INCOSE

CHAPTER CONFERENCE

Towards a SMARTer Requirements

An AI-based Approach for Requirements Quality

INCOSE MBSE SUMMIT · 2025

AUTHORS

Sharmila K A, and Ramakrishnan Raman

CHAPTER

India Chapter

PUBLICATION DATE

July, 2026

SUBMISSION #

28

INCOSE PUBLICATIONS

MBSE SUMMIT 2025
"Conquering Complexity with Simplicity: The MBSE Advantage!!"
13th – 14th June, 2025 | Pune, India
hosted by INCOSE INDIA

Towards SMARTer Requirements: An AI-Based Approach for Requirements Quality

Sharmila K A
ORCID: 0009-0003-1289-4697
sharmilaalagesan@gmail.com

Ramakrishnan Raman
ORCID: 0000-0002-8471-9172
ramakrishnan.raman@incose.net

Abstract. It is well recognized that the quality of textual requirements plays a critical role in engineering of complex systems. When textual requirements suffer from issues such as ambiguity, inconsistency, and lack of verifiability, it causes significant issues in the systems engineering lifecycle of complex systems. This paper presents the design, development and fine-tuning of an AI-based system for evaluating and improving the quality of textual requirements. The system classifies requirements as either SMART or non-SMART based on established guidelines, such as the INCOSE Requirements Working Group rules. Using Natural Language Processing (NLP) techniques and a BERT based model, the system recognizes NON-SMART requirements and suggests improvements towards making the requirements more Specific, Measurable, Achievable, Relevant, and Time-bound (SMART). The AI workflow includes several stages: first, the requirements are preprocessed and tokenized using a BERT tokenizer. The model is then fine-tuned on a labeled dataset of SMART and NON-SMART requirements using binary classification. To ensure robustness and avoid overfitting, cross-validation is used, along with model evaluation through performance metrics like accuracy, precision, recall, and F1-score. Additionally, a rule-based system is integrated to detect violations of specific INCOSE rules, and specific suggestions are provided for converting NON-SMART requirements into SMART ones. Designed with modularity and integration in mind, the system supports interoperability with MBSE workflows by facilitating traceability, consistency checking, and the generation of structured artifacts from unstructured inputs. This capability enables a more seamless transition from textual requirements to formal models within MBSE environments. The system's performance is evaluated using confusion matrices and tested on an independent dataset to validate its performance and generalizability. Preliminary results of the AI-based system demonstrate good accuracy, showing immense potential for adoption towards improving requirements quality across various domains and seamlessly transitioning to MBSE workflows.

Keywords. SMART Requirements, NON-SMART Requirements, INCOSE Rules, Requirements Engineering, Requirement Classification, Requirement Suggestion, NLP, BERT, T5, Transformer Models, AI in Systems Engineering, Requirement Quality Assessment, PLM Integration

Introduction

In the domains of Systems Engineering and Product Lifecycle Management (PLM), the quality of system requirements critically influences the success and sustainability of engineering projects across sectors such as automotive, aerospace, and software development. Requirements serve as the foundation

for translating stakeholder needs into functional, technical, and performance specifications. When well-articulated, they foster clarity, alignment, and testability. However, vague, incomplete, or ambiguous requirements often lead to stakeholder misalignment, extensive rework, budget overruns, and delayed deliveries across the development lifecycle.

To mitigate such issues, industry standards like the SMART criteria—Specific, Measurable, Achievable, Relevant, and Time-bound and the INCOSE (International Council on Systems Engineering) 42 rules have been widely adopted. These frameworks aim to ensure requirements are well-structured, unambiguous, and verifiable. Despite their value, manual enforcement of these standards is labor-intensive, error-prone, and inconsistent, especially in large-scale and complex projects. Existing PLM tools also lack intelligent, automated capabilities for requirement evaluation and refinement, highlighting the need for more robust AI-powered systems.

Recent advancements in Natural Language Processing (NLP) and Machine Learning (ML) offer promising directions for enhancing requirements engineering practices. A review of machine learning algorithms for identification and classification of non-functional requirements (Zhao L, Wu Z & Li Y, 2019) highlighted the progress in classification by analyzing ML approaches and different algorithms, demonstrating the effectiveness of supervised learning methods, and showing that ML techniques outperform traditional NLP approaches. This work underlines the transformation of requirements engineering into a modern expert system supported by intelligent ML techniques.

A survey of natural language processing techniques for requirements engineering (Michael and Chulani, 2019) focused on NLP applications in requirement analysis, presenting a solid theoretical foundation by mapping a wide variety of NLP tools, techniques, and resources to support linguistic analysis tasks across different Requirements Engineering phases. BERT: Pre-training of deep bidirectional transformers for language understanding (Devlin J, Chang M W, Lee K & Toutanova K, 2019) introduced the BERT language model, which has become a cornerstone for NLP tasks due to its powerful contextual understanding of language and versatility in fine-tuning for diverse NLP applications.

Software requirements classification using machine learning algorithms (Canedo E & Mendes B, 2018) investigated classification of functional and non-functional requirements, contributing valuable insights into automated requirements classification, and demonstrating the applicability of ML techniques in requirements engineering. BERT for transforming informal requirements into formal specifications (Nayak A, Timmapathini H P, Murali V, Ponnalagu K, Venkoparao V. G & Post A, 2023) developed an approach to transform informal requirements into formal specifications using BERT, showcasing the potential of deep learning models to support automation in requirements formalization.

Based on the insights from existing research in NLP and machine learning for requirements engineering, the developed system offers a comprehensive, AI-powered, end-to-end solution that combines NLP, ML, and rule-based systems to enhance requirement quality. The system consists of: (1) a fine-tuned BERT model for classifying requirements as SMART or NON-SMART; (2) a T5-based model for generating SMART-compliant suggestions for poorly written requirements; and (3) a rule-based engine that enforces INCOSE's 42 best-practice guidelines. The system identifies missing SMART attributes and provides improved versions of the original statements, thereby facilitating the development of precise, complete, and testable requirements.

The motivation behind this research stems from critical shortcomings observed in real-world engineering projects: lack of clarity in requirement specifications, inefficient manual review processes, and the absence of AI-enhanced tools for proactive refinement. By bridging the gap between requirement assessment and actionable enhancement, this system aims to support high-quality requirements engineering and promote standardization across technical domains.

