System Verification and Validation Plan for ScoreGen

Team #7, Tune Goons
Emily Perica
Ian Algenio
Jackson Lippert
Mark Kogan

April 3, 2025

Revision History

Date	Version	Notes
01/11/2024	0	Initial draft
08/11/2024	0	Include justification for F.R. test cases, improve F.R. test case terminology
04/03/2025	1.0	Issue 314: TA Feedback

Contents

List of Tables

This document outlines the Verification and Validation (VnV) plan for the development of an audio-to-sheet music generator. This includes plans to verify the SRS, system design, the VnV itself, implementation, and software, as well as a plan for an automated testing suite. This document will also describe the specific tests that will be used to ensure all Functional and Non-functional Requirements are met.

1 Symbols, Abbreviations, and Acronyms

See the SRS document for additional definitions.

Table 1: Document Definitions

Symbol	**Description**
MG	Module Guide Specification
MIS	Module Interface Specification
SRS	Software Requirement Specification
T	Test
VnV	Verification and Validation

2 General Information

2.1 Summary

This Verification and Validation Plan describes the testing process for ScoreGen, an audio-to-sheet music generator, such that the following core functionalities perform as desired:

- Signal processing of audio input
- Identification of pitch, rhythm, and timing
- Generation of readable and accurate sheet music

2.2 Objectives

2.2.1 Desired System Qualities

The primary objective of this document is to ensure that the audio-to-sheet music system produces highly accurate, reliable, and usable output, successfully meeting the needs of all stakeholders. Key qualities this VnV will aim to accomplish include the following:

- Demonstrate accuracy in music notation
- Ease of use amongst users with varying characteristics
- Readable, easy to understand outputs
- An enjoyable user experience
- Efficiency and responsiveness
- Portability of the software

2.2.2 Out of Scope Objectives

In order to meet time and cost requirements, the following objectives will be out of the scope of this VnV plan:

- Advanced polyphonic and multi-instrumental audio:

  Monophonic audio is easily processed using signal analysis, but introducing a multitude of simultaneous signals will require advanced algorithms to separate each note.

- Cross-platform support:

  Majority of the team members will be writing the system's software on their personal Windows PC. Testing should thus occur in the same environment it is being developed in in order to reduce performance variability of the final system.

- Advanced music notation, such as dynamics and accents:

  Dynamics and accents can be tricky to differentiate, and attempting to identify them is likely to introduce a level of error to the output that is undesired. As well, a beginner musician may feel overwhelmed being faced with a large variety of notation that they don't recognize.

- Support for non-Western music notation:

  This software will exclusively produce sheet music according to Western music conventions. Expanding to alternative notation conventions would require significant adapting of the output format, and is infeasible given the current scope of the project.

2.3 Challenge Level and Extras

2.3.1 Challenge Level

The ScoreGen project is categorized in the general challenge level, meaning it is not particularly novel and requires a senior highschool level or junior undergraduate level of domain knowledge/implementation. Projects in this category are required to include 'extras' in order to meet the difficulty level expected of a final-year capstone course.

2.3.2 Project Extras

- User manual
- Usability testing
- GenderMag personas

2.4 Relevant Documentation

This document is just one of many written to provide a comprehensive overview of the project and its desired outcomes. The other project documents include:

- Development Plan (Perica, Algenio, Lippert, and Kogan, 2024a): This document acts as a blueprint for the software's creation as well as the overall structuring of the project itself. Leveraging the information outlined in this document will ensure all verification and validation activities are relevant and within the scope of the project.
- Problem Statement and Goals (Perica, Algenio, Lippert, and Kogan, 2024e): This document contains an explicit definition of the problem the ScoreGen system aims to solve. It provides the VnV team with a

clear understanding of how tests may shape the project to align with the system's intended impact.

