Systems Engineering Automation Through Artificial Intelligence (AI) and Natural Language Processing (NLP)-Based Software

Michael Cannizzaro
US Army Futures Command STE CFT
Orlando, FL
michael.r.cannizzaro.civ@army.mil

Xuan Chau, Brian P Parrish
MITRE Corporation
Bedford, MA
xchau@mitre.org, bparrish@mitre.org

ABSTRACT

In the field of both traditional and Agile systems engineering, engineers manually perform a variety of tasks to derive, decompose, trace, rewrite, and evolve large numbers of engineering artifacts – work that is time-consuming and prone to human error. Mistakes made during the project lifecycle have compounding negative impacts on product efficacy and remediation costs grow exponentially problems go unchecked. Thus, improving requirements work through the adoption of modern technologies and methodologies is of significant importance to increase turnaround time and quality of products.

In 2012, Army Training and Doctrine Command (TRADOC) Combined Arms Center for Training (CAC-T) and the MITRE Corporation developed software to accelerate common and repeatable engineering tasks to improve the quality of capability development and acquisition working group efforts. Recent advances in artificial intelligence (AI) natural language processing (NLP) have modernized this software into a web application known as the *Requirements Analysis using Artificial Intelligence for Mapping (RAAM)*, which provides capabilities for the automation of language-related tasks such as generating requirements, improving requirements quality, and performing traceability analysis. In the past two years, the tool has been used by various government organizations to accelerate initial requirements generation by 9,000%, reduce traceability analysis times by 84%, and to improve requirements quality based on International Counsel on Systems Engineering (INCOSE) and Institute of Electrical and Electronics Engineers (IEEE) writing standards.

This paper will discuss several distinct stages of the digital engineering workflow that involve high impact but tedious manual work. It will propose how certain NLP technologies and techniques can be used to accelerate those tasks and improve quality of the outcome through software automation. To further illustrate these capabilities, the paper will focus on the RAAM tool as a case study for real-world application of NLP to digital engineering.

ABOUT THE AUTHORS

Michael Cannizzaro is the senior Science & Technology (S&T) advisor at the Synthetic Training Environment Cross Functional Team (STE CFT) in Orlando, Florida. Mike is responsible for coordinating S&T technologies from across the Department of Defense into the STE strategic roadmap. Mike has nearly 20 years of Army S&T experience as subject matter expert, branch chief and senior engineer in the DEVCOM C5ISR Center in electronic warfare and air/ground survivability equipment, transitioning to the STE CFT in 2019. Mike has a Bachelor of Computer Engineering degree from the University of Delaware and a Master of Computer Engineering degree from Stevens Institute of Technology.

Xuan Chau is a software developer for the MITRE Lab's Mission Center. She has 4 years of industry experience, with an emphasis in full-stack web development, AI, and Natural Language Processing (NLP). Her current focus is on leveraging modern AI/NLP technologies to automate and accelerate language tasks in systems engineering. She holds a Bachelor of Science in Computer Science from the University of Connecticut.

Brian P Parrish is a Principal Modeling and Simulation and Multi-Discipline Systems Engineer who joined the MITRE Corporation in 2005. His work addresses capability development, acquisition, analysis, and complex system of systems engineering for the Army, Marines, Department of Homeland Security (DHS), and Foreign Military Sales (FMS) with Canada and the Republic of Korea. Brian is a Project Leader for the STE CFT and the DEVCOM Soldier

Center. Credentials include Master of Science in Systems Engineering with an emphasis in Information Assurance from Johns Hopkins University, a Bachelor of Science in Computer Science with a minor in mathematics and a Bachelor of Science in Computer Information Systems with a minor in business from Missouri Western State University.

Systems Engineering Automation Through Artificial Intelligence (AI) and Natural Language Processing (NLP)-Based Software

Michael Cannizzaro
US Army Futures Command STE CFT
Orlando, FL
michael.r.cannizzaro.civ@army.mil

Xuan Chau, Brian P Parrish
MITRE Corporation
Bedford, MA
xchau@mitre.org, bparrish@mitre.org

INTRODUCTION

Over the last decade, investments of billions of dollars towards artificial intelligence (AI) and machine learning (ML) research have resulted in rapid advancement in the subfield known as natural language processing (NLP), which encompasses a machine's ability to interpret, manipulate, and comprehend human language (BCC Research, 2023). Statistical and powerful syntactic language models are setting new performance benchmarks and enabling more opportunities than ever to apply NLP to increasingly difficult lexical tasks (Wu, Wu, Wu, Feng, & Tan, 2024).

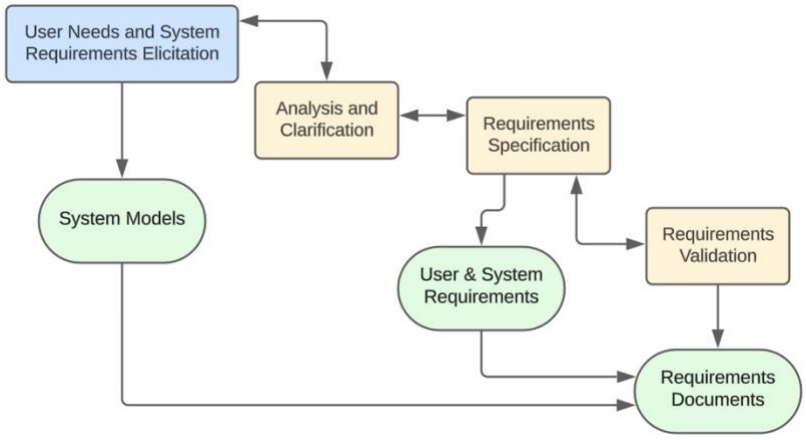

Figure 1. RE process diagram showcases four recursive stages of requirements development and their outputs.

One such opportunity lies in the field of system design work such as requirements engineering (RE), which involves identifying, documenting, and maintaining user needs and systems requirements. Requirements engineers may curate hundreds to thousands of requirements statements throughout the RE lifecycle, which is crucial for the successful integration of a system into its operational environment. This is a time-consuming process due to the large amounts of textual data which is carefully analyzed and processed during routine SE tasks such as source analysis, user needs elicitation, historic system analysis, requirements generation, requirements quality analysis, traceability analysis, architecture development, and test metrics / measures. Oversights in RE are also costly, compounding over time especially within iterative project lifecycles such as Agile (Ambler, 2023).