While the proposed approach is initially tailored for textual requirements in electric vehicle (EV) systems, it is designed to be domain agnostic and scalable to other engineering contexts with minor adaptations. This work presents a significant step towards automating requirement engineering by integrating AI-driven classification, rule-based diagnostics, and generative enhancement, all aligned with industry standards to reduce ambiguity, improve stakeholder communication, and optimize the engineering lifecycle.

System Architecture

The proposed AI-driven SMART Requirements Evaluation and Suggestion System is constructed with a modular, extensible, and scalable architecture to enable end-to-end automation of requirement validation. This system leverages both state-of-the-art transformer-based models and a rigorous rule-based engine rooted in the INCOSE Requirements Working Group's 42 rules. It addresses the SMART criteria Specific, Measurable, Achievable, Realistic, and Time-bound while ensuring adaptability across diverse systems engineering domains. This section presents a comprehensive overview of the architecture's components, inter-module data flow, design rationale, and deployment strategy. At an elevated level, the architecture consists of eight interconnected modules: User Interface, Preprocessing and Tokenization, Classification Module, Suggestion Generation Module, Rule-Based INCOSE Validator, Evaluation and Feedback Engine, Model Fine-Tuning Pipeline, and Integration Layer. Each module is independently testable, allowing for rapid development and debugging. The data flows sequentially through these modules, starting from requirement ingestion and culminating in enhanced, SMART-compliant outputs with transparent justifications. The system design supports plug-and-play upgrades of individual components, enabling seamless adoption of newer models or rule updates in the future.

User Interface (UI): The User Interface is a web-based portal developed using Flask and JavaScript, providing an intuitive environment for end users to input free-text requirements. It includes capabilities such as real-time validation, response rendering, classification visualization, and download/export options for outputs. The interface features form validation, dropdown suggestions for requirement categories (e.g., functional, safety, performance), and supports batch upload of requirements via CSV/Excel files. By leveraging asynchronous API calls through AJAX, the UI offers a smooth and responsive experience, even when processing large datasets or complex NLP tasks on the backend.

Preprocessing and Tokenization Module: The preprocessing module is responsible for cleaning and preparing raw text for downstream NLP tasks. It performs lowercasing, punctuation stripping, normalization of special characters, contraction expansion, and lemmatization where appropriate. Additionally, it identifies named entities and domain-specific keywords using custom NER models to assist with downstream classification and rule validation. Tokenization is managed using the Bert Tokenizer and T5Tokenizer from Hugging Face Transformers, supporting consistent and contextual input formatting for the classifier and generation models, respectively. This stage ensures that linguistic inconsistencies do not impede model performance.

Classification Module: The classification engine uses a fine-tuned Bert-base-uncased model for binary classification of requirements into SMART or NON-SMART categories. The model architecture includes a BERT encoder followed by a dropout layer and a dense classification head with sigmoid activation. Fine-tuning was conducted using a labeled corpus of domain-specific requirements with cross-validation and data augmentation techniques to improve generalizability. The classifier leverages the contextual embedding of the [CLS] token for final predictions. It outputs a probability score alongside the binary label, providing a measure of confidence which can be used for threshold tuning in critical applications.

Suggestion Generation Module: For requirements flagged as NON-SMART, the system invokes a T5-based sequence-to-sequence transformer model (t5-base) to rewrite them into SMART-compliant versions. The generation model is fine-tuned on a curated dataset of paired original and improved

requirements, spanning various industrial contexts including automotive, aerospace, and defense. Using prompt-based learning (e.g., “Convert to SMART requirement:”), the model generates output that is grammatically correct, contextually meaningful, and compliant with the SMART guidelines. Beam search decoding is applied during inference to improve the fluency and determinism of the outputs. Post-processing ensures removal of any residual vagueness or grammatical inconsistencies.

Quality Focus	Rules	Subject
Accuracy	R1	Structured Statements
	R2	Active Voice
	R3	Appropriate Subject-verb
	R4	Defined Terms
	R5	Defined Articles
	R6	Common Units of Measure
	R7	Vague Terms
	R8	Escape Clauses
	R9	Open-Ended Clauses
Concision	R10	Superfluous infinitives
	R11	Separate Clauses
Non-Ambiguity	R12	Correct Grammer
	R13	Correct Spelling
	R14	Correct Condition
	R15	Logical Expressions
	R16	Use of “Not”
	R17	Use of Oblique Symbol
Singularity	R18	Single -thought Sentence
	R19	Combinators
	R20	Purpose Phrases
	R21	Parenthesis
	R22	Enumeration
	R23	Supporting Diagram, Model or ICD
	R24	Pronouns
	R25	Headings
Realism	R26	Absolutes
Conditions	R27	Explicit Conditions
	R28	Multiple Conditions
Uniqueness	R29	Classification
	R30	Unique Expression
Abstraction	R31	Solution Free
Quantifiers	R32	Universal Qualification
Tolerance	R33	Range of Values
Qualification	R34	Measurable Performance
	R35	Temporal Dependencies
Uniformity of Language	R36	Consistent Terms and Units
	R37	Acronyms
	R38	Abbreviations
	R39	Style Guide
	R40	Decimal Format
Modularity	R41	Related Needs and Requirements
	R42	Structured Sets

Table 1: INCOSE rules for writing Requirements

Rule-Based INCOSE Validator: To rigorously enforce the INCOSE 42 rules, the system incorporates a robust rule-based engine that evaluates both user-submitted and AI-generated requirements. The rules are encoded using a combination of regular expressions, dependency parsers, and semantic pattern matchers. Categories such as Accuracy (e.g., ensuring factual consistency), Non-Ambiguity (e.g., avoiding vague words like "fast"), and Quantification (e.g., enforcing measurable targets) are systematically validated. Violations are flagged with detailed diagnostics and suggested corrective actions. This rule engine ensures that generated suggestions are not only SMART but also technically sound and standards compliant. It can be dynamically updated to incorporate domain-specific or organization-specific rulesets. The 42 INCOSE Rules are mentioned in Table 1.

Model Fine-Tuning Pipeline: The fine-tuning pipeline ensures that the system remains adaptive to evolving requirement patterns, emerging terminology, and new industrial domains. It supports automated retraining using a versioned dataset repository and integrated annotation platform. Data curation is semi-automated, with heuristics used to prioritize edge cases and low-confidence outputs. Fine-tuning is conducted using GPU-enabled training environments, with checkpoints stored for reproducibility. The pipeline is orchestrated using Apache Airflow, enabling scheduled updates, rollback on failure, and model comparison using standard metrics such as accuracy, BLEU, and ROUGE. This continual learning framework ensures long-term model relevance and performance robustness.

