System Verification and Validation Plan for Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

November 4, 2024

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

Please refer to Section 4 of Software Requirements Specification found here.

This document is intended to provide a description of the specific validation and verification activites that will be completed throughout the development of the GeoWeb System. The purpose of these activities is to ensure the system requirements agree with stakeholder needs, and to certify the implementation of the system satisifes the stakeholder requirements. This document will start with a review of the system, a description of the testing plan, and a list of tests to be completed.

2 2 General Information

2.1 2.1 Summary

[4]

Orbit Watch is an online platform developed by OKKM Insights to streamline and simplify satellite imagery data analysis. The software addresses the scarcity of high-quality, labeled satellite imagery datasets tailored for specific use cases across various industries such as disaster response, environmental monitoring, urban planning, and defense. Leveraging an AI-powered crowdsourcing model, Orbit Watch enables users to label commercially available satellite images. These labeled datasets are then utilized to train custom computer vision models, enhancing accuracy and efficiency in image analysis. The platform offers a paid service for identifying objects within satellite images and distributes earnings to users who contribute to the labeling efforts.

2.2 2.2 Objectives

[4]

The primary objectives of the Verification and Validation (V&V) plan for Orbit Watch are:

- **Ensure High Data Accuracy**: Build confidence in the software's correctness by verifying that the system achieves high classification accuracy for objects reported in the images. This is crucial for extracting useful information and providing value to stakeholders.
- **Validate System Reliability and Accessibility**: Confirm that the system is reliable with minimal downtime and is accessible remotely

for both purchasers and labelers, ensuring flexibility and efficiency in usage.

The following objectives are out of scope for this V&V plan:

- **Verification of External Components**: We will not verify external libraries or services used for image acquisition, payment processing, or AI algorithms. It is assumed that these components have been adequately verified by their respective development teams.
- **Stretch Goals**: Objectives related to automatic data labeling and multi-source integration are considered stretch goals and will not be included in the current V&V efforts due to time and resource constraints.

2.3 2.3 Challenge Level and Extras

[1] [4]

We aim to classify this project as an advanced level project.

This project can be classified at the **advanced** challenge level due to several factors:

- **Limited Domain Knowledge**: The team has minimal experience with satellite imagery and computer vision models, requiring additional research and learning to implement effective solutions.
- **Complex Implementation**: Developing a web application from scratch involves challenges in front-end development, integrating secure payment systems, handling parallel image labeling tasks, and designing consensus algorithms for data accuracy.
- **Integration of Multiple Components**: The project demands seamless integration of crowd-sourcing models, AI algorithms, payment processing, and user interfaces, increasing the complexity of development and testing.

Extras from the Problem Statement [4]

- **Usability Testing**: This phase involves selecting a group of users, ideally from the target demographic (e.g., professionals in satellite imagery analysis or end-users in agriculture or rescue services). These users interact with the application's interface, performing tasks relevant to the platform's purpose, like labeling images or reviewing AI-generated annotations. They then provide feedback via a structured questionnaire focusing on usability aspects—such as ease of use, intuitiveness, and clarity. This feedback helps refine the user interface to ensure it meets user expectations and requirements.

- **Demonstration Video**: The video will serve as an onboarding tool, guiding users on how to navigate and utilize the platform effectively. It will highlight key features, demonstrate a labeling task from start to finish, and showcase any unique functionalities, such as model-assisted annotations or accuracy checks. The video can be used in training materials or as a quick tutorial accessible directly from the platform to help new users become familiar with the tool's workflow.

- **Formal Proof of Convergence for Labeled Images**: This proof aims to establish that labeled images converge toward a standard of accuracy and consistency. This involves statistical or mathematical validation showing that repeated labeling by different users (or with the assistance of AI) leads to a stable and reliable dataset over time. Proving convergence is crucial in ensuring that the labeling quality remains high, even as new users interact with the platform. This might include metrics such as inter-annotator agreement or AI confidence scores, contributing to the credibility and reliability of the labeled dataset.

2.4 2.4 Relevant Documentation

Created BibTeX entries for our documents and within those entries we have included a hyperlink to the documents. As listed above. Here is a sample of the references for all of the documents. Future references are also included in this sample.

- Development Plan - [1]

- Module Guide - [2]
- Module Interface Specification - [3]
- Verification and Validation Plan - [6]
- Problem Statement - [4]
- Software Requirements Specification - [5]

By referencing these documents, we ensure that our V&V efforts are comprehensive and aligned with the project's goals and requirements. Each document provides specific insights that help us design effective verification and validation activities:

- **Problem Statement** ([4]): Outlines the core challenges and objectives of the Orbit Watch project, providing context and motivation for development. It helps align our V&V activities with the original goals and ensures that the software addresses the identified needs.
- **Development Plan** ([1]): Details the project timeline, milestones, and resource allocation. It assists in scheduling V&V activities appropriately within the development lifecycle, ensuring they are integrated at optimal points for maximum effectiveness.
- **Software Requirements Specification (SRS)** ([5]): Defines all functional and non-functional requirements of Orbit Watch. It serves as the foundation for developing test cases to verify that the software meets all specified requirements and performs as intended.
- **Module Guide (MG)** ([2]): Provides an overview of the system's modular architecture, including the decomposition into modules and their relationships. This information is vital for planning integration testing and ensuring that the modules interact correctly.
- **Module Interface Specification (MIS)** ([3]): Details the interfaces for each module, specifying input and output parameters. This is essential for verifying that modules communicate correctly and adhere to their specified interfaces, which is critical for system reliability.

- **Verification and Validation Plan (V&V Plan)** ([6]): Outlines the strategies, methodologies, and resources allocated for the V&V activities. It guides our testing efforts and ensures that all aspects of the system are adequately verified and validated according to the project's objectives.

By utilizing these documents, our V&V efforts are thorough and focused on delivering a reliable and accurate platform for satellite imagery analysis.

