Streamlining Requirement Management w/ AI

Michael Jordan
michael.jordan@specinnovations.com

Background

Context

- Requirements are the foundation for projects
  - Provides communication between engineering team & stakeholders
  - Aids in system design & overviews the demands of the system
- Artificial Intelligence (AI) refers to the ability of machines to perform tasks that would normally require human intelligence such as learning, problem-solving, decision-making, and pattern recognition
- Effective requirement management is critical for project success and AI can help

Objectives

- Several branches of AI are under evaluation and can be used for requirement development
  - Machine Learning (ML)
  - Computer Vision (CV)
  - Natual Language Processing (NLP)
- Applying AI techniques can streamline and aid users with requirement management
  - Offer real-time quality checks, correlation assessments, decision support, and intelligent input
- AI is going to be used an ‘Assistance’ tool and will always require human interaction

Requirement Management Background

- Requirements can guide all stages of projects
- Requirement management requires to elicit, analyze, and manage requirements
  - AI can help in providing efficient & fast assistance
- AI can improve requirements but its still requires vetting from the user

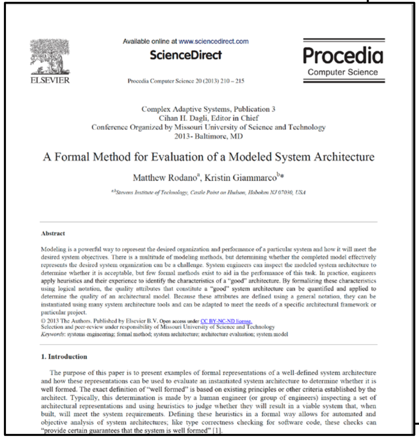
Methodology

Methodology

- Innoslate is a platform that uses AI techniques and applications to aid with requirement management
  - Natural Language Processing (NLP) Algorithms
  - Heuristics
  - Generative Pre-trained Transformers (GPT)
- NLP algorithms can assist with real-time quality checks on requirements
- Heuristics aid in requirement structuring and decision-making
- GPT-powered systems allows for additional feedback from user’s input

How do we determine requirement quality?

- Apply rules to assess the quality of requirements
- Rules come from:
  - Guidebooks
  - Standards
  - Best Practice



Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 20 (2013) 210 – 215

Procedia Computer Science

Complex Adaptive Systems, Publication 3
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2013- Baltimore, MD

A Formal Method for Evaluation of a Modeled System Architecture

Matthew Rodano[a], Kristin Giammarco[b]*

[a,b]Stevens Institute of Technology, Castle Point on Hudson, Hoboken NJ 07030, USA

Abstract

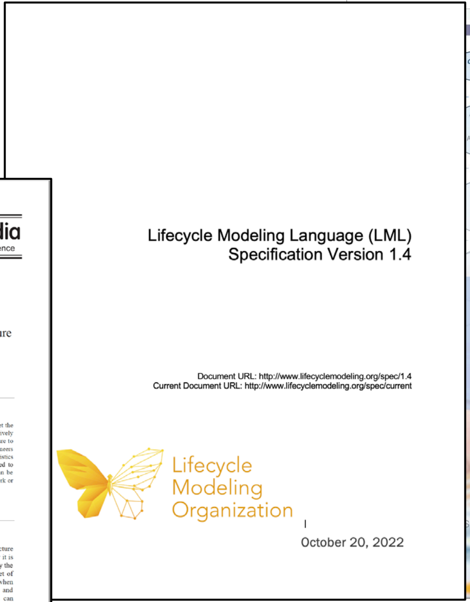
Modeling is a powerful way to represent the desired organization and performance of a particular system and how it will meet the desired system objectives. There is a multitude of modeling methods, but determining whether the completed model effectively represents the desired system organization can be a challenge. System engineers can inspect the modeled system architecture to determine whether it is acceptable, but few formal methods exist to aid in the performance of this task. In practice, engineers apply heuristics and their experience to identify the characteristics of a "good" architecture. By formalizing these characteristics using logical notation, the quality attributes that constitute a "good" system architecture can be quantified and applied to determine the quality of an architectural model. Because these attributes are defined using a general notation, they can be instantiated using many system architecture tools and can be adapted to meet the needs of a specific architectural framework or particular project.

