dependencies	Instead of indefinite ones: 'until', 'simultaneous'	T			O	O			O		
R36	Consistent Terms and Units	Consistent across the project's lifecycle and ontology	T			O					O	O
R37	Use Consistent Acronyms	Consistent throughout requirements and the lifecycle	T			O						O
R38	Avoid Abbreviations	Avoid in requirements, models and lifecycle artefacts	Sy									O
R39	Use a Style Guide	Use a project-wide style guide for all requirements	T				O	O				O
R40	Consistent Decimal Format	Use a consistent format and number of signification digits	T			O	O					O
R41	Related Requirements	Group related requirements together	S				O					O
R42	Structured Sets	Conform to a template for organizing requirements	NA									

Bibliography

[1] Languagetool community: Error rules for languagetool. `https://community.languagetool.org/rule/list?lang=en&offset=300&max=1000`.

[2] Iso/iec/ieee international standard - systems and software engineering–system life cycle processes. *ISO/IEC/IEEE 15288:2023(E)*, pages 1–128, 2023.

[3] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. What works better? a study of classifying requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 496–501. IEEE, 2017.

[4] Daniel Aceituna, Gursimran Walia, Hyunsook Do, and Seok-Won Lee. Model-based requirements verification method: Conclusions from two controlled experiments. *Information and Software Technology*, 56(3):321–334, 2014.

[5] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[6] Muideen Ajagbe and Liping Zhao. Retraining a bert model for transfer learning in requirements engineering: A preliminary study. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*, pages 309–315. IEEE, 2022.

[7] Khalil Al Handawi, Massimo Panarotto, Petter Andersson, Ola Isaksson, and Michael Kokkolaras. Optimization of Design Margins Allocation When Making Use of Additive Remanufacturing. *Journal of Mechanical Design*, 144(1):012001, 07 2021.

[8] Anas Alfaris, Afreen Siddiqi, Charbel Rizk, Olivier de Weck, and Davor Svetinovic. Hierarchical Decomposition and Multidomain Formulation for the Design of Complex Sustainable Systems. *Journal of Mechanical Design*, 132(9):091003, 09 2010.

[9] Waad Alhoshan, Alessio Ferrari, and Liping Zhao. Zero-shot learning for requirements classification: An exploratory study. *Information and Software Technology*, 159:107202, 2023.

[10] Marco A Alvarez and SeungJin Lim. A graph modeling of semantic similarity between words. In *International Conference on Semantic Computing (ICSC 2007)*, pages 355–362. IEEE, 2007.

[11] Sequoia R Andrade and Hannah S Walsh. Discovering a failure taxonomy for early design of complex engineered systems using natural language processing. *Journal of Computing and Information Science in Engineering*, 23(3):031001, 2023.

[12] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated checking of conformance to requirements templates using natural language processing. *IEEE transactions on Software Engineering*, 41(10):944–968, 2015.

[13] W Beitz, G Pahl, and K Grote. Engineering design: a systematic approach. *Mrs Bulletin*, 71, 1996.

[14] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[15] Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesús Gómez-Gardenes, Miguel Romance, Irene Sendina-Nadal, Zhen Wang, and Massimiliano Zanin. The structure and dynamics of multilayer networks. *Physics reports*, 544(1):1–122, 2014.

[16] Barry Boehm, Ricardo Valerdi, and Eric Honour. The roi of systems engineering: Some quantitative results for software-intensive systems. *Systems Engineering*, 11(3):221–234, 2008.

[17] Barry W. Boehm. Verifying and validating software requirements and design specifications. *IEEE software*, 1(1):75, 1984.

[18] Eric Bonjour, Samuel Deniaud, Maryvonne Dulmet, and Ghassen Harmel. A Fuzzy Method for Propagating Functional Architecture Constraints to Physical Architecture. *Journal of Mechanical Design*, 131(6):061002, 04 2009.

[19] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[20] William Brace and Vincent Cheutet. A framework to support requirements analysis in engineering design. *Journal of Engineering Design*, 23(12):876–904, 2012.

[21] Ulrik Brandes. *Network analysis: methodological foundations*, volume 3418. Springer Science & Business Media, 2005.

[22] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational linguistics*, 32(1):13–47, 2006.

[23] Ronald S Carson and Robert A Noel. Formalizing requirements verification and validation. In *INCOSE International Symposium*, volume 28, pages 805–818. Wiley Online Library, 2018.

