Project Title: System Verification and Validation Plan for MCT: A Command Scheduling Application for Mission Operation and Control (MOC) of the McMaster PRESET CubeSat

April 3, 2024

Revision History

Date	Version	Notes
November 3, 2023	1.0	Completed VnV Plan - Q.H, R.V, D.A, D.C, U.R
November 3, 2023	1.0	Updated NFRs - Q.H, R.V, D.A, D.C, U.R
March 3, 2024	1.0.1	Updated Elevation FR and precision NFR - Q.H
March 27, 2024	1.0.2	Updated Document based off feedback - Q.H
April 3, 2024	1.0.3	Added Unit Test Summary - U.R.

Contents

1 Symbols, Abbreviations, and Acronyms

Please refer to the SRS document, section **5 Naming Conventions and Terminologies** for symbols, abbreviation, and acronyms used throughout the document.

This document outlines the verification and validation objectives needed to ensure that the MCT application aligns with its software requirements specifications. The plan detailed in this document acts as a guide to ensure the team produces a verified and validated software solution that meets its requirements.

Roadmap

Testing Strategy	**Date**
Configure ESLint for code base	November 12, 2023
Configure Istanbul code coverag-ing tool	November 12, 2023
Code walkthrough 1	November 19, 2023
Formal Code review 1	November 30, 2023
Configure automated jest testing	January 10, 2024
Code walkthrough 2	January 12, 2024
Formal Code review 2	January 21, 2024

2 General Information

2.1 Summary

The software that is being tested is the MCT web application which is a platform to automatically schedule and send commands to satellites as they pass overhead. The application will be used by authorized operators and administrators. Operators will be registered and given access privileges by an administrator through the GUI of the MCT application. Furthermore, the system will allow operators to schedule both manual and automated command sequences, as well as retain logs of all commands sent and their responses. The GUI of the application provides a means to create, edit, delete, and view scheduled commands and their responses.

2.2 Objectives

In the scope of this project, several critical components demand meticulous attention to ensure the system's effectiveness and reliability. Foremost among these is the graphical user interface (GUI), serving as the central point of interaction between users and the satellite system. The design and functionality of this GUI play a pivotal role, as they directly influence the user experience and the efficiency of data exchange. Ensuring an intuitive and user-friendly interface is paramount, enabling users to seamlessly send and receive essential data from the satellite. A well-designed GUI not only enhances user satisfaction but also contributes significantly to the system's overall usability and accessibility.

Equally vital are the mechanisms of user validation and command verification. Robust protocols for user authentication are imperative to confirm the identity and authorization of individuals accessing the system. Additionally, thorough validation procedures for commands are crucial to prevent any erroneous or malicious instructions from reaching the satellite. These validation processes serve as the first line of defense, ensuring that only legitimate and accurate commands are executed. By implementing stringent validation measures, the project can safeguard the satellite's operations, data integrity, and overall security, thereby instilling confidence in the system's users.

Furthermore, the seamless integration and usability of databases for storing and managing logs represent another cornerstone of the project. Efficient database management is essential for quick and accurate retrieval of data, enabling real-time monitoring and analysis. Properly organized databases facilitate the tracking of system performance, identification of patterns, and swift resolution of potential issues. A well-structured data management system not only enhances the project's overall functionality but also streamlines troubleshooting processes, contributing to the project's long-term sustainability and success.

While validating all aspects of the project is ideal, the reality of limited time and resources poses a challenge. Consequently, strategic prioritization becomes essential. However, it is imperative to acknowledge that certain aspects, such as the libraries used for satellite orbit tracking, are assumed to provide accurate results without exhaustive validation.

2.3 Relevant Documentation

Listed below are some pieces of documentation which may be useful for the reader to gain more context about the system.

1. Project Proposal: This document gives context on the purpose of the application, as well as various stakeholder requirements and constraints.

2. Development Plan: This document provides information on testing and validation tools and technologies that will be implemented through this VnV plan.

3. Software Requirements Specification: This document gives context on the functional and non-functional requirements for which this document will outline tests for.

3 Plan

3.1 Verification and Validation Team

Name and Role	Responsibilites
Buu Ha (Quinn) - Code verification	• Responsible for ensuring that the code adheres to standards and guidelines set for the project.
Rishi Vaya - SRS verification	• Ensures that the MCT application has addressed the functional and non-functional requirements listed in the SRS document.
Diamond Ahuja - Security Verification	• Responsible for ensuring that security measures, including authentication, role-based authorization, and data encryption are implemented correctly.
Dhruv Cheemakurti - Performance Verification	• Ensures the application's load times, response times, and resource utilization meet performance standards.
Umang Rajkarnikar - Data Verification	• Responsible for ensuring the validation and verification of application data. This includes both user and satellite data.

3.2 SRS Verification Plan

In order to perform verification on the SRS, a formal inspection will be used to evaluate the MCT application's implementation of the SRS. These reviews will be conducted by both team members and the Neudose team where both parties can provide valuable feedback based on their knowledge of the system. In these reviews, members will be provided a checklist to ensure that specific aspects of the SRS document are met. This includes that functional requirements, non-functional requirements, assumptions, and use cases are covered. This checklist can be found through the list of requirements and more in the project proposal document.

3.3 Design Verification Plan

The goal of this plan is to verify that the design of the MCT application meets the intended specifications.

Our stakeholders, in particular the Neudose team will participate in hands-on reviews to verify the application's design. This verification will be in the form of design review meetings and two usability testing meeting. Design reviews will be held every two weeks during regular scheduled meetings, where informal feedback can be given. The usability testing meetings will be conducted with additional Neudose team members (aside from primary stakeholders), for which feedback will be collected.

3.4 Verification and Validation Verification Plan

The verification and validation plan is also a document that needs to be verified for correctness and completeness. In order to verify this Verification and Validation Plan document, peer review will be used to ensure that the system and unit tests for the functional and nonfunctional requirements are complete. To accomplish this, a checklist that outlines the key aspects to evaluate will be used as a means of assessing the plan. Team members will meet on scheduled dates to discuss any issues or improvement opportunities found in the current VnV plan.

The checklist is below:

- Title of the V&V Plan is clear and indicative of the system to be tested.

- Document includes version number, authors, and date of creation.
- Purpose and scope of the V&V activities are clearly defined.
- Document structure is logical and follows an easily navigable format.
- Provides a brief description of the system/software to be tested.
- Outlines the system's key functionalities and performance criteria.
- Identifies the stakeholders involved in the V&V process.
- Lists all team members and their roles in the V&V process.
- Specifies the qualifications or expertise required for team members.
- Outlines the communication plan among team members and stakeholders.
- Includes a comprehensive list of functional requirements with detailed descriptions.
- Outlines methods for verifying each functional requirement (e.g., tests, inspections).
- Includes a comprehensive list of non-functional requirements (performance, usability, reliability, etc.) with detailed descriptions.
- Describes the approach for verifying non-functional requirements.
- Describes the process for validating the system against user needs and requirements.
- Details the types of validation tests (e.g., user acceptance testing) and criteria for success.
- Specifies the environment and tools needed for validation testing.
- Defines the test strategy and objectives.
- Outlines the test environment setup, including hardware and software configurations.

