System Verification and Validation Plan for CXR

Team 27, Neuralyzers
Ayman Akhras
Nathan Luong
Patrick Zhou
Kelly Deng
Reza Jodeiri

October 31, 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

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

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

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select,

you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Tests

4.1 Tests for Functional Requirements

This section contains the tests for the Functional Requirements. The subsections for these tests were created based on the Functional Requirements listed in the SRS document. Each test was created according to the Fit Criterion of the requirements they were covering. Traceability for these requirements and tests can be found in the traceability matrix.

4.1.1 X-ray Image Input Tests

This subsection covers Requirement #1 of the SRS document by testing that the system is able to accept chest X-ray images as input from authorized

users, including healthcare professionals and patients.

1. **test-FR1-1**

   Control: Automatic

   Initial State: The system is operational, and the user is logged in as an authorized healthcare professional.

   Input: Uploading a valid chest X-ray image in DICOM format (`sample_image.dcm`).

   Output: The system accepts the image without errors and displays a confirmation message indicating successful upload.

   Test Case Derivation: Authorized users should be able to upload images in supported formats without issues.

   How the test will be performed: The test will be conducted automatically using the Cypress testing framework.

   - *Step 1*: The automated test script will simulate a user logging into the system as an authorized healthcare professional using valid credentials (`username: doctor_user`, `password: SecurePass123`).
   - *Step 2*: Upon successful authentication, the script will navigate to the image upload section located at `/upload`.
   - *Step 3*: The script will simulate the user clicking the "Upload Image" button and selecting the file `sample_image.dcm` from the directory `./test_data/`.
   - *Step 4*: The script will simulate the user confirming the upload by clicking the "Submit" button.
   - *Step 5*: After the upload action, the script will verify that the system displays a confirmation message on the user interface, such as "Image uploaded successfully."

2. **test-FR1-2**

   Control: Automatic

   Initial State: The system is operational, and the user is logged in as an authorized healthcare professional.

   Input: Uploading a valid chest X-ray image in DICOM format (`sample_image.dcm`).

Output: The system stores the image correctly on the server and the API endpoint returns a success status code.

Test Case Derivation: The backend must accurately store uploaded images and respond appropriately to API requests.

How the test will be performed:

- *Step 1*: The automated test script will perform the same steps as in **test-FR1-1** to upload `sample_image.dcm`.
- *Step 2*: The network traffic will be monitored to capture the API request sent to the server during the image upload.
- *Step 3*: The script will verify that the API request includes the correct payload, ensuring that the file `sample_image.dcm` is included under the correct parameter name (e.g., `image_file`).
- *Step 4*: The server's response to the API request will be checked by the automated test, confirming it returns a success status code (HTTP 200 OK) and a response body with a success message (e.g., {`"message": "Upload successful", "image_id": 12345`}).
- *Step 5*: The script will access the server's storage directory (e.g., `.../images/uploads/`) to verify that `sample_image.dcm` is stored correctly. It will check:
  - The file exists in the expected directory.
  - The file name matches the expected naming convention (e.g., `image_{image_id}.dcm`).
  - The file size and checksum match those of the original file to ensure integrity.

4.1.2 Patient Symptom for diagnosis verification

This subsection covers Requirement #2 of the SRS document by testing that the system enables users to input additional patient symptoms, such as cough, chest pain, or fever.

1. **test-FR2-1**

   Control: Manual

   Initial State: The user has successfully uploaded a chest X-ray image and is on the patient information page.

Input: Entering patient symptoms into the symptom input fields.

Output: The system accepts the symptoms and associates them correctly with the uploaded X-ray image.

Test Case Derivation: The system should allow input of symptoms and link them to the corresponding image.

How the test will be performed:

- *Step 1*: The tester uploads `patient_xray_01.dcm` and waits for the AI model to complete analysis.
- *Step 2*: On the patient information page, the tester enters symptoms such as "cough," "chest pain," and "fever" into the symptom fields.
- *Step 3*: The tester submits the symptoms by clicking the "Save" button.
- *Step 4*: The system processes the symptoms and updates the patient's record.
- *Step 5*: The tester views the generated report to verify that:
  - The symptoms are listed and associated with the diagnosis.
  - The report highlights the consistency between the symptoms and the diagnosis.