3 Plan

This section describes the team, verification plans for the system design and documentation, use of automated testing tools, and the validation plan of the software following implementation.

3.1 Verification and Validation Team

When not listed as lead, members of core team will support through team discussion and implementation of feedback.

Name	Role	Responsibilities
Mathew Petronilho	Core team	**SRS** Lead review of SRS **Implementation** Support review of implementation Review Implementation for coding standards, comment quality
Oleg Glotov	Core team	**VnV Plan** Support review of VnV Plan Review VnV Plan for formatting and grammar errors **Implementation** Lead review of Implementation
Kartik Chaudhari	Core team	**Design** Support review of Design Review diagrams and documents for correct notation, formatting and grammar **VnV Plan** Lead review of VnV Plan **Implementation** Support review of implementation
Kyle McMaster	Core team	**SRS** Check SRS for formatting and grammar errors Support review of SRS **Design** Lead review of design Ensure team is following standard design principles
Dr. Swati Mishra	Project Supervisor	Validate all team docs in structured review
Capstone Team 10	Primary Reviewers	Validate all team docs in async review through Git issues

3.2 SRS Verification Plan

The SRS will be verified through several channels. First, the document will be reviewed by another capstone team. This will help the team identify issues which are obscured to the team, due to the additional time they have spent thinking about the project. We expect this feedback to generally con-

sist of omitted definitions or unstated assumptions. Since the team is more familiar with the project, it is likely that we have some information which is obvious to the team, but is necessary to define for others. The team will collect feedback from our primary reviewers in the issue tracker.
The SRS will also be reviewed by our supervisor in a structured review meeting. The team will guide the review with the following checklist:

Constraints & Assumptions
The following sections are informed by the constraints & assumptions. Therefore, it is critical that the these are verified first.

- ☐ All constraints are correct and necessary
- ☐ All constraints are unambigious
- ☐ All constraints which should be present are present
- ☐ All constraints are verifiable
- ☐ All assumptions are correct and necessary
- ☐ All assumptions are unambigious
- ☐ All assumptions which should be present are present
- ☐ All assumptions are verifiable

Data Model
The data model affects how the system will be decomposed in the future design. This in turn affects the requirements, so it should be verified next.

- ☐ Data model is correct
- ☐ Data model is complete
- ☐ Each element of data dictionary is correctly described. There are no extra or missing attributes.
- ☐ Elements of data dictionary are unambigious

Functional Requirements
This section describes the functionality of the system. This will be useful for understanding the context of the NFRs.

- ☐ All requirements are correct and necessary
- ☐ All requirements are unambigious
- ☐ All requirements which should be present are present
- ☐ All requirements are verifiable
- ☐ All requirements are feasible
- ☐ All requirements are traceable

Non-Functional Requirements
The NFRs have now been properly introduced by the other sections and should now be assessed.

- ☐ All requirements are correct and necessary
- ☐ All requirements are unambigious
- ☐ All requirements which should be present are present
- ☐ All requirements are verifiable
- ☐ All requirements are feasible
- ☐ All requirements are traceable

General
Each of the sections reviewed should also be monitored for the following criteria.

- ☐ Document contains properly formatted title, table of contents, references and all necessary sections
- ☐ Tables and figures are correctly formatted
- ☐ No grammar errors are present
- ☐ Each required section is present
- ☐ All requirements are feasible

3.3 Design Verification Plan

Upon completion of the System Design, the team will verify the design. This verification will involve reviews from the primary reviewers, in the same manner as described above. This again will provide fresh prospective on the design, and help identify any omitted or underexplained information. The verification will continue with a systematic review with The project supervisor using the following checklist:

Class Decomposition

- ☐ All functional requirements are covered by a class/subsystem
- ☐ All non-functional requirements are covered by a class/subsystem
- ☐ All classes are correctly sized. That is, each class should be concerned with one purpose
- ☐ When appropriate, interfaces and abstract classes are used
- ☐ All class attributes are complete and necessary
- ☐ When appropriate, design patterns are used to improve design clarity
- ☐ Function permissions are appropriate for all classes
- ☐ Classes cannot be simplified without degrading the understanability of the system

General

- ☐ The types of all attributes are listed
- ☐ The argument and return types of all functions are listed
- ☐ Correct notation is used to describe class relationships
- ☐ Class diagram is legible

3.4 Verification and Validation Plan Verification Plan

This document must be verified and validated, to ensure the validation and verification of other artifacts is correct. Like other documents, this will include review from our primary reviewers, as well as a review with the project supervisor. The review with the project supervisor will be guided by the folllowing checklist:

Testing Plan

- ☐ Testing plan includes a checklist for each artifact
- ☐ Checklist items are unambigious
- ☐ Checklist contains all recessary items
- ☐ Checklist does not contain unnecessary items
- ☐ Plan includes review from multiple parties
- ☐ Plan includes method for collecting feedback on artifacts
- ☐ Plan is verifiable

Tests

- ☐ All functional requirements are covered by one or more tests
- ☐ All non-functional requirements are covered by one or more tests
- ☐ All 'units' are sufficiently covered by one or more tests
- ☐ Tests are unambigious
- ☐ Tests are verifiable
- ☐ Tests are repeatable

3.5 Implementation Verification Plan

As part of the implementation verification plan, the team will conduct a series of static tests. One of which, will be a code walkthrough with members of the team. Before submitting Rev 0 and Rev 1, the members of the team will schedule time to walk through other members of the team through the code they have written. This will be an opportunity for the team to look for code quality and adherence to standards, correctness, and alignment to the system design. During this walkthrough, we expect that the lead developer of the code being displayed will explain the overall flow and control structures. In doing so, we expect they will find the errors in their own code.

The team will also use a suite of tests, which is described in section 4.

The team will used static analyzers to verify adherence to coding standards to perform type checking. Please refer to section 7.2.2 and 7.2.3 in the Development Plan found here.