© 2013 The Authors. Published by Elsevier B.V. Open access under CC BY-NC-ND license.
Selection and peer-review under responsibility of Missouri University of Science and Technology

Keywords: systems engineering; formal method; system architecture; architecture evaluation; system model

1. Introduction

The purpose of this paper is to present examples of formal representations of a well-defined system architecture and how these representations can be used to evaluate an instantiated system architecture to determine whether it is well formed. The exact definition of "well formed" is based on existing principles or other criteria established by the architect. Typically, this determination is made by a human engineer (or group of engineers) inspecting a set of architectural representations and using heuristics to judge whether they will result in a viable system that, when built, will meet the system requirements. Defining these heuristics in a formal way allows for automated and objective analysis of system architectures; like type correctness checking for software code, these checks can "provide certain guarantees that the system is well formed" [1].

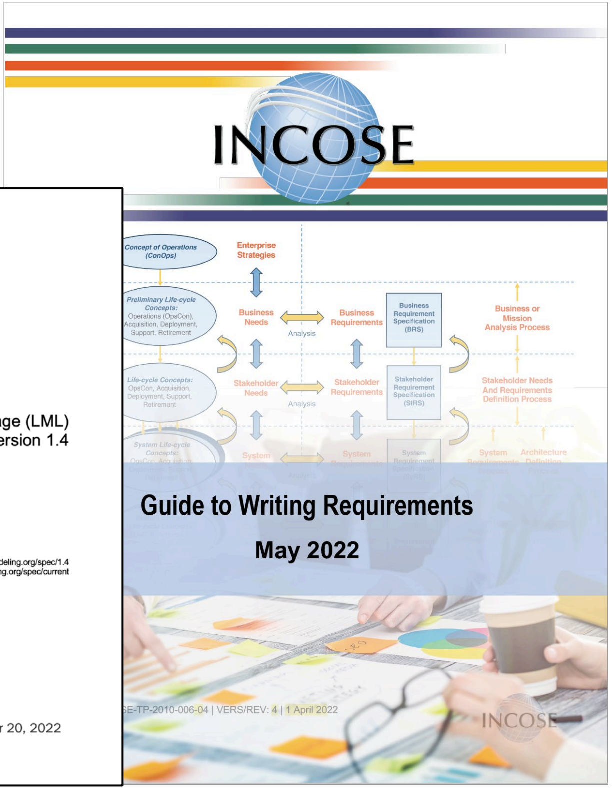


Lifecycle Modeling Language (LML)
Specification Version 1.4

Document URL: http://www.lifecyclemodeling.org/spec/1.4
Current Document URL: http://www.lifecyclemodeling.org/spec/current

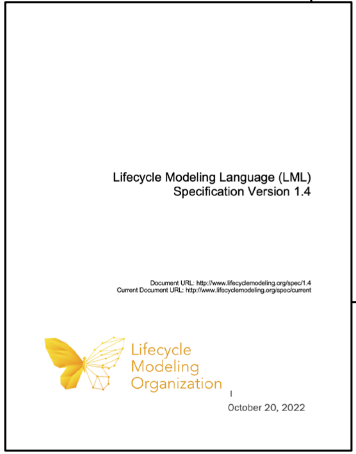
Lifecycle Modeling Organization

October 20, 2022



Requirement Quality in the LML Standard

- Derived from work by Ivy Hooks
- Used as an example, not as a mandatory language element
- Implemented to the NLP algorithm


Lifecycle Modeling Language (LML)
Specification Version 1.4

Document URL: http://www.lifecyclemodeling.org/spec/1.4
Current Document URL: http://www.lifecyclemodeling.org/spec/current