[24] Youssef Chahadi and Herbert Birkhofer. Semantic product requirement network as approaches for a requirement terminology and tool for linguistic analysis of requirements in continuous text. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 43277, pages 223–230, 2008.

[25] Crystal Young Chandan K. Sahu and Rahul Rai. Artificial intelligence (ai) in augmented reality (ar)-assisted manufacturing applications: a review. *International Journal of Production Research*, 59(16):4903–4959, 2021.

[26] Cheligeer Cheligeer, Jiami Yang, Amin Bayatpour, Alexandra Miklin, Stéphane Dufresne, Lan Lin, Nadia Bhuiyan, and Yong Zeng. A hybrid semantic networks construction framework for engineering design. *Journal of Mechanical Design*, 145(4):041405, 2023.

[27] Ting-Ju Chen and Vinayak R Krishnamurthy. Investigating a mixed-initiative workflow for digital mind-mapping. *Journal of Mechanical Design*, 142(10):101404, 2020.

[28] Yan Chen, Ping Cheng, and Jing Yin. Change propagation analysis of trustworthy requirements based on dependency relations. In *2010 2nd IEEE International Conference on Information Management and Engineering*, pages 246–251. IEEE, 2010.

[29] Jane Cleland-Huang, Sepideh Mazrouee, Huang Liguo, and Dan Port. Nfr, 2007.

[30] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated classification of non-functional requirements. *Requirements engineering*, 12(2):103–120, 2007.

[31] Nigel Cross. *Engineering design methods: strategies for product design*. John Wiley & Sons, 2021.

[32] Souvick Das, Novarun Deb, Agostino Cortesi, and Nabendu Chaki. Sentence embedding models for similarity detection of software requirements. *SN Computer Science*, 2:1–11, 2021.

[33] Adailton Ferreira de Araújo and Ricardo Marcondes Marcacini. Re-bert: automatic extraction of software requirements from app reviews using bert language model. In *Proceedings of the 36th annual ACM symposium on applied computing*, pages 1321–1327, 2021.

[34] Juan M Carrillo De Gea, Joaquín Nicolás, José L Fernández Alemán, Ambrosio Toval, Christof Ebert, and Aurora Vizcaíno. Requirements engineering tools: Capabilities, survey and assessment. *Information and Software Technology*, 54(10):1142–1157, 2012.

[35] Thiago C de Sousa, Jorge R Almeida Jr, Sidney Viana, and Judith Pavón. Automatic analysis of requirements consistency with the b method. *ACM SIGSOFT Software Engineering Notes*, 35(2):1–4, 2010.

[36] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[37] Chandan Kumar Sahu Dhruv Patel and Rahul Rai. Security in modern manufacturing systems: integrating blockchain in artificial intelligence-assisted manufacturing. *International Journal of Production Research*, 62(3):1041–1071, 2024.

[38] Jeremy Dick, Elizabeth Hull, and Ken Jackson. *Requirements engineering*. Springer, 2017.

[39] Jeremy Dick and Brendan Jones. 8.7. 2 on the complexity of requirements flow-down structures. In *INCOSE International Symposium*, volume 22, pages 1197–1206. Wiley Online Library, 2012.

[40] Reinhard Diestel. Graduate texts in mathematics: Graph theory. *Heidelb: SpringerVerlag*, 2000.

[41] Telelogic DOORS. Manual "get it right the first time: Writing better requirements". *IBM® Corporation*, 2008.

[42] George T Doran. There's a smart way to write managements's goals and objectives. *Management review*, 70(11), 1981.

[43] Kristen Edwards, Binyang Song, Jaron Porciello, Mark Engelbert, Carolyn Huang, and Faez Ahmed. Advise: Accelerating the creation of evidence syntheses for global development using natural language processing-supported human-ai collaboration. *Journal of Mechanical Design*, pages 1–15, 2023.

[44] Mohan SR Elapolu, Rahul Rai, David J Gorsich, Denise Rizzo, Stephen Rapp, and Matthew P Castanier. Blockchain technology for requirement traceability in systems engineering. *Information Systems*, 123:102384, 2024.

[45] Mohan SR Elapolu, Md Imrul Reza Shishir, and Alireza Tabarraei. Applied machine learning method to predict crack propagation path in polycrystalline graphene sheet. In *ASME International Mechanical Engineering Congress and Exposition*, volume 85680, page V012T12A011. American Society of Mechanical Engineers, 2021.

