System Test Plan

For

Vetronica (Option 9)

Team members: Michael Hall, Troy Neubauer, Zachary Shepley, Dana Merry

Version/Author	Date
1.0/ All	10/18/23
2.0/ All	12/06/23

Table of Contents

1. Introduction

1.1 Purpose

This document is a test plan for Vetronica System Testing, produced by the System Testing team. It describes the testing strategy and approach to testing the team will use to verify that the application meets the established requirements of the business prior to release.

1.2 Objectives

- *Meets the requirements, specifications and the Business rules.*
- *Supports the intended business functions and achieves the required standards.*
- *Satisfies the Entrance Criteria for User Acceptance Testing.*

2. Functional Scope

The Modules in the scope of testing for the Vetronica System Testing are mentioned in the document attached in the following path :

1. The System Requirements Specification document:
   a. https://docs.google.com/document/d/16Nbp9DAgYOBzdLRuxRSvK0EP9S-yN43x/edit
2. Section 3.1 in the document.

3. Overall Strategy and Approach

3.1 Testing Strategy

Vetronica will divide its efforts between two different modeling software: Capella and MagicDraw. These applications will be used to provide a tangible and viewable fulfillment of the determined system requirements. Using these applications, the team will recreate physical infrastructure for an aerospace environment in a virtualized format. From there the system requirements can be recreated and tested in the two application environments.

3.2 System Testing Entrance Criteria

In order to start system testing, certain requirements must be met for testing readiness. The readiness can be classified into:

3.3 Testing Types

3.3.1 Usability Testing

User interface attributes, cosmetic presentation and content will be tested for accuracy and general usability. The goal of Usability Testing is to ensure that the User Interface is comfortable to use and provides the user with consistent and appropriate access and navigation through the functions of the application (e.g., access keys, consistent tab order, readable fonts etc.)

1. The system shall provide the user appropriate instructions and information on how to use the software in the GUI.
2. The system shall take in an .MDXML file or Capella file created from the respective MBsE software as input.
3. The system shall present the test results in an easy-to-read format that allows the user to read the information.

3.3.2 Functional Testing

The objective of this test is to ensure that each element of the component meets the functional requirements of the business as outlined in the:

- Business / Functional Requirements
- Business rules or conditions
- Other functional documents produced during the course of the project i.e. resolution to issues/change requests/feedback

Readable UI regarding the software. Requirements following from the SRS.

Additionally, functionality of inner units of software will be tested atomically, outside of larger integration tests. This will allow us to ensure inner implementation details work before integrating pieces together.

3.3.3 Documentation Testing

Accurate and complete documentation is essential to a good software product. Rust provides rustdoc, a doxygen-like documentation system built into the compiler and ecosystem. Documentation comments are compiled into a html or pdf by the *cargo doc* command with first class support for markdown. An example for a popular argument parsing crate, clap, is available here: https://docs.rs/clap/latest/clap/_derive/.

cargo doc will be ran on each commit as part of our CI/CD pipeline on github actions, and the generated html site will automatically be uploaded to github pages for viewing. Also the *cargo doc* command will fail the CI job if invalid markdown or broken links are detected, allowing us to achieve rapid velocity in delivering high-quality documentation.

3.4 Suspension Criteria and Resumption Requirements

This section will specify the criteria that will be used to suspend all or a portion of the testing activities on the items associated with this test plan.

3.4.1 Suspension Criteria

Testing will be suspended if the incidents found will not allow further testing of the system/application under-test. If testing is halted, and changes are made to the hardware, software or database, it is up to the Testing Manager to determine whether the test plan will be re-executed or part of the plan will be re-executed.

1. Unable to acquire a MagicDraw license, testing will need to be suspended.
2. If the input project files from MagicDraw / Capella are corrupted or invalid, testing will be suspended. It is up to the user to provide correct model files.

3.4.2 Resumption Requirements

Resumption of testing will be possible when the functionality that caused the suspension of testing has been retested successfully.

1. Once the MagicDraw license has been acquired, testing within that software shall be resumed.
2. Likewise, testing will resume when correct model files are provided.

i.

4. Execution Plan

4.1 Execution Plan

The execution plan will detail the test cases to be executed. The Execution plan will be put together to ensure that all the requirements are covered. The execution plan will be designed to accommodate some changes if necessary, if testing is incomplete on any day. All the test cases of the projects under test in this release are arranged in a logical order depending upon their inter dependency.

Additionally, unit tests will be functions marked with Rust's *#[test]* attribute, allowing us to run and verify all unit tests via a single command: *cargo test*. *cargo test* will return a non-zero exit code on failure, allowing us to easily test all commits via CI in Github Actions on every commit.

The test plan for the Vetronica system is as follows:

Requirement (From SRS)	Test Case ID	Input	Expected Behavior	Pass/Fail
4.1.3.1: The software shall allow the user to import MagicDraw model files	1	Valid MagicDraw save of catapult project	Software opens successfully with file loaded	
4.1.3.2: The software shall allow the user to import Capella model files	2	Valid Capella save of catapult project	Software opens successfully with file loaded	
4.1.3.3: The software shall alert the user if features in a particular save are not supported by the tool (positive)	3	File with unsupported features	Software fails to open file	
4.1.3.3: The software shall alert the user if features in a particular save are not supported by the too (negative)	4	Normal file of catapult model	Software opens file successfully	
4.1.3.4: The software shall alert the user if a model is corrupted or cannot be loaded	5	Non-MagicDraw file	Software fails to open file	
4.1.3.5: The software shall display when the model is imported successfully	6	Normal file of catapult model	Software opens to edit screen	
4.2.3.1: The software shall import networks of concern from the user	7	.CSV file	Networks of concern are imported into the software.	
4.2.3.2: The software shall import a prioritized list of threats of concern from the user	8	.CSV file	Threats of concern are imported into the software.	