Lifecycle Modeling Organization

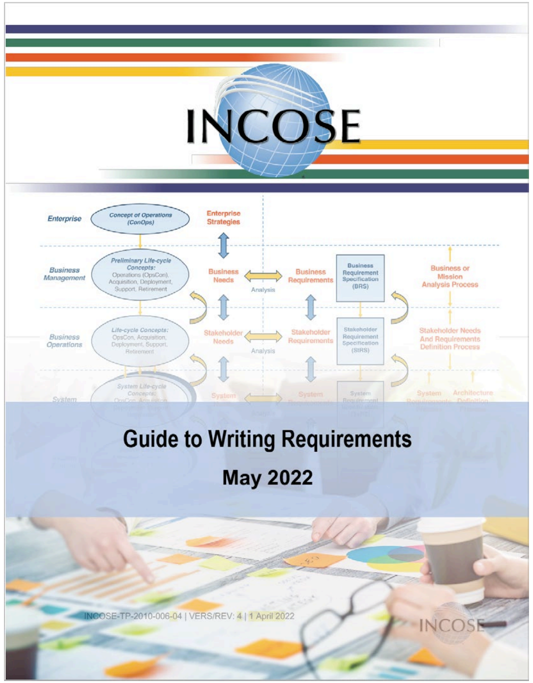
October 20, 2022

Note the quality attributes below are optional. Other sets of quality attributes may be provided by the tool developer, or these may be user-definable. However, some form of quality attributes is recommended.

Attribute	Type	Description
clear	Boolean	*clear* represents if this **Requirement** is unambiguous and not confusing.
complete	Boolean	*complete* represents if this **Requirement** expresses a whole idea.
consistent	Boolean	*consistent* represents if this **Requirement** is not in conflict with other requirements.
correct	Boolean	*correct* represents if this **Requirement** describes the user's true intent and is legally possible.
design	Boolean	*design* represents if this **Requirement** does not impose a specific solution on design; says "what", not "how".
feasible	Boolean	*feasible* represents if this **Requirement** can be implemented with existing technology, within cost and schedule.
modular	Boolean	*modular* represents if this **Requirement** can be changed without excessive impact on other requirements.
traceable	Boolean	*traceable* represents if this **Requirement** is uniquely identified, and able to be tracked to predecessor and successor lifecycle items/objects.
verifiable	Boolean	*verifiable* represents if this **Requirement** is provable (within realistic cost and schedule) that the system meets the requirement.

Requirement Quality from Guidebooks

- INCOSE release the updated Guide to Writing Requirements
- Identified the following on high quality requirements:
  - 14 characteristics for individual requirements
  - 41 rules and associated attributes
- Mapped this set of characteristics and rules to the current application to NLP algorithm to be utilized for requirement management



Requirements Quality from Best Practice

- Heuristics are formal representations of well-defined system architectures
  - Heuristics comes from research done by NPS
- Formal rules derived from analysis of best practice
- 9 out of 68 heuristic focused on requirements
- Implemented these heuristics to Innoslate's Intelligence View

	2. Requirements Traceability Axioms	*Reference*
2.1	Every activity shall be based on some requirement. $(\forall a \in A)(\exists q \in Q)[\mathit{basedon}(a,q)]$	[6]
2.2	Every resource shall be specified by some requirement. $(\forall r \in R)(\exists q \in Q)[\mathit{specifies}(q,r)]$	[6]
2.3	No requirement shall specify more than one resource. $(\forall r_1 \in R)(\forall r_2 \in R)(\nexists q \in Q)[\mathit{specifies}(q,r_1) \wedge \mathit{specifies}(q,r_2) \wedge (r_1 \neq r_2)]$	[12]
2.4	All leaf-level requirements shall be specified by at least one element in the *Activity*, *Connector*, *Performer*, or *Resource* class. $(\forall q_1 \in Q)\left[(\nexists q_2 \in Q)\,\mathit{decomposedby}(q_1,q_2) \rightarrow \begin{pmatrix}(\exists a \in A)\,\mathit{basedon}(a,q_1) \vee (\exists r \in R)\,\mathit{specifiedby}(r,q_1) \vee \\ (\exists p \in P)\,\mathit{specifiedby}(p,q_1) \vee (\exists c \in C)\,\mathit{specifiedby}(c,q_1)\end{pmatrix}\right]$	[6]

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 20 (2013) 210 – 215

Procedia Computer Science

Complex Adaptive Systems, Publication 3
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2013- Baltimore, MD