- Lists the test cases with inputs, expected outputs, and criteria for pass/fail.
- Specifies the process for documenting and resolving defects found during testing.

3.5 Implementation Verification Plan

The following points will outline the plan for verifying the implementation:

- The tests are categorized based on their ability to verify the functional correctness and performance characteristics of the software.
- This includes functional test cases to evaluate FR-SLN1, ensuring that the MCT application loads successfully in the web browser when accessed via a URL path.
- The plan encompasses functional test cases for FR-SLN2 to confirm the accessibility of the implemented communication protocol, wherein a success response from the TCP port indicates the success of the interaction with valid Linux commands.
- Detailed information on more functional requirements can be found in Section 4.1 of this document.
- For non-functional requirements like Usability-1 (NFR: 10.1), outlines a manual usability test in which users engage with the fully functional application or system to provide feedback and complete surveys, leading to the compilation of usability scores on a scale from 1 to 10.
- Detailed information on more non-functional requirements can be found in Section 4.2 of this document.
- In addition to testing, the plan included static verification techniques to catch potential issues in the code without executing it.
  - We will conduct code walkthroughs, where team members review the source code line by line to identify potential issues, such as logic errors and improper coding practices.
  - We will employ static code analysis tools and linters (ESLint) to automatically scan the source code for common programming errors

3.6 Automated Testing and Verification Tools

- Unit Testing Framework: Jest
- Code Coverage Measuring Tool: Jest

Jest was chosen for the unit test framework and code coverage tool for their simplicity, ease of use, and support for various features such as mocking, assertions, and running tests in parallel. In addition, Jest is maintained by Meta, with ample documentation and support available.

- ESLint (extension available on VS Code)

ESLint was chosen as a linter for it's reputation amongst developers as an industry standard. In addition, a VS Code (code editor of choice) extension is available for ease of use and integration.

Detailed information about the technology used and coding standards can be found in Section 6 and 7 of the Development Plan.

3.7 Software Validation Plan

A Software Validation Plan is crucial for ensuring that the developed software meets its intended requirements and functions as expected.

- The MCT interface will be validated using real satellite telemetry data obtained from previous satellites. This data will simulate actual command and response scenarios, allowing us to verify the software's ability to handle and process real satellite communication data.
- We have scheduled bi-weekly review sessions with project stakeholders, including NEUDOSE team members. These sessions will involve walking through the interface and its features, taking feedback on the user interface design, and ensuring that the software aligns with their operational requirements.
- After the Rev 0 demo, we will demonstrate the software's core functionalities to the stakeholders such as Dr. Byun and gather feedback regarding its alignment with the project's goals.

4 System Test Description

4.1 Tests for Functional Requirements

Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed.

Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here.

4.1.1 MCT Application Accessibility

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-1

1. FR-SLN1

   Control: Manual

   Initial State: The MCT system is hosted on a linux server.

   Input: URL path of the MCT user interface to be accessed via a web browser.

   Output: MCT application loads successfully in the web browser

   Test Case Derivation: Upon entering the URL of the MCT application on a web browser, the browser should load the user interface of the application.

   How test will be performed: The test can be performed on several common web browsers, including, but not limited to, Chrome, Edge, and Firefox. A user will enter the URL of the hosted MCT application on the web browser.

2. FR-SLN2

   Control: Manual

   Initial State: The MCT system is hosted on a linux server and has an active TCP port to communicate with.

Input: A valid linux command or a series of commands.

Output: A success response from the TCP port, indicating that the ping was successful.

Test Case Derivation: The active TCP port needs to accept linux-based commands from the command line and a success response from the TCP port can be used to evaluate the accessibility of the implemented communication protocol.

How test will be performed: A user can send linux-commands through the GUI of the application after being authenticated and authorized.

4.1.2 Managing User Roles

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-2
- FR-7

1. FR-SLN3

   Control: Manual

   Initial State: The MongoDB database connected to the MCT application consists of one user with an admin role.

   Input: An email adhering to standard email formats, user role, and a password for a user.

   Output: An updated list of users who are authorized to use the MCT application. This list shall include the user added above.

   Test Case Derivation: Upon adding a new user to the system, the changes should be reflected when retrieving the list of users allowed to access the application.

   How test will be performed: Through the MCT application, a user who has an admin role will be able to add another user to the system. The admin can specify the role of this user. Then, the MongoDB database of users will be updated with the information provided by the admin.

2. FR-SLN4

   Control: Manual

   Initial State: The MongoDB database connected to the MCT application consists of one user with an admin role and five users with operator privileges.

   Input:

   (a) Deleting a user: Email of the user to delete.

   (b) Editing a user: Email of the user to edit and role of the user.

   Output:

   (a) Deleting a user: An updated list of users who are authorized to use the MCT application. This list shall not include the user added above.

   (b) Editing a user: An updated list of users who are authorized to use the MCT application. This list shall include the user added above, with their updated information.

   Test Case Derivation: Upon adding a new user to the system, the changes should be reflected when retrieving the list of users allowed to access the application.

   How test will be performed: Through the MCT application, a user who has an admin role will be able to add another user to the system. The admin can specify the role of this user. Then, the MongoDB database of users will be updated with the information provided by the admin.

4.1.3 Scheduling and Executing Commands

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-3
- FR-4
- FR-5

1. FR-SLN5

   Control: Automated

   Initial State: The MCT application's graphical user interface which currently consists of zero automated command sequences.

   Input: An automated sequence of linux-based commands which is sent through the MCT's graphical command line.

   Output: A table which displays the automated command sequences in a queue.

   Test Case Derivation: Upon entering a set of command sequences to be sent to the satellite, the commands which are now in the queue should be displayed to the operator.

   How test will be performed: An operator will enter a set of linux-based commands to be automated through the MCT's GUI, and a set of linux-based commands via the terminal, on a different schedule. These commands sent should then be received by the satellite.

2. FR-SLN6

   Control: Manual

   Initial State: The MCT application's graphical user interface which currently consists of five command sequences in queue.

   Input: Selected command sequences to be executed. Once executed, the command is sent to the satellite.

   Output: A table which shows the updated statuses of all executed commands. This includes commands which were sent to and received by the satellite with their timestamps.

   Test Case Derivation: Upon executing a set of command sequences, the commands should be logged and displayed to the operator. The logs should also have the timestamp and statuses of the executed commands.

   How test will be performed: An operator will select a previously entered sequence of linux-based commands. Then, an operator can execute all of the selected commands which will forward the request to the satellite. This will be repeated for commands submitted through the terminal.

4.1.4 Cancelling Scheduled Commands

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-6

1. FR-SLN7

   Control: Manual

   Initial State: The MCT application's graphical user interface which currently consists of five command sequences in queue.