3.6 Automated Testing and Verification Tools

Please refer to section 7.2.2 and 7.2.3 in the Development Plan found here.

3.7 Software Validation Plan

As discussed in section 6.2 of the Problem Statement and Goals, found here, we will validate our software through user testing. More specifically, we will conduct testing of the user interface with peers, who will fill in a survey containing both qualitative and quantitative questions. This survery will be used to track progress on the usability of the software by comparing results collecting during different iterations of the user interface. See the appendix for a sample of the user survey.

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

FR0: Customer Account Creation Test

1. **Test ID**: T-FR0
2. **Control**: **Automated**
3. **Initial State**: The system is accessible, and no user account currently exists for the test user.
4. **Input**:
   - Customer provides valid personal information:
     - **Name**: "Alice Smith"
     - **Email**: `alice.smith@example.com`
     - **Password**: "StrongPassword!2021"
   - Customer agrees to the system's privacy policy.
5. **Output**:
   - **Expected Result**:
     - A new user account is created.
     - Account information is securely stored in the database.
     - The customer is redirected to the login page with a success message.
6. **Test Case Derivation**:
   - Based on the requirement that customers must have an account to access system services. This test verifies that with valid input and acceptance of terms, the account creation process is successfully executed and persists data appropriately.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the account creation page using a testing tool like Selenium WebDriver.

(b) **Step 2**: Fill in the registration form fields with the provided valid data.

(c) **Step 3**: Check the agreement box for the privacy policy.

(d) **Step 4**: Submit the registration form.

(e) **Step 5**: Verify that the response indicates successful account creation (e.g., success message displayed).

(f) **Step 6**: Confirm redirection to the login page.

(g) **Step 7**: Query the database to verify that a new user record exists with the email `alice.smith@example.com`.

(h) **Step 8**: Ensure the password is hashed and not stored in plain text.

- **Cleanup**:
  - After verification, delete the test user account from the database to maintain a clean state for future tests.

FR1: Customer Authentication Test

1. **Test ID**: T-FR1
2. **Control**: **Automated**
3. **Initial State**: An existing user account with the following credentials:
   - **Email**: `user.test@example.com`
   - **Password**: "TestPass#123"
4. **Input**:
   - Customer enters the correct login credentials:
     - **Email**: `user.test@example.com`
     - **Password**: "TestPass#123"
5. **Output**:
   - **Expected Result**:
     - Customer is successfully authenticated.

- Access to privileged information (e.g., user dashboard) is granted.
- A session token or cookie is established for the user session.

6. **Test Case Derivation**:

   - Ensures that only authenticated customers can access the system, satisfying security requirements.

7. **How Test Will Be Performed**:

   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the login page.
     (b) **Step 2**: Input the correct email and password.
     (c) **Step 3**: Submit the login form.
     (d) **Step 4**: Verify redirection to the user dashboard or home page.
     (e) **Step 5**: Check for the presence of privileged information on the page.
     (f) **Step 6**: Verify that a session token or authentication cookie has been set.
     (g) **Negative Test**:
       i. **Step 7**: Attempt to access the dashboard with incorrect credentials to ensure access is denied.
   - **Session Validation**:
     (a) **Step 8**: Use the session token to access a secure API endpoint to confirm authentication persistence.

FR2: Customer Account Modification Test

1. **Test ID**: T-FR2
2. **Control**: **Manual**
3. **Initial State**: Customer is logged in and has access to their account information page.
4. **Input**:

- Customer updates personal information fields:
  - **Address**: "123 New Street, Cityville"
  - **Phone Number**: "555-1234"
  - **Profile Picture**: Uploads a new image file.

5. **Output**:
   - **Expected Result**:
     - Updated personal information is saved and reflected in the database.
     - The user receives a confirmation message indicating successful update.

6. **Test Case Derivation**:
   - Confirms that authenticated customers can modify their information, maintaining data accuracy and relevance.

7. **How Test Will Be Performed**:
   - **Manual Steps**:
     (a) **Step 1**: Log in to the customer account.
     (b) **Step 2**: Navigate to the account settings or profile page.
     (c) **Step 3**: Change the address and phone number to the new values.
     (d) **Step 4**: Upload a new profile picture.
     (e) **Step 5**: Save the changes.
     (f) **Step 6**: Verify that the changes are displayed correctly on the profile page.
     (g) **Step 7**: Check the database to ensure the new information is updated.
     (h) **Step 8**: Log out and log back in to confirm that the changes persist.

FR3: Payment Processing Test

1. **Test ID**: T-FR3
2. **Control**: **Automated**
3. **Initial State**: Customer is logged in and ready to make a payment for a service request.
4. **Input**:
   - Valid payment information:
     - **Credit Card Number**: "4111 1111 1111 1111" (Test Visa number)
     - **Expiry Date**: "12/25"
     - **CVV**: "123"
     - **Billing Address**: Matches the address on file.
5. **Output**:
   - **Expected Result**:
     - Payment is processed successfully.
     - A confirmation receipt is generated and emailed to the customer.
     - The service request status is updated to "Paid" or equivalent.
6. **Test Case Derivation**:
   - Verifies payment integration, ensuring that authenticated customers can complete transactions securely.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the payment page for the pending service request.
     (b) **Step 2**: Input the valid payment details.
     (c) **Step 3**: Submit the payment form.

(d) **Step 4**: Mock the payment gateway response if using a sandbox environment.

(e) **Step 5**: Verify that the system displays a payment success message.

(f) **Step 6**: Check that a confirmation receipt is generated and sent to the customer's email.

(g) **Step 7**: Verify that the service request status is updated appropriately in the database.

- **Security Verification**:

  (a) **Step 8**: Ensure that sensitive payment information is not stored in plain text and complies with PCI DSS standards.