[46] Mohan SR Elapolu, Md Imrul Reza Shishir, and Alireza Tabarraei. A novel approach for studying crack propagation in polycrystalline graphene using machine learning algorithms. *Computational Materials Science*, 201:110878, 2022.

[47] Gauthier Fanmuy, Anabel Fraga, and Juan Llorens. Requirements verification in the industry. In *Complex Systems Design & Management: Proceedings of the Second International Conference on Complex Systems Design & Management CSDM 2011*, pages 145–160. Springer, 2012.

[48] Mamdouh Farouk. Sentence semantic similarity based on word embedding and wordnet. In *2018 13th International Conference on Computer Engineering and Systems (ICCES)*, pages 33–37. IEEE, 2018.

[49] Christiane Fellbaum. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, 2010.

[50] Henning Femmer, Axel Müller, and Sebastian Eder. Semantic similarities in natural language requirements. In *Software Quality: Quality Intelligence in Software and Systems Engineering: 12th International Conference, SWQD 2020, Vienna, Austria, January 14–17, 2020, Proceedings 12*, pages 87–105. Springer, 2020.

[51] Shaw C Feng, Tesfaye Moges, Hyunseop Park, Mostafa Yakout, Albert T Jones, Hyunwoong Ko, and Paul Witherell. Functional requirements of software tools for laser-based powder bed fusion additive manufacturing for metals. *Journal of Computing and Information Science in Engineering*, 23(3):031005, 2023.

[52] Alessio Ferrari, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi, and Stefania Gnesi. Natural language requirements processing: A 4d vision. *IEEE Softw.*, 34(6):28–35, 2017.

[53] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505. IEEE, 2017.

[54] Rafael Ferreira, Rafael Dueire Lins, Steven J Simske, Fred Freitas, and Marcelo Riss. Assessing sentence similarity through lexical, syntactic and semantic analysis. *Computer Speech & Language*, 39:1–28, 2016.

[55] David Flanigan and Peggy Brouse. 11.4. 2 measuring the requirements allocation capacity within a system of systems. In *INCOSE International Symposium*, volume 22, pages 1671–1682. Wiley Online Library, 2012.

[56] David Flanigan and Peggy Brouse. System of systems requirements capacity allocation. *Procedia Computer Science*, 8:112–117, 2012.

[57] David Flanigan and Peggy Brouse. Evaluating the allocation of border security system of systems requirements. *Procedia Computer Science*, 16:631–638, 2013.

[58] Jose Fuentes, Anabel Fraga, Gonzalo Génova, Eugenio Parra, Jose María Alvarez, and Juan Llorens. Applying incose rules for writing high-quality requirements in industry. In *INCOSE International Symposium*, volume 26, pages 1875–1889. Wiley Online Library, 2016.

[59] Steve Galgano, Mohamad Talas, Robert Benevelli, David Rausch, Samuel Sim, Chris Stanley, Keir Opie, Denny Stephens, and Douglas Pape. Connected vehicle pilot deployment program phase 1, system requirements specification (syrs) – new york city. `https://rosap.ntl.bts.gov/view/dot/31403`, 2021.

[60] Gonzalo Génova, José M Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. A framework to measure and improve the quality of textual requirements. *Requirements engineering*, 18:25–41, 2013.

[61] Monica Giffin, Olivier de Weck, Gergana Bounova, Rene Keller, Claudia Eckert, and P. John Clarkson. Change Propagation Analysis in Complex Technical Systems. *Journal of Mechanical Design*, 131(8), 07 2009. 081001.

[62] Michael W. Goodman. Wordnet: Taxonomy-based metrics. `https://github.com/goodmami/wn`, 2020.

[63] Orlena CZ Gotel and CW Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of ieee international conference on requirements engineering*, pages 94–101. IEEE, 1994.

[64] Jeffrey O Grady. *System requirements analysis*. Elsevier, 2010.

[65] Emilyn Green, Spenser Estrada, Praveen Kumare Gopalakrishnan, Sogol Jahanbekam, and Sara Behdad. A graph partitioning technique to optimize the physical integration of functional requirements for axiomatic design. *Journal of Mechanical Design*, 144(5):051402, 2022.