   Input: Select scheduled commands to delete. Send the request to delete a command from the queue.

   Output: A table which shows the updated statuses of all executed commands. This includes commands which were sent to and received by the satellite with their timestamps.

   Test Case Derivation: Upon deleting a scheduled command, the updated list of commands should be logged and displayed to the operator. The logs should also have the timestamp and statuses of the executed commands.

   How test will be performed: An operator will select a scheduled command from the list of scheduled commands. Then, an operator can choose to delete the selected command and the changes should be displayed in the MCT application's GUI.

4.1.5 Validating Scheduled Commands

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-7
- FR-8

1. FR-SLN8

   Control: Manual

Initial State: The MCT application has been configured with a list of allowed commands and automated sequences. These command sequences represent the available list of registered commands which operators can use to communicate with the satellite.

Input: A user attempts to enter an unregistered and unknown command through the GUI.

Output: An error message should be displayed, detailing the invalid command sequence.

Test Case Derivation: Only allowed command sequences can be scheduled and sent for communication. Otherwise, the user should be informed of the invalid request.

How test will be performed: An operator will enter a command sequence which is not in the list of available commands. The user interface shall display an error message and show the status of the invalid request.

4.1.6 Permission List Criteria for User

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-10
- FR-11

1. FR-SLN9

   Control: Manual

   Initial State: Ensure the user is logged into the MCT and the command to be executed does not match the user's permission list criteria.

   Input: User attempts to execute a command that does not match their permission list criteria.

   Output: The MCT should reject the command execution attempt and display an appropriate error message indicating the mismatch with the permission list criteria.

Test Case Derivation: Only commands which are under their permission should be listed.

How test will be performed: Multiple users with different access levels will be created. Then the list of commands that each one has access to will be cross-referenced with the allowed commands for their level to determine the correctness of the requirement.

4.1.7 Permission List Criteria for Command Target

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-12

1. FR-SLN11

   Control: Manual

   Initial State: Ensure the user is logged into the MCT and has appropriate permissions to schedule commands.

   Input: User schedules a command or automated command sequence for future execution.

   Output: The MCT should successfully schedule the command or automated command sequence for future execution. The scheduled command or sequence should execute at the specified time without manual intervention.

   Test Case Derivation: The MCT should automatically schedule a set of commands when certain criteria are met, including an acquisition of signal, or if the satellite is lit by the sun during the overpass.

   How test will be performed: A mock overpass will be created, to mimic conditions when the satellite will pass overhead, and the MCT will schedule the commands, and log results.

4.1.8 Managing Scheduled Command Sequences

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-13

1. FR-SLN12

   Control: Manual

   Initial State: Ensure the user is logged into the MCT and has appropriate permissions to manage scheduled commands and sequences.

   Input: User creates, edits, or deletes a scheduled command or sequence using the graphical user interface.

   Output: The MCT's graphical user interface should respond accordingly to the user's action. Created commands or sequences should be displayed, edited changes should be saved, and deleted commands or sequences should be removed from the interface.

   Test Case Derivation: Users should be able to add, edit, and remove commands off a schedule.

   How test will be performed: Endpoints for modifying schedules will be tested to ensure commands can be modified, as well as creating tracking observers between front-end components and back-end components, measuring if a endpoint has been hit to ensure proper connection.

4.1.9 Selecting and Editing Satellites

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-14

1. FR-SLN13

   Control: Manual

   Initial State: Ensure the user is logged into the MCT and has appropriate permissions to select and edit satellites.

   Input: User selects a satellite and edits its settings using the graphical user interface.

Output: The MCT's graphical user interface should allow the user to select and edit satellites of interest. Changes made to satellite settings should be saved and reflected in the interface.

Test Case Derivation: Users should be able to select and edit satellites of interest.

How test will be performed: Testing front-end UI will be done to add and delete satellites of interests for a particular user. These changes will reflect on the UI as well as in the back-end database, and will be cross referenced to see if the states match.

4.1.10 Viewing Configured Satellites

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-15

1. FR-SLN14

   Control: Manual

   Initial State: Satellites of interest are configured in the MCT.

   Input: User accesses the MCT interface.

   Output: The MCT interface should display the current orbital state for each satellite of interest. Information displayed should include elevation, orbital state, and solar illumination for each satellite.

   Test Case Derivation: Information should be dynamically shown for each user's satellite of interest.

   How test will be performed: Manual verification and cross-referencing with n2yo.org to verify accuracy and precision of data.

4.1.11 Configuring Elevation Threshold

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-16

1. FR-SLN15

   Control: Manual

   Initial State: User is logged into the MCT and the satellite's current elevation is below the specified threshold.

   Input: User sets a specific elevation threshold for a satellite.

   Output: The MCT should automatically schedule a command when the satellite's elevation reaches the user-specified threshold. The scheduled command should execute as per the defined parameters.

   Note: this requirement is no longer being implemented due to stakeholder request.

   Test Case Derivation:

   How test will be performed:

4.1.12 Detecting Satellite and Scheduling Command

The tests below provide a means to evaluate the following functional requirements referred to in the SRS document:

- FR-17

1. FR-SLN16

   Control: Manual

   Initial State: User is logged into the MCT and the satellite is within the covered area.

   Input: Satellite enters or exits the covered area.

   Output: The MCT should automatically schedule a command when the satellite enters the covered area. The scheduled command should execute as per the defined parameters.

   Test Case Derivation: The MCT should automatically schedule a set of commands when there is an acquisition of signal (above a certain elevation).

   How test will be performed: A mock overpass will be created, to mimic conditions when the satellite will pass overhead, and the MCT will schedule the commands, and log results

4.2 Tests for Nonfunctional Requirements

4.2.1 Usability and Humanity Requirements

Our usability testing module employs a comprehensive approach to evaluate user experience, learnability, and accessibility. We deploy a targeted suite of manual tests to capture direct user interaction feedback, assessing the software's intuitiveness and inclusiveness.

Usability Testing: We engage users in 2 hands-on sessions where users can interact with the app, collecting their feedback to derive a quantitative usability score. This score reflects the software's ease of use from the perspective of new users.

Learnability Assessment: Observing new users as they navigate the software without prior training allows us to gauge the software's learning curve and identify potential user experience roadblocks.

Accessibility Review: A diverse range of users, including those with different accessibility needs, are invited to test the software. Their experiences are crucial in informing necessary improvements to achieve an accessible and barrier-free user experience.

Each test within this module is carefully crafted to target essential usability facets, ensuring that the software lives up to the highest user-friendliness and accessibility standards. The usability questionnaire mentioned in future parts of the document cant be found here.

1. Usability-1

   NFR: 10.1

   Type: Usability, Manual

   Initial State: Application or system is fully functional and ready for user interaction without prior exposure.

   Input/Condition: User engagement with the application or system.

   Output/Result: Compilation of usability scores derived from user feedback and completed surveys.