Although researchers have been investigating the application of NLP to automate system engineering tasks for decades, digital engineering (DE) currently benefits little from this technology (Zhao, et al., 2021). Existing DE tools primarily aid in organization and change management, but experts are still required to enable an effective data driven process. An observation during our experience developing requirements for military training systems is that third party NLP tools are mostly suited for software developers, underutilized by engineers, and not currently usable by busy capability development or acquisition experts. Meanwhile, well-established SE and RE software (e.g., International Business Machines Corporation [IBM]™ Dynamic Objective Oriented Requirements System [DOORS], NoMagic MagicDraw™) are often expensive and may only offer basic AI capabilities for data analysis. Engineers and requirements stakeholders may also be reluctant to rely on software automation due to the complex and highly subjective nature of language-based tasks, and the presence of domain specific terminology common in engineering artifacts that confuse generic AI models. It should be noted that although NLP can undoubtedly accelerate certain tasks and reduce human error in specific applications, it cannot produce perfect results or emulate review processes. NLP software tools should therefore aim to expedite both the analysis and review process, providing engineers both automation and insight through an interactive and intuitive interface.

To demonstrate how software can be integrated into current SE workflows, this paper proposes NLP solutions to the requirements-related tasks outlined in Table 1. The implementation and application of these technologies will be explored by discussing the software known as the *Requirements Analyzer using AI for Mapping (RAAM)* – a recently developed web-based DE tool that utilizes state-of-the-art NLP to process requirements language. The RAAM tool automates a variety of RE tasks and was designed to allow engineers to verify the outputs of NLP algorithms through user friendly graphical user interfaces (GUIs), supporting interactive human-in-the-loop development. In the past several years, the RAAM tool has been used experimentally by government organizations such as the Synthetic Training Environment Cross Functional Team (STE CFT), Program Executive Office for Simulation Training and Instrumentation (PEO STRI) and Combined Arms Center for Education (CAC-E) to expedite requirements generation, tracing, and quality analysis, the successful outcome and metrics of which will be discussed in this paper. This paper performs a case study of the RAAM and its recent applications to demonstrate the benefits and limitations of NLP techniques on SE work.

Table 1

Task	Description	Solution
Comprehend domain-specific language	Address the presence of domain-specific terminology in text which can confuse generic large language models (LLMs).	Use automated extraction methods to aid users in curating a *domain model* dictionary mapping terms to in-vocabulary replacements.
Derive requirements from elicited documents	Create, organize, and refine requirement statements based on available system documentation, Concept of Operations (CONOPS), and other material procured during elicitation.	Use basic algorithms, generative AI (genAI), and trained tokenization pipelines to isolate, rewrite, and analyze requirements statements from source.
Architecture modeling	Create graphical diagrams from requirements showcasing the needs of the system.	Perform parts of speech analysis to identify components of an architecture diagram. Use text-to-visual tools such as MagicDraw™ or Draw.IO to create diagrams.
Quality checking	Ensure that requirements adhere to good writing standards. This typically means ensuring they are unambiguous, complete, singular, and verifiable.	Use a variety of text algorithms to check for rule violations based on the International Counsel of Systems Engineering (INCOSE) & Institute of Electrical and Electronic Engineers (IEEE) requirements best writing practices.
Artifact comparison	Compare text to other text (e.g. requirements to architecture) to maintain and establish traceability to sources, derivatives, or other versions during the management phase.	Use word embeddings of LLMs to perform semantic-aware document similarity analysis to establish relationships between software artifacts such as requirements, test cases, and more.

DOMAIN-SPECIFIC LANGUAGE

A common difficulty in automating RE language tasks is the abundance of domain-specific terminology, acronyms, and word meanings. Well-performing LLMs such as Meta's Large Language Model Meta AI (LLaMA) and Google's Bidirectional Encoder Representations from Transformers (BERT) model are trained using the text corpora of everyday language, such as that of Wikipedia or news articles. As a result, most NLP pipelines are not designed to process domain-specific engineering artifacts (e.g., Army training requirements), resulting in inaccuracies (Ling, et al., 2023). Generative AIs (genAIs) based on such LLMs also struggle to properly understand and incorporate domain-specific words and syntax when prompted to generate or analyze engineering artifacts. Consider the ambiguity of the bolded terms within the following illustrative user story:

> "*As a **division**, we require a **STE** to consolidate gains continuously to ensure lasting outcomes and a more favorable security environment within the **AOR** which enables maintaining and protecting decision advantages."* (Army Future Command, 2021)

The meanings of certain words and phrases (e.g. "division") must be interpreted within the military domain of the original Army Future Command document. However, a standard language model lacks this context. For the same

reason, LLMs may incorrectly interpret ambiguous acronyms such as "AOR" (Area of Responsibility) or "STE" (Synthetic Training Environment), which have specific definitions within the original document, or would fail to derive meaning from the acronym entirely.

The most proposed solution to underfitted LLMs is to fine-tune models for each desired domain. Although effective, fine-tuning requires significant time and computing resources, the development of large, highly specialized datasets, and a training algorithm to perform correctly (Jeong, 2024). As systems engineers often work with highly specialized program- or even project-specific domains, fine-tuning is not always feasible. The following section discusses alternative solutions to fine-tuning, including those used by the RAAM software.

Proposed Solutions

There are effective alternative methods to integrating knowledge of a domain into an LLM pipeline. One approach is to curate dictionaries of domain-specific terminology to increase comprehension by substituting otherwise ambiguous text. The RAAM, for example, facilitates the creation of user-curated dictionaries coined *domain models* that map out-of-vocabulary (OOV) terminology to in-vocabulary (IV) synonyms. These definitions are applied through a simple substitution method during text pre-processing to improve the accuracy of engineering artifact comparison, similarity analysis, or other tasks that rely on LLMs to evaluate a document's semantic meaning. Generally, the expansion of acronyms tends towards a more descriptive artifact. Consider the following example text and its rewrite after substituting domain model definitions, the latter of which is significantly more comprehensible to a generic LLM:

- **Original Text**: "*TMT* enables greater visibility of training metrics that support *Objective-T* and the *Sustainable Readiness Model (SRM)*." (U.S. Army, 2019)
- **Substituted Text**: "The *Training Management Tool* enables greater visibility of training metrics that support *the combat readiness test* and the *Regionally Aligned Readiness and Modernization Model (ReARMM).*"

The construction and maintenance of domain models can be tedious for users. The RAAM software alleviates this issue by automatically identifying unknown terminology in text and by providing an interface through which users can define and organize the resulting terminology. By repeatedly checking for missing terms and carefully curating findings, systems engineers can more feasibly maintain domain models to use as knowledge bases.