[66] Sarah Gregory. Requirements engineering: The quest for meaningful metrics: Time for a change? *IEEE Software*, 36(6):7–11, 2019.

[67] Carl A Gunter, Elsa L Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, 2000.

[68] Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20. IEEE, 2017.

[69] Ji Han, Serhad Sarica, Feng Shi, and Jianxi Luo. Semantic networks for engineering design: state of the art and future directions. *Journal of Mechanical Design*, 144(2):020802, 2022.

[70] Jane Huffman Hayes, Alex Dekhtyar, and Jared Payne. The requirements tracing on target (retro). net dataset. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 424–427. IEEE, 2018.

[71] Jane Huffman Hayes, Jared Payne, and Mallory Leppelmeier. Toward improved artificial intelligence in requirements engineering: metadata for tracing datasets. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 256–262. IEEE, 2019.

[72] George A. Hazelrigg and Philip Stolfi. The Cost on System Performance of Requirements on Differentiable Variables. *Journal of Mechanical Design*, 143(5):054501, 01 2021.

[73] Erik Herzog. *An approach to systems engineering tool data representation and exchange*. PhD thesis, Linköping University Electronic Press, 2004.

[74] Steven R Hirshorn, Linda D Voss, and Linda K Bromley. Nasa systems engineering handbook. Technical report, 2017.

[75] Elizabeth Hull, Ken Jackson, Jeremy Dick, et al. *Requirements Engineering*. London: Springer London: Imprint: Springer,, 2010.

[76] INCOSE. *INCOSE systems engineering handbook*. John Wiley & Sons, 2023.

[77] INCOSE. *INCOSE systems engineering handbook*. John Wiley & Sons, 2023.

[78] Maddalena Jackson and Marcus Wilkerson. Mbse-driven visualization of requirements allocation and traceability. In *2016 IEEE Aerospace Conference*, pages 1–17. IEEE, 2016.

[79] Shraddha Joshi and Joshua D Summers. Requirements change: Understanding the type of changes in the requirements document of novice designers. *International Journal of Mechanical Engineering Education*, 43(4):286–304, 2015.

[80] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[81] D Juraksky and James H Martin. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. *Pearson education, Asia*, 2000.

[82] Olivia Kenney and Matt Cooper. Automating requirement quality standards with qvscribe. In *REFSQ Workshops*, 2020.

[83] Vinayak Khade, Nafiseh Masoudi, Dane Acena, Guo Freeman, Rahul Rai, David Gorsich, Denise Rizzo, and Matt Castanier. Requirements elicitation: Impacts of gamification on variety, novelty, and completeness. In *ASME International Mechanical Engineering Congress and Exposition*, volume 86663, page V004T06A019. American Society of Mechanical Engineers, 2022.

[84] Raj Pradip Khawale, Suparno Bhattacharyya, Rahul Rai, and Gary F Dargush. Efficient dynamic topology optimization of 2d metamaterials based on a complementary energy formulation. *Computers & Structures*, 299:107371, 2024.

[85] Raj Pradip Khawale, Greg Vinal, Rahul Rai, William W Menasco, and Gary F Dargush. Tiling-based lattice generation for structural property exploration. *Materials & Design*, 247:113391, 2024.

[86] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[87] Stefan Klink, Andreas Dengel, and Thomas Kieninger. Document structure analysis based on layout and textual features. In *Proc. of International Workshop on Document Analysis Systems, DAS2000*, pages 99–111, 2000.

[88] Eric Knauss and Daniel Ott. (semi-) automatic categorization of natural language requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 39–54. Springer, 2014.

[89] H Komoto and T Tomiyama. Multi-disciplinary system decomposition of complex mechatronics systems. *CIRP annals*, 60(1):191–194, 2011.

[90] Gerald. Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. J. Wiley, 1998.

[91] Chandan Kumar Sahu, Vedant Rai, and Vinit Roshan. Well-formed quality of system requirements for verifying to iso 29148-2018: A natural language processing (nlp) based framework and quantitative metric. volume Volume 2B: 44th Computers and Information in Engineering Conference (CIE) of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, page V02BT02A013, 08 2024.

[92] Tolga Kurtoglu and Irem Y. Tumer. A Graph-Based Fault Identification and Propagation Framework for Functional Design of Complex Systems. *Journal of Mechanical Design*, 130(5):051401, 03 2008.