A Formal Method for Evaluation of a Modeled System Architecture

Matthew Rodano[a], Kristin Giammarco[b]*

[a,b]Stevens Institute of Technology, Castle Point on Hudson, Hoboken NJ 07030, USA

Abstract

Modeling is a powerful way to represent the desired organization and performance of a particular system and how it will meet the desired system objectives. There is a multitude of modeling methods, but determining whether the completed model effectively represents the desired system organization can be a challenge. System engineers can inspect the modeled system architecture to determine whether it is acceptable, but few formal methods exist to aid in the performance of this task. In practice, engineers apply heuristics and their experience to identify the characteristics of a "good" architecture. By formalizing these characteristics using logical notation, the quality attributes that constitute a "good" system architecture can be quantified and applied to determine the quality of an architectural model. Because these attributes are defined using a general notation, they can be instantiated using many system architecture tools and can be adapted to meet the needs of a specific architectural framework or particular project.

Selection and peer-review under responsibility of Missouri University of Science and Technology
Keywords: systems engineering; formal method; system architecture; architecture evaluation; system model

1. Introduction

The purpose of this paper is to present examples of formal representations of a well-defined system architecture and how these representations can be used to evaluate an instantiated system architecture to determine whether it is well formed. The exact definition of "well formed" is based on existing principles or other criteria established by the architect. Typically, this determination is made by a human engineer (or group of engineers) inspecting a set of architectural representations and using heuristics to judge whether they will result in a viable system that, when built, will meet the system requirements. Defining these heuristics in a formal way allows for automated and objective analysis of system architectures; like type correctness checking for software code, these checks can "provide certain guarantees that the system is well formed" [1].

AI Process

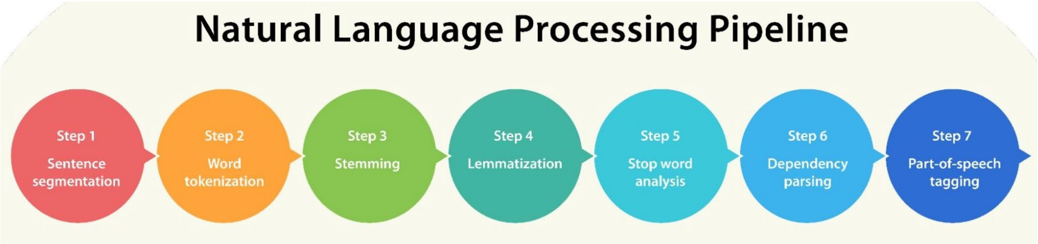
NLP Algorithms

- NLP algorithm is used to evaluate and assess the quality of requirements
- Use 6 of the 8 quality indicator attributes to assist the NLP algorithms
  - Map directly to the INCOSE guide characteristics
  - ‘Correct’ & ‘Feasible’ are not automated & left for user review