4.1.3 AI Model Inferece Tests

This subsection covers Requirement #3 of the SRS document by testing that the system analyzes chest X-ray images using a convolutional neural network (CNN)-based AI model to detect the presence or absence of specific diseases with an accuracy of 85% or higher.

1. **test-FR3-1**

   Type: Functional, Dynamic, Automatic

   Initial State: The AI model is deployed and integrated into the system, ready for analysis.

   Input: A test dataset of 1,000 chest X-ray images with known diagnoses (500 positive cases, 500 negative cases).

Output: The AI model achieves an accuracy of at least 85%, correctly identifying at least 850 out of 1,000 cases.

Test Case Derivation: The AI model must meet or exceed the specified accuracy threshold to be considered effective.

How the test will be performed:

- *Step 1*: The automated test script will load the test dataset from `./test_data/test_set/`.
- *Step 2*: Preprocess each image according to FR17 requirements (e.g., resize to 224x224 pixels, normalize pixel values).
- *Step 3*: Feed the preprocessed images into the AI model in batches (e.g., batch size of 32).
- *Step 4*: Record the AI model's predictions for each image.
- *Step 5*: Compare the predictions with the ground truth labels from `fr3_test_set_labels.csv`.
- *Step 6*: Calculate performance metrics including accuracy, precision, recall, and ROC curves.
- *Step 7*: Verify that the accuracy is at least 85%, using assertions.

4.1.4 Patient Condition Progression Between Scans

This subsection covers Requirement #4 of the SRS document by testing that the system indicates whether a patient's condition has improved, worsened, or remained stable between scans.

1. **test-FR4-1**

   Type: Functional, Dynamic, Manual

   Initial State: The patient has previous chest X-ray images and analysis results stored in the system.

   Input: Uploading a new chest X-ray image for the same patient.

   Output: The system analyzes the new image, compares it with previous scans, and indicates the progression status (improved, worsened, or stable).

   Test Case Derivation: The system should accurately assess and report changes in the patient's condition over time.

How the test will be performed:

- *Step 1*: The tester logs into the system as a healthcare professional and selects the patient with prior scans.
- *Step 2*: Upload the new chest X-ray image `xray_current.dcm`.
- *Step 3*: Wait for the system to analyze the new image and retrieve previous analysis results.
- *Step 4*: The system compares the new image with the previous one using progression algorithms.
- *Step 5*: Review the progression status indicated by the system (e.g., "Condition has worsened").
- *Step 6*: Verify the accuracy of the assessment by comparing it with known and reliable clinical data.

4.1.5 Visual Aid Generation Tests

This subsection covers Requirement #5 of the SRS document by testing that the system generates visual aids by highlighting affected areas on the chest X-ray images.

1. **test-FR5-1**

   Type: Functional, Dynamic, Manual

   Initial State: The system has completed analysis on a chest X-ray image with detected abnormalities.

   Input: Accessing the analysis results and viewing the image.

   Output: The image displays highlighted areas indicating the locations of the detected abnormalities.

   Test Case Derivation: Visual aids help clinicians quickly identify areas of concern on the images.

   How the test will be performed:

   - *Step 1*: The tester logs into the system and navigates to the patient's analysis results for `abnormal_xray.dcm`.
   - *Step 2*: Open the analyzed image with visual aids.

- *Step 3*: Examine the highlighted areas on the image.
- *Step 4*: Compare the highlighted areas with expected abnormalities based on known data.
- *Step 5*: Assess whether the visual aids accurately represent the detected abnormalities.
- *Step 6*: Document observations and any discrepancies.

2. **test-FR5-2**

   Control: Automatic

   Initial State: The system has completed analysis on a chest X-ray image (`normal_patient_xray.dcm`) with no detected abnormalities.

   Input: Accessing the analysis results for `normal_patient_xray.dcm`.

   Output: The image displays no highlights or visual markers.

   Test Case Derivation: The system should not produce false positives by highlighting areas in normal images.