[93] Michael A Kurtz, Ruoyu Yang, Mohan SR Elapolu, Audrey C Wessinger, William Nelson, Kazzandra Alaniz, Rahul Rai, and Jeremy L Gilbert. Predicting corrosion damage in the human body using artificial intelligence: in vitro progress and future applications. *Orthopedic Clinics*, 54(2):169–192, 2023.

[94] Michael A Kurtz, Ruoyu Yang, Dinghe Liu, Mohan SR Elapolu, Rahul Rai, and Jeremy L Gilbert. Deep neural network predicts ti-6al-4v dissolution state using near-field impedance spectra. *Advanced Functional Materials*, 34(4):2308932, 2024.

[95] Andrew Kusiak and Fang Qin. Requirements allocation. *Handbook of Measuring System Design*, 2005.

[96] Carl Lamar. *Linguistic analysis of natural language engineering requirements.* PhD thesis, Clemson University, 2009.

[97] Jesko G Lamm, Tim Weilkiens, M Maurer, and SO Schulze. Functional architectures in sysml. *Proceedings of the TdSE*, 2010.

[98] Phillip A Laplante and Mohamad Kassab. *Requirements engineering for software and systems.* Auerbach Publications, 2022.

[99] Alex Lash, Kevin Murray, and Gregory Mocko. Natural language processing applications in requirements engineering. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 45011, pages 541–549. American Society of Mechanical Engineers, 2012.

[100] Claudia Leacock. Combining local context and wordnet similarity for word sense identification. *WordNet: A Lexical Reference System and its Application*, pages 265–283, 1998.

[101] Dean Leffingwell and Don Widrig. *Managing software requirements: a unified approach.* Addison-Wesley Professional, 2000.

[102] Mark T Lemke, Robert B Stone, and Ryan A Arlitt. Ontologies to support customer requirement formulation in aerospace design. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 58219, page V007T06A018. American Society of Mechanical Engineers, 2017.

[103] Haomin Li, Xinai Zhang, Chao Tang, and Chao Zhan. A functional requirements development and management approach applied to a civil aircraft program. In *The Proceedings of the 2018 Asia-Pacific International Symposium on Aerospace Technology (APISAT 2018) 9th*, pages 3025–3039. Springer, 2019.

[104] Yuhua Li, David McLean, Zuhair A Bandar, James D O'shea, and Keeley Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE transactions on knowledge and data engineering*, 18(8):1138–1150, 2006.

[105] Grischa Liebel, Andreea Olaru, Henrik Lönn, Henrik Kaijser, Sunith Rajendran, Urban Ingelsson, and Richard Berntsson Svensson. Addressing model complexity in automotive system development: selection of system model elements for allocation of requirements. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 168–175. IEEE, 2016.

[106] Mitch Lubars, Colin Potts, and Charles Richter. A review of the state of the practice in requirements modeling. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 2–14. IEEE, 1993.

[107] Mark W Maier. *The art of systems architecting*. CRC press, 2009.

[108] Song Mao, Azriel Rosenfeld, and Tapas Kanungo. Document structure analysis algorithms: a literature survey. *Document recognition and retrieval X*, 5010:197–207, 2003.

[109] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. Easy approach to requirements syntax (ears). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322. IEEE, 2009.

[110] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. Easy approach to requirements syntax (ears). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322, 2009.

[111] James R McCoy. Nasa software tools for high-quality requirements engineering. In *Proceedings 26th Annual NASA Goddard Software Engineering Workshop*, pages 69–69. IEEE Computer Society, 2001.

[112] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[113] AR Miller and DR Herber. Digital engineering transformation of requirements analysis within model-based systems engineering. In *World Conference of the Society for Industrial and Systems Engineering*, 2021.

[114] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[115] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244, 1990.

[116] Aashay Mokadam, Shrikrishna Shivakumar, Vimal Viswanathan, and Mahima Agumbe Suresh. Online product review analysis to automate the extraction of customer requirements. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 85420, page V006T06A049. American Society of Mechanical Engineers, 2021.

[117] Faisal Mokammel, Eric Coatanea, Francois Christophe, Mohamed Ba Khouya, and Galina Medyna. Towards an approach for evaluating the quality of requirements. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 55867, page V02BT02A024. American Society of Mechanical Engineers, 2013.

