Verification and Validation Report: Software Engineering

Team 17, Team RAdiAIdance
Allison Cook
Ibrahim Issa
Mohaansh Pranjal
Nathaniel Hu
Tushar Aggarwal

March 6, 2024

1 Revision History

Date	Version	Notes
03/01/2024	0.0	Initial Document
03/02/2024	0.1	Completed Symbols, Abbreviations and Acronyms, Automated Testing
03/04/2024	0.2	Worked on Functional Requirements Evaluation, Nonfunctional Requirements Evaluation
03/05/2024	0.3	Worked on Functional Requirements Evaluation, Nonfunctional Requirements Evaluation, Completed Changes due to Testing
03/06/2024	0.4	Completed Functional Requirements Evaluation, Nonfunctional Requirements Evaluation, Unit Testing, Trace to Requirements, Reflection
03/06/2024	0.5	Completed Trace to Modules, Code Coverage Metrics; formatting and final review

2 Symbols, Abbreviations and Acronyms

The following table, Table 2, includes the definitions and descriptions of all relevant symbols, abbreviations and acronyms used throughout this VnV Report document.

Symbol, Abbreviation or Acronym	**Definition or Description**
AI/ML	Artificial Intelligence/Machine Learning
AUC	Area Under (ROC) Curve
BUC	Business Use Case
DICOM	Digital Imaging and Communications in Medicine, the standard for medical images to ensure data and quality necessary for clinical use.
FR	Functional Requirement
ICU	Intensive Care Unit
IT	Information Technology
MC	Mandated Constraints
MG	Module Guide
MIS	Module Interface Specification
NFR	Non-functional Requirement
PoC	Proof of Concept
ROC curve	Receiver Operating Characteristic curve, a graph to show the performance of a model, plots true positive rate and false positive rate
SRS	Software Requirements Specification
T	Test
TC	Test Case
VnV	Verification and Validation

Table 1: Symbols, Abbreviations and Acronyms

Contents

List of Tables

List of Figures

This document intends to describe in detail the testing performed and the subsequent results in the verification and validation process of (NAME). In detail, the document outlines the evaluation of Functional system tests, Non Functional tests, and Unit tests. The Functional and Non Functional tests and their connection to the requirements can be found in the VnV Plan. As well, the unit tests are based on the individual modules that can be found in the MIS.

3 Functional Requirements Evaluation

The following are the test cases carried out as part of the functional requirements evaluation of the project software application. For each test case, the test case name, input, expected output, actual output, whether the actual output matched the expected output, and the relevant FR(s) are outlined in detail in this section. The test cases are organized into subsections based on what overarching functionality is being tested (i.e. handling input data, display, classification of diseases, access of database and authorization).

3.1 Handling Input Data

In this subsection, the test cases evaluating the software application for its input data handling functionalities are outlined in detail alongside the testing results.

Test Case Name: DICOM Input Acceptance (TC.1)

Input: A sample chest X-ray image in DICOM format.

Expected Output: The DICOM input is accepted and converted into a JPEG successfully, with no error messages or system anomalies.

Actual Output: The given DICOM image is converted to JPEG format.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.1, FR.7

Test Case Name: Non-DICOM Input Rejection (TC.2)

Input: A sample chest X-ray image in a format other than DICOM.

Expected Output: The non-DICOM input is rejected, and an appropriate error message is returned.

Actual Output: The image is rejected and an error message is returned.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.1

Test Case Name: DICOM conversion to JPEG (TC.3)

Input: A sample chest X-ray image DICOM format.

Expected Output: The output is the same X-ray image in JPEG format.

Actual Output: The given image in DICOM format is displayed as a JPEG image.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.8

3.2 Display

In this subsection, the test cases evaluating the software application for its display functionalities are outlined in detail alongside the testing results.

Test Case Name: Image Display Correctness (TC.4)

Input: Processed chest X-ray images with known medical conditions.

Expected Output: The correct display of chest X-ray images on the user interface, and reflects the processed medical conditions accurately.