FR4: Service Request Submission Test

1. **Test ID**: T-FR4
2. **Control**: **Manual**
3. **Initial State**: Customer is logged in and has a confirmed payment.
4. **Input**:
   - Customer fills out the service request form with necessary details:
     - **Service Type**: "Image Analysis"
     - **Description**: "Analysis of satellite images for deforestation."
     - **Preferred Completion Date**: "2023-12-31"
5. **Output**:
   - **Expected Result**:
     - Service request is accepted and logged in the system.
     - Customer receives a confirmation message and request ID.
6. **Test Case Derivation**:
   - Ensures that the system accepts valid requests from authenticated customers.

7. **How Test Will Be Performed**:

    - **Manual Steps**:
        (a) **Step 1**: Navigate to the new service request page.
        (b) **Step 2**: Fill in the form with the input data.
        (c) **Step 3**: Submit the form.
        (d) **Step 4**: Verify that a confirmation message with a unique request ID is displayed.
        (e) **Step 5**: Check the database to confirm that the service request is logged with correct details.
        (f) **Step 6**: Ensure that the service request appears in the customer's list of active requests.

FR5: Service Report Delivery Test

1. **Test ID**: T-FR5
2. **Control**: **Automated**
3. **Initial State**: Customer is logged in and has a completed service request.
4. **Input**:
    - Customer navigates to the "My Reports" section after being notified of service completion.
5. **Output**:
    - **Expected Result**:
        – The service report is available for viewing and download.
        – Report contents are accurate and correspond to the service request.
6. **Test Case Derivation**:
    - Confirms that customers receive reports on completed services, fulfilling the system's purpose.

7. **How Test Will Be Performed**:

    - **Automated Test Script Execution**:
        (a) **Step 1**: Simulate the completion of a service request in the system (can be mocked or set up in a test environment).
        (b) **Step 2**: Log in as the customer.
        (c) **Step 3**: Navigate to the "My Reports" or equivalent section.
        (d) **Step 4**: Verify that the completed service report is listed.
        (e) **Step 5**: Open the report and check that it loads correctly.
        (f) **Step 6**: Validate that the report content matches the expected output based on the service provided.
        (g) **Step 7**: Attempt to download the report and ensure the file is intact and accessible.

FR6: Image Upload Test

1. **Test ID**: T-FR6
2. **Control**: **Manual**
3. **Initial State**: Customer is logged in with an active service request requiring image uploads.
4. **Input**:
    - Customer uploads multiple image files:
        – `Image1.jpg`: 2 MB
        – `Image2.png`: 3 MB
        – `Image3.tif`: 5 MB
5. **Output**:
    - **Expected Result**:
        – All images are successfully uploaded and stored.
        – Images are correctly linked to the specific service request.
        – Customer receives an upload success message.

6. **Test Case Derivation**:

    - Ensures that customers can upload images for requested services, fulfilling the fit criterion.

7. **How Test Will Be Performed**:

    - **Manual Steps**:
        (a) **Step 1**: Navigate to the image upload section of the active service request.
        (b) **Step 2**: Select the image files for upload.
        (c) **Step 3**: Initiate the upload process.
        (d) **Step 4**: Monitor progress indicators for each file.
        (e) **Step 5**: Verify that a success message is displayed after upload completion.
        (f) **Step 6**: Check the service request details to ensure images are listed.
        (g) **Step 7**: Confirm that the images are stored in the correct directory or database location.

FR7: Satellite Image Request Test

1. **Test ID**: T-FR7

2. **Control**: **Automated**

3. **Initial State**: Customer is logged in with an active service request that requires satellite images.

4. **Input**:

    - Geographic coordinates:
        – **Latitude**: 37.7749° N
        – **Longitude**: 122.4194° W (San Francisco, CA)
        – **Date Range**: “2023-01-01” to “2023-01-31”

5. **Output**:

- **Expected Result**:
  - The system retrieves and stores satellite images corresponding to the provided coordinates and date range.
  - Customer is notified of successful retrieval.

6. **Test Case Derivation**:

   - Confirms the system can source images using specified geographical data, aiding in analysis.

7. **How Test Will Be Performed**:

   - **Automated Test Script Execution**:
     (a) **Step 1**: Input the geographic coordinates and date range into the request form.
     (b) **Step 2**: Submit the request.
     (c) **Step 3**: Mock the satellite data provider's API response if necessary.
     (d) **Step 4**: Verify that the system processes the input without errors.
     (e) **Step 5**: Check that the images are retrieved and stored in the system.
     (f) **Step 6**: Confirm that the images are linked to the correct service request.
     (g) **Step 7**: Validate that the customer receives a notification or confirmation message.

FR8: Service Request Failure Alert Test

1. **Test ID**: T-FR8

2. **Control**: **Manual**

3. **Initial State**: Customer has initiated a service request that cannot be fulfilled due to invalid parameters.

4. **Input**:

- Service request with unfulfillable criteria:
  - **Service Type**: "Image Analysis"
  - **Geographic Coordinates**: Invalid coordinates (e.g., Latitude: 95° N)

5. **Output**:
   - **Expected Result**:
     - Customer receives an alert indicating that the service request cannot be processed.
     - An explanation of the failure is provided.
6. **Test Case Derivation**:
   - Ensures customers are promptly notified when requests cannot be processed, enhancing user experience.
7. **How Test Will Be Performed**:
   - **Manual Steps**:
     (a) **Step 1**: Attempt to submit the service request with invalid coordinates.
     (b) **Step 2**: Observe the system's response.
     (c) **Step 3**: Verify that an alert or error message is displayed to the customer.
     (d) **Step 4**: Ensure the message clearly explains the reason for failure.
     (e) **Step 5**: Check that no service request is logged in the system for the invalid input.

FR9: Labeler Account Creation Test

1. **Test ID**: T-FR9
2. **Control**: **Automated**
3. **Initial State**: No labeler account exists for the test user in the system.