[118] Beshoy Morkos, James Mathieson, and Joshua D Summers. Comparative analysis of requirements change prediction models: manual, linguistic, and neural network. *Research in Engineering Design*, 25(2):139–156, 2014.

[119] Beshoy Morkos, Prabhu Shankar, and Joshua D Summers. Predicting requirement change propagation, using higher order design structure matrices: an industry case study. *Journal of Engineering Design*, 23(12):905–926, 2012.

[120] Jesse Mullis, Cheng Chen, Beshoy Morkos, and Scott Ferguson. Deep neural networks in natural language processing for classifying requirements by origin and functionality: An application of bert in system requirements. *Journal of Mechanical Design*, 146(4):041401, 2024.

[121] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[122] Mark Newman. *Networks*. Oxford university press, 2018.

[123] Charles Egerton Osgood, George J Suci, and Percy H Tannenbaum. *The measurement of meaning*. Number 47. University of Illinois press, 1957.

[124] Anja-Karina Pahl, Linda Newnes, and Chris McMahon. A generic model for creativity and innovation: overview for early phases of engineering design. *Journal of Design Research*, 6(1-2):5–44, 2007.

[125] Gerhard Pahl and Wolfgang Beitz. *Engineering design: a systematic approach*. Springer Science & Business Media, 2013.

[126] Hyeoun-Ae Park, InSook Cho, and NamSoo Byeun. Modeling a terminology-based electronic nursing record system: an object-oriented approach. *International journal of medical informatics*, 76(10):735–746, 2007.

[127] Eugenio Parra, Christos Dimou, Juan Llorens, Valentín Moreno, and Anabel Fraga. A methodology for the classification of quality of requirements using machine learning techniques. *Information and Software Technology*, 67:180–195, 2015.

[128] Stefania Passera. Beyond the wall of text: How information design can make contracts user-friendly. In *Design, User Experience, and Usability: Users and Interactions: 4th International Conference, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II 4*, pages 341–352. Springer, 2015.

[129] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[130] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology*, 64:1–18, 2015.

[131] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[132] Tonio Pinna, Danilo Nicola Dongiovanni, and Francesco Iannone. Functional analysis for complex systems of nuclear fusion plant. *Fusion Engineering and Design*, 109:795–800, 2016.

[133] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[134] Rahul Rai and Chandan K. Sahu. Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. *IEEE Access*, 8:71050–71073, 2020.

[135] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[136] Douglas LT Rohde, Laura M Gonnerman, and David C Plaut. An improved model of semantic similarity based on lexical co-occurrence. *Communications of the ACM*, 8(627-633):116, 2006.

[137] Michael Ryan, Ryan Wheatcraft, and Requirements Working Group, INCOSE. *INCOSE Guide to Writing Requirements, Jul-2023*. INCOSE, 2023.

[138] Michael J Ryan and Louis S Wheatcraft. On a cohesive set of requirements engineering terms. *Systems Engineering*, 20(2):118–130, 2017.

[139] Michael J Ryan and Louis S Wheatcraft. On the use of the terms verification and validation. In *INCOSE international symposium*, volume 27, pages 1277–1290. Wiley Online Library, 2017.

[140] Andrew P Sage and William B Rouse. *Handbook of systems engineering and management*. John Wiley & Sons, 2011.

[141] Chandan K. Sahu, Vinayak S. Khade, Mohan S. R. Elapolu, Rahul Rai, Matthew P. Castanier, and David Gorsich. Reqcity++: A City-based visualization for the verification of system Requirements to conform to the iso/iec/ieee 29148:2018 (e) standard. *IEEE Transactions on Software Engineering*, pages 1–21, 2025.

[142] Chandan Kumar Sahu. Boon of hybrid approaches over the bane of model based and machine learning approaches in modelling cyber-physical systems. Master's thesis, State University of New York at Buffalo, 2020.

[143] Chandan Kumar Sahu. Chandanksahu/reqlist_reqnet_reqsim: Zenodo release, April 2024.

[144] Chandan Kumar Sahu, Rahul Rai, Matthew Castanier, and David Gorsich. A semantic network of requirements from the contextual embeddings of req-sbert. *Under Review at the ASME Journal of Mechanical Design*, pages 1–15, 2025.

[145] Chandan Kumar Sahu, Rahul Rai, and Margaret Wiecek. Requirement allocation from the tree structure of requirement specification documents. *Under Review at the ASME Journal of Mechanical Design*, pages 1–14, 2025.