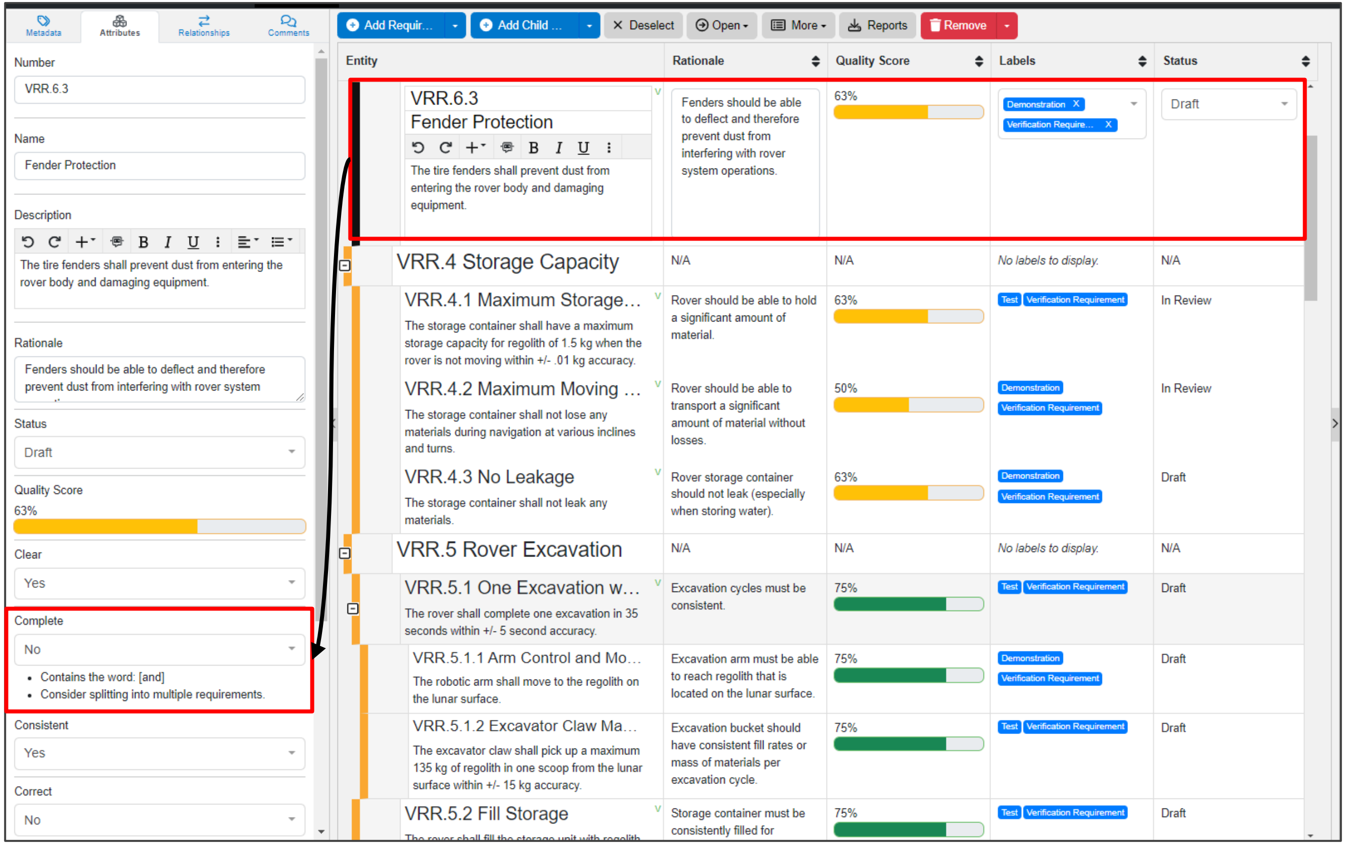
Innoslate Quality Checker	INCOSE Characteristics	INCOSE Rules
Clear – passive voice	C3 – Unambiguous	R2 – Use Active Voice
Complete - begin with capital letter, end with period, contains comma, contains and, contains multiple sentences	C4 – Complete	R1 – Sentence Structure, R18 – Single Sentence, R19 – Avoid Combinators
Consistent – potential duplicate	C5 – Singular	R30 – Express Once
Correct – not automated	C8 – Correct	R27 – Conditions/Explicit
Design – does not impose on a specific solution	C2 – Appropriate	R31 – Solution Free
Feasible – not automated	C6 – Feasible	R26 – Realism/AvoidAbsolutes R33 – Tolerance/ValueRange
Traceable – unique number	C1 – Necessary	R20 – Singularity/AvoidPurpose
Verifiable – looks for ambiguities (it), looks for number of modal verbs (shall, will, must)	C7 - Verifiable	R1 – Sentence Structure, R24 – Avoid Pronouns, R32 - Universals

Utilizing NLP w/ Requirements

- NLP will scan the requirement documentation based on the attributes
- A quality score will be provided to Innoslate user
  - Quality score is out of 100%
- Any errors that result in low score:
  - AI shall identify which attribute the requirement is missing
  - Provide suggestion to revise error

