Overview and Introduction:

Thank you so much for participating in this experiment!

This experiment is built around a *notional* DoD acquisitions program. It is not intended to represent an actual program, but should provide enough material to permit evaluation of requirement verification techniques. The performance data within is entirely fictional, with little to no analysis completed to determine whether or not the performance is actually feasible.

Please review it with this context.

As part of the review, you have been provided with documents and models to support the analysis, as well as a review tool to present the model data and capture your review data. This review tool will be the only method for officially documenting your review. It will start you on a random requirement and enable you to select whether you consider the requirement 'met' or 'not met.' You will be required to annotate how long you spent reviewing the requirement, as well as the errors or inconsistencies you found during your review of the requirement.

Some requirements will provide you assistance in your review by presenting data collected from models. You will *always* have the ability to review documents to help you make your decisions.

Some additional notes:

- Use this documents, data and the tool *exclusively* to conduct your review. This example exists nowhere outside of this experiment, so searching won't help.
- To ensure controlled test execution, make your best effort with the data provided… the test team will not be able to answer questions about the test.
- Do not communicate with others regarding the test.
- Please document any errors you believe would affect your review of the requirement. This data will be evaluated *even if the review was correct*. For example, if you find a single glaring error that caused you to determine that the requirement was not met, *keep looking.* There may be other errors. In real reviews, this is extremely important, because documenting this information may be the difference between a sustained protest and a successful award.
- Keep track of the time you spend on the review. Each requirement will be evaluated statistically to determine if a particular approach was faster, so accurate information is vital.

Part 1: What is being tested?

1. Background

A 2013 Study by the Project Management Institute found that "On average, two in five projects do not meet their original goals and business intent, and one-half of those unsuccessful projects are related to ineffective communications." Similarly, at the 2017 Australian Systems Engineering Workshop, a panel of industry experts determined that "the root cause of all project failures is a lack of shared understanding between the acquirer and the supplier." (Hallet 2018) Perhaps the most important concepts to understand are the user's needs (as specified in requirements) and the system's ability to meet those needs (as shown through attributes of the system implementation).

Currently, most projects attempt to achieve common understanding by creating and sharing a series of documents including reports, checklists, requirements databases and verification matrices. Stakeholders review these documents throughout the program lifecycle to independently evaluate progress towards program goals. Multiple researchers, including Madni (2018), Friedenthal (2014), and Bernard (2012) have found that document based approaches increase program risk due inconsistencies in language, differences in assumptions and semantics, and reliance on inconsistent, out-of-date, disconnected documentation. These approaches often lack technical rigor and standardization, resulting in incompatibilities across organizational lines. As a result, it becomes infeasible to check for correctness or ensure unambiguous statement of needs. (Madni 2018) To ensure successful program execution, engineering methods capitalizing on modeling and machine assistance have been developed and are beginning to be adopted.

This experiment aims to test model based approaches to utilize machine assistance to verify that system requirements have been met. You will evaluate requirements for a notional system utilizing four different approaches, ranging from purely document based to utilizing an approach that integrates requirements and systems models. The integrated approach is called Machine Assisted Requirements Verification for Stakeholders (MARVS), which is being developed as part of doctoral research at George Washington University. The remaining two approaches balance a model based approach (for either requirements or implementation) with a legacy document based approach.

2. Requirements Model

Requirements modeling is focused on writing requirements in formal language and connecting them together in a manner that explicitly defines their relationships. Often, this takes the form of a mathematical relationship that relates an attribute of the system to a required value, as in the following example:

System::Weight < 50 pounds

Similarly, requirements can reference other requirements. For example:

R1: System::Weight < 50 pounds
R2: System::Cost < $10,000
R3: R1 and R2

In this example, the system must weigh less than 50 pounds AND must also cost less than $10,000. This type of relationship can be expanded using boolean expressions to express all system requirements, enabling rapid evaluation of the requirement model.