[146] Chandan Kumar Sahu, Rahul Rai, Margaret Wiecek, and David Gorsich. ReqNet and ReqSim: A Network and Semantic Similarity Dataset of Requirements from the Tree Structure of System Requirement Specifications. *Journal of Computing and Information Science in Engineering*, pages 1–15, 06 2024.

[147] Alberto Sardinha, Ruzanna Chitchyan, Nathan Weston, Phil Greenwood, and Awais Rashid. Ea-analyzer: automating conflict detection in a large set of textual aspect-oriented requirements. *Automated Software Engineering*, 20:111–135, 2013.

[148] Serhad Sarica, Jianxi Luo, and Kristin L Wood. Technet: Technology semantic network based on patent data. *Expert Systems with Applications*, 142:112995, 2020.

[149] Robert P Scheurer and Michael E Volz. Capturing and taming derived requirements. In *INCOSE International Symposium*, volume 4, pages 77–83. Wiley Online Library, 1994.

[150] Chiradeep Sen, Alolika Mukhopadhyay, John Fields, and Farhad Ameri. An approach for measuring information content of textual engineering requirements using a form-neutral representation. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 46292, page V01BT02A006. American Society of Mechanical Engineers, 2014.

[151] Tobias Senger. A unified open-and closed-source software requirements dataset. B.S. thesis, 2022.

[152] Khurram Shahzad, Ifrah Pervaz, and Adeel Nawab. Wordnet based semantic similarity measures for process model matching. In *BIR Workshops*, pages 33–44, 2018.

[153] Md Imrul Reza Shishir, Mohan Surya Raja Elapolu, and Alireza Tabarraei. A deep convolutional neural network-based method to predict accurate fracture strength of poly-crystalline graphene. In *ASME International Mechanical Engineering Congress and Exposition*, volume 85680, page V012T12A012. American Society of Mechanical Engineers, 2021.

[154] Md Imrul Reza Shishir, Mohan Surya Raja Elapolu, and Alireza Tabarraei. A deep learning model for predicting mechanical properties of polycrystalline graphene. *Computational Materials Science*, 218:111924, 2023.

[155] John Slankas and Laurie Williams. Automated extraction of non-functional requirements in available documentation. In *2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE)*, pages 9–16. IEEE, 2013.

[156] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems*, 33:16857–16867, 2020.

[157] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[158] Nick Spivey, Joshua Ortiz, Akash Patel, Brian Davenport, and Joshua D Summers. Analysis of the impact of requirement-sketch sequencing on requirement generation in conceptual design. *Journal of Mechanical Design*, 143(12):121402, 2021.

[159] Jonette M Stecklein, Jim Dabney, Brandon Dick, Bill Haskins, Randy Lovell, and Gregory Moroney. Error cost escalation through the project life cycle. In *14th Annual International Symposium*, number JSC-CN-8435, 2004.

[160] Sean Szumlanski and Fernando Gomez. Automatically acquiring a semantic network of related concepts. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 19–28, 2010.

[161] Executive Office Of The President Office Of Management And Budget. North american industry classification system (naics), united states. `https://www.census.gov/naics/reference_files_tools/2022_NAICS_Manual.pdf`, 2022.

[162] IEEE Std 1233-1998. Ieee guide for developing system requirements specifications. *IEEE Std 1233, 1998 Edition*, pages 1–36, 1998.

[163] IEEE Std 830-1998. Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, 1998.

[164] IFB No. IOMSKP 086/16. Supply and delivery of local deployable coordination communication centre, mobile surveillance system equipped with eo-ir cameras. `https://www.iom.int/sites/g/files/tmzbdl486/files/procurement/Technical%20Specifications_Mobile%20Surveillance%20System_Item%203.pdf`, 2016.

[165] International Council on Systems Engineering, INCOSE-TP-2003-002-05. *Systems Engineering Handbook, A Guide for System Life Cycle Processes and Activities.* John Wiley & Sons Ltd., 2023.

[166] ISO 29148:2018(E). Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2018(E)*, pages 1–104, 2018.

[167] New York City Department of Transportation. Connected vehicle pilot deployment program phase 1 systems requirements specification (syrs) – new york city, 2017.

[168] Office of Inspector General. 2023 report on nasa's top management and performance challenges, 2023.