Actual Output: The given chest X-ray image and the identified medical condition is correctly displayed.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.2, FR.3

Test Case Name: Report and Terms Correctness (TC.5)

Input: Processed chest X-ray images with known medical conditions.

Expected Output: The correct display of medical reports and associated terms on the user interface accurately reflects the processed information.

Actual Output: The correct diagnosis report for the processed image is displayed.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.4, FR.5

3.3 Classification of Diseases

In this subsection, the test cases evaluating the software application for its classification of disease functionalities are outlined in detail alongside the testing results.

Test Case Name: Disease Classification Correctness Performance (TC.6)

Input: DICOM chest X-ray images, approximately 20% of the dataset DATASET_NAME will be reserved and used to run this test/experiment, with known medical conditions representing various diseases.

Expected Output: The AUC for the ROC curves for the classification of diseases based on the expected area under the ROC curves is given in Table 6 in the SRS.

Actual Output: The expected AUC of each disease classification matches the expected AUC given in Table 6 of the SRS.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.6

3.4 Access of Database

In this subsection, the test cases evaluating the software application for its access to database functionalities are outlined in detail alongside the testing results.

Test Case Name: Database Search Functionality (TC.7)

Input: User-initiated search queries for patient records based on specified criteria such as patient ID (unique), name, date of X-ray, or medical condition(s).

Expected Output: The accurate retrieval of patient records matching the specified search criteria, is displayed on the user interface.

Actual Output: The requested patient data is accurately displayed to the user.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.6, FR.9, FR.11

3.5 Authorization

In this subsection, the test cases evaluating the software application for its authorization functionalities are outlined in detail alongside the testing results.

Test Case Name: Correct User Login (TC.8)

Input: User-initiated login attempts with valid credentials.

Expected Output: The successful login of authorized users, granting them access to the system.

Actual Output: The authorized user is able to log in with valid credentials and be granted access to the system.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.10

Test Case Name: Incorrect User Login (TC.9)

Input: User-initiated login attempts with invalid credentials.

Expected Output: The rejection of login attempts with invalid credentials, accompanied by the appropriate error message.

Actual Output: The user login attempts with invalid credentials are rejected, and the user is not granted access to the system.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.10

Test Case Name: Creation of New Authorized User (TC.10)

Input: User-initiated creation of a new authorized user account through the user interface.

Expected Output: The expected result is the successful creation of a new authorized user account recorded and saved in the system.

Actual Output: The new authorized user account is created successfully and saved in the system.

Expected and Actual Output Match: True

Relevant Functional Requirement(s): FR.10

4 Nonfunctional Requirements Evaluation

The following are the test cases carried out as part of the nonfunctional requirements evaluation of the project software application. For each test case, the test case name, input, expected output, actual output, whether the actual output matched the expected output, and the relevant NFR(s) are outlined in detail in this section. The test cases are organized into subsections based on what overarching property is being tested (i.e. usability, performance and security).

4.1 Usability

In this subsection, the test cases evaluating the software application for its usability properties are outlined in detail alongside the testing results.

Test Case Name: Software Application Usability I (TC.16A)

Input: Usability testers will attempt to perform a given set of tasks (e.g. retrieving patient data) using the software application's GUI.

Expected Output: The usability testers will perform the tasks and complete a survey documenting their experiences and thoughts on the GUI's usability.

Actual Output: The testers submitted a survey on their experience using the GUI.

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.A0, NF.C0, NF.EUR0, NF.LR0, NF.UPR0

Test Case Name: Software Application Usability II (TC.16B)

Input: A given set of tasks (e.g. retrieving patient data) using the software application's GUI will be completed during a walk-through and demo with the supervisor and/or other peers.

Expected Output: The supervisor and/or peers will give feedback documenting their thoughts on the GUI's usability.

Actual Output: The supervisor was given a walkthrough and demo of the software application and gave verbal feedback to the team that was documented. See the Changes due to Testing section for more information on the exact feedback.