3. Systems Model

The systems model describe the components of a systems, their attributes, and the explicit links between components. In addition, system models may have the ability to link requirements to components. SysML, a common systems modeling language, utilizes the ‘satisfy’ relationship to connect components to requirements, implying that the linked component has attributes that satisfy the originating requirement. While promising, current versions of SysML only support textual requirement rather than mathematically rigorous language. Discrete expressions of systems models are called implementations. For example:

Model:	System::Weight : pounds
Implementation A:	System::Weight : 30 pounds
Implementation B:	System::Weight : 60 pounds

In this example, the system model identifies an attribute belonging to the “System” component called “Weight” that is expressed in “pounds.” To be conformant to the model, any potential implementation would have a System::Weight attribute expressed in the same data type. The example shows two implementations of the same system model, one that weighs 30 pounds and one that weighs 60 pounds.

4. Integrated Requirements – Systems Model (IRSM)

Both the requirements model and the systems model provide important information about the system. However, neither model by itself provides enough information to assert that an implementation of the system meets its requirements. Combining the two models into a single integrated model allows for traceability from a system requirement model all the way to the lowest relevant system attribute. Linking the requirements and systems models is accomplished by ensuring that (1) the requirements are expressed in formal language using system attributes and (2) the system attributes expressed in the requirements are present in the system model.

Another important benefit of integrating the two models is that any parameters required for validation are required to be present in the model implementation, ensuring accurate and complete documentation. In the MARVS approach documented in this research, this capability is used as an integral part of the requirement verification process. As part of the verification artifacts for each requirement, a full implementation is identified that satisfies the requirement. This implementation corresponds with requirement verification event and includes all system parameters that were expressed at the time of the event. In the above examples, Implementation A could represent the flight testing implementation, meaning the Implementation A’s System::Weight attribute would be the ‘as tested’ flight test weight. Implementation B could represent a sensor test, with an associate ‘as tested’ weight.

The MARVS approach has the entire requirement tree evaluated for each requirement by default, rather than only the attributes associated with the single requirement being verified. For the above example, the System::Weight would have caused the requirement tree to fail during the sensor test. This does not mean that the sensor requirement was not met; instead, it informs the user that aspects of the system were out of specification during the test event. The reviewer would be responsible for determining whether this information results in a ‘not-met’ status for the requirement. This approach is intended to be conservative, ensuring that the reviewer must make a strong determination that the requirement was not relevant rather than having that decision made by default.

If done manually, this approach would likely add a significant burden to reviewing requirements. The MARVS approach is designed to utilize machine assistance to automate as much as possible this effort, while still giving the reviewer the last say. One goal of this experiment is to determine whether machine assistance makes this approach feasible.

Part 2: Test Scenario

The system provided for this experiment is designed to meet the requirements of the Organic Tactical Surveillance System – Expeditionary (OTSS-E) program. This notional program is to procure a small unmanned aircraft system (SUAS) for tactical operations in austere deployed environments. OTSS-E will provide imagery capability for small units, including full motion video (FMV) and wide area imagery. The system will be rapidly re-deployable, with a small lightweight ground control station (GCS). For more detailed information, please see the Systems Requirements Document (SRD) found in the same folder as these instructions.

The OTSS-E program completed an initial market study and concluded that no Commercial Off The Shelf (COTS) solution existed that completely satisfied the requirements. However, multiple companies were identified that had solutions deemed ideal for 'rapid tailoring' to the military requirements. The OTSS-E program was structured around a two phase approach, with 5 contractors selected for a Phase 1 effort to tailor their COTS solutions under USAF contract followed by a source selection for the Phase 2 production effort. As part of this effort, bidders were encourages to leverage previous analysis, data, testing and certifications to reduce cost and risk.

This experiment is set after the completion of Phase 1. You will review the submission by XYZ Aerospace, one of the 5 competitors. The award is based on a Lowest Price Technically Acceptable (LPTA) evaluation. You are a member of the technical team determining whether the offering is technically acceptable (meets all requirements). The Source Selection Authority has determined that you will not see any price or contracting aspects of the proposal to ensure that your review will not be biased. In addition, because the proposal will either be considered acceptable or not, not objective values have been identified for requirements.

XYZ Aerospace has chosen to submit their Nomad SUAS, a quadcopter design based on their commercially successful Wanderer SUAS. Wanderer has been used in multiple industries, including mining, private security, and film. However, it does not meet military requirements for sensor accuracy, communications security, and degraded GPS performance (among others). In addition, multiple Wanderer components are produced in prohibited countries. In Phase 2, XYZ chose to leverage Wanderer flight data and FAA certification to dramatically reduce flight testing. Consequently, Phase 2 was primarily focused on integrating new sensors and software, along with implementing a domestically produced flight computer.