Software using genAI pipelines may also improve performance by integrating domain models and similar context documents into a retrieval augmented generation (RAG) database. RAG has been shown to improve genAI's ability to perform language tasks such as question answering, text generation, and fact verification when dealing with specialized domains (Barron, et al., 2024). By extension, RAG can be applied to improve AI-based RE automation. For example, when tasked to generate initial requirements, the user may specify a domain model along with a concept of operations (CONOPS) document describing the characteristics of the proposed system. These references bolster the AI's vocabulary and guide generation towards requirements that better describe the system. More specific applications of RAG will be discussed in subsequent sections.

REQUIREMENTS GENERATION

UUID	Hierarchy ID	Paragraph #	Heading	Description	Subject	Verb(s)	Object(s)
1	1	1	Appendix D AFCC-C2 Dependencies on Other Concepts				
5	1.1.2	4	D-2. C2 dependencies on other warfighting functional and supporting concepts				
63	1.1.2.58	39		The AFCC-C2 depends on the maneuver functional concepts for the capability to consolidate gains continuously to ensure lasting outcomes and a more favorable security environment within the AOR which enables maintaining and protecting decision advantages.	AFCC-C2	depends, enables	gains, advantages, AOR, outcomes

Figure 2. Initial Requirements Analysis of Source Document (Army Future Command, 2021)

The first stage of the requirement engineering (RE) lifecycle is the authoring of an initial set of user needs, system requirements, and a Requirements Specification (RS) document which summarizes the design, implementation, and goals of the system to be built. During the elicitation stage of the SE workflow, requirements engineers and stakeholders gather requirements documents, which may include CONOPS files, operational mode summaries / mission profiles (OMS/MPs), capability documents, doctrines, training manuals, and others that describe the operational needs of the system. From these documents, engineers manually construct sets of formalized user, functional, and/or non-functional requirements statements. These are then often organized in a tabular format with appropriate metadata (see Figure 2). This is a time-consuming process, as engineers must thoroughly analyze large

amounts of text, derive large quantities of requirements, and ensure their validity, uniqueness, and overall quality (Pohl, 1996).

To expedite RS generation using AI software, measures can be taken to 1) automate the conversion of multiple text documents of variable format to organized spreadsheets, 2) automate the conversion of system descriptions to formal shall statements or user stories, and/or 3) aid engineers in performing other analysis like categorization and identifying redundancies or ambiguities. GenAI provides new opportunities to automatically derive requirements from source material by wholistically analyzing documents to not only generate user needs statements and requirements, but also to organize them by file structure or category.

Implementation

This section will provide an overview of the pipelines and technologies supporting the RAAM's Requirements Derivation Tool (RDT) capability which uses various LLM pipelines, including genAI, to automate RS generation. As input, the tool ingests Word documents describing the operational needs of a system (e.g. CONOPS, OMS/MPs, etc.) and others as previously described. The file's contents are then parsed by chunks of text (paragraphs between 1 and 10 sentences in length) and a Llama 3-based genAI is prompted to convert each text chunk into one or more shall requirements. Once parsed, these shall requirements are returned in a spreadsheet formatted output. This output acts as a more advanced baseline from which experts can produce a requirements specification, reducing time otherwise spent manually converting and aggregating an initial set of requirements. Figure 3 depicts a simplified operational diagram of the pipeline and its stages, while Table 2 describes each stage in more detail with examples. Note that although the RDT does not currently implement RAG to provide context documents to the genAI to guide the generation of requirements, providing context is crucial to improving the quality of output. Therefore, a RAG system is included in the proposed diagram and table.

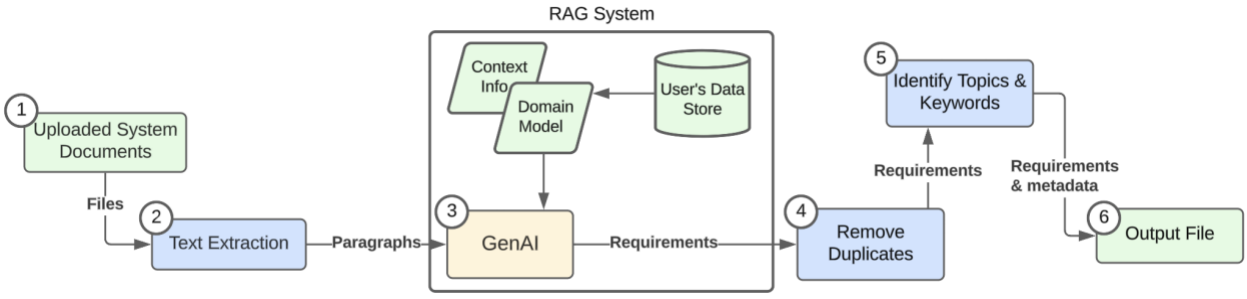

Figure 2. Flow chart shows how source requirements documents are processed by the RAAM's RDT to produce an organized, low-level requirements specification table

Table 2

Stage Description	Example
1) Upload documents: User provides system documents and a file hierarchy identification marker	File: Word document (.docx) Hierarchy identification method: (e.g., Hierarchy prefix, heading style level, no hierarchy)
2) Text extraction: Chunks of text are extracted from each file – including tables and lists. Metadata for each chunk such as the file (source) name and hierarchy value are collected.	Chunk: "The vendor should organize all work-related documents and files (CDRL A007), store them on the designated collaboration tool and provide a file plan outlining the file structure. See Addendum B for details. Status for each project needs to be documented, to include recent, current, and pending actions. (Add more notes later)" Metadata: "Filename.docx" (filename), "1.2.3" (hierarchy ID)
3) Generate requirements: Paragraphs (chunks of text separated by new lines, or entire lists) between 1 and 10 sentences long are formatted into prompts for shall requirements. RAG implementation retrieves relevant context documents (original document,	Input Prompt: "Extract 6 shall statements from the following text: <chunk>" Generated Output: "Here are 6 "shall" statements extracted from the given text: 1. The vendor shall organize all work-related documents and files (CDRL A007).