   How the test will be performed:

   - *Step 1*: The automated test script will process `normal_patient_xray.dcm` through the system.
   - *Step 2*: The system analyzes the image and generates an output image.
   - *Step 3*: The script retrieves the output image from `.../images/outputs/`.
   - *Step 4*: The script uses image comparison techniques to compare the output image with the original.
   - *Step 5*: The script checks for any added visual markers or differences.
   - *Step 6*: The script asserts that no highlights are present.

4.1.6 Structured Report Generation Tests

This subsection covers Requirement #6 of the SRS document by testing that the system produces a structured, human-readable report summarizing key findings, disease detection results, and progression status.

1. **test-FR6-1**

   Control: Automatic

   Initial State: The system has completed analysis on a patient's chest X-ray image, and all relevant data is available. The report generation module is integrated and configured to use an LLM API (e.g., BERT).

   Input: Request to generate a structured report summarizing the analysis.

   Output: A human-readable report containing key findings, detected diseases with confidence levels, progression status, and any input symptoms.

   Test Case Derivation: The system should produce comprehensive reports without errors, leveraging LLMs where appropriate.

   How the test will be performed:

   - *Step 1*: The automated test script will initiate the report generation process via the API endpoint `/generate_report` and corresponding patient ID.
   - *Step 2*: The script will monitor the API call to the LLM service, ensuring that:
     - The API token is included and valid.
     - The prompt sent to the LLM is correctly formatted and contains all necessary information.
   - *Step 3*: The script will check the response from the LLM service, verifying:
     - The response status code is success (e.g., HTTP 200 OK).
     - The content includes the structured report.
   - *Step 4*: The script will parse the generated report to ensure it includes:
     - Key findings from the AI analysis.
     - Detected diseases with confidence levels.
     - Progression status compared to previous scans.
     - Input symptoms provided by the user.

- *Step 5*: The script will verify that the report format adheres to predefined templates or standards (e.g., headings, sections).
- *Step 6*: The script will check for any placeholders or missing information that could indicate issues with LLM integration.
- *Step 7*: The test passes if the report is complete, accurate, and correctly formatted; otherwise, it fails.

2. **test-FR6-2**

   Control: Manual

   Initial State: The system is operational, and a report has been generated for a patient using the LLM integration.

   Input: The generated report for review.

   Output: Confirmation that the report does not contain misinformation and accurately reflects the patient's analysis results.

   Test Case Derivation: Reports must be accurate and free from misinformation to ensure patient safety.

   How the test will be performed:

   - *Step 1*: The tester (a qualified healthcare professional) will retrieve the generated report for patient ID `12345`.
   - *Step 2*: The tester will review the report in detail, verifying:
     - All clinical findings are accurately reported.
     - Diagnoses match the AI model's outputs.
     - Confidence levels are correctly stated.
     - Progression status aligns with the comparative analysis.
   - *Step 3*: The tester will cross-reference the report with the raw data and analysis results.
   - *Step 4*: The tester will check for any signs of misinformation, such as incorrect interpretations or unsupported conclusions.
   - *Step 5*: The tester will document any discrepancies or errors found.
   - *Step 6*: The test passes if the report is accurate and free of misinformation; otherwise, it fails.

4.1.7 Patient Data Storage Tests

This subsection covers Requirement #7 of the SRS document by testing that the system securely stores patient data, including images and analysis results, for future reference.

1. **test-FR7-1**

   Control: Automatic

   Initial State: The system's secure database and storage solutions are operational. Access to patient data is regulated by role-based access controls.

   Input: Patient data including images (`test_patient_image.dcm`), analysis results, and reports to be stored.

   Output: Data is securely stored in the database and can be retrieved accurately by authorized users. Unauthorized access is prevented.

   Test Case Derivation: The system must ensure data integrity and security for patient information.

   How the test will be performed:

   - *Step 1*: The automated test script will simulate a user with appropriate permissions uploading patient data:
     - Image file: `test_patient_image.dcm`.
     - Analysis results: Generated by the AI model.
     - Report: Generated as per FR6.
   - *Step 2*: The script will verify that the data is stored in the secure database:
     - Confirm entries in the database tables for patients, images, analyses, and reports.
     - Verify that images are stored in the secure storage directory (e.g., `/secure_storage/patient_images/`).
   - *Step 3*: The script will retrieve the stored data using authorized credentials and compare it with the original inputs to ensure integrity:
     - Check that the image file retrieved matches the original file (e.g., via checksum comparison).