- Software Requirement Specification (SRS) (Perica, Algenio, Lippert, and Kogan, 2024f): States the functional and non-functional requirements that this VnV plan is developing tests for.
- Hazard Analysis (Perica, Algenio, Lippert, and Kogan, 2024b): Similar to the SRS, this document will guide VnV test plans to ensure all possible hazards to the system are mitigated.
- Verification and Validation Report (Perica, Algenio, Lippert, and Kogan, 2024h): This document will be guided entirely by the frameworks described in this VnV plan. Its outcome will depend on the relevance and quality of the described plans.
- User Guide (Perica, Algenio, Lippert, and Kogan, 2024g): This guide is intended to be used directly by the end-user to inform them on usage of the software. The VnV aims to verify all use cases and user interactions, and can thus be used to identify the necessary features or steps whose inclusion in the User Guide is critical.
- Module Interface Specification (MIS) (Perica, Algenio, Lippert, and Kogan, 2024d): This document defines how different software modules within the system interact with one another, making note of all inputs and outputs. These inputs and outputs will require verification outlined in the VnV Plan to ensure feasibility and accurate constraints are put into place.
- Module Guide (MG) (Perica, Algenio, Lippert, and Kogan, 2024c): Where the MIS defines module interactions, this document defines the modules themselves. The VnV tests should target all modules in the system - since the MG will be written after the VnV Plan, it may be necessary to revisit this plan and add/remove tests where necessary as the modules are formalized.

3 Plan

This section will provide formal validation and verification plans for various aspects of the project. Starting with a definition of all individuals involved in

the VnV process, it goes on to inform the reader of VnV plans for the SRS, design, VnV plan itself, implementation, automated tests, and the system software.

3.1 Verification and Validation Team

The first 4 entries in the below table are the software developers creating the system; they will all contribute to VnV activities to ensure the work is evenly distributed, but each has a specific responsibility they will oversee. The last 3 entries in the table are not involved in ideation/implementation of VnV, but their feedback will shape the final plan.

Table 2: VnV Team

Name	**Role**
Emily Perica	**Validation analyst**. Generates test reports and ensures the test results can be linked back to the SRS.
Ian Algenio	**FR tester**. Leads the testing process for functional requirements.
Jackson Lippert	**NFR tester**. Leads the testing process for non-functional requirements.
Mark Kogan	**Verification specialist**. Defines the process for establishing ground truths and ensures all tests being implemented follow the VnV plans.
Dr. Martin von Mohrenschildt	**Project supervisor**. Provides insight and suggests strategies for creating relevant tests.
Dr. Spencer Smith	**Course supervisor**. Similar to the project supervisor; provides insight and guides the team in creating a robust VnV plan.
Hunter Ceranic	**Teaching assistant**. Will provide feedback to the team on the feasibility, scope, and overall quality of the VnV plan.

3.2 SRS Verification Plan

To verify the SRS for our sheet music generation project, we plan to use a combination of informal feedback and structured reviews to ensure accuracy and alignment with the project's goals. The following subsections describe these approaches - the combination of peer and supervisor reviews, structured meetings, and issue tracking will provide comprehensive coverage for SRS verification.

3.2.1 Ad Hoc Feedback from Peers

We will request informal reviews from our primary reviewers (team 8). Their feedback will help identify potential ambiguities or areas needing clarification. We will ask them to focus on the clarity and feasibility of requirements, ensuring alignment with the intended functionality of the app.

3.2.2 Supervisor Review Meeting

Structured Review Meeting: We plan to conduct a concise, structured meeting with our project supervisor, Dr. von Mohrenschildt, to leverage his expertise effectively while respecting his time constraints. This meeting will include the following steps:

1. **Overview Presentation**: We will present a high-level overview of our initial requirements, focusing on the key features and functionalities we believe are essential for the project. The purpose of this presentation is to quickly bring the supervisor up to speed on the project's direction.

2. **Discussion on Feasibility and Priority**: We will ask the supervisor to evaluate the feasibility of the proposed requirements, help identify any that are overly ambitious or unrealistic, and highlight those that are core to the project's success. This will help us prioritize our work based on expert advice.

3. **Focused Questions**: To guide the discussion and make the best use of the supervisor's time, we will prepare targeted questions that focus on critical project areas. Examples include:

    - Which requirements are technically challenging or might present significant roadblocks?

- Are there any high-priority requirements that we might have overlooked?
- Which elements align closely with the project's objectives, and which might be deferred?

4. **Utilizing Issue Tracker**: To maintain transparency and track action items, we will record the supervisor's feedback and any new insights or tasks identified during the meeting in our issue tracker. This will enable us to follow up efficiently and ensure that the supervisor's input is incorporated into subsequent iterations of our requirements document.