4. **Input**:
    - Labeler provides required account information:
        - **Name**: "Bob Labeler"
        - **Email**: `bob.labeler@example.com`
        - **Password**: "LabelerPass789!"
        - **Expertise Area**: "Satellite Image Annotation"
5. **Output**:
    - **Expected Result**:
        - A new labeler account is created and securely stored.
        - Labeler is prompted to complete any additional onboarding steps.
6. **Test Case Derivation**:
    - Ensures labelers can create accounts to access the system, which is essential for workflow.
7. **How Test Will Be Performed**:
    - **Automated Test Script Execution**:
        (a) **Step 1**: Navigate to the labeler registration page.
        (b) **Step 2**: Fill in the registration form with the input data.
        (c) **Step 3**: Submit the form.
        (d) **Step 4**: Verify that a success message is displayed.
        (e) **Step 5**: Check the database to ensure the new labeler account exists with the correct details.
        (f) **Step 6**: Confirm that the password is stored securely (hashed).
    - **Cleanup**:
        - After verification, delete the test labeler account from the database to maintain a clean state for future tests.

FR10: Labeler Authentication Test

1. **Test ID**: T-FR10
2. **Control**: **Automated**
3. **Initial State**: Labeler account exists with credentials:
   - **Email**: `labeler.test@example.com`
   - **Password**: "LabelerSecure!2022"
4. **Input**:
   - Labeler enters correct login credentials.
5. **Output**:
   - **Expected Result**:
     - Labeler is authenticated successfully.
     - Access to the labeler dashboard is granted.
6. **Test Case Derivation**:
   - Confirms authentication mechanisms for labelers, maintaining system security.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the labeler login page.
     (b) **Step 2**: Input the correct credentials.
     (c) **Step 3**: Submit the login form.
     (d) **Step 4**: Verify redirection to the labeler dashboard.
     (e) **Step 5**: Check for access to labeler-specific features and data.
     (f) **Negative Test**:
         i. **Step 6**: Attempt login with incorrect credentials to ensure authentication fails appropriately.

FR11: Labeler Account Modification Test

1. **Test ID**: T-FR11
2. **Control**: **Manual**
3. **Initial State**: Labeler is logged in and on the account settings page.
4. **Input**:
   - Update personal information:
     - **Expertise Area**: Add "Aerial Photography Annotation"
     - **Contact Number**: "555-6789"
5. **Output**:
   - **Expected Result**:
     - Personal information is updated and stored in the database.
     - Labeler receives a confirmation of successful update.
6. **Test Case Derivation**:
   - Ensures labelers can maintain current information, which is crucial for assignment matching.
7. **How Test Will Be Performed**:
   - **Manual Steps**:
     (a) **Step 1**: Navigate to account settings.
     (b) **Step 2**: Modify the expertise area and contact number.
     (c) **Step 3**: Save the changes.
     (d) **Step 4**: Verify that the updated information is displayed.
     (e) **Step 5**: Check the database for updated records.
     (f) **Step 6**: Log out and log back in to confirm persistence.

FR12: Labeler Earnings Transfer Test

1. **Test ID**: T-FR12
2. **Control**: **Automated**
3. **Initial State**: Labeler is logged in with available earnings exceeding the minimum transfer threshold.
4. **Input**:
   - Transfer request to linked banking platform:
     - **Amount**: Total available earnings.
     - **Bank Account Details**: Pre-verified and linked.
5. **Output**:
   - **Expected Result**:
     - Earnings are transferred successfully.
     - Transaction record is created.
     - Labeler receives confirmation and updated earnings balance.
6. **Test Case Derivation**:
   - Ensures labelers are compensated accurately and promptly, critical for system trust.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the earnings or wallet section.
     (b) **Step 2**: Initiate a transfer request for the total available amount.
     (c) **Step 3**: Mock the banking API response if necessary.
     (d) **Step 4**: Verify that the system processes the transfer without errors.
     (e) **Step 5**: Confirm that the earnings balance is updated to reflect the transfer.

(f) **Step 6**: Check that a transaction record is logged in the database.

(g) **Step 7**: Validate that a confirmation message or email is sent to the labeler.

FR13: Image Annotation Test

1. **Test ID**: T-FR13
2. **Control**: **Manual**
3. **Initial State**: Labeler is logged in with images assigned for annotation.
4. **Input**:
   - Labeler annotates an image using the provided tools:
     - Draws bounding boxes around objects.
     - Adds classification labels to each object.
     - Saves the annotation.
5. **Output**:
   - **Expected Result**:
     - Annotated image is stored in the system.
     - Annotation data is correctly linked to the image and service request.
     - Labeler receives confirmation of successful submission.
6. **Test Case Derivation**:
   - Confirms annotation capabilities, essential for image analysis services.
7. **How Test Will Be Performed**:
   - **Manual Steps**:

     (a) **Step 1**: Access the image annotation interface.

     (b) **Step 2**: Use annotation tools to mark objects.

(c) **Step 3**: Assign appropriate labels to each annotation.
(d) **Step 4**: Save and submit the annotations.
(e) **Step 5**: Verify that a confirmation is received.
(f) **Step 6**: Check the database to ensure annotations are stored correctly.
(g) **Step 7**: Attempt to re-access the annotated image to confirm annotations persist.

FR14: Consolidated Annotation Report Test

1. **Test ID**: T-FR14
2. **Control**: **Automated**
3. **Initial State**: All required labeler annotations are complete for a service request.
4. **Input**:
   - System triggers consolidation process for annotations.
5. **Output**:
   - **Expected Result**:
     - Consolidated report is generated if label accuracy meets the predefined threshold.
     - Report is stored and made accessible to the customer.
6. **Test Case Derivation**:
   - Verifies that the system consolidates annotations accurately, ensuring quality results for customers.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Simulate completion of all annotations for a service request.

(b) **Step 2**: Initiate the consolidation process (this may be automatic upon annotation completion).