4.4.3.2: The software shall run a simulation by executing each programming element in order	9	Sample program that sums integers 0 to 10	Sum shown is 55	
4.4.3.3: The software shall respect branches diverging control flow based on standard comparison operators	10	Branch on 0 == 0	Branch taken	
4.4.3.4: The software shall support loops jumping back to a programming element based on conditional operators	11	Sample program that sums integers 0 to 10	Sum shown is 55	
4.4.3.5: The software shall delay execution for a constant or dynamically determined amount of time when the delay programming element is encountered	12	Sample program that sums integers 0 to 10 with one second delay on each iteration	Execution takes about 10 seconds	
4.4.3.7: The software shall assign to the global key value pair when encountering a store programming element	13	Sample program stores to a variable, and later prints variable	Value stored to variable is the same as printed value	
4.4.3.8: The software shall modify number values in the global key pair when encountering a math operation such as addition, subtraction, multiplication, or division acting on two numbers in the global key value pairs	14	Test a <OP> b for all a in [0, 10], b in [0, 10] where OP is +, -, *, /	Ensure result is correct from addition/subtraction/multiplica tion/division table	
4.5.3.2: The software shall store a list of paths to registered plugins	15	Open plugin in software. Software opened to file -> plugins menu	Recently used plugins displayed	
4.5.3.3: On startup the software shall attempt to load each registered plugin from the list	16	Freshly installed software is opened. Plugin is loaded. Software is closed. Software is opened	Software loads saved plugin	
4.5.3.4: Each plugin will export a function named plugin_init which takes a pointer to a struct containing functions that can be called by the plugin to interact with the core software	17	Example no-op plugin	Assert plugin_init functions is exported and callable via C abi	
4.5.3.5: The plugin will export a functio called plugin_version that returns an int containing the plugin's version and required version of the software	18	Example no-op plugin	Assert plugin_version functions is exported and callable via C abi	
4.5.3.6: The software shall reject plugins which return an error code from plugin_init or require a future version of the software	19	Malformed plugin with future version	Plugin is rejected by software and not loaded	

5. Traceability Matrix & Defect Tracking

5.1 Traceability Matrix

List of requirement, corresponding test cases

Requirement CRITICAL: SRS 4.1.3.1 The software shall allow the user to import MagicDraw model files.

Test Cases: Check that the imported file is of .MDXML format.

Requirement CRITICAL: SRS 4.1.3.2 The software shall allow the user to import Capella model files.

Test Cases: Check that the imported file is of the appropriate Capella file format (format TBD).

Requirement CRITICAL: SRS 4.1.3.3 The software shall alert the user if features in a particular save are not supported by the tool.

Test Cases: Load a file with features that aren't supported and confirm that the software reads this and alerts the user.

Requirement CRITICAL: SRS 4.1.3.4 The software shall alert the user if a model is corrupted or cannot be loaded.

Test Cases: Load a corrupted model file and determine if the software reads and alerts the user.

Requirement MEDIUM: SRS 4.1.3.5 The software shall display when the model is imported successfully.

Test Cases: The software shall read the file and if completed with no error, display the notification.

5.2 Defect Severity Definitions

Critical	The defect causes a catastrophic or severe error that results in major problems and the functionality rendered is unavailable to the user. A manual procedure cannot be either implemented or a high effort is required to remedy the defect. Examples of a critical defect are as follows: ● System abends ● Data cannot flow through a business function/lifecycle ● Data is corrupted or cannot post to the database
Medium	The defect does not seriously impair system function can be categorized as a medium Defect. A manual procedure requiring medium effort can be implemented to remedy the defect. Examples of a medium defect are as follows: ● Form navigation is incorrect ● Field labels are not consistent with global terminology
Low	The defect is cosmetic or has little to no impact on system functionality. A manual procedure requiring low effort can be implemented to remedy the defect. Examples of a low defect are as follows: ● Repositioning of fields on screens ● Text font on reports is incorrect

6. Environment

6.1 Environment

- The System Testing Environment will be used for System Testing.
- There will be two MBsE tools utilized for System Testing:
  - MagicDraw Version 2022x R1 HF1 and up
  - Capella 6.1.0 and up
- The (TBD) analysis program
  - rustup with stable toolchain v1.73

7. Assumptions

- There will be no physical infrastructure on which to test the solution
- All testing and modeling will be performed via one of two Model-Based Systems Engineering tools:
  - MagicDraw
  - Capella
- The project requirements and their implementation into the provided modeling software will be tested as opposed to the modeling software itself

8. Risks and Contingencies

Define risks and contingencies.

- Modeling is contingent on receiving licensing for MagicDraw
- Balancing time investment between performing analysis of MagicDraw and Capella to obtain requirements (research) and developing analysis tool for MagicDraw and Capella projects (software implementation).

9. Appendices

Magic Draw User's guide: https://www.magicdraw.com/files/manuals/MagicDraw%20UserManual.pdf