3.2.3 Issue Tracking System

We will use the git issue tracker to log any concerns, suggestions, or changes requested by reviewers. This system will help maintain a clear record of all feedback and revisions made to the SRS, ensuring transparency and accountability in addressing reviewer comments.

3.3 Design Verification Plan

To ensure the design of the sheet music generation app meets all functional and non-functional requirements, the following verification plan will be implemented:

Peer Review and Feedback

The design will undergo reviews by our classmates, focusing on key design elements such as the user interface and algorithm selection for audio-to-sheet music conversion. Feedback from peers will help identify usability issues, potential improvements, and ensure that the design aligns with the project goals.

Checklist-Based Review

A checklist will be created to guide reviewers in assessing each design aspect:

- Are the core algorithms well-suited for the required audio-to-sheet music conversion?

- Are system constraints (e.g., processing time, memory usage) addressed in the design?
- Is the design modular and maintainable?

This checklist will help standardize feedback, ensuring a thorough and consistent review process.

Supervisor Review

In addition to peer review, we will present the design to our project supervisor for further validation. The supervisor will be provided with the design documents and checklist before the review session to facilitate a structured examination of each design component. Key discussion points will include scalability, performance optimization, and potential risks.

Issue Tracking and Follow-Up

Git Issues will be used to log any design issues or recommendations identified during the review process. This will provide a centralized record of all feedback and allow for systematic tracking of design revisions, ensuring that each suggestion is addressed or appropriately documented.

This design verification plan, incorporating both peer and supervisor feedback along with checklist-guided reviews and structured follow-up, will ensure the design is robust, user-friendly, and aligns with project requirements.

3.4 Verification and Validation Plan Verification Plan

Verification and Validation Plan Checklist:

- **Plan Review by Peers**: Ensure the verification and validation plan is reviewed by classmates to gain diverse insights and identify room for improvement.
- **Requirement Coverage**: Verify that all requirements are adequately covered by the verification and validation plan, ensuring no critical areas are omitted.

- **Traceability**: Confirm traceability of each verification and validation activity to specific project requirements to maintain alignment with project goals.
- **Clarity and Completeness**: Review the plan for clarity, making sure all instructions and testing procedures are clear and complete for future reference.

3.5 Implementation Verification Plan

The Implementation Verification Plan outlines the strategies and methodologies we will employ to verify the correctness and quality of our implementation.

- **Unit Testing Plan**: A comprehensive unit testing plan (section 5) will be developed, which includes a detailed list of tests designed to verify the functionality of individual components of the system. Each unit test will target specific functions or classes to ensure they perform as expected under various conditions. The results of these tests will be crucial for identifying and resolving issues early in the development process.
- **Static Verification Methods**: In addition to dynamic testing, we will implement several static verification techniques to enhance the quality of our code. These methods allow us to analyze the code without executing it, thereby identifying potential issues at an early stage. The techniques we plan to use include:
  - **Code Walkthroughs**: We will conduct peer reviews in the form of code walkthroughs, where team members will systematically review code segments. This collaborative approach encourages knowledge sharing and helps catch defects or design flaws that may not be evident to the original developer.
  - **Static Analyzers**: We will employ static analysis tools to automatically analyze the code for common programming errors, potential security vulnerabilities, and adherence to coding best practices. These tools provide insights that can help us improve code quality and reduce technical debt.

By integrating both unit testing and static verification methods into our implementation verification plan, we aim to ensure that our code is robust, maintainable, and free from critical defects. The combination of unit testing and static verification not only enhances the reliability of our implementation but also streamlines the development process by identifying issues early on. Implementing this plan will reduce the possibility of accruing technical debt, and overall increase the ease of software development.

3.6 Automated Testing and Verification Tools

For our project, we plan to utilize GitHub Actions to create a continuous integration pipeline, which will automate the build and testing processes. We will employ the Google Test framework for unit testing, as it provides a robust and easy-to-use interface for writing and executing C++ test cases.

In addition to unit testing, we will implement linters to verify coding standards and maintain consistency across the codebase. The coding standards we will follow include:

- Google's C++ Style Guide (Google, 2024) for C++.
- PEP8 Style Guide (Python Software Foundation, 2024): for Python.
- BEM Methodology (BEM Community, 2024): for HTML and CSS.

We will summarize code coverage metrics by integrating a coverage tool like gcov with our CI pipeline, allowing us to visualize the effectiveness of our tests and identify untested code paths.