(c) **Step 3**: Verify that the system calculates label accuracy and compares it against the threshold.

(d) **Step 4**: Confirm that if the accuracy meets or exceeds the threshold, the report is generated.

(e) **Step 5**: Check that the report contains consolidated data from all annotations.

(f) **Step 6**: Ensure the report is stored and accessible to the customer linked to the service request.

(g) **Step 7**: Validate that notifications are sent to the customer about the report availability.

- **Edge Case Testing**:

  (a) **Step 8**: Repeat the test with annotations that do not meet the accuracy threshold to verify that the report is not generated and appropriate actions are taken (e.g., re-annotation request).

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Operational and Environmental

Release Tests

1. Road Map Consistency: T-OE6

   Type: Manual, Static

   Initial State: Application has a release road map that is publicly accessible.

   Input/Condition: Team member conducts a review.

   Output/Result: At least MIN_ON_TIME_MILESTONE% of the listed milestones have been met on time.

   How test will be performed: The team member looks over the road map and cross references the completion date of milestones to the dates listed in the road map.

2. Beta Testing: T-OE7

   Type: Dynamic, Exploratory

   Initial State: Beta version of application is deployed and accessible

   Input/Condition: At least BETA_TESTERS beta testers are provided access to use the application.

   Output/Result: Feedback on any bugs, navigation issues, or aesthetic problems is provided. Less than MAX_BUGS_FOUND bugs are found.

   How test will be performed: Testers will be recruited and identified. They will be from fields of interest that include scientists, labelers, and domain experts. Then, the beta testing environment will be set up and the url will be distributed to the testers along with any other set up resources. Specific tasks are provided for testers to complete that focus on the annotation tools, sign up process and project creation. Feedback will be collected through direct comments from the tester.

3. Regression Testing: T-OE8

   Type: Dynamic, Automated

Initial State: Application is deployed.

Input/Condition: Run regression test suite, consisting of unit tests.

Output/Result: All regression tests are passed.

How test will be performed: An automated script with regression tests will run when updates are made to the production build.

4.2.2 Maintainability and Support

Maintenance Tests

1. Ease of Change: T-MS0

   Type: Manual, Static

   Initial State: Application's source repository contains complete documentation.

   Input/Condition: Competent software developer who has not previously worked on the app reviews documentation and attempts to perform tasks.

   Output/Result: The developer can easily make a minor update to a specified part of the application.

   How test will be performed: Give the developer time to read through the documentation. Give them a maintenance task, such as updating the size of the title font to 20px. Observe them and document how long it takes them and if they encountered any troubles.

4.2.3 Security

Access Tests

1. Logged Out Permissions: T-SE0

   Type: Manual, Dynamic, White-box

   Initial State: Application is deployed.

   Input/Condition: Tester who is not signed in tries to access application paths for project creation and image labeling (Ex. /projects or /label).

Output/Result: The tester is denied access to these paths and is told to sign in.

How test will be performed: On the deployed application, the tester will visit all possible paths as a logged out user.

2. Labeler Permissions: T-SE1

   Type: Manual, Dynamic, White-box

   Initial State: Application is deployed.

   Input/Condition: Tester who is signed in as a labeler tries to access application paths for project creation.

   Output/Result: The tester is denied access to these paths. However, the tester has access to paths related to image labeling.

   How test will be performed: On the deployed application, the tester will visit all possible paths as a labeler.

3. Invalid Email Format: T-SE2

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Email with invalid format, such as an empty string or a string missing '@', is entered.

   Output/Result: Application rejects email and tells the user that the email format is wrong.

   How test will be performed: A unit test will be performed where the input is entered into the email section of the registration form.

4. Duplicate Email: T-SE3

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Email that is already in database is entered.

Output/Result: Application rejects email and tells the user that the email is in use.

How test will be performed: A unit test will be performed where the input is entered into the email section of the registration form.

5. Invalid Password Format: T-SE4

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Password with invalid format, such as an empty string or a string with no numbers, is entered.

   Output/Result: Application rejects password and tells the user what requirements they have not met.

   How test will be performed: A unit test will be performed where the input is entered into the password section of the registration form.

6. System Error: T-SE5

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Purposely invoke a system failure, and attempt to perform an action such as a label submission.

   Output/Result: Application provides an error message on the user interface. The database has not changed in anyway.

   How test will be performed: Go on to the application, start a labeling task, purposely disconnect from the internet, and try to submit a labeled image.

Integrity Tests

1. Duplicate Entries: T-SE6

   Type: Manual, Dynamic

Initial State: Database is deployed.

Input/Condition: Duplicate database entry is inserted into the database.

Output/Result: Database has only one of the inputted entry and the duplicate has been removed.

How test will be performed: Attempt to insert the same entry twice into the database through the database UI.

Privacy Tests

1. Encrypted User Data: T-SE7

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester registers an account.

   Output/Result: All sensitive user data that is stored in the database is encrypted.

   How test will be performed: Tester will create a new account, then check the corresponding user entry in the database and see if the sensitive information is encrypted.

2. Encrypted Payments: T-SE8

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester enters sample payment details to pay for a labeling project that has been created.

   Output/Result: These details are encrypted and can not be read through packet analyzers. The amount in the request can not be modified by an adversary.

   How test will be performed: Tester will enter sample payment details, and submit their payment. Using a packet analyzer (such as Wireshark), packets from this request will be looked at to ensure all information is encrypted.

Immunity Tests

1. SQL Injection: T-SE9

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: A malicious SQL statement is entered into a text field.

   Output/Result: The system raises an error telling the user that it is invalid.

   How test will be performed: Tester will enter a SQL statement such as "{valid email}'–" into an input such as the email input. This example has the potential to bypass a password check by commenting out the rest of the SQL query. The tester will check that when this statement is entered, the system gives feedback that it is invalid.

4.2.4 Cultural

Language Tests

1. Support of Different Languages: T-CU0

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester selects a language from a list of available languages.