Additional detail can be found in the documents provided with this instruction.

Part 3: Using the Tool

1. Tool Overview

Your review will be documented using the prototype MARVS reviewer tool, which will present the requirements to you and enable you to input your review data. You can execute the reviewer tool by opening a program called MARVS.exe located in the same directory as this instruction document. It has been compiled and tested for Windows 10. If you need a version for another OS, please let the test team know.

If you are reading this, then you have already retrieved the files from GitHub. The directory structure is as follows:

- **Documents**
- **readOnlyFiles**
- MARVS.exe
- System Requirements Document.doc
- userInstructions.doc
- user_xxx.dbc

The Documents folder contains all the documents that you will use to review Nomad. The readOnlyFiles folder contains the internal files needed for the MARVS program, and should not be edited or removed. If any files in the readOnlyFiles directory are modified, please re-download them from the repository. MARVS.exe is the review program. The Systems Requirement Document is the notional SRD provided by the OTSS-E program office; it contains the textual requirements that Nomad was built towards. User_xxx.dbc is the output file containing our review data. At the completion of your review, this file will be emailed to the test team.

2. Tool Walkthrough

The prototype MARVS tool presents the reviewer with a single interface for reviewing requirements as shown in Figure 1. When the program is run, it will automatically load all necessary data and configure itself for review. You will be presented with requirements in random order, one at a time. Once you submit a requirement, you will not be able to go back and edit that requirement again. Each time MARVS is run, it will present you the first unsubmitted requirement remaining.