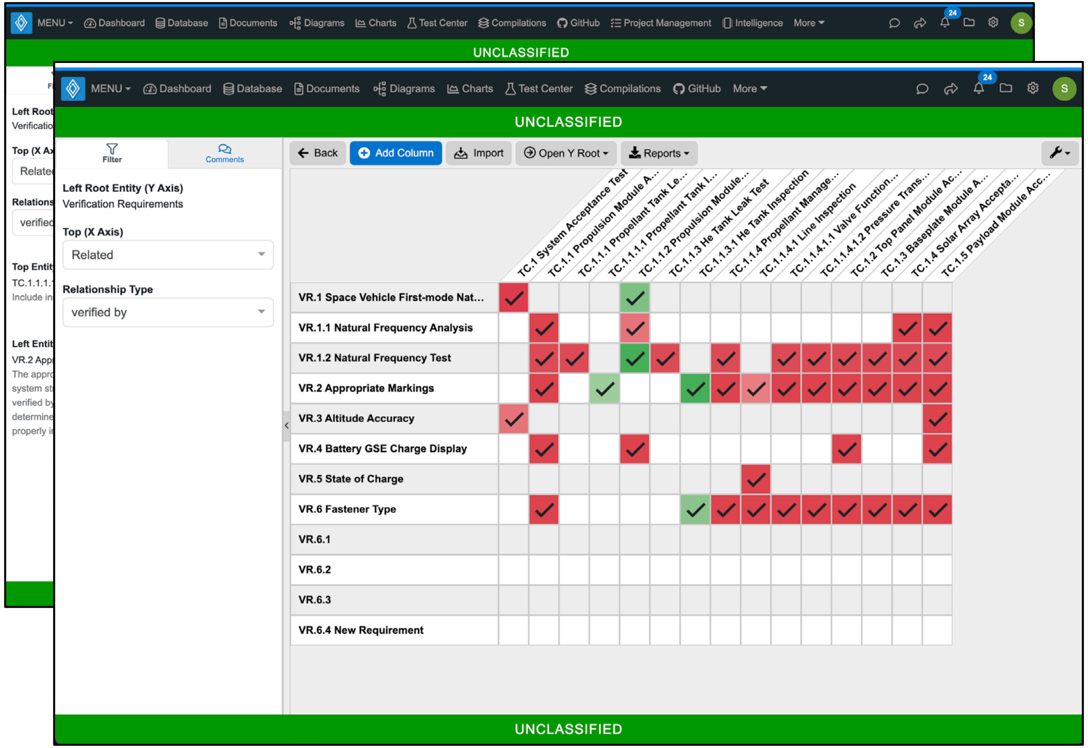


Requirement Management w/ NLP View



AI & Requirement Traceability

- Traceability Matrix provides a means to establish traceability using any relationships
- Traceability and Suspect Assists using NLP to aid users in making good traceability decisions



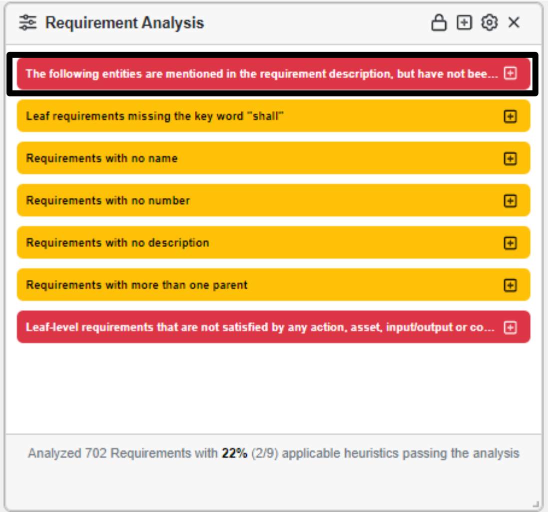
Heuristics w/ Requirement Management

- Heuristics confirms the requirements are properly set based on the project's database
  - Relationships
  - Decompositions
  - Proper Hierarchy
  - Unnecessary Redundant Traceability
- Innoslate's Intelligence View uses heuristics to allow users to review, fix, and/or ignore configurations
  - Provides an update for the user