domain models, and other supporting information) to aid in requirements generation. Shall requirements are then extracted from the genAI output.	2. The vendor shall store them on the designated collaboration tool. 3. The vendor shall provide a file plan outlining the file structure. 4. Status for each project shall be documented. 5. The status shall include recent, current, and pending actions. 6. The vendor shall document the status for each project."
4) Remove duplicates: After all requirements have been generated, cosine similarity of vectorized sentences is used to find highly similar requirements. Duplicates are removed.	Sentence 1: "Status for each project needs to be documented, to include recent, current, and pending actions." Sentence 2: "Project leaders are responsible for documenting the status of their projects, including current and future tasks." Cosine similarity percentage: 67% These requirements are above the similarity threshold, and one is discarded
5) Get topics / keywords: Requirements topics are determined using LDA and used to group requirements. Keywords are identified using part-of-speech tagging and sentence decomposition pipelines.	Sentence: "The vendor should organize all work-related documents and files (CDRL A007), store them on the designated collaboration tool and provide a file plan outlining the file structure." Keywords: "Vendor", "documents", "file", "work-related", "collaboration", "tool", "organize", "store", "provide" Topics: "vendor", "file", "work"
6) Export final table: Final requirements are assigned an ID and hierarchy value. The final spreadsheet is generated and returned.	See **Error! Reference source not found.**.

While this implementation significantly accelerates the initial conversion of source documents to a tabular format and performs first-pass analysis, subsequent refinement of the output will still constitute a significant amount of time. Tasks will include comparing extracted requirements to the source material to determine what should be rewritten or discarded, reorganizing the document structure, disambiguation and consolidation of requirements, and topic modeling (Pohl, 1996). Therefore, future efforts may focus on providing NLP-assisted GUIs that aid engineers in the post-processing stage. For example, such a GUI could provide side-by-side comparisons of source paragraphs to derived requirements, in-program text editing and reordering capabilities, and offer suggestions on how to categorize requirements through topic identification methods such as Latent Dirichlet Allocation (LDA). In doing so, we accelerate the validation process as well as increase the usability of NLP-assisted digital engineering.

ARCHITECTURE GENERATION

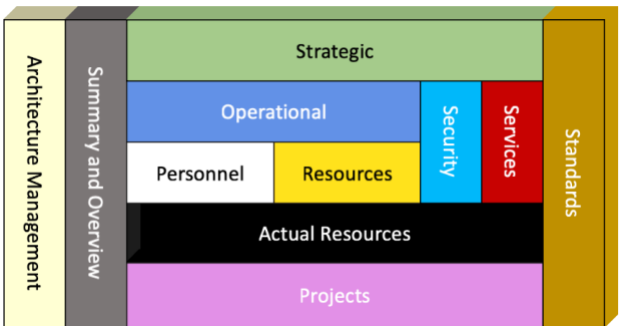

Figure 3. Unified Architecture Framework Viewpoint Interrelationships (OMG, 2022)

In most SE methodologies (e.g., agile, waterfall), systems engineers perform requirements and architecture analysis as an iterative, but sequential activity. Once initial user needs and system requirements are written, engineers author architecture diagrams that detail the operational, services, information exchanges, and other viewpoints of the system (see **Error! Reference source not found.**). When working on these diagrams, engineers perform a significant amount of duplicative work as those authoring requirements (e.g., analyzing source materials, deriving syntactically structured phrases, and categorizing artifacts). Critically, when performed in a DE environment, engineers must ensure requirements and architecture are bidirectionally traceable to each other (e.g., architecture elements reference the requirements they satisfy). Unfortunately, architecture efforts are often performed by multiple teams working in parallel with requirements writing efforts to accelerate the schedule, resulting in continuous desynchronization (e.g., phrasing inconsistencies) between requirements and architecture artifacts that necessitates the additional overhead of a change management process. Current DE tools track traceability using a manually curated matrix, requiring engineers to

memorize both bodies of work to efficiently identify traceability relationships or to search for key words and phrases that likely indicate a relationship.

We propose that the task of architecture generation can be greatly accelerated through NLP linguistic analysis by automatically identifying information exchange requirements between a requirement's subject, verbs, and direct objects. These building blocks can be used to automatically generate *word map* diagrams with entities and action flows representing a system architecture diagram. Standardizing the use of software to simultaneously generate requirements from source document and architecture from said requirements also enforces the use of consistent phrases, maintaining traceability to the source materials, and maintaining traceability between requirements and architecture from inception. By keeping both architecture and requirements generation to a DE environment that automatically tracks relationships, such software helps sustain traceability between artifacts.

Implementation

Producing an architecture diagram requires the identification of user/operational and system components which become entities within the diagram. These can be acquired from requirements text. Specifically, nouns (subjects and objects) can be represented by individual nodes, and verbs by the unidirectional arrows between them, creating a *word map* depicting the interactions between system entities. Parts of speech (PoS) analysis can be used to identify the subjects, verbs, and direct objects of each requirement (Chiche & Yitagesu, 2022). Diagrams can be produced using script-to-visual tools such as Draw.IO or MagicDraw™. The output of this process includes the final *word map* along with a traceability matrix linking original requirements to elements in the architecture diagram (see Figure 5). From these outputs, an engineer can easily gather a high-level understanding of the system architecture. By automating both the generation of architecture and requirements, we also ensure traceability is documented between the original source document, user need statements, low-level requirements, and architecture components.