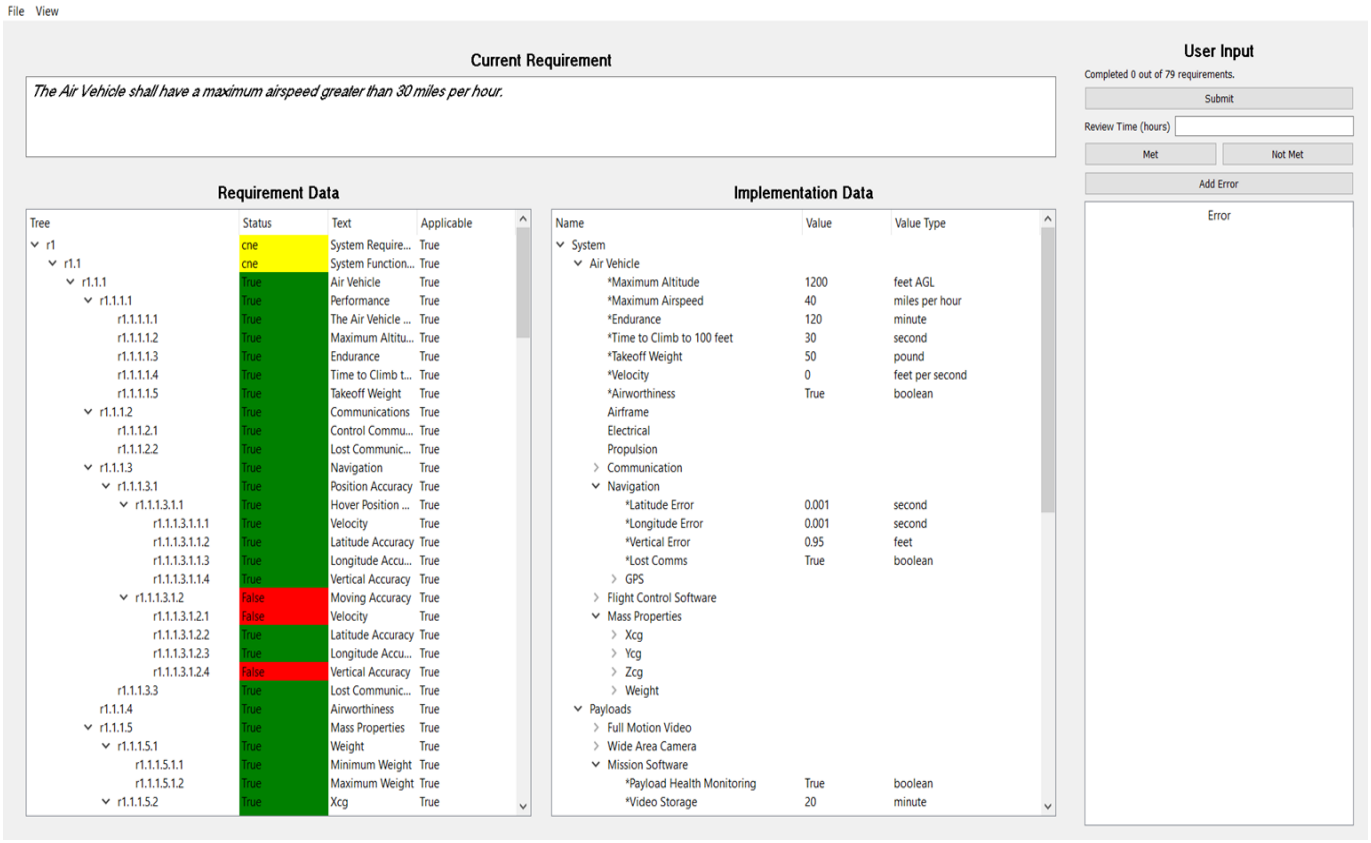

Figure 1: MARVS Review Tool Interface

The current requirement is displayed in textual form at the top of the interface, as shown in Figure 2.

Figure 2: Current Requirement Text

The user input is provided at the right of the user interface, as shown in Figure 3. The Submit button saves the requirement, and automatically moves to the next requirement. You are also presented your current status of the review.

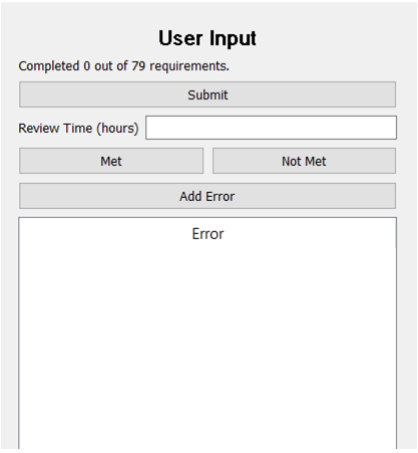

Figure 3: User Input Pane

Prior to submission, you must select either ‘Met’ or ‘Not Met’ for the requirement. A ‘Met’ requirement asserts that the vendor has met the requirement sufficiently. You will also input the number of hours you spent reviewing this requirement. You may input decimal hours if needed.

In addition to evaluating the requirement, you will annotate errors that you identify during the review. Each ‘Not Met’ should have at least one error showing what caused you to determine that the vendor did not meet the requirement. You can also add additional errors corresponding to concerns you identified during the review. For example, when reviewing determine that the system did not meet a basic requirement for speed, which would be an error. You may also notice that during the speed testing, the system weight was outside of design limits. This could be a second error. You may identify as many errors as you like. You may also have errors that exist in ‘Met’ requirements, but do not cause you to evaluate the requirement as unmet. Please keep each input to a single error, rather than bundling errors. This makes it much easier for the team to review. When you click the add error button, you will have the opportunity to name the error and write a short description of the error. If you need to delete the error, you can simply right click on it in the error list and select delete.

3. Requirements Pane

The Requirements Pane shows the requirements, and is shown on the left hand side of the interface. TIf you are evaluating a requirement that has a requirements model, the model hierarchy will be shown within the pane (see Figure 4), enabling you to see how the implementation used to verify the selected requirement meets the overall requirement model.