   How test will be performed:

- Users will be granted access to the system or application for task execution or navigation.
- Upon using the application or system, participants will be requested to fill out a survey evaluating the usability aspects.
- The accumulated surveys will be examined to extract an aggregate usability score on a scale from 1 to 5.

2. Usability-2

   NFR: 10.3

   Type: Usability, Manual

   Initial State: Application or system is ready for testing with users who are inexperienced and have not been instructed on its use.

   Input/Condition: Users attempt self-guided learning and navigation of the application or system.

   Output/Result: Documentation of the learning curve, user errors, inquiries, and the time required for users to reach proficiency.

   How test will be performed:

   - Users will be allowed access to the software for exploratory learning through any available guides or tooltips.
   - Observers will document any user encounters with difficulties, posed questions, committed mistakes, and the duration to complete specified tasks efficiently.
   - Post-testing feedback sessions may be organized to gather additional insights on user learning experience.
   - The results from the surveys and observations will be synthesized to formulate an overall learnability score from 1 to 5.

3. Usability-3

   NFR: 10.5

   Type: Accessibility, Manual

Initial State: Application or system is operational and ready for assessment by users with various accessibility needs.

This will not be tested for the current scope of the project

4.2.2 Performance Requirements

The Performance Testing Module is structured to rigorously assess the application's responsiveness, stability, and efficiency through a series of automated and manual tests:

Response Time and Latency: Measures the application's speed and latency under varying loads, ensuring performance stays within specified benchmarks.

System Availability: Monitors uptime to ensure the application's continuous operation, capturing any periods of unavailability.

Calculation Precision: Validates the accuracy of calculations to the second decimal point, critical for applications relying on spg4's positioning and elevation computations.

Exception Handling: Evaluates the system's capability to effectively manage internal exceptions, maintaining functionality without data loss.

Resource Optimization: Tracks CPU, RAM, and disk space usage to guarantee optimal resource utilization under different workload scenarios.

Longevity Assessment: Conducts comprehensive reviews to project the system's operational viability up to the year 2026, ensuring robustness and maintainability.

This module is designed to ensure our software not only meets current performance standards but is also poised for future demands and growth.

1. performance-1

   NFR: 11.1

   Type: Performance, Automated

Initial State: Application or system fully functional and ready for testing, ideally in a controlled environment similar to production.

Input/Condition: Specific tasks or actions are performed on the application or system to measure response times and latency.

Output/Result: Measurements of speed (how fast a task is completed) and latency (time delay between the input and the expected output).

How test will be performed:

- Jest, a performance testing framework will be set up to simulate user actions or tasks.
- The tool will execute these actions, mimicking real-world usage scenarios.
- The software's response times for each task will be measured to assess speed.
- Latency, or the delay between a user's action and the system's response, will be measured.
- The test will be run multiple times, with increasing user load, to see how the system behaves under stress and if there are any degradation in speed or increases in latency.
- Collected data will be analyzed to determine if the software meets predefined speed and latency benchmarks or thresholds.

2. performance-2
   NFR: 11.2

   Type: Availability, Automated

   Initial State: Application or system deployed in a production-like environment.

   Input/Condition: Continuous monitoring tools check the application or system's availability at regular intervals.

   Output/Result: Measurements of the system's uptime and any periods of unavailability or downtime outside of scheduled maintenance windows.

   How test will be performed:

- Vercel's and DigitalOcean's (linux hosting providers) availability monitoring tool will be set up to check the application or system's status at regular intervals, e.g., every minute.
- The tool will send requests to the system to ensure it is responsive and available.
- Any periods of unavailability or downtime will be logged.
- Scheduled maintenance windows will be noted, and any downtime during these periods will be excluded from the final availability calculations.
- The system's overall uptime will be calculated as a percentage of the total time minus any downtime outside of scheduled maintenance.
- The collected data will be analyzed to determine if the software meets the required 24/7 availability criteria, excluding scheduled maintenance periods.

3. performance-3
   NFR: 11.3

   Type: Precision, Manual/Automated

   Note: This NFR is no longer being tested due to outsourcing of astrological calculations

   Initial State: Application or system fully functional and ready for testing in a controlled environment.

   Input/Condition: Specific tasks or calculations are executed in the application or system that involve spg4's calculation for position and elevation.

   Output/Result: Measurements of the system's calculation precision, ensuring results are accurate to the nearest 2nd decimal point.

   How test will be performed:

   - A set of known input data will be fed into the system which uses spg4's calculation.
   - The system's calculated results will be captured and compared against expected values with known precision.

- Any variation from the expected values beyond the 2nd decimal point will be logged as a precision error.
- Automated testing tools or scripts will be used to run multiple iterations with varying input data to ensure consistent precision across different scenarios.
- The collected data will be analyzed to determine if the software meets the required precision criteria, ensuring accuracy to the nearest 2nd decimal point for spg4's calculation.

4. performance-4
   NFR: 11.4

   Type: Exception Handling, Automated/Manual

   Initial State: Application or system fully functional and ready for testing in a controlled environment.

   Input/Condition: Specific actions or tasks are performed on the application or system that are known to trigger internal exceptions or are likely to do so.

   Output/Result: Logs or notifications of the system's ability to catch and handle internal exceptions.

   How test will be performed:

   - Scenarios known to cause internal exceptions will be identified. These scenarios will be executed on the system, through automated testing tools.
   - The system's behavior will be observed to check if it appropriately catches and handles the exceptions without crashing or causing data loss.
   - Logs, error messages, or any other relevant system output will be captured and analyzed to ensure the internal exceptions are caught and documented.
   - Any failure to catch or handle exceptions will be logged as an error.
   - The collected data will be analyzed to determine if the software effectively catches and manages internal exceptions as per the required criteria.

5. performance-5
   NFR: 11.5

   Type: Resource Optimization, Automated

   Initial State: Application or system fully functional and ready for testing in a controlled environment with monitoring tools in place.

   Input/Condition: The application or system is subjected to typical workloads or specific tasks that challenge its resource utilization.

   Output/Result: Measurements of the system's CPU, RAM, and disk space utilization during the test.

   Note: Since the server is hosted by external providers, this NFR no longer needs testing

   How test will be performed:

   - Baseline measurements of CPU, RAM, and disk space utilization will be captured when the system is idle.
   - Automated testing tools or scripts will be used to simulate typical user actions, workloads, or specific stress scenarios on the system.
   - Monitoring tools will continuously track and log the system's CPU, RAM, and disk space utilization throughout the test.
   - Once testing is complete, the peak and average resource utilizations will be identified.
   - These values will be compared to predefined benchmarks or acceptable limits to ensure the software operates within the desired resource constraints.
   - If resource utilization exceeds acceptable limits, the specific scenarios causing excessive usage will be identified for optimization.

6. performance-6
   NFR: 11.7

   Type: Longevity Assessment, Manual

   Initial State: Application or system fully functional, with complete documentation about its architecture, components, dependencies, and operational environment.

Input/Condition: Review of the system's architecture, codebase, dependencies, licensing, infrastructure, and any third-party services it relies on.