Integration Layer: The integration layer abstracts system functionalities into secure RESTful APIs, enabling third-party tools and enterprise systems to leverage the classification and suggestion services. It supports openAPI/Swagger documentation for ease of adoption and includes endpoints for requirement submission, SMART status retrieval, rule validation results, and improvement suggestions. Authentication and authorization are managed using OAuth 2.0, and role-based access controls are implemented to support different user tiers (e.g., analyst, reviewer, admin). This modularity allows integration with tools such as PTC Windchill, IBM DOORS, JIRA, and Azure DevOps, facilitating automation within broader systems engineering workflows.

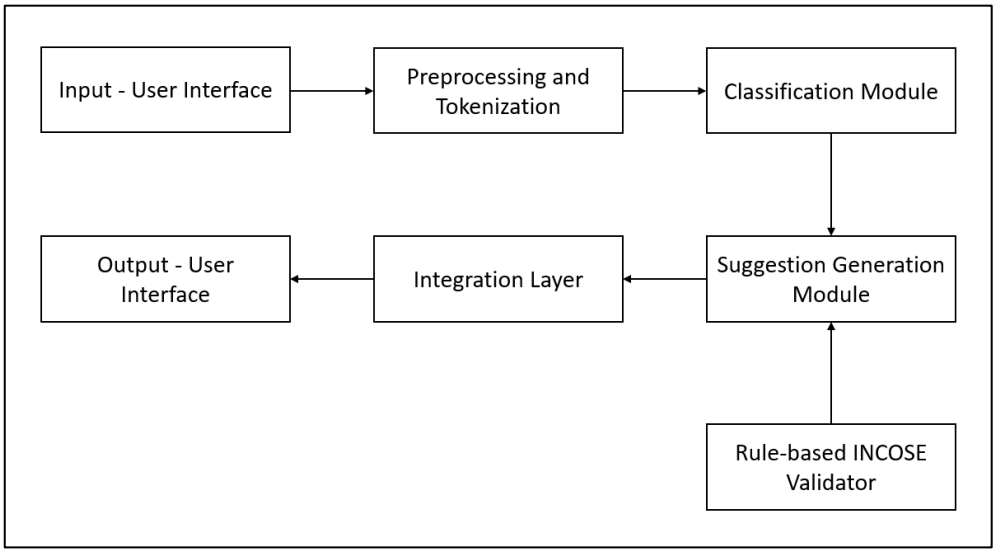

Figure 1. Flow of AI-Driver System for SMART Requirements

Dataset and Implementation

Dataset

The dataset used in this study is specifically designed around Electric Vehicle (EV) Car Requirements, capturing a diverse range of real-world and synthetically generated system-level specifications. It has

been meticulously curated to support both the classification and suggestion generation tasks, forming the foundational component for training, validating, and evaluating the AI-driven SMART requirement evaluation system. This dataset (10000 rows of data) comprises textual requirements labeled as either SMART or NON-SMART, along with detailed annotations that provide insight into which SMART criteria are unmet and which INCOSE (International Council on Systems Engineering) rules are violated. The inclusion of this rich metadata allows the system to perform fine-grained analysis and offer tailored feedback and improvements. Each entry in the dataset contains the following key fields:

- Requirement Text: The raw textual requirement statement, which may range from one-liner feature requests to multi-clause system specifications.
- Classification Label: A binary indicator (1 for SMART, 0 for NON-SMART), derived using a rubric based on SMART criteria—Specific, Measurable, Achievable, Realistic, and Time-bound.
- Missing SMART Criteria: A multi-label field indicating which of the SMART aspects are absent in the NON-SMART requirement (e.g., Measurable, Time-bound).
- Violated INCOSE Rule Categories: Tags indicating the broader rule categories from the 42 INCOSE guidelines that the requirement fails to satisfy (e.g., Non-Ambiguity, Completeness, Quantification).
- Specific INCOSE Rules Violated: IDs and descriptions of the specific rules violated, such as Rule 34 - Measurable Performance, Rule 12 - Avoid Subjective Language, etc.
- SMART Suggestion: A revised version of the requirement which conforms to both the SMART criteria and the INCOSE rules. This serves as the ground truth for training the suggestion generation model.

Implementation

The implementation strategy for building the AI-driven requirement evaluation system includes stages such as data loading and preprocessing, training of two separate transformer-based models (classification and suggestion) and deploying them through a unified Flask API for end-to-end user interaction. The implementation is modular and optimized for scalability, reusability, and integration with external systems such as PLM platforms.

Data Loading And Preprocessing

Dataset Loading: The dataset is stored in an Excel (.xlsx) file format and is loaded using the pandas library. It contains multiple fields critical for training both the classification and suggestion generation models. These include:

- Requirement: The original requirement statements.
- Label: Indicates whether the requirement is SMART (1) or NON-SMART (0).
- Suggestion: The SMART-compliant version of the NON-SMART requirement.
- Missing Category: A comma-separated list of missing SMART criteria (e.g., Specific, Measurable).
- Missing_INCOSE_Rule: Lists violated INCOSE rule categories such as “Non-Ambiguity” or “Accuracy.”

Text Preprocessing: Text preprocessing is essential for standardizing inputs to the language models. It includes the following:

- Lowercasing: To normalize text and remove case sensitivity.
- Removing Punctuation and Special Characters: Cleans noisy elements that can negatively impact model learning.
- Emoji and Non-ASCII Filtering: Ensures that the dataset is clean and well-formatted for tokenizer compatibility.

Preprocessing is applied to both the Requirement and Suggestion columns, and results are stored in Cleaned_Requirement and Cleaned_Suggestion.

Label Mapping: The labels are mapped from categorical (SMART/NON-SMART) to numerical (1/0). This is crucial for training the binary classification model using standard loss functions like cross-entropy.

Feature Engineering for Suggestion Generation: To give contextual information to the suggestion model, a new Input field is created. This field is constructed as:
Cleaned_Requirement + Missing_Category + Missing_INCOSE_Rule
This hybrid representation gives the model both the faulty requirement and the metadata that explains why it is considered NON-SMART, enabling more targeted suggestions.