3.7 Software Validation Plan

For our project, we plan to validate the functionality of our application using imported music files. These music files will serve as test cases to ensure that our software correctly processes and generates sheet music from audio inputs.

To further validate the product we will conduct user testing to gather feedback on the functionality and usability of our application, allowing us to make improvements based on user experiences.

There will also be a quality assurance testing effort to ensure a robust adherence to the defined SRS requirements, as further defined in the SRS Verification Plan

4 System Tests

System tests are categorized into two subsections. Section 4.1 lists test cases for functional requirements defined in the SRS, and section 4.2 lists test cases for nonfunctional requirements defined in the SRS.

4.1 Tests for Functional Requirements

Tests for the functional requirements of the application naturally follow the division of the major types of functional requirements in section 9 of the SRS. Thus, tests for each major type are grouped similarly, resulting in five subcategories.

4.1.1 Input Handling

1. **Test for Correct Audio File Formats**

   Test ID: FR-AR1-3-1
   Control: Automatic
   Initial State: Application is running and idle, awaiting audio file upload from the user.
   Input: Audio file(`.WAV`)
   Output: File acceptance without errors, entrance into file processing state.
   Test Case Derivation: For validation of functional requirements FR-AR1 and FR-AR3 in section 9.1 of the SRS.
   Justification: This test case ensures the application only accepts supported file formats and only begins processing these formats. This test case acts as a preventative measure against further error progogation through the file processing stage.
   How Test Will Be Performed:

   (a) Select a prepared audio sample file.
   (b) Upload the audio file for transcription.
   (c) Confirm the application accepts the file through success message or by entrance into the file processing state.

2. **Test for Incorrect File Formats**

Test ID: FR-AR1-3-2
Control: Automatic
Initial State: Application is open and idle, awaiting audio file upload from the user.
Input: Non-audio file (e.g., `.PDF`, `.MP4`, `.JPEG`, etc.)
Output: Denial of upload attempt with error message.
Test Case Derivation: For validation of functional requirements FR-AR1 and FR-AR3 in section 9.1 of the SRS.
Justification: Complements test case FR-AR1-3-1. It will ensure that various, unsupported file formats are not accepted by the application.
How Test Will Be Performed:

(a) Select a prepared file of an unsupported format (i.e. non-audio file format).

(b) Upload the file for transcription.

(c) Confirm the application denies the file and provides an error message that specifies the unsupported file format.

3. **Test User Device Microphone**

Test ID: FR-AR2
Control: Manual
Initial State: Application is open and idle, user has navigated to the audio recording interface and microphone permissions have been granted.
Input: Audio recorded by the user device's microphone.
Output: The application captures the correct audio and saves it in a format processable by the application (e.g., `.WAV`).
Test Case Derivation: For validation of functional requirement FR-AR2 in section 9.1 of the SRS. Guarentees that in addition to file uploads, the user is able to capture raw, custom audio with their device hardware.
Justification: It also validates that the audio is saved in a compatitble format for processing.
How Test Will Be Performed:

(a) Navigate to the application's audio recording interface.

(b) Start a recording using the application's "Record" element.

(c) Wait for 10 seconds and stop the recording.

(d) Confirm that the audio was recorded through:

- Playback using an audio player on the device.
- File metadata analysis (duration, file size, etc.).
- Waveform inspection using an external tool (e.g., Audacity).

(e) Confirm that the captured audio matches the expected input via playback, metadata, and/or waveform inspection.

4. **Test Generated Score Alignment with Selected Instrument**

Test ID: FR-AR4
Control: Automatic
Initial State: Application is running and idle, pre-transcription state awaiting input.
Input: Sample audio file or user-recorded audio and selected instrument type.
Output: A generated sheet music file in MusicXML format that aligns with the selected instrument's key signature and note pitches.
Test Case Derivation: For validation of functional requirement FR-AR4 in section 9.1 of the SRS.
Justification: Matching an instrument's specific key and pitch requirements is essential for sheet music accuracy.
How Test Will Be Performed:

(a) Select and submit an instrument type within the application.

(b) Upload an audio input for the selected instrument.

(c) Upon transcription completion, parse the generated score.

(d) Verify that the key signature and note pitches match the expected result for the selected instrument.