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.A0, NF.C0, NF.EUR0, NF.LR0 and NF.UPR0

4.2 Performance

In this subsection, the test cases evaluating the software application for its performance properties are outlined in detail alongside the testing results.

Test Case Name: System Response Time (TC.12)

Input: Automated requests for all the various system functionalities, including image processing and database searches. Additionally, this includes website launch and load time.

Expected Output: The system provides responses within the defined acceptable time thresholds for each functionality as specified in section 3.3.1 of the SRS.

Actual Output: The system responses are provided within the defined acceptable time thresholds for each functionality as specified.

The average load time of the website from launch was 4 seconds.

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.SLR0

Runs	Time (seconds)
#1	3.2
#2	3.8
#3	3.8
#4	4.2
#5	4.4
#6	5.6

Table 2: Website Load Time

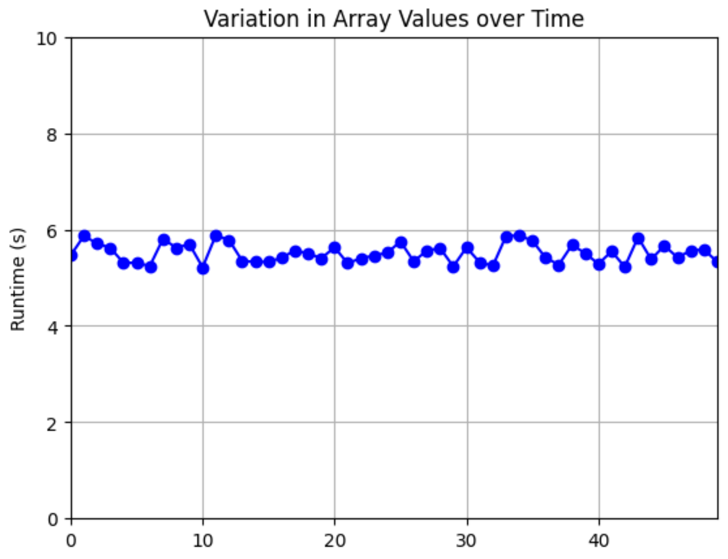

Figure 1: Variation in ML Model Runtime

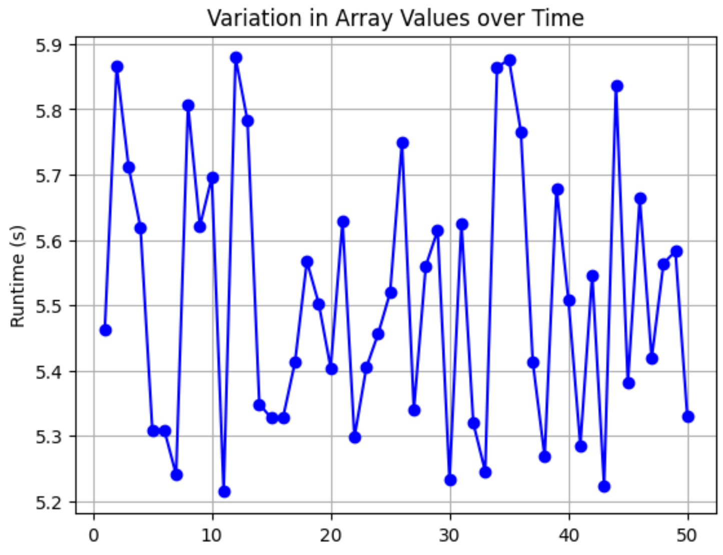

Figure 2: Variation in ML Model Runtime (Expanded Y-Axis)

Test Case Name: Accuracy of the ML Model (TC.13)

Input: See FR test case Disease Classification Correctness Performance for reference

Expected Output: Likewise refer to Disease Classification Correctness Performance for more details.

Actual Output: AUC of each disease is accurate according to Table 6 in the SRS.

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.PAR0

Test Case Name: Patient Data Collection (TC.14)

Input: A New patient is added to the system.