Data Splitting: Two different splits are prepared:

- Classification: 80/20 training-testing split on the entire dataset.
- Suggestion: Filtered to only NON-SMART requirements, also split 80/20 for model generalization.

This ensures both models are evaluated rigorously on unseen data.

Displaying Preprocessed Data: Previewing the processed dataset helps validate all transformations. This includes checking for data integrity, verifying balanced class distribution, and ensuring no NULL or malformed entries exist.

Training The Classification Model

The Hugging Face Bert-base-uncased tokenizer is used to tokenize input text into IDs. The classification model is a pre-trained BERT model with an additional classification head for binary classification. A custom PyTorch Dataset class encapsulates the tokenization and batching logic. Each data sample includes Input token IDs, Attention mask and Numerical label. This modular structure facilitates easy experimentation and hyperparameter tuning. PyTorch Data Loader enables efficient mini-batch training and shuffling. It abstracts out the complexity of data fetching, batching, and feeding into the GPU.

The AdamW optimizer is chosen for its weight decay regularization and proven efficiency with transformers. The CrossEntropyLoss function is used due to its effectiveness in binary classification.
The loop includes Device-aware training (CPU/GPU), Gradient calculation and backpropagation, Weight updates and Real-time monitoring using tqdm for each epoch. Additional features such as early stopping, learning rate schedulers, and loss tracking can be added for production-grade systems. The fine-tuned model and tokenizer are saved using torch.save() and save_pretrained() functions to ensure quick reuse for inference tasks without re-training.

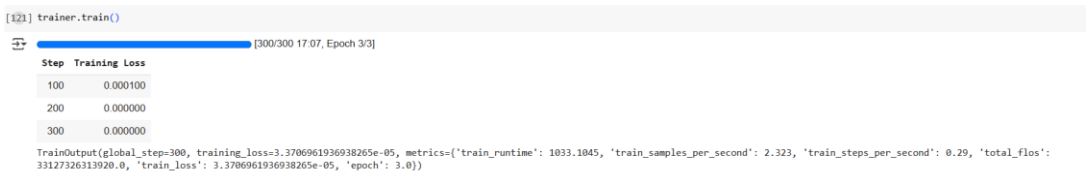

Figure 2. Classification Model Training Loss

Training The Suggestion Model

The suggestion model uses a sequence-to-sequence architecture (T5ForConditionalGeneration) fine-tuned for text transformation. The tokenizer follows the "text-to-text" approach, making it suitable for this task. A custom dataset class tokenizes both the inputs and outputs. Input: NON-SMART requirement + context. Output: Corresponding SMART-compliant suggestion. The text is truncated or padded

to a max length (typically 128 tokens), making it efficient for training without memory overflow. Like the classification model, but specifically tailored for text-to-text tasks. Ensures efficient and shuffled data feeding into the model.

The AdamW optimizer with a learning rate of 5e-5 is used. Learning rate scheduling and gradient clipping can be added for fine-tuning stability. Includes forward pass, loss calculation, backward pass, and optimizer step. The model is trained to minimize the difference between predicted suggestions and ground-truth SMART requirements. The trained T5 model and tokenizer are stored using save pretrained () to ensure reproducibility and seamless inference.

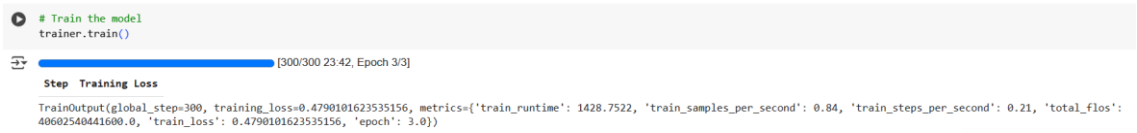

Figure 3. Suggestion Model Training Loss

Model Integration Using Flask Api

Input Handling: The Flask server listens for POST requests at the /classify endpoint. Input is JSON formatted and includes:

```
json
{
  "requirement": "Battery should be long-lasting."
}
```

The API validates this structure and extracts the requirement text.

Classification Logic: The BERT model classifies the requirement. Tokenization and prediction are executed on-the-fly. The highest-scoring class is selected using argmax.

Suggestion Generation Logic: If the requirement is NON-SMART, The corresponding contextual input is generated (based on missing categories and INCOSE rules). This is passed through the T5 model. The output sequence is decoded to form the SMART-compliant suggestion.

Output Formatting: The response contains:

```
json
{
  "requirement": "Battery should be long-lasting",
  "classification": "NON-SMART",
  "suggestion": "Battery should last at least 400 km on a single charge, as tested under standard conditions."
}
```

Device Management: Dynamic allocation to GPU or CPU ensures the API is hardware-agnostic and performant.

Tokenization Strategy: All sequences are tokenized with a max length of 128 tokens. This ensures uniformity, speed, and compatibility with hardware limitations.

Simplified Output: The user-facing API omits internal rule violations for simplicity. The goal is clarity and usability for system engineers and analysts.

Evaluation Metrics and System Performance

Evaluation Metrics

The evaluation metrics for this system are divided into two parts, corresponding to the two models in use: the classification model and the suggestion generation model. Below is a detailed explanation of the metrics, their significance, and how they relate to the system's performance.

Classification Model Evaluation Metrics

The classification model predicts whether a requirement is SMART or NON-SMART. The following evaluation metrics are used to assess its performance:

Accuracy: The percentage of correctly classified requirements (both SMART and NONSMART) out of the total requirements. It provides an overall measure of the model's effectiveness but does not distinguish between false positives and false negatives.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Samples} \quad (1)$$

Precision: The proportion of requirements classified as NON-SMART that are actually NON-SMART. High precision ensures fewer false positives, meaning SMART requirements are rarely misclassified as NON-SMART.

$$Precision = \frac{True\ Positives}{Total\ Positives + False\ Positives} \quad (2)$$

Recall (Sensitivity): The proportion of actual NON_SMART requirements correctly classified by the model. High recall ensures that most NON-SMART requirements are identified, even if some SMART requirements are misclassified.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (3)$$

F1 – score: The harmonic mean of precision and recall, balancing the trade-off between the two metrics. It provides a single metric to evaluate the model's ability to classify correctly while minimizing false positives and false negatives.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (4)$$

Suggestion Model Evaluation Metrics

The suggestion generation model predicts a SMART version of a NON-SMART requirement.
The following metrics evaluate its performance:

BLEU Score (Bilingual Evaluation Understudy): Measures the similarity between the generated suggestion and the reference (ground truth) suggestion using n-gram overlap. BLEU evaluates how closely the generated suggestion matches the expected SMART requirement.

$$BLEU = \mathrm{BP} * \exp(\sum_{n=1}^{N} w_n\ logp_n) \quad (5)$$

ROUGE Score (Recall-Oriented Understudy for Gisting Evaluation): Measures the overlap between the generated suggestion and the reference suggestion, using unigrams, bigrams, and longest common subsequences.

$$ROUGE - N = \frac{Total\ n-grams\ in\ the\ reference}{Number\ of\ matching\ n-grams} \quad (6)$$

$$ROUGE - L = F_\beta = \frac{(1+\beta^2)*Precision*Recall}{\beta^2*Precision+Recall} \quad (7)$$

For the suggestion generation model, the BLEU score was 0.556, indicating a reasonable overlap between the generated suggestions and the reference SMART requirements.
The ROUGE score, which measures the longest common subsequence, was 0.621, suggesting that the generated suggestions capture a significant portion of the content of the reference requirements.

Performance

The performance of the system demonstrates its effectiveness in both classification and suggestion generation tasks. The classification model reliably identifies requirements as SMART or NON-SMART, achieving high accuracy and balanced performance across precision, recall, and F1-score metrics. This ensures the model correctly flags requirements needing improvement while minimizing misclassifications. The suggestion generation model, built using a T5 architecture, excels in transforming NON-SMART requirements into actionable SMART requirements. Evaluation metrics such as BLEU and ROUGE scores indicate a strong alignment between generated suggestions and reference requirements, reflecting the model's ability to address missing SMART criteria like specificity, measurability, and time-bound constraints.

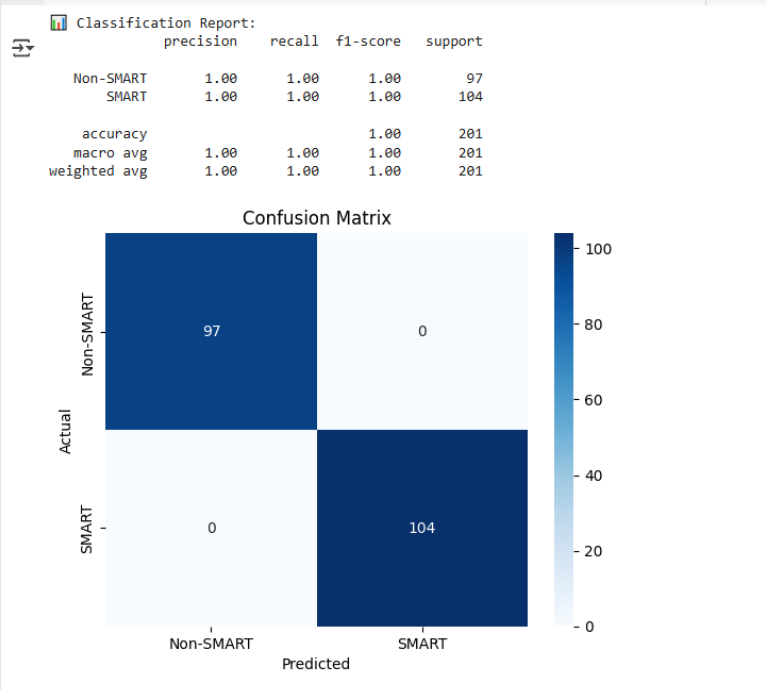

Figure 4. Classification Model Evaluation

Overall, the system provides robust and coherent outputs, making it a practical and reliable tool for automating requirement evaluation and refinement in various domains. To further enhance the system's

performance and adaptability, the models are continuously fine-tuned using domain-specific (Automotive Domain – EV Cars Requirements) datasets. This fine-tuning process allows the classification model to better understand the nuances of requirements from various industries, such as automotive, software, or aerospace, improving its ability to classify requirements accurately. Similarly, the suggestion generation model is fine-tuned to produce SMART suggestions that are more aligned with industry-specific terminology and practices. By incorporating additional examples, particularly for edge cases and diverse requirement styles, the system evolves to handle complex and ambiguous inputs effectively. This iterative fine-tuning ensures that the models remain robust and reliable, adapting to changing requirements and standards over time. The performance of the model is illustrated in Figure 4.

```python
import evaluate

def compute_bleu_rouge(predictions, references):
    bleu = evaluate.load("bleu")
    rouge = evaluate.load("rouge")

    bleu_result = bleu.compute(predictions=predictions, references=[[r] for r in references])
    rouge_result = rouge.compute(predictions=predictions, references=references)

    return bleu_result["bleu"], rouge_result["rougeL"]
```

```python
bleu_score, rougeL_score = compute_bleu_rouge(preds, refs)

print(f"📊 BLEU Score: {bleu_score:.4f}")
print(f"📊 ROUGE-L Score: {rougeL_score:.4f}")
```

```
📊 BLEU Score: 0.5385
📊 ROUGE-L Score: 0.5650
```

Figure 5. Suggestion Model Evaluation

Output

This Output shows a Flask-based web application designed to assist with analyzing and improving the quality of requirements based on the SMART criteria.

Use Case 1:

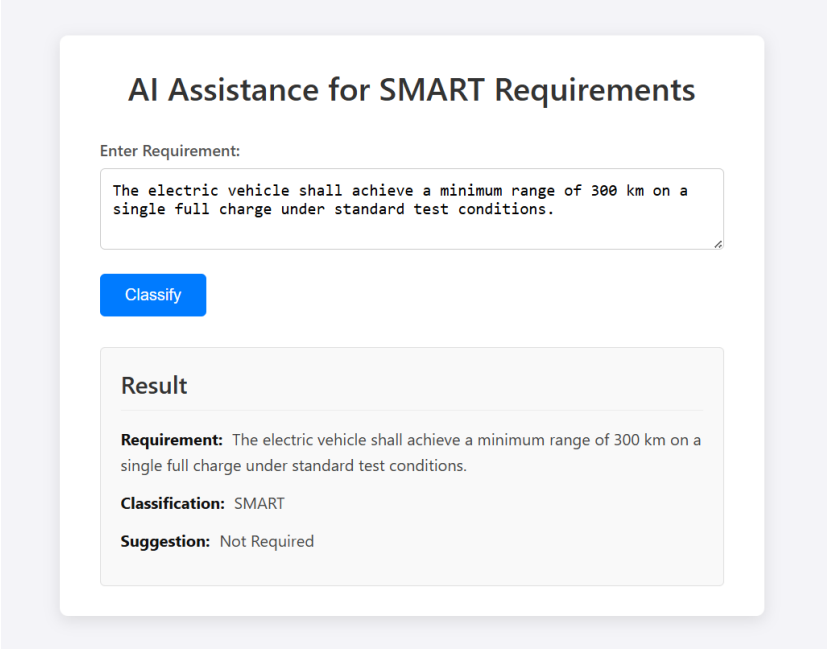

Figure 6. User Interface output of use case 1

Requirement: The electric vehicle shall achieve a minimum range of 300 km on a single full charge under standard test conditions.
Classification: SMART
Suggestion: Not Required