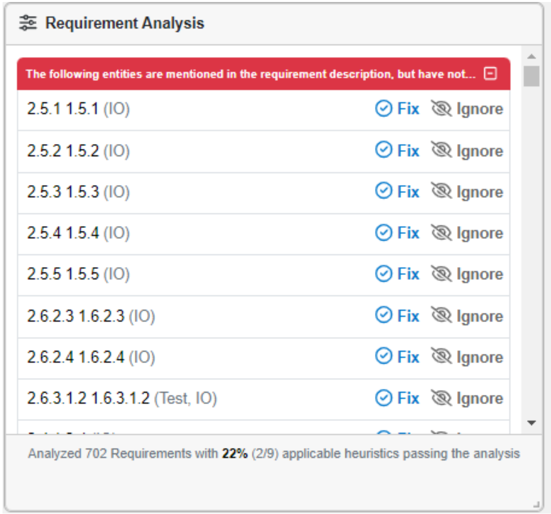
Intelligence View's Requirement Heuristics
An IO/Asset/Action called out in a requirement should be related to that requirement.
Shall use in leaf level requirements.
Each requirement has a name.
Each requirement has a number.
Each requirement has a description.
Each requirement has a child or parent.
No requirement has more than one parent.
All leaf-level requirements are related to at least one entity in the Action, Conduit, Asset, or Input/Output class.
No requirement is satisfied by more than one input/output.