  - Verify that analysis results and reports are accurate.
- *Step 4*: The script will attempt to access the data using unauthorized credentials or as an unauthorized user:
  - Expect access to be denied.
  - Verify that appropriate error messages are displayed.
- *Step 5*: The script will check security measures:
  - Data at rest is encrypted (e.g., verify encryption settings in the database).
  - Access logs are updated with the retrieval attempts.
- *Step 6*: The test passes if data integrity is maintained, authorized access is successful, and unauthorized access is prevented.

4.1.8 Alert Mechanism Tests

This subsection covers Requirement #8 of the SRS document by testing that the system generates alerts to clinicians when significant changes in a patient's condition are detected.

1. **test-FR8-1**

   Control: Automatic

   Initial State: The system has prior scans and analysis results for corresponding patient ID. Alerting mechanisms via email, SMS, and in-app notifications are configured.

   Input: A new scan (`patient_xray_alert.dcm`) showing significant changes exceeding predefined thresholds.

   Output: The system generates and delivers alerts to clinicians indicating the significant change.

   Test Case Derivation: Clinicians should be promptly notified of critical changes in a patient's condition.

   How the test will be performed:

   - *Step 1*: The automated test script will upload `xray_alert.dcm`.
   - *Step 2*: The system analyzes the new scan and compares it with previous scans.

- *Step 3*: The system detects changes exceeding thresholds (e.g., lesion size increase by 30%).
- *Step 4*: The system generates an alert message containing:
  - Patient ID and relevant details.
  - Description of the significant change.
  - Urgency level or recommended actions.
- *Step 5*: The system sends the alert via configured channels:
  - Email to the clinician's registered email address.
  - SMS to the clinician's registered phone number.
  - In-app notification within the clinician's dashboard.
- *Step 6*: The script will monitor each channel to verify delivery:
  - Check the email inbox for the alert message.
  - Use a mock SMS gateway to confirm SMS delivery.
  - Log into the system as the clinician to verify the in-app notification.
- *Step 7*: The script will record the time taken from detection to alert delivery to ensure it meets timeliness requirements (e.g., within 5 minutes).

4.1.9 Treatment Plan Adjustment Tests

This subsection covers Requirement #9 of the SRS document by testing that the system allows healthcare professionals to adjust a patient's treatment plan based on analysis results.

1. **test-FR9-1**

   Control: Manual

   Initial State: A healthcare professional is logged into the system and has access to patient ID `12345`'s analysis results.

   Input: Adjustment to the patient's treatment plan based on the analysis results.

   Output: The system allows the professional to modify the treatment plan, linking the changes to the corresponding X-ray analysis results.

Test Case Derivation: Clinicians should be able to update treatment plans within the system, ensuring continuity of care.

How the test will be performed:

- *Step 1*: The tester logs into the system as a clinician with credentials.
- *Step 2*: The tester navigates to patient's profile and reviews the analysis results.
- *Step 3*: The tester accesses the treatment plan section and makes adjustments:
  - Changes medication dosage.
  - Adds a new therapy recommendation.
- *Step 4*: The tester saves the changes.
- *Step 5*: The system validates the changes according to clinical guidelines.
- *Step 6*: The system updates the patient's treatment plan and logs the changes with timestamps and the clinician's ID.
- *Step 7*: The tester verifies that the updated treatment plan is correctly reflected in the patient's records.
- *Step 8*: The tester checks that the changes are linked to the latest analysis results.
- *Step 9*: The test passes if the treatment plan is updated accurately and linked appropriately; otherwise, it fails.

4.1.10 Disclaimer Message Tests

This subsection covers Requirement #10 of the SRS document by testing that the system allows patients to upload their own chest X-ray images for analysis, providing appropriate disclaimers.

1. **test-FR10-1**

   Control: Automatic

   Initial State: A patient user account (`username: patient_user`, `password: PatientPass123`) is registered and the user is logged out.