Use Case 2:

Requirement: The system should be energy efficient.
Classification: NON-SMART
Suggestion: The system shall consume no more than 100 Wh per operational hour during standard usage conditions.

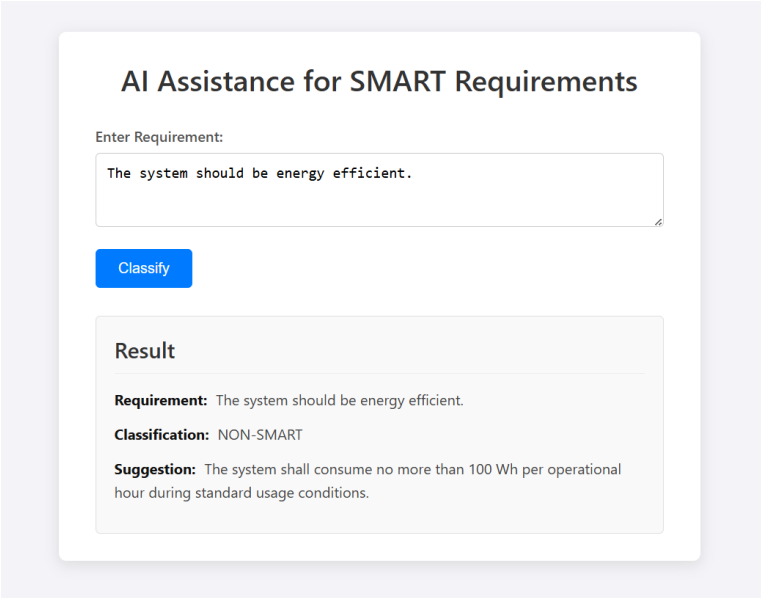

Figure 7. User Interface output of use case 2

Use Case 3:

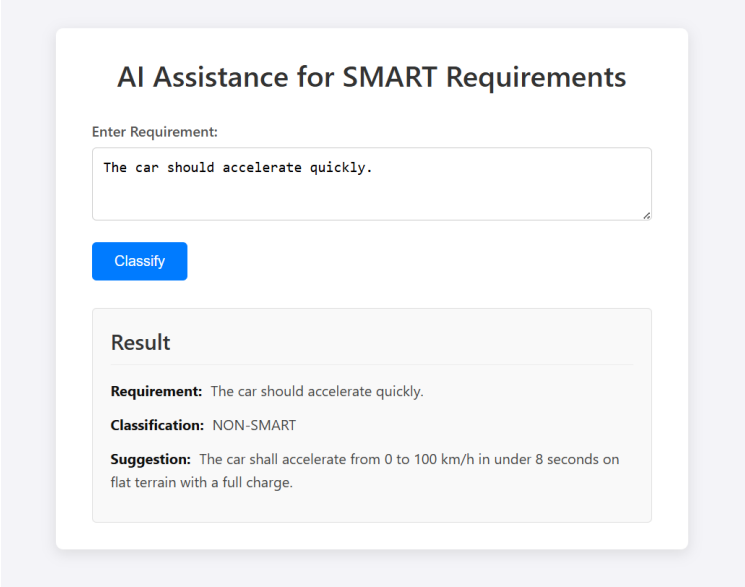

Figure 8. User Interface output of use case 3

Requirement: The car should accelerate quickly.
Classification: NON-SMART
Suggestion: The car shall accelerate from 0 to 100 km/h in under 8 seconds on flat terrain with full charge.

Use Case 4:

Requirement: The battery shall provide a minimum driving range of 400 km per charge for at least 4 years, with no more than a 10% decrease in range over time.
Classification: SMART
Suggestion: Not Required

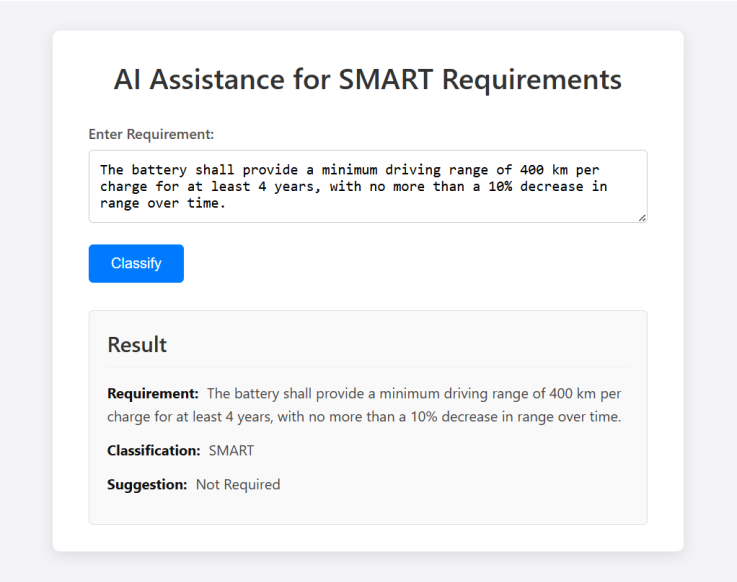

Figure 9. User Interface output of use case 4

Integration with MBSE Workflow

The workflow integrates AI Assistance for SMART Requirements within the Model-Based Systems Engineering (MBSE) lifecycle. It begins in the Requirement (R) layer, where inputs such as system context, use-cases, and composition elements are used to define the Initial System Requirements. These requirements are then fed into the AI Assistance for SMART Requirements module, which classifies them based on SMART criteria—Specific, Measurable, and Time-bound. If a requirement is found to be non-compliant, the AI system generates improved Revised System Requirements using rule-based checks and language models.