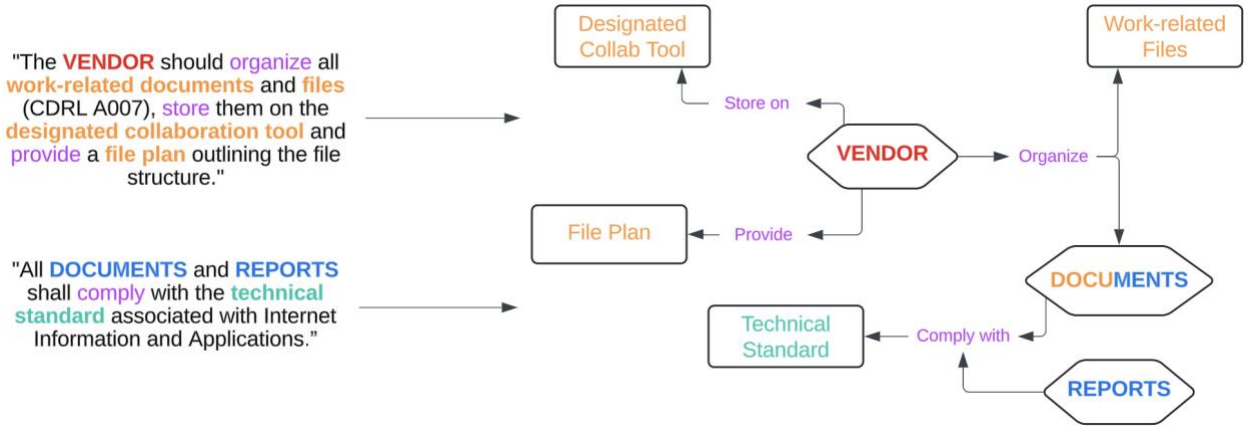

Figure 5. (Hypothetical) Two requirements sourced from a CONOPS document on the left are programmatically converted to the word map on the right, showcasing key entities and their capabilities.

Although word maps may serve as a useful starting point for architecture generation, there is significantly more processing needed before a final product is achieved. For instance, many architecture diagrams follow Department of Defense Architecture Framework (DODAF) or Unified Architecture Framework (UAF) models which require the generation of diagrams for each *viewpoint* in the system (e.g., Operational, capability, data and information, systems, services). Another common architecture format is a sequence diagram (swimlane) for showing information exchanged between entities. This traditionally requires a trained architect to also be a domain expert with the ability to identify users or user concepts (e.g., Army Division), user processes, systems, system processes, etc. To automatically identify these entities, named entity recognition (NER) – a technique used to identify words in text that fit a particular category (e.g., person, tool, organization) – can be used. While NER performs very successfully on generic sentences (Naseer, et al., 2022), the presence of domain-specific terminology in system requirements will require the fine-tuning of NER models.

REQUIREMENTS QUALITY ANALYSIS AND VALIDATION

Writing high-quality requirements ensures that a system will be designed to meet the needs of its stakeholders and prevents costly reworks later in development. According to the INCOSE and IEEE, high-quality requirements are characterized by being unambiguous, complete, feasible, and verifiable amongst numerous other characteristics (INCOSE, 2022). When these qualities are not met, problems in development arise due to misinterpretations of user needs, resulting in delays, lost productivity, and decreased customer satisfaction (Rajkumar, 2022). The detriment of poor requirements quality is well-studied especially in software, with research estimating that requirements errors cost US businesses more than $30 billion per year and often result in failed or abandoned projects (Marasco, 2007). Furthermore, research has found that the cost of fixing requirements issues grows exponentially while left unaddressed, becoming up to a thousand times costlier to fix during the operational phase than during the requirements engineering phase (Haskins et al., 2004). By these metrics, investment in early, thorough requirements quality assurance is of great importance.

However, requirements validation is a difficult task even for the experienced engineer. IEEE's Guide for Developing System Requirements Specifications proposes 9 properties of high-quality requirements (Banerjee, et al., 1999). Similarly, INCOSE's 2022 Guide to Writing Requirements proposes 41 rules across 14 characteristics for writing high-quality shall requirements, many of which require a thorough understanding of the system at large to align with (INCOSE, 2022). It is time-consuming for engineers to check for adhesion to good writing practices, especially for large projects encompassing thousands of requirements. As a result of human error and other factors such as compressed schedules, tight budgets, and limited access to subject matter experts (SMEs), quality issues still frequently make it past the review stage (G. Firesmith, 2007).

Upon analyzing the rulesets proposed by IEEE and INCOSE, it becomes evident that a majority can be evaluated for automatically through simple algorithmic and NLP techniques (see Table 3). In fact, many tools have emerged in recent years to automate quality analysis in this way, such as IBM™ DOORS' Requirements Quality Assistant (RQA). These have yet to see wide adoption, likely due to high costs of the software and poor marketing (Zhao, et al., 2021). Nevertheless, programmatically checking for poor writing practices or *rule violations* accelerates the process of verification and validation, reduces the likelihood of missed issues from human error, encourages engineers to perform quality analysis more frequently and iteratively, and educates engineers on good writing practices.

Implementation

Based on our investigations, a combination of basic text analysis, NLP techniques, and LLM-based analysis pipelines can be used to automate or significantly accelerate roughly 32 out of 41 INCOSE rules (78%). Table 3 shows each of these analysis strategies and the specific INCOSE rule(s) they check for. These strategies range greatly in complexity, with some require only basic keyword matching. As of the research conducted for this paper, the RAAM's *Requirements Quality Analyzer* (RQA) implements 19 of these viable rules to automate INCOSE rule checks.

Table 3. Rule descriptions in this table have been abbreviated (INCOSE, 2022)

Strategy	Relevant INCOSE rules
Keyword matching – Violations identified by a finite set of trigger words.	R5 – Use definite article "the" rather than the indefinite article "a" R8 – Avoid escape clauses such as "if necessary" & "where possible" R9 – Avoid open-ended clauses such as "etc." & "and so on" R16 – Avoid the use of "not" R24 – Avoid the use of pronouns and indefinite pronouns
Conditional keyword matching / Regular Expressions (RegEx) – Violations identified by text patterns	R6 – All numbers should have units of measure explicitly stated R17 – Avoid the use of the oblique ("/") symbol except in units (km/hr) R19 – Avoid combinators R21 – Avoid parentheses and brackets containing subordinate text
Sentence structural analysis with pretrained pipelines – Violations detected if sentence follows or does not follow a certain word dependency structure	R1 – Use a structured, complete sentence R2 – Use the active voice with the responsible entity as the subject R11 – Use a separate clause for each condition or qualification R18 – Write a single sentence containing a single thought R15 – Use a defined convention to express logical expressions R25 – Avoid relying on headings to understand the requirement
Combinatory techniques (keyword matching & structural analysis) –	R7 – Avoid the use of vague terms such as "some", "about", etc. R10 – Avoid superfluous infinitives such as "be able to"