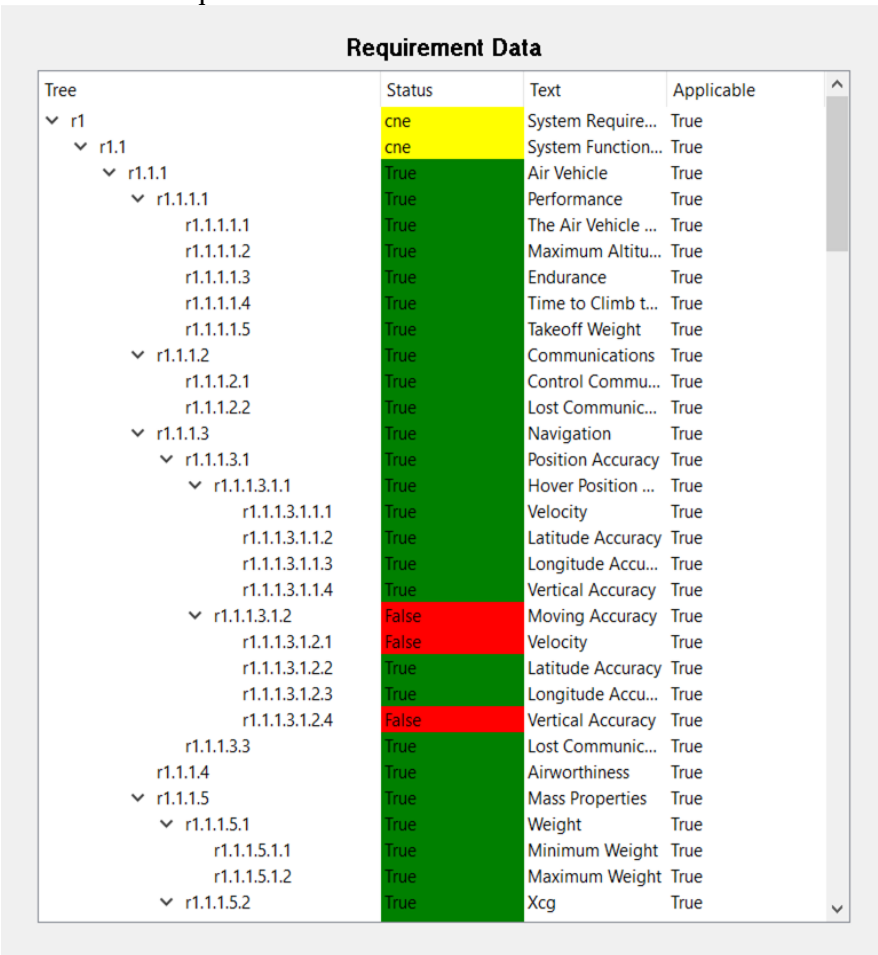

Figure 4: Requirements Pane

This tree will show the requirement number, as well as the requirement text. If implementation data has been provided, the status will reflect the mathematical evaluation of the requirement model in regards to the associated implementation data.

Table 1: Requirements Status Type

Status	
True	Implementation satisfies the equation from the requirements model.
False	Implementation does not satisfy the equation from the requirements model.
cne	The requirement cannot be evaluated. This may be due to many factors, including missing data

If you believe that the requirement in the hierarchy does not apply to the requirement you are currently reviewing, you have the ability to tell the interface that it does not apply. To do this, right click and select 'Does Not Apply.' This will remove that requirement (and all its children) from evaluation in the requirement model.

If the requirement you are reviewing does not have a requirement model, the requirement hierarchy will not appear. An example of this will be shown in a later section.

4. Types of Review

You will be presented with four different types of review, with the potential to be presented requirement data and implementation data via documents or models. This section will walk through each method, and how the interface changes.

A. Model Based Requirements and Implementations

When both requirements and implementations are documented via a model, the MARVS tool will automatically pull data from both sources and evaluate the requirement tree. Figure 5 shows an example of this. You will use this information to determine whether the current requirement is met or not, and to identify any errors related to the requirement.