Input: The patient logs in, uploads a chest X-ray image (`patient_image.dcm`), and requests analysis.

Output: The system displays an appropriate disclaimer, requires acknowledgment before proceeding, accepts the image, and provides analysis results.

Test Case Derivation: Patients should be informed about limitations and must consent before using the service.

How the test will be performed:

- *Step 1*: The automated test script simulates the patient logging into the system.
- *Step 2*: The script navigates to the "Upload Image" section and uploads an image
- *Step 4*: The script verifies that a disclaimer is presented:
  - Check that a modal or page displays the disclaimer text, including:
    * The analysis is not a substitute for professional medical advice.
    * The need to consult a healthcare professional.
    * Information about data usage and privacy.
  - Ensure that the "I Agree" or acknowledgment checkbox/button is present.
- *Step 5*: The script attempts to proceed without acknowledging the disclaimer:
  - Click "Continue" without checking "I Agree".
  - Verify that the system prevents progression and displays a prompt to acknowledge the disclaimer.
- *Step 6*: The script acknowledges the disclaimer:
  - Check the "I Agree" box.
  - Click "Continue".
  - Verify that the system accepts the acknowledgment and proceeds to process the image.
- *Step 7*: The script monitors the analysis process:

  - Check for a progress indicator or status message.
  - Wait for the analysis to complete.
- *Step 8*: After analysis completion, the script verifies that results are displayed:
  - Confirm that detected diseases and confidence levels are shown.
  - Ensure that a reminder is present advising consultation with a healthcare professional.
- *Step 9*: The script attempts to access features restricted to clinicians:
  - Try to access the "Patient Records" section.
  - Verify that access is denied with an appropriate message.
- *Step 10*: The test passes if all steps function as expected and the disclaimer process cannot be bypassed; otherwise, it fails.

4.1.11 Role Based Access Control Tests

This subsection covers Requirement #11 of the SRS document by testing that the system enforces role-based access control, ensuring users only access permitted features.

1. **test-FR11-1**

   Control: Automatic

   Initial State: The system is operational with user accounts assigned different roles:

   - Physician: `username: physician_user`, `password: PhysicianPass123`.
   - Radiologist: `username: radiologist_user`, `password: RadiologistPass123`.
   - Patient: `username: patient_user`, `password: PatientPass123`.

   Input: Users attempt to access features outside their permissions.

   Output: Access is granted only to features appropriate for each role; unauthorized access is denied with an appropriate error message.

   Test Case Derivation: Role-based access control is essential for security and compliance.

   How the test will be performed:

- *Step 1*: For each user role, the automated test script will perform the following steps:
  - **Physician User**:
    * Log in using physician credentials.
    * Access permitted features:
      · View and edit assigned patient records.
      · Adjust treatment plans.
      · Verify access is granted and functions correctly.
    * Attempt to access restricted features:
      · Administrative settings.
      · Other clinicians' patient records.
      · Verify access is denied with appropriate error messages.
    * Log out.
  - **Radiologist User**:
    * Log in using radiologist credentials.
    * Access permitted features:
      · View imaging studies.
      · Perform image analyses.
      · Verify access is granted and functions correctly.
    * Attempt to access restricted features:
      · Modify treatment plans.
      · Access administrative settings.
      · Verify access is denied with appropriate error messages.
    * Log out.
  - **Patient User**:
    * Log in using patient credentials.
    * Access permitted features:
      · View own medical records.
      · Upload images for analysis.
      · Verify access is granted and functions correctly.
    * Attempt to access restricted features:
      · Other patients' records.
      · Clinical tools and analysis features.

        - Administrative settings.
        - Verify access is denied with appropriate error messages.
      * Log out.
  - *Step 2*: The script will record all access attempts and the system's responses.
  - *Step 3*: The script will verify that:
    - Authorized actions are allowed without errors.
    - Unauthorized actions are blocked with clear error messages.
  - *Step 4*: The test passes if access controls function correctly for all roles; otherwise, it fails.

4.1.12 Preservation of Original Data Tests

This subsection covers Requirement #12 of the SRS document by testing that the system preserves the original chest X-ray images by creating copies for analysis.