[169] PMI. *Requirements Management: A Core Competency For Project And Program Success.* Project Management Institute (PMI), 2014.

[170] QRA Corp. Automating the incose guide for writing requirements.

[171] Sparx Systems and Stephen Maguire. Enterprise architect requirements engineering - version 1.0. `https://sparxsystems.com/downloads/whitepapers/requirements-engineering.pdf`, 2016.

[172] The Reuse Company. RQA - Quality Studio.

[173] David Ullman. *EBOOK: The mechanical design process.* McGraw hill, 2009.

[174] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[175] Axel Van Lamsweerde et al. Engineering requirements for system reliability and security. *NATO Security Through Science Series D-Information and Communication Security*, 9:196, 2007.

[176] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[177] Alvaro Veizaga, Seung Yeob Shin, and Lionel C Briand. Automated smell detection and recommendation in natural language requirements. *IEEE Transactions on Software Engineering*, 2024.

[178] Zdenek Vintr and R Holub. R&m requirements allocation in upgrading a system. In *Annual Reliability and Maintainability Symposium. 2001 Proceedings. International Symposium on Product Quality and Integrity (Cat. No. 01CH37179)*, pages 258–263. IEEE, 2001.

[179] Ademir-Paolo Vrolijk and Zoe Szajnfarber. Requirements, objectives, both, or neither: How to formulate complex design problems for innovation contests. *Journal of Mechanical Design*, 146(3):031402, 2024.

[180] Hannah S Walsh and Sequoia R Andrade. Semantic search with sentence-bert for design information retrieval. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 86212, page V002T02A066. American Society of Mechanical Engineers, 2022.

[181] Shen Wan and Rafal A Angryk. Measuring semantic similarity using wordnet-based context vectors. In *2007 IEEE international conference on systems, man and cybernetics*, pages 908–913. IEEE, 2007.

[182] Jing Wang and Mian Li. Redundancy allocation for reliability design of engineering systems with failure interactions. volume Volume 2B: 40th Design Automation Conference of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, page V02BT03A042, 08 2014.

[183] Nanxin Wang. Ermm: An engineering requirements management method. *Journal of Computing and Information Science in Engineering*, 6(2):196–199, 04 2006.

[184] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

[185] Michael W. Whalen, Anitha Murugesan, and Mats P. E. Heimdahl. Your what is my how: Why requirements and architectural design should be iterative. In *2012 First IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, pages 36–40, 2012.

[186] Lou Wheatcraft, Tami Katz, Michael Ryan, Raymond Wolfgang, and Requirements Working Group, INCOSE. Needs and requirements manual: Needs, requirements, verification, validation across the lifecycle, may-2022. Technical report, International Council on Systems Engineering (INCOSE), 2022.

[187] Louis S Wheatcraft. 5.1. 2 triple your chances of project success risk and requirements. In *INCOSE International Symposium*, volume 21, pages 543–558. Wiley Online Library, 2011.

[188] Jon Whittle and Praveen K. Jayaraman. Generating hierarchical state machines from use case charts. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 19–28, 2006.

[189] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[190] Glen Williams, Nicholas A Meisel, Timothy W Simpson, and Christopher McComb. Design for artificial intelligence: Proposing a conceptual framework grounded in data wrangling. *Journal of Computing and Information Science in Engineering*, 22(6):060903, 2022.

[191] Grant Williams and Anas Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.

[192] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[193] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. *arXiv preprint cmp-lg/9406033*, 1994.

[194] Chenxi Yuan, Tucker Marion, and Mohsen Moghaddam. Leveraging end-user data for enhanced design concept evaluation: A multimodal deep regression model. *Journal of Mechanical Design*, 144(2):021403, 2022.

[195] Ke-Shi Zhang, Wei-Ji Li, and Hong-Yan Wei. A new method for optimum allocation of design requirements in aircraft conceptual design. *Chinese Journal of Aeronautics*, 19(3):203–211, 2006.

[196] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. Natural language processing for requirements engineering: a systematic mapping study. *ACM Computing Surveys (CSUR)*, 54(3):1–41, 2021.

[197] Shubhendu Kumar Singh Rahul Rai Zhuo Yang Zhibo Zhang, Chandan Kumar Sahu and Yan Lu. Machine learning based prediction of melt pool morphology in a laser-based powder bed fusion additive manufacturing process. *International Journal of Production Research*, 62(5):1803–1817, 2024.