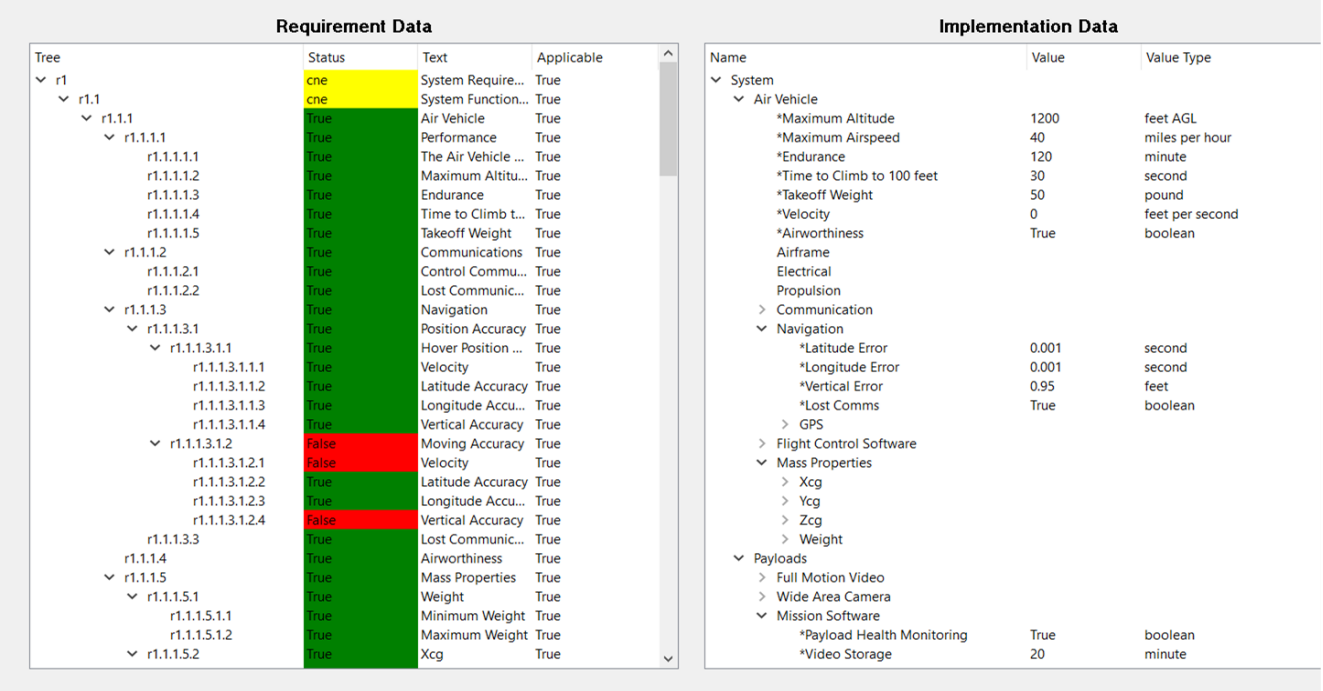

Figure 5: Model - Model View

B. Model Based Requirement with Document Based Implementation

When the requirement model is provided, but no model based implementation data is available, the tool will still automatically evaluate the requirement tree. However, it will be up to you to look through the documents and find the implementation data. The Implementation Data pane enables you to input values by right-clicking on the desired attribute. You will be presented with a dialog to input the data, and then the tool will evaluate the requirement model with the updated implementation data. Note that any requirement that uses data missing from the model will have a status of ‘cne.’