The revised requirements are then propagated into the Functional (F) layer, where they guide system modeling from a Blackbox and Whitebox perspective. Functional roles are defined and mapped through a Function Allocation process. These allocations inform the Logical (L) layer, which constructs the Logical System Structure, Interface Allocations, and Logical Behaviors with reference to a generic resource model. Finally, the Technical (T) layer implements the Technical System Structure based on logical definitions. It performs Technical Allocation, defines Internal Technical Structures, and connects to configuration and resource models. This integrated AI-assisted workflow ensures traceability, correctness, and compliance of system requirements throughout the design process.

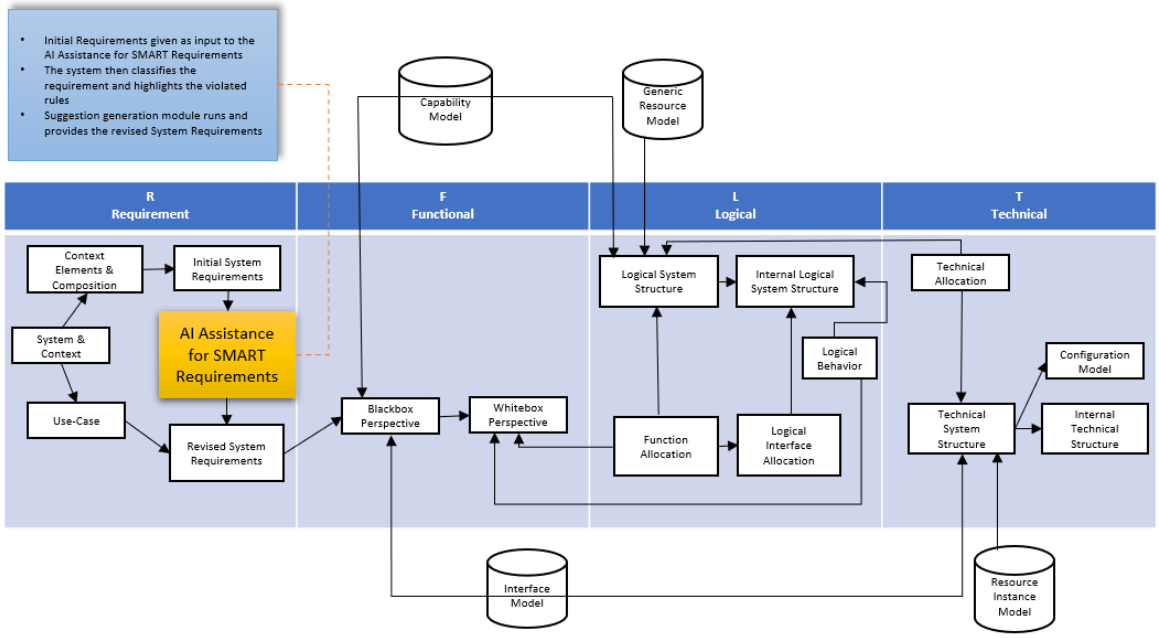

Figure 10. Integration with MBSE workflow (Workflow adapted from L. Beers, M. Weigand, H. Nabizada and A. Fay, 2023)

Limitations

- The tool is trained primarily on EV domain requirements, limiting its effectiveness across other domains without further adaptation. Domain-specific vocabulary, contextual nuances, and requirement patterns may vary significantly in sectors such as aerospace, healthcare, or defense. Without retraining or fine-tuning on domain-specific datasets, the system's classification and suggestion capabilities may underperform.
- The current dataset used in this study consists of single-line, standalone requirements. Due to the nature of these requirements, it is not feasible to evaluate completeness, consistency across a set, or detect duplicates meaningfully.
- This tool mainly focuses on only three SMART criteria (Specific, Measurable, Time-bound), potentially overlooking Achievable and Relevant aspects. This partial implementation could result in requirements being classified as SMART even if they are not feasible (Achievable) or aligned with project objectives (Relevant).
- Automated suggestions may not fully capture stakeholder intent, requiring human validation. While the system generates improved SMART-compliant requirements, the nuances of user intent, organizational policies, and technical constraints are not always captured by NLP models. This may lead to suggestions that, although syntactically and structurally correct, are semantically misaligned with project goals.
- Integration into existing MBSE workflows may face compatibility issues or require additional customization effort. Ensuring seamless interaction between this AI-driven tool and legacy MBSE platforms may necessitate development of custom APIs or middleware.

Conclusion

The developed system provides a robust and automated approach to assessing and improving the quality of requirements. By integrating a classification model and a suggestion generation model, the system ensures that requirements are evaluated against SMART criteria and INCOSE rules, addressing key

gaps in clarity, specificity, and measurability. This approach significantly reduces ambiguity in requirements, leading to enhanced understanding and better alignment with project goals. The classification model effectively identifies whether a requirement is SMART or NONSMART. This initial step is crucial in flagging requirements that need refinement, ensuring that only those requiring attention are processed further.

For NON-SMART requirements, the suggestion generation model dynamically generates SMART versions, making requirements more actionable, testable, and consistent with industry standards. One of the key strengths of the system is its ability to integrate and apply the 42 INCOSE rules, ensuring that generated suggestions are not only SMART but also adhere to best practices in requirement engineering. This compliance ensures that the requirements are practical and precise, facilitating smoother implementation in real-world scenarios. The system's focus on automation reduces manual effort, allowing requirement engineers to focus on high-level tasks while maintaining consistency and accuracy in requirement documentation.

This system also offers significant potential for scalability and adaptability. It can be finetuned with domain-specific data to cater to various industries, such as automotive, aerospace, and software development. Additionally, its integration into existing tools and workflows ensures it can seamlessly fit into enterprise processes, enhancing productivity and standardization.

In conclusion, this system represents a step forward in modernizing requirement engineering practices. By automating the evaluation and refinement of requirements, it addresses familiar challenges such as ambiguity and lack of specificity. With continuous updates and domain-specific adaptations, the system is well-positioned to become a valuable tool for industries where clear and actionable requirements are critical for success.

