Final Project – Testing Document

Goals of this Document

The goal of this document is to detail efforts towards testing and debugging the application. This includes specifying tests for functional and non-functional requirements, describing the use of Test-Driven Development (TDD), and discussing random and automatically-generated tests, UI testing, and testing for build, integration, and deployment.

1. Testing Functional Requirements

1. System Initialization and Configuration

Test Objective: Validate that the initialization and configuration of the system components occur correctly, handling any errors gracefully.

Test Name:

a. ***Test_InitializeProject_HandlesErrors***
   - **Description**: Verifies that errors during project initialization are correctly caught and handled by the *InitializeProject* method of *InitializeProjectBL*.
   - **Test Steps**:
     1. Simulate initialization with erroneous input.
     2. Assert that appropriate error handling mechanisms are triggered.
     3. Validate that the system remains in a stable state post-error.

2. Metadata Reception and Processing

Test Objective: Ensure accurate reception and processing of PLP metadata.

Test Name:

a. ***Test_IsValidPLP_ReturnsTrueForValidPLP***
   - **Description**: Confirms that the *IsValidPLP* method correctly identifies valid PLP metadata.
   - **Test Steps**:
     1. Provide a set of valid PLP metadata.
     2. Assert that *IsValidPLP* returns true for these inputs.

b. ***Test_IsValidPLP_ReturnsFalseForInvalidPLP***
   - **Description**: Validates that the *IsValidPLP* method correctly identifies invalid PLP metadata.
   - **Test Steps**:
     1. Provide a set of invalid PLP metadata.
     2. Assert that *IsValidPLP* returns false for these inputs.

3. Communication Monitoring

N/A.

4. Robot Infrastructure Compatibility

Test Objective: Ensure compatibility with ROS and ROS2 middleware generation.

Test Name:

a. ***Test_GetBuildRosMiddlewareBashFile_ReturnsCorrectScript***
   - **Description**: Verifies that the *GetBuildRosMiddlewareBashFile* method generates the correct bash script for ROS middleware.
   - **Test Steps**:
     1. Call *GetBuildRosMiddlewareBashFile*.
     2. Assert the generated script matches the expected ROS middleware build script.

b. ***Test_GetBuildRos2MiddlewareBashFile_ReturnsCorrectScript***
   - **Description**: Validates that the *GetBuildRos2MiddlewareBashFile* method generates the correct bash script for ROS2 middleware.
   - **Test Steps**:
     1. Call *GetBuildRos2MiddlewareBashFile*.
     2. Assert the generated script matches the expected ROS2 middleware build script.

5. Remote User Interface

N/A.

6. Integration with AOS Planning Engine

Test Objective: Ensure proper integration with the AOS planning engine.

Test Name:

a. ***Test_GetRunSolverBashFile_ReturnsCorrectScript***
   - **Description**: Checks that the *GetRunSolverBashFile* method generates the correct bash script for running the solver.
   - **Test Steps**:
     1. Call *GetRunSolverBashFile*.
     2. Assert the generated script matches the expected solver execution script.

7. System Interface with External Devices

N/A.

2. Testing Non-Functional Requirements

Data Integrity

Test Objective: Verify the integrity of data processed by the application.

Test Name:

a. ***Test_IsValidPLP_EnsuresDataIntegrity***

- **Description**: Ensures data integrity by validating both valid and invalid PLP JSON content.
- **Test Steps**:
    1. Provide valid and invalid PLP JSON data.
    2. Assert that the *IsValidPLP* method correctly identifies valid and invalid data.

3. Test-Driven Development

Description: TDD principles are integral to the development process, ensuring that all functionality is thoroughly tested before implementation.

Implementation:

All tests in this document follow a TDD approach, where test cases are defined before coding begins. This ensures that all requirements are met and edge cases are considered during development.

4. Random & Automatically-Generated Tests

N/A.

5. Testing the User Interface

N/A.

6. Testing Build, Integration & Deployment

1. Build

Test Objective: Ensure that build scripts are generated correctly.

Test Cases:

a. ***Test_GetBuildRosMiddlewareBashFile_ReturnsCorrectScript***
    - **Description**: Validates that the *GetBuildRosMiddlewareBashFile* method generates the correct ROS middleware build script.

b. ***Test_GetBuildRos2MiddlewareBashFile_ReturnsCorrectScript***
    - **Description**: Confirms that the *GetBuildRos2MiddlewareBashFile* method generates the correct ROS2 middleware build script.