Violations detected if sentence follows or does not follow a certain word dependency structure marked by a finite set of keywords	R20 – Avoid phrases that indicate the purpose of the requirement R26 – Avoid unachievable absolutes such as “100%”, “never”, etc. R27 – State applicability conditions explicitly R28 - Express the propositional nature of a condition explicitly R32 – Use “each” instead of “all”, “any'' or “both' R33 – Define quantities with a range of appropriate values R34 – Provide specific measurable performance targets R35 – Define temporal dependencies explicitly instead of using indefinite temporal keywords such as “eventually”, “after”, “once”
Grammar & spell-checking software	R12, 13, 14 – Use correct grammar, spelling, punctuation
Assisted topic classification – Can be used to automatically assign categories	R29 – Classify the need or requirement R40 – Group related requirements together
Document similarity analysis – Finds semantically similar requirements	R30 – Express each need and requirement once and only once
Other algorithms	R37 – Use a consistent set of acronyms (e.g., CMDP != C.M.D.P)
Not applicable / not significantly automatable	R3 – Ensure the subject and verb are appropriate to the referred entity R4 – Define terms in a glossary, data dictionary, etc. R22 – Enumerate sets explicitly instead of using a group noun R23 – Refer to a supporting diagram or model for complex behavior R31 – Design inputs avoid stating a solution unless there is rationale it R36 – Use each term consistently throughout the requirement sets R38 – Avoid the use of abbreviations R39 – Use a project-wide style guide R41 – Conform to a defined structure or template

Viable INCOSE rule checks can be performed extremely efficiently by software for large sets of requirements provided by an engineer. RAAM’s RQA, for instance, performs scans at a rate of 97 requirements per second on a single thread. The algorithms not only determine if a given requirement violates a rule, but can identify the problematic word or phrase. The RQA returns results in a matrix format (see Figure 5) that provides a visualization of overall set quality, and which rules are violated most frequently. The RQA provides further assistance to engineers by providing a digital GUI through which users can review potential violations and the responsible text, edit requirements, and re-analyze requirements to see improvements (see Figure 6). Furthermore, the GUI supports dismissal of false positives and the splitting of compound requirements to address singularity rules, increasing the usability of NLP quality analysis software.

R1	R2	R5	R7	R8	R9	R10	R16	R17	R18	R19	R20	R21
Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok
Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok
Ok	Wrong	Ok	Wrong	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok
Ok	Ok	Ok	Wrong	Wrong	Ok	Ok	Wrong	Ok	Ok	Ok	Ok	Ok
Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok

Figure 5. INCOSE rule violations analysis results showing requirements on the X axis and INCOSE rules on the Y axis, with issues highlighted in red, as output by the RAAM’s RQA

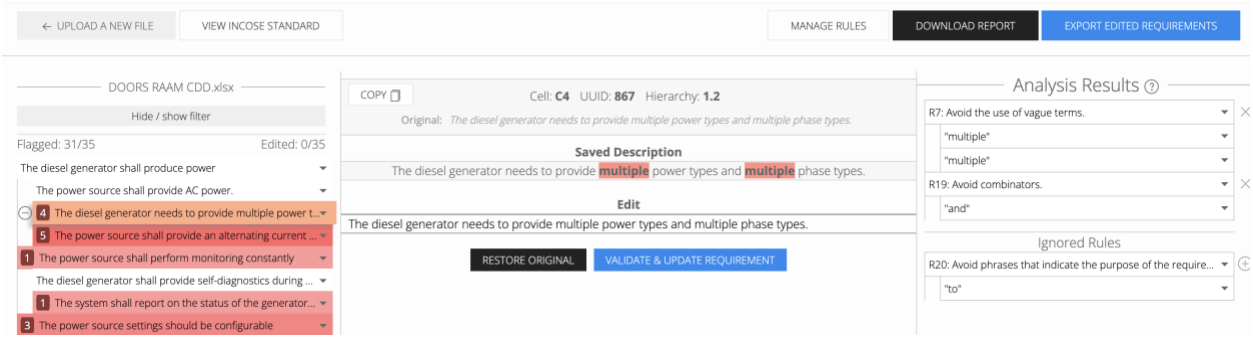

Figure 6. The RAAM RQA GUI with functions for reviewing quality issues and editing requirements

TRACING AND COMPARING ENGINEERING ARTIFACTS

The tracing and comparison of large numbers of engineering artifacts occurs for a multitude of use cases throughout the systems engineering lifecycle. For instance, there is a common need to reestablish lost traceability between source artifacts, low-level requirements, and architecture as discussed in the architecture generation section of this paper. Other instances in which artifacts must be compared in-masse include mapping mission objectives to system capabilities, requirements to Science and Technology (S&T) solutions, source selection proposals to Requests for Proposals (RFP), requirements to test and evaluation criteria, as well as performing gaps and overlaps analysis. Large-scale comparisons between thousands of artifacts are tedious and time-consuming to do manually, even for an engineer who is deeply familiar with the artifacts being analyzed.

Traditionally, an engineer conducts a comparison of two sets of artifacts by manually searching for semantically similar pairs. To accelerate this, they may use DE tools to search for shared words or phrases. Because keyword searches are exact, it can be a tedious and limiting to have to search for variety of word tenses, acronyms and expansions, and words with similar meanings or connotations (e.g., Mission Command, AFATDS, Nett Warrior, JBC-P). Many word processors don't provide the average user with intuitive features for complex, compound searches (e.g., AND, OR, CONTAINS, NOT, Wildcards). Furthermore, while search tools can help, determining whether two artifacts are related requires a deep understanding of the semantics, context, and implications of each artifact, which is more than keyword matching alone can provide. NLP software, combined with user curated domain models, can provide opportunities to conduct search engine-like analysis to mass-identify relationships between engineering artifacts. Akin to using a search engine however, a qualified engineer is still needed to refine these results and produce a final summary of relationships. All in all, automating document similarity analysis is a simple yet effective method of significantly reducing time spent on traceability work, gaps and overlaps analysis, and other related tasks.

Implementation

In simplified terms, the similarity of two sentences can be quantified by first vectorizing both texts (creating a vector representation) and then calculating the difference between those vectors using a distance function such as the cosine similarity function (Chandrasekaran & Mago, 2021). It should be noted that a high degree of similarity between artifacts does not always indicate a meaningful relationship, and that quantifying the exact degree of similarity between two sentences is challenging even with the most state-of-the-art NLP pipelines (Chandrasekaran & Mago, 2021). Other issues include the fact that engineering artifacts may depend on context for full comprehension (taking into consideration hierarchy, organization, or headings), and may contain confusing domain-specific language. However, these difficulties can be mitigated with an appropriately designed software pipeline and human-in-the-loop workflow.