4.1.2 Signal Processing and Element Identification

1. **Test Effect of Increased Noise in Audio Input**

Test ID: FR-SP1
Control: Automatic

Initial State: Application is running and idle, awaiting file upload from the user.
Input: Two sample audio files—one unedited, the other with 10% noise interference.
Output: Two identical sheet music files in MusicXML format.
Test Case Derivation: For validation of functional requirement FR-SP1 in section 9.2 of the SRS.
Justification: Confirms that the application can handle expected levels of background noise in input audio, and that it can maintain accuracy despite the noise.
How Test Will Be Performed:

(a) Upload the unedited audio file to the application and generate a sheet music file.

(b) Upload the noisy audio file and generate another sheet music file.

(c) Parse both files and identify discrepancies, if any.

2. **Test for Pitch and Rhythm Identification**

Test ID: FR-SP2
Control: Manual
Initial State: Application is running and idle, awaiting audio input from the user.
Input: Sample audio file or user-recorded audio with a known sheet music equivalent.
Output: Sheet music correctly identifying all notes in the input audio.
Test Case Derivation: For validation of functional requirement FR-SP2 in section 9.2 of the SRS.
Justification: Verfies the transcription stage as well as the pitch and rhythm identification algorithms employed by the applications implementation.
How Test Will Be Performed:

(a) Prepare an audio file containing an ascending and descending C major scale.

(b) Upload the audio input to the application.

(c) Generate and save the sheet music in a viewable file format.

(d) Compare the generated sheet music visually with the sample sheet music for note discrepancies.

3. **Test for Key Signature and Time Signature Identification**

   Test ID: FR-SP3
   Control: Automatic
   Initial State: Application is running and idle, awaiting audio input from the user.
   Input: Sample or user-recorded audio file that has a known key signature and time signature.
   Output: Sheet music that has the same key signature and time signature as the input's sheet music.
   Test Case Derivation: For validation of functional requirement FR-SP3 in section 9.2 of the SRS.
   Justification: Further verifies the transcription stage and accuracy of other sheet music elements.
   How Test Will Be Performed:

   (a) Upload the sample file or recorded audio file to the application.

   (b) Save the generated sheet music in MusicXML file format.

   (c) Parse the generated file and extract the key signature and time signature.

   (d) Compare the extracted signatures to the known input's signatures.

4. **Test for Polyphonic Audio Identification**

   Test ID: FR-SP5
   Control: Automatic
   Initial State: Application is running and idle, awaiting audio input from the user.
   Input: Sample file or user-recorded audio that contains known chords.
   Output: Sheet music file in MusicXML format that matches the input file's chords.
   Test Case Derivation: For validation of functional requirement FR-SP5 in section 9.2 of the SRS.
   Justification: Tests the application's ability to decipher and tran-

scribe multiple simultaneous notes in the audio input.
How Test Will Be Performed:

(a) Upload the sample audio file to the application.

(b) Ensure the sample audio file includes at least two distinct chords; single notes may or may not be interleaved.

(c) Save the generated sheet music in MusicXML file format.

(d) Parse the generated file.

(e) Compare the processed and identified chords to the known input audio chords and notes.

5. **Test for Monophonic Audio Identification**
Subsumed by test case FR-SP2 (see section 4.1.2.2)

4.1.3 Sheet Music Generation

1. **Test General Notation and Layout**

Test ID: FR-SMG1
Control: Automatic with manual inspection
Initial State: Application is running and idle, awaiting audio input from the user.
Input: Sample or user-recorded audio file with equivalent sheet music available.
Output: Sheet music using the same layout and notation as the input.
Test Case Derivation: For validation of functional requirement FR-SMG1 in section 9.3 of the SRS.
Justification: Ensures readability and usability, using improper notation and layout defeats the purpose of generating sheet music.
How Test Will Be Performed:

(a) Upload the sample or user-recorded audio file for transcription.

(b) Save the generated sheet music in MusicXML format and a viewable document format.

(c) Check for discrepancies:

- Parse the MusicXML file and compare it to the input file.

- View the document formatted file (e.g., `.PDF`) and compare it to the input's sheet music.