Output/Result: Recommendations and actions to ensure the system remains operational up to 2026.

How test will be performed:

- Infrastructure Review: Check the lifespan of the infrastructure components. Ensure that servers, databases, and other key components are robust and maintained to last through 2026.
- Dependency Audit: Analyze the system's dependencies, including libraries, frameworks, and third-party services. Ensure that they are actively maintained and are expected to be supported through 2026.
- Code Quality Assessment: High-quality, maintainable code is more likely to last longer without major issues. Perform a code review to identify any potential problem areas or technical debt that might cause problems in the future.
- Redundancy and Failover: Ensure that the system has adequate redundancy and failover mechanisms in place to handle failures and maintain uptime.
- Backup and Recovery Plan: Ensure that there's a robust backup and recovery plan in place and that it's tested regularly.
- Documentation Review: Well-documented systems are easier to maintain and update. Ensure that system documentation is up-to-date and comprehensive, covering both the technical aspects and operational procedures.

4.2.3 Operational and Environmental Requirements

1. Environmental-1

   NFR: 12.2.1

   Type: Staging Environment, Manual/Automated

Initial State: Development and testing of the application or system have been completed, and the production environment is ready for deployment.

Input/Condition: The application or system is prepared for deployment to the production environment, and the staging environment is configured to mirror the production environment.

Output/Result: Verification that the staging environment is available and functional for testing before production deployment.

How test will be performed:

(a) Perform automated load and performance testing in the staging environment to ensure it can handle expected traffic and workloads.

(b) Ensure that the staging environment is set up with configurations that match the production environment, including hardware, software, and network settings.

(c) Confirm that monitoring and logging tools are in place to track the environment's status and performance.

(d) Conduct manual testing to verify that the application or system's core functionalities work as expected in the staging environment.

2. Environmental-2
NFR: 12.2.2

Type: Development environment, Manual

Initial State: Development environment infrastructure is set up, and developers are ready to begin working on the MCT.

Input/Condition: Developers need to set up their local development environment without installing additional software or tools on their local machines.

Output/Result: Verification that the local development environment is self-contained and functional without the need for additional software installations.

How test will be performed:

(a) Developers will attempt to set up their local development environment following the provided instructions. Developers will launch the local development environment to ensure it initializes and runs without errors.

(b) Developers will work on the MCT application, making changes, writing code, and performing tests within the self-contained environment.

(c) Verify that the MCT's required dependencies are included and functioning properly within the self-contained development environment.

3. Environmental-3

NFR: 12.5

Type: Release Requirement, Manual

Initial State: The project for developing the MCT is in its planning and development phase.

Input/Condition: The project plan includes the requirement to have the MCT ready for use by September 2024.

Output/Result: Project is on track to meet the commission date and validation that the MCT is indeed ready for use by September 2024.

How test will be performed:

(a) Implement a project management and monitoring system to track the progress of the MCT development against the project plan. Continuously monitor project tasks and milestones to identify any potential delays.

(b) Schedule regular project status meetings to review progress, discuss challenges, and address any deviations from the project plan. Monitor and report on progress to stakeholders.

(c) As the September 2024 deadline approaches, assess the readiness of the MCT. Verify that all features, functions, and necessary preparations are in place for use.

4.2.4 Maintainability and Support Requirements

1. Maintenance-1
   NFR: 13.1.1

   Type: Database Management, Manual and Automated

   Initial State: The MCT development project is in progress, and MongoDB has been chosen as the database technology for storing application data.

   Input/Condition: The development team is tasked with implementing MongoDB and ensuring it aligns with the project's requirements.

   Output/Result: Verification that MongoDB is successfully implemented, and the use of text or plain files for data storage is avoided.

   How test will be performed:

   (a) Verify that MongoDB is configured and optimized for the project's specific requirements.

   (b) Ensure that MongoDB is compatible with the MCT's software stack, programming languages, and frameworks.

   (c) Test data retrieval and query performance to ensure that MongoDB can deliver data to the application in a timely manner.

   (d) Ensure that MongoDB backup and recovery mechanisms are in place and tested to safeguard against data loss.

   (e) Simulate expected workloads and assess MongoDB's performance, including read and write operations, query execution times, and throughput.

2. Maintenance-2
   NFR: 13.1.2

   Type: CI/CD Process, Automated and Manual

   Initial State: The MCT development project is underway, and the development team has decided to use GitHub Actions for CI/CD.

   Input/Condition: The development team is responsible for implementing CI/CD with GitHub Actions, incorporating automated testing and static analysis.

Output/Result: GitHub Actions is successfully set up for CI/CD, including automated testing and static analysis, and that it aligns with the requirement.

How test will be performed:

(a) Confirm that GitHub Actions is correctly configured for the project, including the definition of workflows, jobs, and triggers.

(b) Ensure that the CI/CD pipeline in GitHub Actions automates the build and deployment process of the MCT.

(c) Introduce a code change and push it to the version control system. Confirm that the CI/CD pipeline automatically triggers unit tests and reports the test results.

(d) Validate that CI/CD pipelines integrate seamlessly with issue tracking systems and version control repositories. Confirm that commits trigger automatic builds.

(e) Use Git for code reviews to ensure high quality, readable code is being committed to the database. Confirm that formatting matches ESLint formatting standards.

3. Maintenance-3
NFR: 13.1.3

Type: System updates, Manual/Automated

Initial State: The MCT development is in progress, and the system is expected to have the ability to update libraries and dependencies.

Input/Condition: The system must be designed to detect, download, and apply updates for libraries and dependencies from trusted sources.

Output/Result: Verification that the MCT can effectively update its libraries and dependencies to enhance security, performance, or functionality.

How test will be performed:

(a) Use a test environment to initiate an update for a specific library. Confirm that the system processes the update request and begins the update process.

(b) Manually downgrade a library version to simulate an outdated component. Verify that the system identifies the outdated version and triggers an update.

(c) Apply an update to a library in the test environment. Verify that automated tests are executed, and manual validation is performed to ensure that the update doesn't introduce new issues.

(d) Introduce deliberate code formatting issues and verify that they are promptly addressed and corrected in the codebase.

4. Maintenance-4
NFR: 13.1.4

Type: Code Quality Assessment, Manual

Initial State: The MCT development project is ongoing, and the code is being actively developed.

Input/Condition: The development team is responsible for writing and maintaining the MCT's source code.

Output/Result: Source code is well-organized, modular, and adheres to fundamental software engineering principles.

How test will be performed:

(a) Review the project's directory structure to ensure that source code files are organized into logical folders, like "front-end" and "back-end".

(b) Verify that variable and function names follow a consistent naming convention (e.g., camelCase), ensuring that code is self-consistent.

(c) Conduct a code review session with team members, identifying and discussing any code quality issues or deviations from best practices.

(d) Review code from multiple team members to ensure that it adheres to the project's coding standards and guidelines.