The RAAM's Artifact Traceability Tool (ATT) is a capability which can rapidly compare large sets of engineering artifacts and identify potential relationships between individual texts. After identifying potential matches using the aforementioned distance-of-vectors method, an engineer reviews these matches and utilizes the provided GUI (see Figure 8) to formally identify and/or establish linkages between the two sets. Ultimately, the tool produces a *linking table* showing the relationships between artifacts. The ATT's document similarity pipeline is comprised of multiple stages outlined in the diagram in Figure 7. Note that there are two vectorization methods used in parallel: 1) term frequency over inverse document frequency (TF-IDF) and 2) word embeddings generated using a Google BERT-based LLM. While TF-IDF vectors are built using statistical measures of word frequency, word embeddings are pretrained and capture the semantic meaning of whole sentences (Chandrasekaran & Mago, 2021). As a result, TF-IDF places better emphasis on shared keywords (beneficial for short texts such as requirements statements), while word embeddings encompass the semantic meanings of sentences, allowing the cosine-similarity function to identify similarities even in the absence of shared keywords. These strategies result in two different similarity scores which are then averaged.

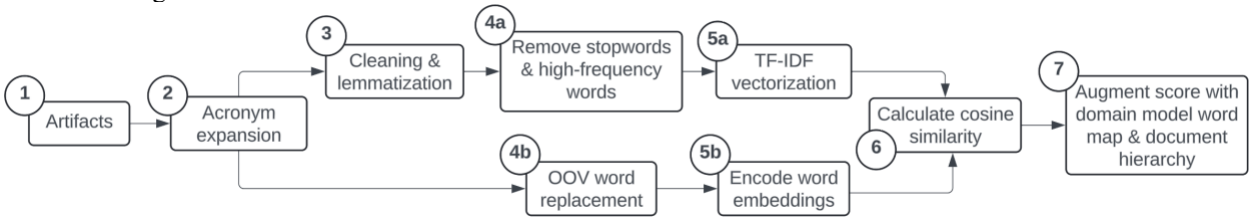

Figure 7. Document similarity analysis pipeline for the RAAM ATT

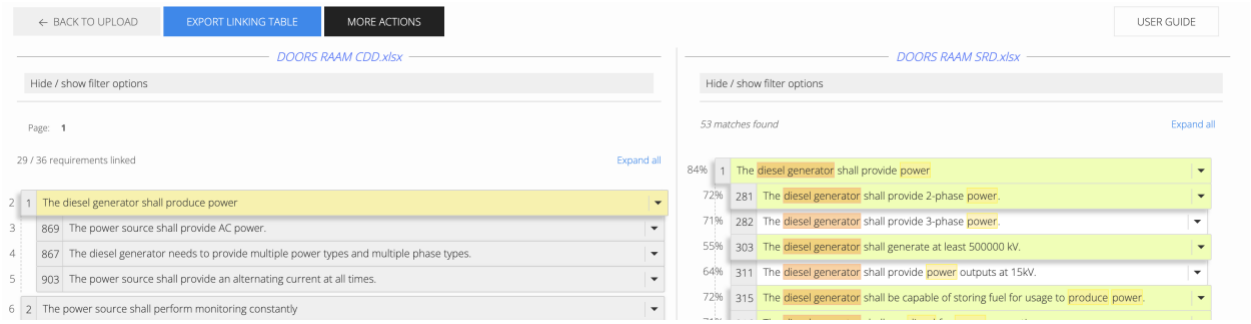

Figure 8. The RAAM ATT GUI with functions for reviewing and confirming potential linkages

As seen in Figure 7, standard preprocessing measures such as tokenization, lemmatization, and stopword removal are only necessary for TF-IDF vectorization, not vectorization into word embeddings, which works better with the original text. Also of note is the importance of RAAM's user-curated domain models in many stages in the pipeline to enhance comprehension of domain-specific text. In Step 2 for example, the expanded forms of OOV acronyms defined by the appropriate domain model are inserted to provide additional keywords (in the case of TF-IDF vectorization) or meaningful context (in the case of word embeddings). In Step 4b, OOV terminology and phrases are also replaced with IV synonyms defined by the domain model before generating word embeddings.

USE CASES AND PERFORMANCE RESULTS

In 2023, MITRE support to the STE CFT employed the RAAM's *Requirements Derivation Tool* (RDT) to conduct requirements engineering and analysis. The STE CFT extracted and linked user stories and requirements from nine Army Futures Command Multi-Domain Operations (MDO) concept documents (containing roughly 5,200 pages total). Engineers conducted the initial analysis of all nine documents in 30 minutes with the help of RAAM NLP automation, which included using the tool itself and post-processing the generated output. In contrast, it took the same engineers approximately 5 hours to extract and process the contents of each document manually, or at least 45 hours for all nine documents. Overall, it was estimated that the tool saved 44.5 hours of labor, accelerating the effort by 9,000%.

In 2012, MITRE support to the TRADOC Capability Manager for the Integrated Training Environment (TCM ITE) created the first RAAM *Requirements Quality Analyzer* (RQA) prototype to automate the evaluation of several thousand requirements using the INCOSE and IEEE best practices for writing requirements. The objective was to improve clarity and reduce downstream costs incurred by clarifying vague requirements. Proposed requirements updates were managed by a working group that was mostly focused on operational/functional requirements rather than clarity. Employing a manual process, engineers would read requirements one at a time and evaluate each of the 50+ INCOSE rules and characteristics. With the highly optimistic assumption that each rule required an average of five seconds to identify violations, each requirement would consume over 4 minutes of time for quality checking, resulting in hundreds of hours spent processing the set of several thousand requirements. Applying an NLP approach greatly improved this process by enabling engineers to run an automatic pipeline and receive a detailed evaluation of all requirements in only around a minute of processing time. In addition to identifying violations significantly faster, this enabled working groups to initially focus on only the requirements that failed evaluation, further increasing productivity, but to a non-quantifiable degree.