   Output/Result: All text on the website is translated and displayed in the selected language.

   How test will be performed: Tester will check that the language selection list is accessible, and that the most popular languages are included. When a language is selected, the tester will check that the translation has been applied and there is no untranslated or gibberish text. This can be checked for each language.

4.2.5 Compliance

Financial Tests

1. Compliant Payment Process: T-CO0

   Type: Manual, Static

   Initial State: Application is deployed.

   Input/Condition: Qualified Security Assessor (QSA) assesses the application.

   Output/Result: They determine that it meets the PCI-DSS standard.

   How test will be performed: A QSA will be found and contacted to perform an assessment. The QSA will be shown all parts of the application that deal will financial transactions and will be able to make a determination on if it meets the standard.

Legal Tests

1. System Availability: T-CO1

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester changes country they are accessing the application from using a tool such as a VPN.

   Output/Result: Application is blocked in countries facing economic sanctions by the Government of Canada.

   How test will be performed: A list of the countries facing economic sanctions by Canada will be compiled. Then, the tester will simulate that they are accessing the application from these countries, and ensure it is unreachable.

2. Taxes: T-CO2

   Type: Manual, Dynamic

   Initial State: Application is deployed.

Input/Condition: Tester redeems a cash balance.

Output/Result: If the cash balance exceeds a threshold, a tax form will be issued.

How test will be performed: Tester creates a test account with an account balance over the threshold. When they withdraw, they check that a tax form has been emailed to the email associated with the account. The tax form should reflect the withdrawal balance.

3. Project Availability: T-CO3

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester changes country they are accessing the application from using a tool such as a VPN.

   Output/Result: Specific project is not shown.

   How test will be performed: A project will be specified to only be distributed in a specific country. Then, the tester will simulate that they are accessing the application from other countries, and ensure the project does not show up.

4.2.6 User Documentation and Training

1. Helpfulness of User Aids: T-UDT0

   Type: Manual, Dynamic

   Initial State: Application is deployed with all help features. Tutorials and user documentation have been created.

   Input/Condition: Users attempt to complete a basic labeling task using only the platform's built-in help resources (help system, quick start guide, tutorials and contextual tooltips).

   Output/Result: At least MIN_USER_HELP_SATISFACTION% of users who used a help feature found that feature helpful. With the assistance of the help tools, the user was able to perform the task within MAX_TASK_TIME minutes.

How test will be performed: The purpose of the platform will be explained to the users and the built-in help features will be shown. Then, the labeling task will be given to them. Time to complete task is observed and the help tools they use are recorded. Participants will then fill out a usability survey, which can be viewed in the appendix.

2. Usefulness of Sandbox: T-UDT1

   Type: Automatic, Dynamic

   Initial State: Application is deployed with all help features. Tutorials and user documentation has been created.

   Input/Condition: A new user has accessed the platform.

   Output/Result: At least MIN_PRACTICE_USAGE% of new users utilize the practice environment, with self-assessment scores indicating an average improvement of IMPROVE_IN_ACC% in labeling accuracy over their first three attempts.

   How test will be performed: Practice environment utilization will be tracked by the application. Improvement in accuracy will also be tracked. If the metrics meet or succeed our thresholds, then we can conclude the sandbox is useful.

...

4.3 Traceability Between Test Cases and Requirements

FR #	Test Case														
	T-FR0	T-FR1	T-FR2	T-FR3	T-FR4	T-FR5	T-FR6	T-FR7	T-FR8	T-FR9	T-FR10	T-FR11	T-FR12	T-FR13	T-FR14
FR0	X														
FR1		X													
FR2			X												
FR3				X											
FR4					X										
FR5						X									
FR6							X								
FR7								X							
FR8									X						
FR9										X					
FR10											X				
FR11												X			
FR12													X		
FR13														X	
FR14															X

NFR #	Test Case

	T-LF0	T-LF1	T-LF2	T-UH0	T-UH1	T-UH2	T-UH3	T-UH4	T-UH5	T-UH6	T-UH7	T-UH8	T-UH9	T-PR0	T-PR1
LF0	X														
LF1		X													
LF2			X												
UH0				X											
UH1					X										
UH2						X									
UH3							X								
UH4								X							
UH5									X						
UH6										X					
UH7											X				
UH8												X			
UH9													X		
PR0														X	
PR1														X	
PR2															X

NFR #	Test Case														
	T-PR1	T-PR2	T-PR3	T-PR4	T-PR5	T-PR6	T-PR7	T-OE0	T-OE1	T-OE2	T-OE3	T-OE4	T-OE5	T-OE6	T-OE7
PR3		X													
PR4			X												
PR5				X											
PR6					X										
PR7	X														
PR8						X									
PR9							X								
OE0								X							
OE1									X						
OE2										X					
OE3											X				
OE4												X			
OE5												X			
OE6													X		
OE7														X	
OE8															X

NFR #	Test Case														
	T-OE8	T-MS0	T-SE0	T-SE1	T-SE2	T-SE3	T-SE4	T-SE5	T-SE6	T-SE7	T-SE8	T-SE9	T-CU0	T-CO0	T-C01
OE9	X														
MR0		X													
SE0			X												
SE1				X											
SE2					X	X									
SE3							X								
SE4								X							
SE5					X	X	X								
SE6								X							
SE7									X						
SE8										X					
SE9											X			X	
SE10												X			
CU0													X		
CO0															X

NFR #	Test Case			
	T-C02	T-C03	T-UDT0	T-UDT1
CO1	X			
CO2		X		
UD0			X	
UD1			X	
UD2			X	
TR0			X	
TR1				X

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case,

explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

[1] OrbitWatch. *Development Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf. McMaster University, 2024.

[2] OrbitWatch. *Module Guide (MG)*. Tech. rep. https://example.com/mg. McMaster University, 2024.