Recommendations

Several key enhancements are recommended to significantly improve the system’s performance, usability, and applicability across different domains. Expanding the dataset to include a broader range of domain-specific requirements is essential. By incorporating data from various industries such as healthcare, finance, automotive, and software development, the model can achieve better generalization and robustness. Feedback mechanism by integrating user input to allow the system to learn and adapt continuously. Users can provide valuable feedback on the correctness of classifications and the relevance of suggestions, which can be used to iteratively refine the models. This dynamic learning process will help the system stay aligned with real-world needs and evolving practices. Improving explainability is crucial for user trust and effectiveness.

Integrating the system into widely used requirement management tools like Jira to facilitate seamless adoption. Embedding the solution within existing workflows ensures that users can access its capabilities without disrupting their current processes, thereby enhancing productivity and user experience. Supporting multilingual capabilities by leveraging advanced multilingual models will expand the system’s usability globally. This feature is particularly important for international teams and projects where requirements may be documented in various languages. Focusing on scalability through continuous fine-tuning and updates will ensure that the models remain effective as industry standards evolve and new application areas emerge. This ongoing maintenance will keep the system relevant and capable of addressing diverse and changing requirements.

Acknowledgements

This work was conducted as part of the engineering masters MTech. Project work by the first author under the Work Integrated Learning Program (WILP) at BITS Pilani. The authors would like to express

their sincere gratitude to Dr. Yogananda Jeppu, who served as the external examiner for this masters degree project, for his valuable insights and feedback. The authors would also like to thank Dr. Prajna Devi Upadhyay for her constructive suggestions as part of the project work viva.

References

Ahmad, K., Almorsy, M., Arora, C., Bano, M., & Grundy, J. C. (2022). Requirements engineering for artificial intelligence systems: A systematic mapping study. Information and Software Technology, 158, 107176. https://doi.org/10.1016/j.infsof.2022.107176

Arora, C., Bano, M., Ahmad, K., Almorsy, M., & Grundy, J. (2021). What's up with requirements engineering for artificial intelligence systems? In Proceedings of the 29th IEEE International Requirements Engineering Conference (RE 2021) (pp. 1–12). IEEE. https://doi.org/10.1109/RE51729.2021.00013

Bano, M., Ahmad, K., Almorsy, M., Arora, C., & Grundy, J. (2019). Requirements engineering for artificial intelligence systems: A systematic mapping study. arXiv preprint arXiv:2212.10693. https://arxiv.org/abs/2212.10693

Beers, L., Weigand, M., Nabizada, H., and Fay, A., (2023) MBSE Modeling Workflow for the Development of Automated Aircraft Production Systems. IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 2023, pp. 1-8, https://doi.org/10.1109/ETFA54631.2023.10275444

Canedo, E., & Mendes, B. (2018). Software requirements classification using machine learning algorithms. In Proceedings of the 2018 IEEE/ACM International Conference on Software Engineering (ICSE) (pp. 251–261). IEEE. https://doi.org/10.1145/3180155.3180243

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

Franch, X., Mairiza, D., & Orue-Echevarria, L. (2023). Requirements engineering for artificial intelligence systems. Information and Software Technology, 155, 107070. https://doi.org/10.1016/j.infsof.2023.107070

Knauss, E., Felderer, M., & Vogelsang, A. (2023). Requirements engineering for artificial intelligence: Results from a systematic mapping study. Journal of Systems and Software, 198, 111613. https://doi.org/10.1016/j.jss.2023.111613

Kurtanović, Z., & Maalej, W. (2019). Automatically classifying functional and non-functional requirements using supervised machine learning. IEEE Transactions on Software Engineering, 45(11), 1092–1107. https://doi.org/10.1109/TSE.2018.2870781

Lalwani, T., Bhalotia, S., Pal, A., Bisen, S., & Rathod, V. (2021). Implementation of a chat bot system using AI and NLP. In Proceedings of the 2021 International Conference on Artificial Intelligence and Data Engineering (ICAIDE) (pp. 210–217). IEEE. https://doi.org/10.1109/ICAIDE52030.2021.00039

Mairiza, D., Franch, X., & Orue-Echevarria, L. (2023). Requirements engineering for artificial intelligence systems: A review of current approaches and challenges. Science of Computer Programming, 231, 102882. https://doi.org/10.1016/j.scico.2023.102882

Malerba, E. D., & Cegarra, F. A. (2020). Machine learning for requirements engineering: A systematic review. IEEE Transactions on Software Engineering, 46(6), 611–629. https://doi.org/10.1109/TSE.2020.2986090

Michael, J. B., & Chulani, S. (2018). A survey of natural language processing techniques for requirements engineering. IEEE Access, 6, 67910–67928. https://doi.org/10.1109/ACCESS.2018.2879768

Nayak, A., Murali, V., Ponnalagu, K., Timmapathini, H. P., Venkoparao, V. G., & Post, A. (2023). Req2Spec: Transforming software requirements into formal specifications using large language models. In Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ). Springer. https://doi.org/10.1007/978-3-030-98464-9_8

Vogelsang, A., Borg, M., & Wnuk, K. (2023). Requirements engineering practices for AI/ML-based systems: A systematic literature review. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5080988

Zhao, L., Wu, Z., & Li, Y. (2019). A review of machine learning algorithms for identification and classification of non-functional requirements. Journal of Systems and Software, 147, 1–16. https://doi.org/10.1016/j.jss.2018.09.047

Biography

Sharmila K A holds Bachelor of Engineering degree from PSG Institute of Technology, and a Master of Technology degree from BITS Pilani. With experience in industrial IOT and automation systems, she contributes to building scalable, innovative solutions for manufacturing sectors. Her area of interest includes AI/ML, digital transformations, and system integrations. She is currently a Senior Developer at Schneider Electric India, specializing in Software application development for PLM systems. Prior to her current role, she was with Volvo Group as a Software Engineer, and with ITC InfoTech as an Associate IT Consultant. ORCID: 0009-0003-1289-4697

Ramakrishnan Raman has B.Tech and MS degrees from IIT Madras and PhD from IIIT Bangalore. He is an INCOSE Fellow and ESEP. He has to his credit several publications in refereed international conferences & journals on complex systems architecture design and artificial intelligence – machine learning. He is currently Senior Chief Engineer at Eaton, where he oversees the Chief Engineers Office for Systems Engineering. Prior to Eaton, he was a Fellow at Honeywell Aerospace. ORCID: 0009-0003-1289-4697