1. **test-FR12-1**

   Control: Automatic

   Initial State: A chest X-ray image (`original_image.dcm`) is uploaded and stored in the system's image repository (`/var/www/images/uploads/`).

   Input: Initiation of the AI model analysis on the uploaded image.

   Output: The system creates a new copy of the image for analysis, preserving the original unaltered.

   Test Case Derivation: Data integrity is maintained by not altering original images.

   How the test will be performed:

   - *Step 1*: The automated test script uploads `original_image.dcm` to the system:
     - Use API endpoint `/api/upload` with authorized credentials.
     - Record the file size and compute the checksum of `original_image.dcm`.
   - *Step 2*: The script initiates the analysis process:

  - Call the analysis API endpoint `/api/analyze` with the image ID returned from the upload.

- *Step 3*: The script monitors the processing directory (`.../images/processing/`):
  - Verify that a copy of the image (`processing_image.dcm`) is created for analysis.
  - Record the file size and compute the checksum of `processing_image.dcm`.
- *Step 4*: The script compares the original and copied images:
  - Confirm that the checksums of the two files match, ensuring the copy is identical before processing.
- *Step 5*: After analysis completion, the script re-computes the checksum of `original_image.dcm`:
  - Verify that the checksum matches the original value from Step 1.
  - Confirm that the file size remains unchanged.
- *Step 6*: The script verifies that all processing operations were performed on `processing_image.dcm`:
  - Check logs or metadata to confirm the processed image's ID matches `processing_image.dcm`.
- *Step 7*: The test passes if the original image is unmodified and a copy was used for analysis; otherwise, it fails.

4.1.13 Multiple Modality Support Tests

This subsection covers Requirement #13 of the SRS document by testing that the system supports multiple medical imaging modalities beyond chest X-rays.

1. **test-FR13-1**

   Control: Automatic

   Initial State: The system is configured to accept multiple medical imaging modalities, and processing modules for CT and MRI images are operational.

   Input: Uploading a CT scan image (`sample_ct_scan.dcm`) and an MRI image (`sample_mri_image.dcm`).

Output: The system accepts and processes each image correctly, providing accurate analysis results specific to each modality.

Test Case Derivation: The system should handle different imaging modalities appropriately.

How the test will be performed:

- *Step 1*: The automated test script logs into the system with clinician credentials.
- *Step 2*: The script uploads `sample_ct_scan.dcm`:
  - Use the API endpoint `/api/upload` with the image file.
  - Include metadata indicating the modality (if not automatically detected).
- *Step 3*: The script verifies that the system recognizes the image as a CT scan:
  - Check the response from the server for confirmation.
  - Verify that the image is stored in the appropriate directory (e.g., `/images/ct_scans/`).
- *Step 4*: The script initiates the analysis process for the CT scan:
  - Call the analysis API endpoint `/api/analyze` with the CT image ID.
- *Step 5*: The script monitors the analysis pipeline to ensure the CT-specific processing module is used:
  - Check logs or system messages indicating the module invoked.
- *Step 6*: Upon analysis completion, the script retrieves the results:
  - Verify that the results are appropriate for CT images.
  - Compare results against expected findings for `sample_ct_scan.dcm`.
- *Step 7*: The script repeats Steps 2-6 for `sample_mri_image.dcm`, ensuring that:
  - The MRI image is correctly recognized.
  - The MRI-specific analysis module is used.
  - Results are accurate and modality-specific.
- *Step 8*: The test passes if both images are processed correctly with accurate results; otherwise, it fails.

4.1.14 Regular AI Model Update Tests

This subsection covers Requirement #14 of the SRS document by testing that the system can update the AI model regularly without disrupting ongoing services.

1. **test-FR14-1**

   Control: Automatic, ¡anual

   Initial State: The system is functional with the current AI model version (`model_v1.0`). A new AI model version (`model_v1.1`) is ready for deployment.

   Input: Deployment of the updated AI model to the system generated by the .ipynb file.

   Output: The system integrates the new AI model seamlessly without disrupting ongoing services and continues processing user requests during the update.