Requirement Management w/ Heuristic View

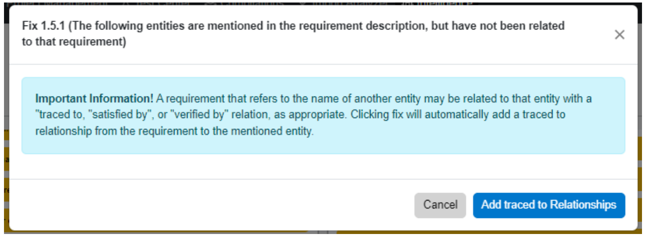
1. Requirement Heuristics



2. Heuristics Violations



3. Heuristic Violation Overview

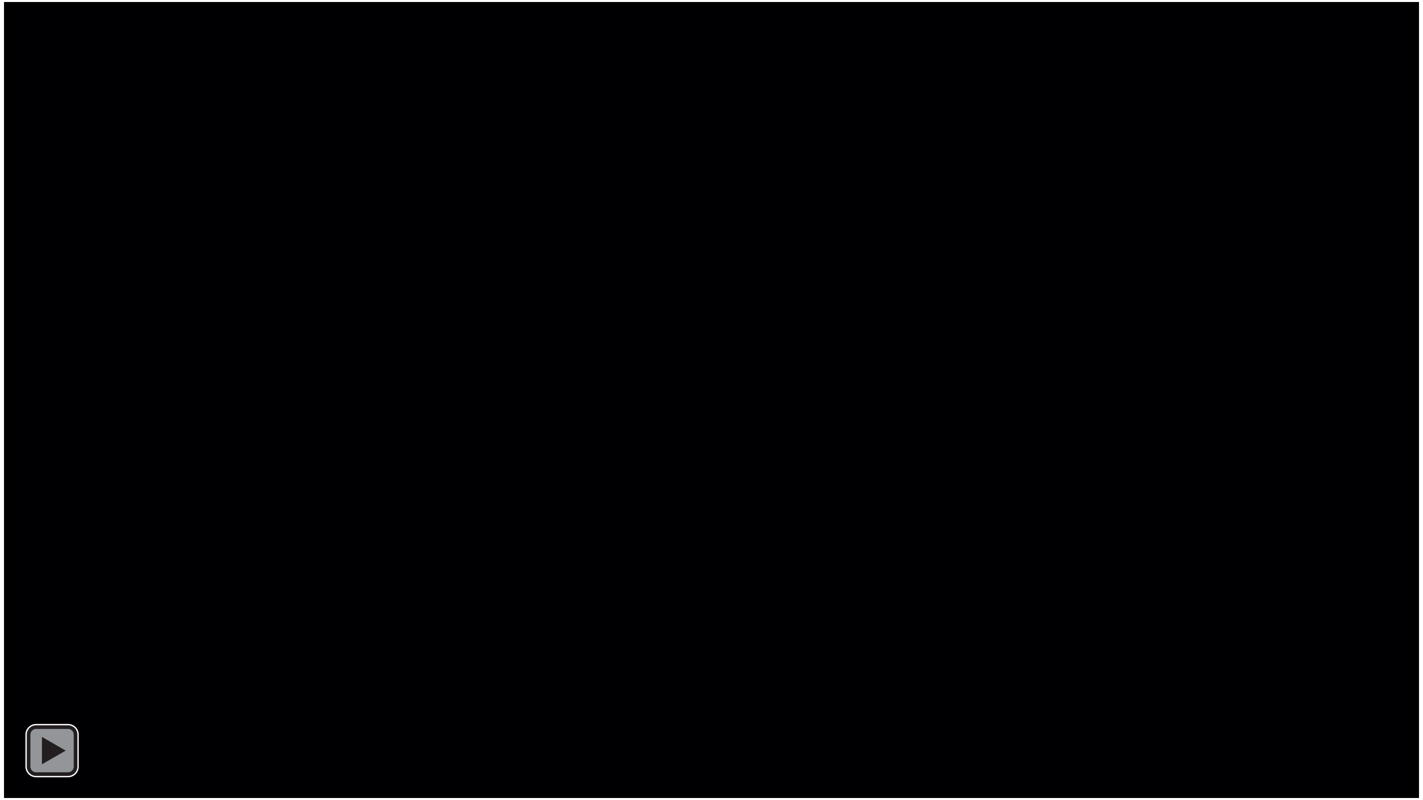


SPEC INNOVATIONS

GPT Integration w/ Requirement Management

- GPT allows to take diverse inputs and provide additional suggestions
  - Great assistance tool for requirement management
- User shall prompt chat box that will provide suggested outputs for their response
  - Provides quick and efficient feedback
- Capability is useful in brainstorming, refining, clarifying, and correlating requirements
- Lead to improved outcomes and efficient development processes

GPT Integration Use Case for Requirement Management



Innoslate's GPT Integration Video

Outcome of AI-Enhanced Requirement Management

- Enhance analysis and feedback on requirements through AI
- More robust requirements traceability matrix through ML
- Streamlined requirements creation through generative input
- Overall increased efficiency and effectiveness of management

Significance of AI Integration

- Seamlessly integrated AI improves user experience
- Comprehensive guidance empowers users to optimize requirements
- AI- enhance management represents major advancement for requirement management

Mitigating Risks w/ AI

- AI is new technology that requires further evaluation for its trustworthiness and user interface
- SPEC Innovation has been researching and experimenting with AI to:
  - Ensure NLP algorithms properly trained on diverse quality data sets
  - Validate ML models to prevent incorrect or missed correlations
  - Use heuristics judiciously to augment not replace human judgment
  - Monitor generative output for misinformation, bias, overreliance

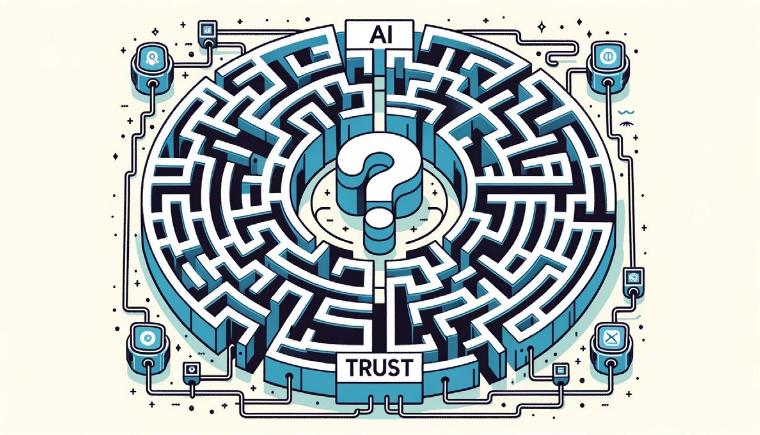


Image generated using Dall·E 3

Conclusion

- AI techniques can transform requirement management
- Meaningful integration provides contextual decision support
- Critical to mitigate risks when applying AI to requirement processes
- AI-enhanced management enables for great project outcomes

Q&A