Expected Output: The system will retain the patient name, past diagnoses, assigned physician, past visit dates and other relevant data, while not retaining any unnecessary data.

Actual Output: The system only retains the needed patient data to identify patients and link them to their chest X-ray images and scan results, medical history, assigned physician and other relevant data.

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.SCR0

Test Case Name: Capacity (TC.15)

Input: Three sample chest X-ray images in DICOM format divided between three computers using the system.

Expected Output: The correct display of chest X-ray images on the user interface, reflects all the processed medical conditions accurately for each upload.

Actual Output: The images are correctly displayed on the user interface with all the processed information.

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.CR0

4.3 Security

In this subsection, the test cases evaluating the software application for its security properties are outlined in detail alongside the testing results.

Test Case Name: Data Encryption/Secure Access (TC.11)

Input: Software Application requests patient data as an authenticated user.

Expected Output: The encrypted data (not plain text data) is returned only for an authenticated user.

Actual Output: The data is returned only for an authenticated user (non-authenticated are denied access to the patient data).

Expected and Actual Output Match: True

Relevant Nonfunctional Requirement(s): NF.IR0, NF.PR1

Also, see Authorization in the Functional Requirements Evaluation section for NF.AR0, NF.AR1, NF.PR0 and NF.PR1 as they all relate to limiting access to records for authorized individuals.

5 Unit Testing

Unit testing was conducted on the different functions in each of the modules. The goal of these tests was to verify the correctness of independent units of code based on the separation of concerns in the system. The table below lists the functions tested in the corresponding modules, and the results of those tests.

Module	**Function Tested**	**Result**
ChestXRayRead	dcmFileValid	Passed
	parseDcm	Passed
	convertDcm2Jpg	Passed
ResultGen	reportResults	Passed
RepCompGen	nlpGen	Passed
	processImage	Passed
AIModel	processImage	Passed
	predictions	Passed
Backend	createANewUser	Passed
	loginAUser	Passed
	logoutAUser	Passed
	deleteAUser	Passed
DatabaseOps	writeUserData	Passed
	readUserData	Passed
	createNewUserData	Passed
	deleteUserData	Passed
UserAuthMgmt	createNewUser	Passed
	loginUser	Passed
	logoutUser	Passed
	deleteUser	Passed
AppGUI	displayLoginPage	passed
	displayScanPage	passed
	displayResultsPage	passed
PerformScan	initiateScan	passed
ViewResults	displayReport	passed

Table 3: Unit Testing Results

Overall, unit testing serves as a fundamental component of our software development process, enabling us to maintain code quality, identify and fix defects early, and ensure the overall reliability and robustness of our software application. We aim to achieve thorough test coverage across the entire software stack, from frontend user interfaces to backend business logic and data processing pipelines. By adhering to best practices in unit testing and continuously refining our test suites, we can deliver a high-quality software product that meets the functional requirements, performance expectations, and user satisfaction criteria outlined in our project specifications.

6 Changes Due to Testing

During the testing phase, several important changes were identified and implemented based on feedback from users and supervisors, particularly following the Rev 0 demo. These changes were aimed at enhancing the functionality, usability, and overall effectiveness of the software application. Here are the key changes:

- **Integration of NLP Report Generation:** Initially, generating NLP reports from chest X-ray images was a stretch goal. However, based on feedback and the importance of this functionality, it was elevated to one of the project's main goals.
  A new feature was introduced to generate NLP reports from chest X-ray images. This involved developing a separate ML model dedicated to producing textual reports, distinct from the model used for disease identification. By including this capability, the application can provide comprehensive diagnostic insights in textual format, enhancing its utility for healthcare professionals.
- **Modification of Image Upload Process:** Based on feedback and usability considerations, the process for uploading images or DICOM files was restructured. Instead of uploading images/DICOM files directly, it was decided to assume that the files had already been uploaded to the server. As a result, the user interface was modified to automatically populate relevant patient data and linked images/DICOM files when selecting a patient. This modification aimed to simplify the user experience and improve efficiency by eliminating redundant steps.
- **Store Results in Database:** To ensure comprehensive record-keeping and facilitate future reference, consideration was given to storing diagnostic results in the database. While this change was optional, it aimed to

provide a centralized repository for patient data and corresponding analyses, enhancing data management capabilities and enabling further analysis and reporting.