2. **Test Instrument Specific Concert Pitch**

   Test ID: FR-SMG2
   Control: Manual
   Initial State: Application is running and idle, awaiting audio input from the user.
   Input: Two audio files—one from a non-transposing instrument and another from a transposing instrument.
   Output: Two sets of sheet music that structurally and visually match their corresponding instrument's concert pitch.
   Test Case Derivation: For validation of functional requirement FR-SMG2 in section 9.3 of the SRS.
   Justification: Some instruments are transposing while others are non-transposing, this test case confirms the application's ability to support a broad range of instruments.
   How Test Will Be Performed:

   (a) Upload a non-transposing instrument's audio file for transcription and save the generated sheet music in a viewable file format.

   (b) Upload a transposing instrument's audio file for transcription and save the generated sheet music in a viewable file format.

   (c) Compare the two sets of sheet music to ensure appropriate notes are in concert pitch relative to the input sheet music.

3. **Test Post-Processing Edit and View Functionalities**

   Test ID: FR-SMG3
   Control: Manual
   Initial State: Application has just finished processing and transcribing audio input.
   Input: Edit requests, save requests, open requests.
   Output: Viewable file containing sheet music that reflects the requested edits.
   Test Case Derivation: For validation of functional requirement FR-SMG3 in section 9.3 of the SRS.

Justification: Ensures that users can change any elements of the sheet music and that these changes are reflected properly by the application.
How Test Will Be Performed:

(a) View the generated sheet music in the application.

(b) Perform three edit operations:

- Note deletion.
- Note addition.
- Note pitch and/or duration change.

(c) Save the edited sheet music in a viewable file format.

(d) Open and view the edited sheet music file to confirm that changes are present and correct.

4.1.4 User Interface (UI)

1. **Test Application Feedback After Audio File is Uploaded**

Test ID: FR-UI1
Control: Automatic
Initial State: Application is running and idle, awaiting audio input from the user.
Input: A sample audio file.
Output: Visual cue(s) on the GUI in 2 seconds or less.
Test Case Derivation: For validation of functional requirement FR-UI1 in section 9.4 of the SRS.
Justification: Validates the funcationality that contributes to the non-functional requirement of human-centered design principles (see Section 4.2.2.1). One of the four fundamental principles is feedback, without the proper creation and display of visual feedback cues, the application does not fulfill this principle to the best of its ability.
How Test Will Be Performed:

(a) Upload a sample audio file to the application.

(b) Start a timer and detect visual element changes on the GUI using a testing framework (e.g., Jest).

(c) Confirm the following conditions are met:

- The appropriate visual cue is detected.
- The elapsed time from upload start to display of visual feedback is at most 2 seconds.

2. **Test Availability of System Documentation**

   Test ID: FR-UI2
   Control: Manual
   Initial State: Application is running and idle.
   Input: Text-search queries.
   Output: Navigation to appropriate documentation/user guide sections.
   Test Case Derivation: For use during user testing to meet fit criteria for functional requirement FR-UI2 in section 9.4 of the SRS.
   Justification: Further supports usability and humanity testing (see Section 4.2.2). Without documentation, users may find navigating and using the appliation difficult.
   How Test Will Be Performed:

   (a) Navigate to the location of user documentation in the application.

   (b) Perform a broad text search for major application features or for anything the user requires clarification on.

3. **Test User Feedback Report Mechanism**

   Test ID: FR-UI3
   Control: Automatic
   Initial State: Application is running and idle, prepared to receive input through the feedback mechanism.
   Input: Plain text.
   Output: GUI submission success cue and message, return of the input text.
   Test Case Derivation: For validation of functional requirement FR-UI3 in section 9.4 of the SRS.
   Justification: Ensures users are able to submit basic feedback or issues and that the application provides adequate feedback in the form of confirmation of a successful submission.
   How Test Will Be Performed:

(a) Submit a plain text report via the user feedback mechanism to the development team (e.g., through email).

(b) Parse through inbox messages for user feedback reports and confirm the reception of the submitted report and entire plain text input.

4.1.5 Save/Load

1. **Test Application Save Function**

   Test ID: FR-SL1
   Control: Manual
   Initial State: Post-audio processing state with generated sheet music file(s) and original audio files accessible for download.
   Input: Request to save file(s) to local drive.
   Output: Non-corrupt file(s) in destination directories.
   Test Case Derivation: For validation of functional requirement FR-SL1 in section 9.5 of the SRS.
   Justification: Confirms that users are able to track their progress and save sheet music and audio files to their local storage without data corruption.
   How Test Will Be Performed:

   (a) Transcribe a sample audio file using the application.