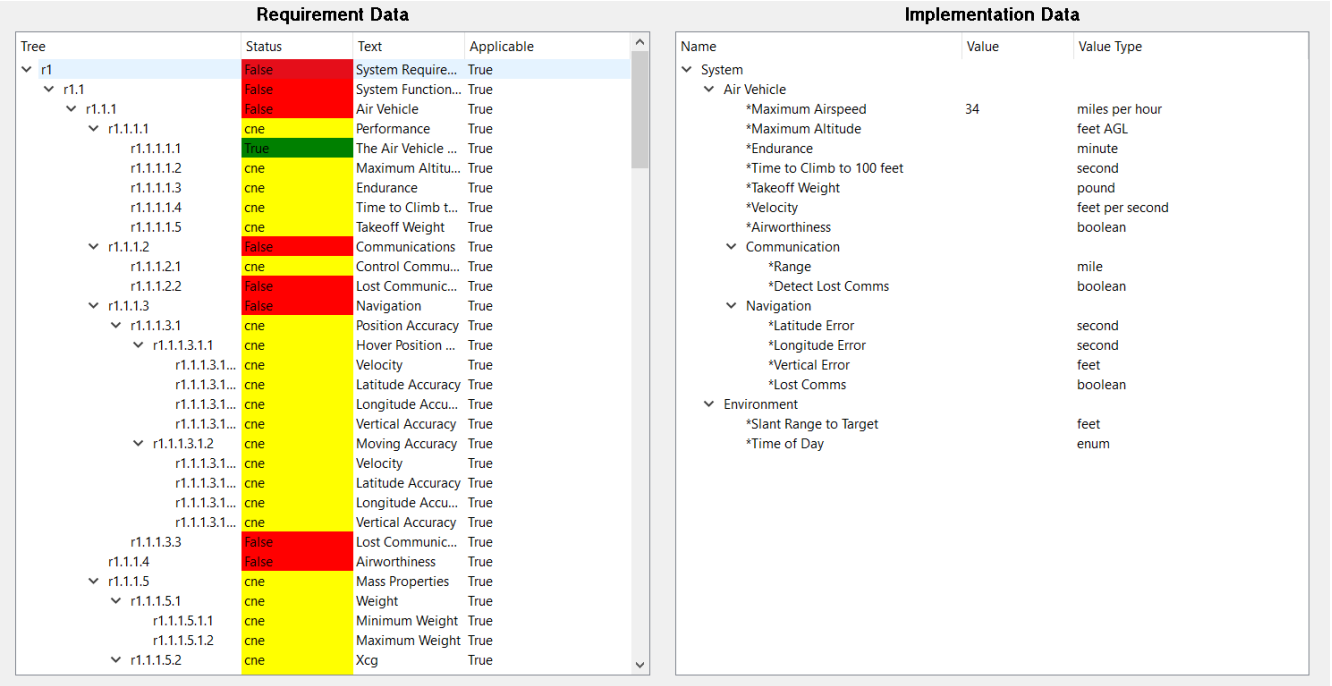

Figure 6: Model - Document View

C. Document Based Requirements with Model Based Implementation

When you are presented with a requirement that does not have a requirement model, but *does* have an implementation model, you will see an interface similar to Figure 7. In this case, no data will be shown in the Requirement Data pane. The Implementation Data pane will show the attributes of the system component linked to the requirement. This replicates model behavior similar to the SysML 'Satisfy' relationship. In SysML, a Requirement Block has requirement language specified via text and may be linked to a system component that is asserted to have properties that satisfy the requirement. For MARVS, the interface uses this relationship to find the component within the implementation model and present this data to the user. The component will have data that is relevant to the requirement, but there is no guarantee that it will provide all data necessary to determine whether all implicit or explicit assumptions of the requirement are met. The Implementation Data pane may also show component attributes that are completely unrelated to the requirement being evaluated.