(e) Use Git for code reviews to ensure high quality, readable code is being committed to the database. Confirm that formatting matches ESLint formatting standards.

5. Maintenance-5
   NFR: 13.1.5

   Type: Code Formatting, Automated

   Initial State: The MCT development project is in progress, and source code is actively being written and maintained.

   Input/Condition: The development team is responsible for writing and formatting the source code, and ESLint is integrated into the development workflow.

   Output/Result: Verification that the source code adheres to consistent formatting standards and is easy to read.

   How test will be performed:

   (a) Confirm that ESLint, a code formatting tool is integrated into the development environment and configured to enforce code formatting standards.

   (b) Run the code formatting tool on a code file known to have inconsistent formatting. Verify that the tool automatically formats the code.

   (c) Select code files at random and confirm that they follow defined code style guidelines, including aspects like indentation, variable naming, and code structure.

6. Support-1
   NFR: 13.2.1

   Type: Containerization, Manual and Automated

   Initial State: The source code of the MCT application is actively being developed and is in a state where it needs to be containerized for deployment.

   Input/Condition: The team has access to containerization tools and technologies, such as Docker, and is responsible for integrating these tools into the development workflow.

   Output/Result: Verification that the MCT is successfully containerized, allowing for portability, scalability, and efficiency.

   How test will be performed:

(a) Confirm that Docker is installed and configured on the development environment. Execute the containerization process for the MCT application. Verify that the container is successfully created and can be run as intended.

(b) Deploy the MCT container to a local development environment and a testing environment. Confirm that the same container runs consistently in both environments.

(c) Store data in the containerized application and restart it. Confirm that data is not lost when containers are restarted.

7. Support-2
NFR: 13.2.2

Type: Backup Testing, Manual

Initial State: The MCT system is operational, and data is being actively processed and stored.

Input/Condition: The development team has implemented data backup capabilities within the MCT system, and administrators are responsible for initiating and managing data backup operations.

Output/Result: Verification that backups are created and can be accessed in case of data loss.

How test will be performed:

(a) After creating a backup, compare the backed-up data with the original data to ensure that no data corruption occurred during the backup process.

(b) Perform a backup of a large data set and measure the time it takes to complete. Verify that the system can handle large backups efficiently.

(c) Initiate a data restoration process from a backup to confirm that administrators can successfully restore data in case of loss or corruption.

8. Support-3
NFR: 13.2.3

Type: Application Log Implementation, Automated and Manual

Initial State: The MCT application is operational and actively processing requests and data. Logging mechanisms are implemented within the application.

Input/Condition: The MCT application is configured to generate logs, and the log storage and retrieval mechanisms are in place. The development team and administrators have access to log data for debugging purposes.

Output/Result: Verification that the MCT application effectively generates and stores logs to assist in debugging, and that the logs can be retrieved and analyzed.

How test will be performed:

(a) Trigger specific events or actions in the MCT application that should result in log entries being created. Confirm that log entries are generated as expected.

(b) Inspect log entries to ensure they contain relevant information, such as timestamps, error messages.

(c) Use logs to investigate and resolve specific issues or errors that occur within the MCT application. Confirm that logs facilitate the debugging process by identifying the causes of errors.

4.2.5 Security and Access Requirements

9. Access-1
NFR: 14.1.1

Type: Multi-Factor Authentication Implementation, Manual

Initial State: The application's multi-factor authentication feature is enabled and configured. Users have registered email accounts linked to their respective profiles.

Input/Condition: User attempts to log in to the application and provides the required login credentials. The user is prompted to verify their identity through their registered email account by entering a verification code sent to the email.

Output/Result: Verification that the application successfully prompts users for multi-factor authentication via their registered email accounts, preventing unauthorized access to sensitive user and system data.

How test will be performed:

(a) Attempt to log in to the application using valid credentials.

(b) Verify that the application prompts the user to verify their identity through their registered email account.

(c) Access the registered email account and retrieve the verification code.

(d) Enter the verification code into the application to complete the login process.

(e) Attempt to log in with invalid credentials and verify that the multi-factor authentication process prevents unauthorized access.

10. Access-2
NFR: 14.1.2

Type: Account Lockout and Timeout Implementation, Manual

Initial State: The application's account lockout and timeout settings are configured to restrict access after ten consecutive failed login attempts. The timeout period is set to five minutes.

Input/Condition: A user attempts to log in to the application using the same credentials consecutively for ten times.

Output/Result: Verification that the application denies access to the user account after ten consecutive failed login attempts and enforces a five-minute timeout period before the user can attempt to authenticate again.

How test will be performed:

(a) Attempt to log in to the application with the same credentials ten times, ensuring each attempt is unsuccessful.

(b) Verify that the application denies access to the user account after the tenth failed attempt.

(c) Wait for the designated timeout period to pass (five minutes).

(d) Attempt to log in to the application again with the correct credentials and verify that the user can authenticate successfully.

11. Integrity-1
NFR: 14.2.1

Type: TLE Update and Overpass Calculation, Automated

Initial State: The application has a scheduled process to periodically fetch TLE data and recalculate satellite overpasses based on the most recent information.

Input/Condition: The scheduled time for the TLE update and overpass calculation process is reached.

Output/Result: Verification that the application successfully fetches the most recent TLE data and recalculates satellite overpasses using the updated information, ensuring accurate predictions of satellite overpasses.

How test will be performed:

(a) Wait for the scheduled time for the TLE update and overpass calculation process to be initiated.

(b) Verify that the application fetches the most recent TLE data from the appropriate source.

(c) Confirm that the application utilizes the updated TLE data to recalculate satellite overpasses.

(d) Compare the recalculated overpasses with the previously calculated overpasses to ensure the predictions are more accurate.

12. Integrity-2
NFR: 14.2.2

Type: Minimum Elevation Connection Evaluation, Automated

Initial State: The application is configured with a minimum elevation threshold to evaluate satellite connections, and the necessary satellite data and connectivity settings are in place.

Input/Condition: The application attempts to establish a connection with a satellite positioned below the set minimum elevation threshold.

Output/Result: Verification that the application correctly assesses whether a satellite is within the acceptable range for connection based on the minimum elevation threshold, ensuring that scheduled commands are not executed if the satellite is out of range.

How test will be performed:

(a) Simulate a scenario where the application attempts to establish a connection with a satellite positioned below the minimum elevation threshold.

(b) Verify that the application evaluates the satellite's elevation and determines whether it is within the acceptable range for a connection.

(c) Confirm that the application does not execute scheduled commands if the satellite is deemed out of range based on the minimum elevation threshold.

13. Integrity-3

NFR: 14.2.3

Type: Error Notification and Operator Responsibility, Manual

Initial State: The application is operational, and the operators have the necessary access and privileges to send and schedule commands to the satellite.

Input/Condition: An operator attempts to schedule a command that contains errors or invalid data.

Output/Result: Verification that the application promptly notifies the operator of any errors found within the scheduled command, while emphasizing the operator's responsibility to ensure the validity of the commands.