- **Usability:** A majority of feedback received during our walk-through and revision 0 demo focused on the redundancy of information that needed to be uploaded to utilize the application. We've decided to update the UI to allow for the users to select from existing patients to conduct a study and allow new X-ray images the be the only new information required to conduct new studies.

7 Automated Testing

7.1 Linters

The first step in our automated testing pipeline involves the use of linters such as Prettier, ESLint, Flake8, and Black. These tools are configured to run automatically upon code changes to enforce coding standards, detect syntax errors, and ensure consistent formatting across the entire codebase.

- **Prettier:** Ensures consistent code formatting across the project, eliminating debates over style preferences and enhancing code readability.
- **ESLint:** Identifies problematic patterns or code that does not adhere to specified coding rules for TypeScript.
- **Flake8:** Performs linting for Python code, enforcing PEP 8 style guidelines and detecting syntax errors and code smells.
- **Black:** Enforces a consistent code style for Python, automatically formatting code to adhere to the Black code style.

7.2 Unit testing

- **Jest:** Jest is used to test the frontend components and functionalities of our application developed using React.js. We write Jest test cases to verify the behaviour of individual React components, including their rendering, state management, event handling, and integration with external libraries or APIs. By leveraging Jest's comprehensive testing capabilities, we ensure the robustness and correctness of our frontend codebase, allowing us to catch bugs and issues early in the development process.

- **Pytest:** Pytest is employed to test the backend logic, APIs, and data processing functionalities of our software application developed using Python. We create Pytest test suites to validate the functionality and performance of various backend modules, including data retrieval, processing, storage, and interaction with external services or databases. Pytest allows us to write concise and expressive test cases, enabling thorough testing of different aspects of our backend codebase. By leveraging Pytest's robust testing features, we ensure the correctness, stability, and efficiency of our backend implementation, facilitating the delivery of a high-quality software product.

7.3 Continuous Integration

Continuous Integration (CI) is seamlessly integrated into our development workflow using GitHub Actions, enabling automated testing to occur with every code change. CI workflows are configured to trigger automated tests on each pull request and push to the main branch, ensuring that any introduced changes undergo thorough testing before integration into the main codebase.

- **Trigger:** CI workflows are triggered automatically on every pull request and push to the main branch, ensuring that all changes are validated before being merged.
- **Feedback:** Upon completion of the CI workflow, developers receive immediate feedback on the status of their code changes. If any tests fail, developers are alerted, allowing them to quickly address issues and iterate on their code.

8 Trace to Requirements

In this section, the traceability matrices linking the functional and non-functional requirements to the test cases are shown.

TC / FR	TC.1	TC.2	TC.3	TC.4	TC.5	TC.6	TC.7	TC.8	TC.9	TC.10
FR1	X	X								
FR2				X						
FR3				X						
FR4					X					
FR5					X					
FR6						X	X			
FR7	X									
FR8			X							
FR9							X			
FR10								X	X	X
FR11							X			

Table 4: Traceability Matrix for Functional Requirements

TC / NFR	TC.1	TC.2	TC.3	TC.4	TC.5	TC.6	TC.7	TC.8	TC.9	TC.10
NF.PAR0						X				
NF.RIAS0	X	X	X	X	X					
NF.PR0	X	X	X	X	X		X	X	X	X
NF.SR0	X	X	X	X	X		X	X	X	X
NF.AR0								X	X	X
NF.AR1								X	X	X
NF.PR0								X	X	X
NF.PR1								X	X	X

Table 5: Traceability Matrix for Nonfunctional Requirements I

NFR \ TC	TC.11	TC.12	TC.13	TC.14	TC.15	TC.16
NF.A0						X
NF.EUR0						X
NF.LR0						X
NF.UPR0						X
NF.SLR0		X				
NF.SCR0				X		
NF.PAR0			X			
NF.CR0					X	
NF.PR0	X	X	X	X	X	X
NF.SR0	X	X	X	X	X	X
NF.IR0	X					
NF.PR1	X					
NF.C0						X

Table 6: Traceability Matrix for Nonfunctional Requirements II

9 Trace to Modules

In this section, the traceability matrices linking the software modules to the test cases are shown.