The RAAM's ATT has been leveraged consistently between 2021 and 2023 by the STE CFT and associated government organizations to accelerate traceability work. For instance, in 2021 the STE CFT, TRADOC Proponent Office (TPO) STE, TPO Army Training Information System (ATIS), employed the traceability tool to identify capability requirements overlaps that resulted in the opportunity to conduct engineering deep-dives that identified roles and responsibility in system-to-system information exchanges. Observing case studies between 2021 and 2023, the RAAM ATT consistently reduced time spent on traceability work from months to weeks. In another instance in 2023, the MITRE Corporation used the RAAM to perform requirements engineering and analysis to consolidate, merge, and analyze three requirements sets to create user needs for the ArmyIgnitED's Government off the Shelf (GOTS) education management system. The MITRE systems engineer estimated the time to complete this work

without the RAAM would be 85 hours (estimated based on the number of requirements and an estimated time per requirement). The actual time needed to complete the work with RAAM automation was 14 hours. This resulted in an 84% reduction in time needed to complete the task (percent change).

CONCLUSIONS

Comparing the efficiency of SE work conducted with and without the aid of RAAM's AI capabilities show positive impacts of AI automation in several stages of the SE lifecycle. The RAAM RQA and ATT tools in particular have seen high adoption and performance success within the government userbase where the software has been experimentally used. AI software undoubtedly has potential to severely cut down on time and costs spent on SE tasks and has shown be non-disruptive to traditional engineering workflows, as it can be integrated wherever needed. The RAAM also does not enforce adherence to specific project or SE methodologies.

Contrasting with the experimental success of SE software such as RAAM and in spite of high costs associated with manual labor and human error within the field, there has been little investment thus far into the automation or digitization of repetitive aspects of systems engineering (Zhao, et al., 2021). This can be attributed to the complexity of language-based SE work, which will require the review of experienced engineers even with the use of advanced AI capabilities, especially as oversights can be costly. We believe that a large contributor of the RAAM's success in real-world applications lies in the user experience its GUI delivers, especially in support of human-in-the-loop development. For example, consider the ATT's matching interface depicted in Figure 8 which enables users to review and selected matches found by the tool's document similarity algorithm (with shared word highlighting as visual aid), but also to search for artifacts which had not been identified.

Recent advancements in NLP, namely genAI and other word embeddings-based trained pipelines make NLP tools more dependable and open new avenues for automation. As developers of the RAAM, future work will involve the integration of genAI into more capabilities, such as requirements quality analysis and user story generation. In domain-specific areas, the additional application of RAG and other systems such as RAAM's domain models will be critical to improve genAI performance. Regardless, future software should be designed with human-in-the-loop development in mind by making it easy for engineers to review, accept, and correct the output of AI tools, as such technology is still far from being able to fully replicate the work of human engineers. Overall, this paper points out the significant potential and benefits of applying modern NLP technology to systems engineering. The subject warrants the attention and investment of systems engineers and stakeholders concerned with the high costs of this work.

APPROVAL FOR RELEASE

Approved for Public Release, Distribution Unlimited.

Public Release Case Number 24-2152.

This technical data deliverable was developed using contract funds under Basic Contract No. W56KGU-18-D-0004.

REFERENCES

Ambler, S. (2023). *Examining the Agile Cost of Change Curve.* Retrieved from Agile Modeling: https://agilemodeling.com/essays/costofchange.htm

Army Future Command. (2021, July 21). *AFC Pamphley 71-20-9.* Retrieved from Army Futures Command Concept for Command and Control 2028: Pursuing Decision:

https://api.army.mil/e2/c/downloads/2021/10/06/ffd892d0/afc-concept-for-command-and-control-2028-pursuing-decision-dominance-oct21.pdf

Banerjee, B., Byrch, D., Cady, K., Diehr, L., Droz, C., Forrest, L., . . . Williams. (1999). *IEEE Guide for Developing System Requirements Specifications.*

Barron, R. C., Grantcharov, V., Wanna, S., Eren, M. E., Bhattarai, M., Solovyev, N., . . . Alexandrov, B. S. (2024, October 3). Domain-Specific Retrieval-Augmented Generation Using Vector Stores, Knowledge Graphs, and Tensor Factorization. Los Alamos, New Mexico, United States of America.

BCC Research. (2023). *Natural Language Processing (NLP): Global Market Analysis and Insights.*

Chandrasekaran, D., & Mago, V. (2021). Evolution of Semantic Similarity—A Survey. *ACM Computing Surveys, 54*.

Chiche, A., & Yitagesu, B. (2022). Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data*.

G. Firesmith, D. (2007, January-February). Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *Journal of Object Technology, 6*, 17-33.

INCOSE. (2022). *INCOSE Guide to Writing Requirements V3.1 – Summary Sheet.* Retrieved from INCOSE: https://www.incose.org/docs/default-source/working-groups/requirements-wg/rwg_products/incose_rwg_gtwr_summary_sheet_2022.pdf?sfvrsn=a95a6fc7_2

Jeong, C. (2024). Fine-tuning and Utilization Methods of Domain-specific LLMs. *arXiv preprint arXiv:2401.02981*.

Ling, C., Zhao, X., Lu, J. D., Zheng, C., Wang, J., Chowdhury, T., . . . Chen, Z. (2023). Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey.

Marasco, J. (2007). What Is the Cost of a Requirement Error. *StickyMinds*.

Naseer, S., Ghafoor, M. M., Alvi, S. B., Kiran, A., Rahmand, S. U., Murtazae, G., & Murtaza, G. (2022). Named Entity Recognition (NER) in NLP Techniques, Tools Accuracy and Performance. *Pakistan Journal of Multidisciplinary Research, 2*(2).

OMG. (2022, July). *OMG Unified Architecture Framework v1.2.* Retrieved from Unified Architecture Framework Domain Metamodel: https://www.omg.org/spec/UAF/1.2/DMM/PDF

Pohl, K. (1996). *Requirements Engineering: An Overview.* Aachen: RWTH: Fachgruppe Informatik.

Rajkumar, A. (2022). *Your bad requirements are costing you money.* Deloitte. Retrieved from Deloitte.

U.S. Army. (2019, 03 29). *Annex H: CSE SoN – Reconfigurable Virtual Collective Trainer – 29-March-19.* Retrieved from National Security Technology Accelerator (NSTXL): https://nstxl.org/wp-content/uploads/2019/04/RVCT-SoN_Annex-H-CSE-SoN_29Mar.pdf

Wu, S., Wu, S. H., Wu, J., Feng, L., & Tan, K. C. (2024). Evolutionary computation in the era of large language model: Survey and roadmap. *arXiv preprint arXiv:2401.10034*.

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. *ACM Computing Surveys, 54*.