How test will be performed:

(a) Intentionally create a command with errors or invalid data and attempt to schedule it.

(b) Confirm that the application promptly notifies the operator of the errors found within the command.

(c) Ensure that the error notification provides sufficient information to help the operator identify and rectify the errors within the command.

(d) Verify that the application does not execute the command until the errors have been addressed by the operator.

14. Integrity-4

NFR: 14.2.4

Type: Backup and Recovery Mechanism, Automated and Manual

Initial State: The system has a backup and recovery mechanism in place to restore the database to its most recent correct state in the event of unexpected data failures. The backup schedule is established and functioning properly.

Input/Condition: Introduce simulated data corruption or loss within the database to trigger the need for the backup and recovery mechanism.

Output/Result: Verification that the system successfully restores the database to its most recent correct state using the backup and recovery mechanism, ensuring that the information in the database remains accessible despite the data failures.

How test will be performed:

(a) Introduce simulated data corruption or loss within the database.

(b) Trigger the backup and recovery mechanism to restore the database to its most recent correct state.

(c) Verify that the system retrieves and reinstates the most recent backup data without any loss or corruption.

(d) Confirm that the restored database contains all the necessary information and is fully operational.

15. Integrity-5

NFR: 14.2.5

Type: Error Message Standardization and Data Validation, Automated **Initial State**: The application's data validation is set up to identify the missing data fields and return appropriate error messages adhering to a standardized format.

Input/Condition: Simulate a scenario where the system or satellite failure leads to responses with missing data attributes.

Output/Result: Ensure that the application correctly labels all missing data fields with appropriate error messages, ensuring that it sticks to the standardized format and maintaining consistent data across all validation processes.

How test will be performed:

(a) Simulate a system, or satellite failure to generate responses with missing data attributes.

(b) Verify that the application identifies the missing data fields during the data validation process.

(c) Confirm that the application labels the missing data fields with appropriate error messages following the standardized format.

(d) Cross-verify that the standardized error messages effectively communicate the nature of the missing data attributes and their respective fields.

16. Integrity-6

NFR: 14.2.6

Type: Cross-Browser Testing, Manual and Automated **Initial State**: The application is developed, and the cross-browser testing framework is set up to evaluate the application's performance across a range of browsers, including Safari, Chrome, and Firefox.

Input/Condition: Execute the application on different browsers, including Safari, Chrome, and Firefox, to assess its functionality and consistency. **Output/Result**: Verification that the application functions consistently across all tested browsers, ensuring a uniform user experience regardless of the browser being used. **How test will be performed**:

(a) Execute the application on Safari, Chrome, and Firefox, covering the range of browsers specified in the requirement.

(b) Verify that all essential functionalities of the application perform as expected on each browser.

(c) Confirm that the layout, design, and user interface elements remain consistent across all tested browsers.

(d) Identify and document any discrepancies or inconsistencies in the application's behavior across different browsers.

17. Integrity-7

NFR: 14.2.7

Type: HTML and CSS Compatibility Checks, Automated **Initial State**: The code base is set up with automated HTML and CSS checks to ensure compatibility across different devices and browsers. The necessary testing tools and frameworks are in place.

Input/Condition: Run automated HTML and CSS checks on the code base to identify any compatibility issues across various devices and browsers. **Output/Result**: Verification that the code base passes the HTML and CSS compatibility checks, ensuring that the application is compatible and displays correctly across different devices and browsers. **How test will be performed**:

(a) Execute the automated HTML and CSS checks on the code base using appropriate testing tools and frameworks.

(b) Verify that the code base adheres to the HTML and CSS standards for compatibility across different devices and browsers.

(c) Identify and address any compatibility issues or errors reported by the automated checks.

(d) Validate that the application's user interface remains consistent and functional across different devices and browsers.

18. Integrity-8

NFR: 14.2.8

Type: Efficiency Evaluation of Scheduling Components, Automated

Initial State: The application's scheduling components are in place, and the efficiency evaluation pipeline is set up to assess each component's performance, ensuring that each component completes within the specified minimum time frame.

Input/Condition: Execute the efficiency evaluation pipeline to test the duration of each scheduling component and verify that they complete within the designated minimum time frame.

Output/Result: Verification that all scheduling components complete their tasks within the minimum time frame, ensuring efficient scheduling operations within the application. **How test will be performed**:

(a) Execute the efficiency evaluation pipeline to assess the duration of each scheduling component's operation.

(b) Verify that each scheduling component completes its tasks within the specified minimum time frame.

(c) Identify and address any components that do not meet the specified efficiency requirements.

(d) Ensure that the scheduling operations remain efficient and that any inefficiencies are rectified promptly.

19. Privacy-1

NFR: 14.3.1

Type: Network Port Security with Cryptographic Protocol, Automated

Initial State: The MCT's network ports are configured and exposed, and a reputed cryptographic protocol is implemented to secure the network communication.

Input/Condition: Initiate an automated security test to verify that the network ports exposed by the MCT are secured using the specified reputable cryptographic protocol.

Output/Result: Verification that the network ports exposed by the MCT are effectively secured using the reputable cryptographic protocol, ensuring the confidentiality and integrity of the network communication.

How test will be performed:

(a) Conduct an automated security test to assess the cryptographic protocol implementation on the network ports exposed by the MCT.

(b) Confirm that the cryptographic protocol in use is recognized as reputable and meets security requirements.

(c) Validate that the network communication remains confidential and secure during the automated security test.

(d) Address any identified security vulnerabilities or weaknesses to ensure the robustness of the cryptographic protocol.

4.3 Traceability Between Test Cases and Requirements

Req. ID	**System Test ID**
FR-1	FR-SLN1, FR-SLN2
FR-2	FR-SLN3, FR-SLN4
FR-3	FR-SLN5, FR-SLN6
FR-4	FR-SLN5, FR-SLN6
FR-5	FR-SLN5, FR-SLN6
FR-6	FR-SLN7
FR-7	FR-SLN3, FR-SLN8
FR-8	FR-SLN8
FR-10	FR-SLN9
FR-11	FR-SLN10

FR-12	FR-SLN11
FR-13	FR-SLN12
FR-14	FR-SLN13
FR-15	FR-SLN16
FR-17	FR-SLN16
NFR-10.1	Usability-1
NFR-10.3	Usability-2
NFR-10.5	Usability-3
NFR-11.1	Performance-1
NFR-11.2	Performance-2
NFR-11.3	Performance-3
NFR-11.4	Performance-4
NFR-11.5	Performance-5
NFR-11.6	Performance-6
NFR-12.2.1	Environmental-1
NFR-12.2.2	Environmental-2
NFR-12.2.5	Environmental-3
NFR-13.1.1	Maintenance-1
NFR-13.1.2	Maintenance-2
NFR-13.1.3	Maintenance-3
NFR-13.1.4	Maintenance-4
NFR-13.1.5	Maintenance-5
NFR-13.2.1	Support-1

NFR-13.2.2	Support-2
NFR-13.2.3	Support-3
NFR-14.1.1	Access-1
NFR-14.1.2	Access-2
NFR-14.2.1	Integrity-1
NFR-14.2.2	Integrity-2
NFR-14.2.3	Integrity-3
NFR-14.2.4	Integrity-4
NFR-14.2.5	Integrity-5
NFR-14.2.6	Integrity-6
NFR-14.2.7	Integrity-7
NFR-14.2.8	Integrity-8
NFR-14.3.1	Privacy-3

5 Unit Test Description

5.1 Unit Testing Scope

The Satellite Users, Schedule, and Database modules will be evaluated using unit tests. Since the Satellite Users and Schedule modules uses the Database modules for creating, updating, and removing records, the evaluation of the Satellite Users and Schedule modules will encapsulate test cases from the Database module. In addition, these unit tests only evaluate the functionality of the backend of the application and all tests can be found here.