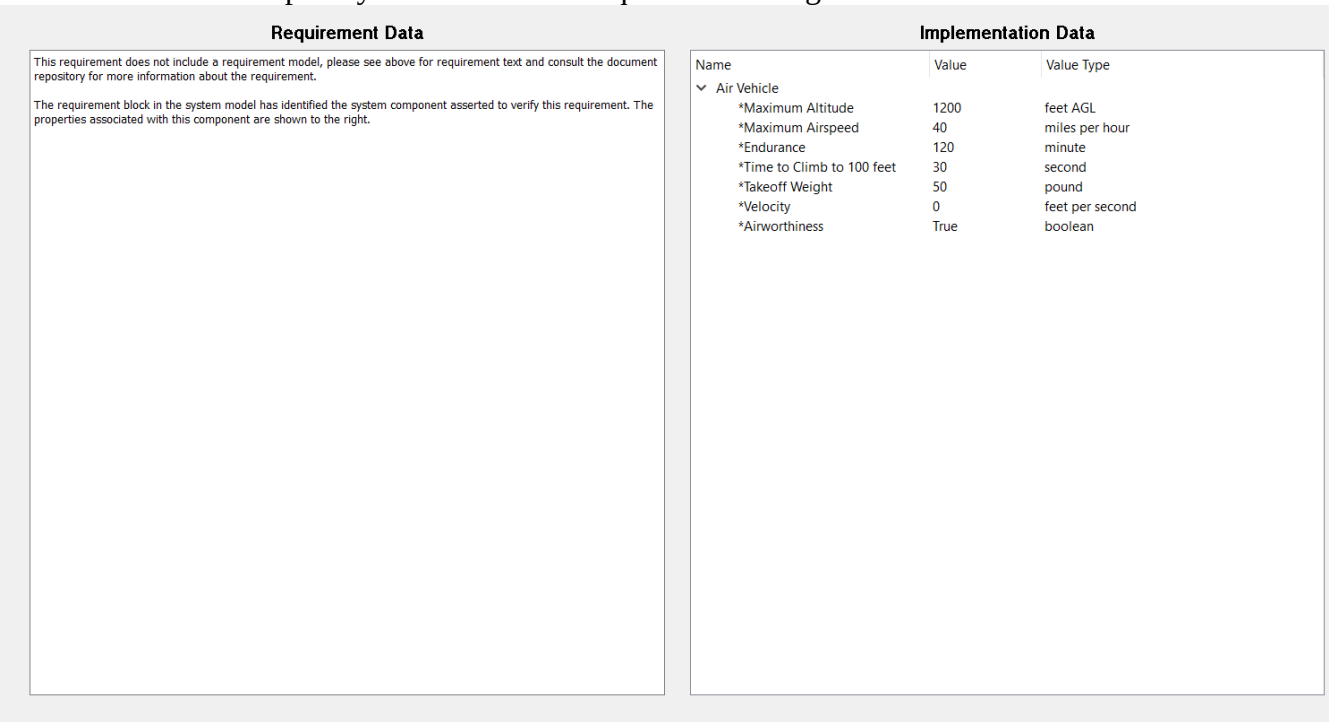

Figure 7: Document - Model View

D. Purely Document Based

When no models are available, the interface provides no additional information. In this case, you must utilize the provided documents to determine whether the requirement was met or not.

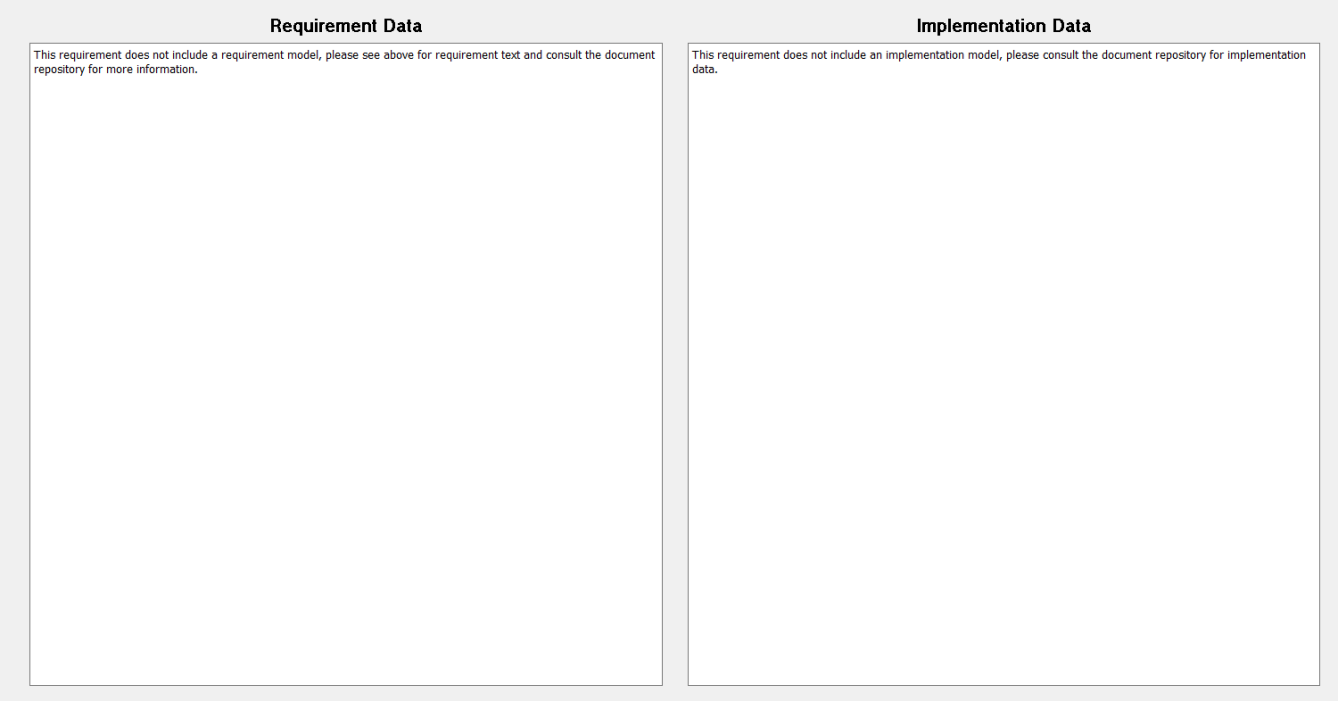

Figure 8: Document - Document View