NFR \ TC	TC.1	TC.2	TC.3	TC.4	TC.5	TC.6	TC.7	TC.8	TC.9	TC.10
M1					X	X				
M2					X	X				
M3					X	X				
M4					X					
M5					X					
M6								X	X	X
M7										
M8								X	X	X
M9										
M10	X	X	X	X	X		X	X	X	X
M11	X	X	X	X	X		X	X	X	X
M12								X	X	X
M13	X	X	X	X	X					
M14				X	X		X			

Table 7: Traceability Matrix for Software Modules I

NFR \ TC	TC.11	TC.12	TC.13	TC.14	TC.15	TC.16
M1		X	X		X	X
M2		X	X		X	X
M3		X	X		X	X
M4		X			X	X
M5		X			X	X
M6	X	X		X		X
M7	X	X		X		X
M8	X	X		X		X
M9		X		X		X
M10	X	X		X	X	X
M11	X	X		X	X	X
M12	X	X		X		X
M13		X			X	X
M14		X			X	X

Table 8: Traceability Matrix for Software Modules II

10 Code Coverage Metrics

In this section, the code coverage metrics are summarized for each software module. The code coverage values are given as percentages of code covered with a combination of automatic and/or unit tests.

Module	Statement Coverage %	Condition Coverage %	Decision Coverage %
M1: AIModel	100	100	100
M2: ChestXRayRead	100	100	100
M3: ResultGen	100	100	100
M4: NLPModel	100	100	100
M5: RepCompGen	100	100	100
M6: Backend	100	100	100
M7: DatabaseOps	100	100	100
M8: UserAuthMgmt	100	100	100
M9: MedInstInter	100	100	100
M10: AppController	100	100	100
M11: AppGUI	100	100	100
M12: Login	100	100	100
M13: PerfScan	100	100	100
M14: ViewResults	100	100	100

Table 9: Code Coverage Metrics

Appendix — Reflection

In reflecting on the Verification and Validation (VnV) Plan, it is evident that the actual activities deviated somewhat from the initial plan. Prior to development, our team outlined a VnV Plan based on our understanding of the project scope and requirements. Our VnV Plan provided a structured framework for verification and validation processes, outlining test case creation, execution, and analysis. However, as the development progressed, we realized that the scope of our actual implementation had evolved beyond what was initially envisioned.

For instance, we initially planned for the AI model to solely detect common chest diseases from X-ray images. However, during implementation, we decided to expand the functionality to include generating natural language processing (NLP) reports based on the detected conditions and incorporating new functionalities, such as enhanced data management capabilities, which were deemed essential for meeting user needs and project goals. Consequently, the VnV Plan had to be adapted to incorporate these new requirements and ensure comprehensive testing coverage. This underscored the importance of flexibility and adaptability in project management.

Reflecting on this experience, we acknowledge that our initial VnV Plan lacked a comprehensive understanding of the project's full scope and potential functionalities. We now recognize the importance of conducting thorough research and planning before finalizing the VnV Plan to anticipate potential changes in project scope more effectively. Additionally, better communication and collaboration among team members during the planning phase could have facilitated a clearer vision of the project's requirements and goals, helping to align the VnV Plan more closely with the actual implementation.

Looking ahead, incorporating more robust mechanisms for gathering requirements, conducting feasibility studies, anticipating potential scope changes and stakeholder feedback can help in designing a more flexible VnV Plan for future projects, enabling teams to accommodate adjustments without compromising the overall testing efficacy.