5.2 Tests for Functional Requirements

The tests for functional requirements are grouped by modules and helper functions.

5.2.1 Satellite Users Module

The unit tests for the Satellite Users Module is written using Jest.js. These tests focus on managing information for a satellite and user records by interfacing with the Database Module. These tests are automatically executed through Github Actions.

5.2.2 Schedule Module

The unit tests for the Schedule Module is written using Jest.js. These tests focus on managing scheduled command sequences for a specific satellite by interfacing with the Database Module. These tests are automatically executed through Github Actions.

5.2.3 Helper Functions

The unit tests for the helper functions used across the Satellite, Satellite Users, and Schedule Module is written using Jest.js. These tests focus on verifying and validating the functionality of common helper functions used throughout the application. These tests are automatically executed through Github Actions.

5.3 Tests for Nonfunctional Requirements

Unit Tests will not cover Nonfunctional Requirements.

5.4 Traceability between Unit Test Cases and Requirements

Req. ID	**System Test ID**
FR-1	SM3
FR-4	SM3
FR-5	SM3
FR-10	SM10, SM20, SM21

FR-11	SM4, SM9, SM11, SM21
FR-12	SM1
FR-13	SM5, SM6, SM7, SM8
FR-14	SM7, SM8, SM18, SM19, SM29, SM30, SM31
FR-15	SM7, SM8, SM10, SM11, SM12, SM23, SM24, SM25, SM26, SM27, SM28
FR-16	SM1, SM2, SM12, SM13, SM14, SM22
FR-17	SM1, SM2, SM12, SM22

References

6 Appendix

6.1 Symbolic Parameters

N/A

6.2 Usability Survey Questions

The questions for the usability survey can be found here.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

Appendix — Reflection

This section is not required for CAS 741

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

Umang's Reflection

1. In order to make sure this project achieves all of its targeted goals, the team will have to acquire knowledge and skills to become more familiar with the verification and validation steps of a software project. The team will need to acquire knowledge in dynamic testing techniques such as integration testing when verifying how well the application adheres to the SRS. Also, unit testing can be used to streamline the testing process when evaluating different components of the MCT application. Software tools such as Selenium and jest are available to automate the testing of client and server-side applications. In addition, the performance of the application needs to be verified and validated, which can be done through acquiring testing skills related to regression testing.

2. There are several resources, whether it is in the form of books or videos, to learn more about the technologies and processes mentioned above. I will learn by reading documentation and articles to learn about best practices of testing depending on the use case. Then, I will research into testing tools and frameworks to help implement these best practices.

Quinn's Reflection

1. For the team to complete the verification and validation of the project, thorough testing knowledge will have to be learned and applied, specifically reflecting on past coursework in SFWRENG 3SO3, Software Testing. This knowledge can be applied while using various testing tools to statically and dynamically test FRs and NRFs. In addition, we can draw our knowledge from SFWRENG 4HC3, Human Computer Interfaces, for usability testing of NFR's, and conducting usability testing based off designs.

2. To approach static, dynamic, and usability testing, it would be good to reflect and utilize past coursework, as well as use online resources/tutorial to aid in learning. As my interests lie primarily in the UI/UX component of the application, I will look into usability testing, and drawing my experiences from past coursework.

Diamond's Reflection

1. To successfully complete the verification and validation of our project, the team will need a range of knowledge and skills. This includes understanding testing methodologies, the ability to create effective tests, proficiency in relevant tools and languages, database management, and effective communication with stakeholders. Acquiring these competencies is essential for the project's success.

2. In addition to taking webinars for practical testing, team members have the option of enrolling in online tutorials and courses to gain information about dynamic testing. They can use linters and static analysis tools to find problems in the code, and they can research and implement industry best practises for efficient code reviews while the product is being developed.

Dhruv's Reflection

1. To ensure the successful verification and validation of our project, the team will require a diverse set of knowledge and skills. This encompasses familiarity with various testing methodologies for which I will refer to a previous course we took on testing in a previous year, 3S03: Software Testing. Acquiring and honing these competencies are imperative for achieving the project's objectives. To do this, we will continue to keep touching base to ensure team communication is maintained and

I will find resources so I can learn how to dynamic test more effectively. Utilizing some softwares such as Jira can help in this aspect.

2. To enhance their understanding of practical testing, team members can participate in webinars and consider enrolling in online Udemy courses to become better dynamic testers. Additionally, looking at online videos on how people approach it is a good way to understand the different techniques involved. This is something I look forward to doing because it can allow me to think from many perspectives while ensuring our application is robust and reliable.

Rishi's Reflection

1. To ensure the successful verification and validation of our project, the team must possess a diverse set of knowledge and skills. This encompasses familiarity with various testing methods, the capability to design meaningful tests, proficiency in relevant programming languages and tools, adeptness in database management, and the ability to communicate effectively with stakeholders. Acquiring these competencies is vital for the project's overall success, as they enable the team to thoroughly assess the project's functionality, identify potential issues, and maintain clear communication channels with all involved parties. With these essential skills in place, the project can achieve its objectives efficiently and meet the requirements of stakeholders and users alike.

2. Numerous resources, including books and videos, are available to delve deeper into the technologies and processes discussed. My approach involves studying documentation and articles to grasp testing best practices tailored to specific use cases. Subsequently, I plan to explore various testing tools and frameworks, utilizing this research to effectively implement the identified best practices. This methodical approach ensures a comprehensive understanding of the subject matter and facilitates the practical application of learned concepts in the project.

Next steps:

- Dynamic Testing (integration testing): Quinn and Rishi will take the initiative to gain a deeper understanding of dynamic testing techniques such as integration testing. This will be used to verify the completeness and correctness of the code and SRS.

- Regression Testing: Dhruv will take the initiative to gain a deeper understanding of regression testing. The goal is to research into the available regression testing tools and frameworks which will subsequently be utilized to verify the application's performance.
- Unit Testing: Diamond and Umang will take the initiative to gain a deeper understanding of testing tools such as Selenium and jest. This will be used to automate testing of the application's client and server-side functions.