2. Integration

Test Objective: Validate integration capabilities, particularly with running the solver.

Test Case:

a. ***Test_RunSolver_CallsRunBashCommand***
    - **Description**: Checks that the *RunSolver* method executes the solver bash command as expected.

- **Test Steps**:
  1. Call *RunSolver*.
  2. Verify that the solver bash command is invoked correctly.

3. Deployment

Test Objective: Ensure smooth deployment and error handling during initialization.

Test Case:

a. Test_InitializeProject_HandlesErrors

- **Description**: Verifies that the *InitializeProject* method handles initialization errors appropriately.
- **Test Steps**:
  1. Invoke *InitializeProject* with erroneous conditions.
  2. Ensure that errors are caught and handled without disrupting system stability.

7. Functional Testing for Translation Modules

1. TranslateSD Functionality

Description: Tests for translating SD files into JSON representation using the visitor pattern approach.

Tests:

a. ***Test_TranslateSD_ValidContent***
- **Description**: Validates translation of valid *SD* content.
- **Test Steps**:
  1. Provide a sample valid *SD* file.
  2. Assert that the translated JSON matches the expected structure.

b. ***Test_TranslateSD_InvalidContent***
- **Description**: Ensures proper handling of invalid *SD* content.
- **Test Steps**:
  1. Provide an invalid *SD* file.
  2. Verify that appropriate exceptions or error messages are raised.

c. ***Test_TranslateSD_ParameterSpecificCases***
- **Description**: Covers translation scenarios based on different parameters in *SD* files.
- **Test Steps**:
  1. Test various combinations of parameters.
  2. Validate that translations are accurate under different conditions.

d. ***Test_TranslateSD_EdgeCases***
- **Description**: Tests edge cases for *SD* file translation.
- **Test Steps**:
  1. Include scenarios with minimal or maximal inputs.
  2. Verify that the system handles edge cases gracefully.

2. TranslateAM Functionality

Description: Tests for translating *AM* files into JSON representation using the visitor pattern approach.

Tests:

a. ***Test_TranslateAM_ValidContent***
   - **Description**: Validates translation of valid *AM* content.
   - **Test Steps**:
     1. Provide a sample valid *AM* file.
     2. Assert that the translated JSON matches the expected structure.

b. ***Test_TranslateAM_InvalidContent***
   - **Description**: Ensures proper handling of invalid *AM* content.
   - **Test Steps**:
     1. Provide an invalid *AM* file.
     2. Verify that appropriate exceptions or error messages are raised.

c. ***Test_TranslateAM_ParameterSpecificCases***
   - **Description**: Covers translation scenarios based on different parameters in *AM* files.
   - **Test Steps**:
     1. Test various combinations of parameters.
     2. Validate that translations are accurate under different conditions.

d. ***Test_TranslateAM_EdgeCases***
   - **Description**: Tests edge cases for *AM* file translation.
   - **Test Steps**:
     1. Include scenarios with minimal or maximal inputs.
     2. Verify that the system handles edge cases gracefully.

3. TranslateEF Functionality

Description: Tests for translating *EF* files into JSON representation using the visitor pattern approach.

Tests:

a. ***Test_TranslateEF_ValidContent***
   - **Description**: Validates translation of valid *EF* content.
   - **Test Steps**:
     1. Provide a sample valid *EF* file.
     2. Assert that the translated JSON matches the expected structure.

b. ***Test_TranslateEF_InvalidContent***
   - **Description**: Ensures proper handling of invalid *EF* content.
   - **Test Steps**:
     1. Provide an invalid *EF* file.
     2. Verify that appropriate exceptions or error messages are raised.

c. ***Test_TranslateEF_ParameterSpecificCases***
   - **Description**: Covers translation scenarios based on different parameters in *EF* files.
   - **Test Steps**:
     1. Test various combinations of parameters.
     2. Validate that translations are accurate under different conditions.

d. ***Test_TranslateEF_EdgeCases***
   - **Description**: Tests edge cases for *EF* file translation.
   - **Test Steps**:
     1. Include scenarios with minimal or maximal inputs.
     2. Verify that the system handles edge cases gracefully.