   Test Case Derivation: Regular updates should not impact system availability.

   How the test will be performed:

   - *Step 1*: The automated test script triggers the deployment of `model_v1.1` via the CI/CD pipeline:
     - Use deployment scripts to initiate the update.
   - *Step 2*: While the deployment is in progress, the script simulates continuous user activity:
     - Upload test images at regular intervals (e.g., every 30 seconds).
     - Initiate analysis requests for each uploaded image.
   - *Step 3*: The script monitors system performance metrics:
     - Response times for API calls.
     - Error rates or any failed requests.
     - Server resource utilization (CPU, GPU, memory).
   - *Step 4*: After deployment completion, the script verifies that the new model is active:

  - Check the model version via a dedicated API endpoint (`/api/model_version`).
  - Confirm that `model_v1.1` is returned.
- *Step 5*: The script analyzes the results from test images submitted during deployment:
  - Verify that all analysis requests were processed successfully.
  - Compare results before and after the model update for consistency or expected improvements.
- *Step 6*: The script checks for any logged errors or warnings during deployment.
- *Step 7*: The test passes if there are no service disruptions, the new model is correctly integrated, and all user requests are handled successfully; otherwise, it fails.

4.1.15 Image Preprocessing Tests

This subsection covers Requirement #15 of the SRS document by testing that the system preprocesses chest X-ray images by normalizing pixel values and resizing images to the input dimensions required by the AI model.

1. **test-FR15-1**

   Control: Automatic

   Initial State: The preprocessing module is operational and correctly configured.

   Input: A batch of raw chest X-ray images with varying resolutions and pixel intensity ranges, located in `./test_data/preprocessing_test_set/`.

   Output: Preprocessed images resized to 224x224 pixels with normalized pixel values suitable for input into the AI model.

   Test Case Derivation: Proper preprocessing ensures consistency and compatibility with the AI model.

   How the test will be performed:

   - *Step 1*: The automated test script loads the batch of raw images:
     - Read images from `./test_data/preprocessing_test_set/`.
     - Record original metadata for each image (dimensions, pixel value ranges).

- *Step 2*: The script passes each image through the preprocessing module:
  - Use the preprocessing API or function.
  - Capture any preprocessing logs or outputs.
- *Step 3*: For each preprocessed image, the script verifies:
  - Image Dimensions:
    - Confirm that the image dimensions are 224x224 pixels.
  - Pixel Value Normalization:
    - Check that pixel values are within the expected range (e.g., 0 to 1).
- *Step 4*: The script compares the preprocessed images to expected outputs:
  - Use reference images or calculated expected values.
  - Compute the difference between preprocessed images and references.
  - Verify that any differences are within acceptable tolerances.
- *Step 5*: The test passes if all images meet the specified preprocessing requirements; otherwise, it fails.

2. **test-FR15-2**

   Control: Automatic

   Initial State: The preprocessing module is operational.

   Input: An image file with an unsupported format (`invalid_image.bmp`) and a corrupted image file (`corrupted_image.dcm`).

   Output: The system handles the errors gracefully without crashing and logs informative error messages.

   Test Case Derivation: The system should be robust against invalid inputs and maintain stability.

   How the test will be performed:

   - *Step 1*: The automated test script attempts to preprocess `invalid_image.bmp`:
     - Pass the file to the preprocessing module.
     - Monitor the module's response.

- *Step 2*: The script verifies that the preprocessing module:
  - Detects the unsupported file format.
  - Does not crash or throw unhandled exceptions.
  - Logs an error message indicating the issue, including the file name and reason (e.g., "Unsupported file format: .bmp").
- *Step 3*: The script repeats the process with `corrupted_image.dcm`:
  - Attempt to preprocess the corrupted file.
  - Monitor for exceptions or errors.
- *Step 4*: The script verifies that the preprocessing module:
  - Detects the file corruption.
  - Handles the error without crashing.
  - Logs an informative error message (e.g., "Failed to read image data: corrupted or incomplete file").

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 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   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:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

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

Author Author. System requirements specification. https://github.com/..., 2019.

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.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

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.

1. What went well while writing this deliverable?

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

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.

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?