[3] OrbitWatch. *Module Interface Specification (MIS)*. Tech. rep. https://example.com/mis. McMaster University, 2024.

[4] OrbitWatch. *Problem Statement*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf. McMaster University, 2024.

[5] OrbitWatch. *Software Requirements Specification*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/SRS/SRS.pdf. McMaster University, 2024.

[6] OrbitWatch. *Verification and Validation (VnV) Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/VnVPlan/VnVPlan.pdf. McMaster University, 2024.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Parameter	Value	Unit	Description
MIN_ON_TIME_MILESTONE	80	%	Minimum percent of milestones that have been met on time
BETA_TESTERS	50	People	Number of beta testers
MAX_BUGS_FOUND	10	Bugs	Number of software bugs found
MIN_USER_HELP_SATISFACTION	80	%	Minimum percent of users satisfied with help feature
MAX_TASK_TIME	15	Minutes	Maximum time it takes a user to complete a task
MIN_PRACTICE_USAGE	80	%	Minimum percent of new users who have used the practice sandbox
IMPROVE_IN_ACC	20	%	Improvement in accuracy of a user after practicing

6.2 Usability Survey Questions

1. Did you use the help system to aid in completing your task? Yes/No
2. If you answered yes, please rate how useful it was in helping you accomplish your task: 1 (Not Useful) - 5 (Very Useful)

3. Did you use the quick start guide to aid in completing your task? Yes/No

4. If you answered yes, please rate the clarity and helpfulness of it: 1 (Not Helpful) - 5 (Very Helpful)

5. Did you notice the tool-tips or pop-ups providing contextual help as you worked? Yes/No

6. If you answered yes, please rate the clarity and usefulness of the in-app help indicators: 1 (Not Helpful) - 5 (Very Helpful)

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   While working on this deliverable, we found that several things went particularly well for our team. From the very beginning, we laid out a clear plan and divided the tasks based on each person's strengths, which made the workload manageable and played to our individual skills. Getting an early start was a game-changer; it allowed us ample time to really delve into each aspect without feeling rushed. We also benefited from having some spare time due to lighter workloads in our other courses, which we used to do deeper research on each topic. Revisiting our SRS and applying it to our VnV plan challenged us to think critically and solidified our understanding of the project. Our communication was smooth throughout—we checked in regularly and supported each other whenever questions or issues came up. Overall, the combination of good planning, effective collaboration, and the extra time we invested made the process both successful and satisfying.

2. What pain points did you experience during this deliverable, and how did you resolve them?

While working on this deliverable, we did face a few pain points that tested our teamwork and project management skills. One of the main challenges was investigating topics related to our project and tests that we hadn't openly discussed before; this led to some confusion and extra time spent getting everyone on the same page. Maintaining consistency across the entire document was also tricky—different writing styles and interpretations made parts of the doc feel uneven. We encountered build issues as well; some of us had to wait for others' pull requests to merge before we could proceed with our own work, which slowed down our progress. To tackle these problems, we discussed it during our meeting on Monday to openly discuss and clarify the unclear topics, ensuring everyone understood the direction we were heading. We also discussed that why working on documents is extremely important since it's worth a lot of our grade. Ensuring the deliverable met high standards despite any time pressures or setbacks is a key thing we discussed. By addressing these issues head-on, we were able to smooth out the rough patches and keep the project moving forward.

3. 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, Valgrind etc. You should look to identify at least one item for each team member.

   To successfully verify and validate our project of processing high-definition satellite images for segmentation, our team needs to acquire a few specific skills. One team member plans to deepen his understanding of dynamic testing so he can effectively test his segmentation algorithms under various conditions. One of us will focus on static testing techniques, learning to use tools like Valgrind to identify and fix any code issues or inefficiencies. Another team member is going to get up to speed with image processing tools like OpenCV, which will be crucial for validating our segmentation results. The fourth member will work on performance profiling to ensure our application can handle large HD images smoothly without bogging down. We'll all need to become more familiar with handling and processing large satellite datasets, and possibly learn about machine learning model validation since we will be

incorporating AI techniques for most part. By each of us focusing on these areas, we'll collectively cover all the bases needed for thorough verification and validation of our project.

4. 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?

   This is the potential list of what each team member might be doing (subject to changes):

   - Kartik – Dynamic Testing
   - Mathew – Static Testing
   - Kyle – OpenCV
   - Oleg – Performance Profiling

   For dynamic testing, one of us considered enrolling in an online course or jumping straight into writing and running test cases on our code. We chose the hands-on route because some of us learn best by doing, and it lets us contribute directly to the project while improving our skills. The team member focusing on static testing and tools like Valgrind thought about attending a workshop or studying the documentation and applying it themselves. They decided to dive into the documentation and start using Valgrind on our code, preferring self-paced learning and immediate application. Our teammate working with OpenCV weighed taking a formal course versus following online tutorials and building small projects. They opted for the practical approach, believing that working on example projects would help them understand how to apply OpenCV to our needs more effectively. The fourth member, handling performance profiling, considered seeking guidance from a mentor or learning from online resources and applying them directly. They chose to start with online resources to quickly begin identifying and addressing performance issues in our application. Each of us picked the approach that aligns with our learning styles and allows us to make meaningful contributions to the project.

References

[1] OrbitWatch. *Development Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf. McMaster University, 2024.

[2] OrbitWatch. *Module Guide (MG)*. Tech. rep. https://example.com/mg. McMaster University, 2024.

[3] OrbitWatch. *Module Interface Specification (MIS)*. Tech. rep. https://example.com/mis. McMaster University, 2024.

[4] OrbitWatch. *Problem Statement*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf. McMaster University, 2024.

[5] OrbitWatch. *Software Requirements Specification*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/SRS/SRS.pdf. McMaster University, 2024.

[6] OrbitWatch. *Verification and Validation (VnV) Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/VnVPlan/VnVPlan.pdf. McMaster University, 2024.