   (b) Request to download both the original audio and the generated sheet music to a directory on the local drive.

   (c) Compare the contents of the downloaded files to their original copies to check for consistency.

2. **Test Application's Ability to Load Existing Files**

   Test ID: FR-SL2
   Control: Manual
   Initial State: Application running and idle, waiting for user input to modify and/or view.
   Input: An existing audio file or existing file containing previously generated sheet music.
   Output: Successful loading of files without errors.

Test Case Derivation: For validation of functional requirement FR-SL2 in section 9.5 of the SRS.
Justification: Complements test case FR-SL1. Ensures users can load previously saved sheet music or audio files into the application without errors.
How Test Will Be Performed:

(a) Request to load an existing file on the local drive.

(b) Manually inspect the opened file in the editor to ensure the files appear as they were saved.

4.2 Tests for Nonfunctional Requirements

The following tests are for nonfunctional requirements. There are a few things to note about the tests:

1. Tests will be carried out by a member of our team (led by NFR tester Jackson Lippert as shown in table 2), the tests have not been assigned at this time as we are unsure what the future of testing will look like at this time.

2. Wherever the tests refer to a 'sample group of people' (such as for usability testing), we will decide at a later time who that will be.

3. Usability Tests are left semi-ambiguous at this time to give us room to expand in the future when implementation details are better known.

4.2.1 Look and Feel Testing

1. **Test for LF-A1 Discoverability**

   Test ID: LF-A1
   Type: Usability, Dynamic, Manual
   Initial State: The application is launched, showing the main interface.
   Input/Condition: The user views the interface for the first time without guidance.
   Output/Result: 75% of users should be able to locate the interactive element that initiates the score generation process.
   How Test Will Be Performed:

(a) Recruit a sample group of users unfamiliar with the app and explain that they need to initiate the score generation process.

(b) Record if each user is able to locate the score generation button within 10 seconds.

(c) Ask users to describe why they selected the specific element as the score generation option.

Success Criterion: If 75% or more of the users identify the correct element, the test is marked as passed.

2. **Test for LF-A2 Colour Cohesion**

Test ID: LF-A2
Type: Static, Manual, Visual Inspection
Initial State: The application interface is fully designed.
Input/Condition: Inspect the interface to verify colour usage.
Output/Result: To meet the fit criterion, the colour scheme should have at least three distinct colours: primary, secondary, and accent.
How Test Will Be Performed:

(a) Identify and document the primary, secondary, and accent colours as specified in the app's design guidelines.

(b) Conduct a visual inspection of each screen and interactive element within the app to confirm adherence to these colours.

(c) Verify there are no unintended or extraneous colours used across the app.

Success Criterion: If all screens consistently use the defined colour palette, the test passes.

3. **Test for LF-A3 Resolution**

Test ID: LF-A3
Type: Functional, Manual, Dynamic
Initial State: All graphics are loaded within the application.
Input/Condition: Inspect images to verify resolution.
Output/Result: All graphics should display at 720p resolution or higher.
How Test Will Be Performed:

(a) Compile a list of all graphical assets used in the app, including their file locations.

(b) Use image inspection tools to verify the resolution of each image.

(c) Document the resolution of each graphic and flag any below 720p for replacement.

Success Criterion: If all images meet or exceed 720p resolution, the test passes.

4. **Test for LF-S1 Authority and Trust**

Test ID: LF-S1
Type: Usability, Dynamic, Survey-Based
Initial State: The user has interacted with the application once.
Input/Condition: The user provides feedback on their perception of the app's authority and trustworthiness.
Output/Result: 70% of surveyed users report feeling that the application is reliable and trustworthy.
How Test Will Be Performed:

(a) After a user's initial interaction with the app, provide them with a survey containing the following questions:

- Q1: On a scale of 1-5, how trustworthy does this app feel when handling your data? (1 = Not Trustworthy, 5 = Very Trustworthy)
- Q2: Do you feel confident using this app for your music notation needs? (Yes/No)
- Q3: Please briefly explain any factors that contributed to your level of trust in the app.

(b) Collect responses and analyze the data to calculate the percentage of users rating the app as 4 or 5 for trustworthiness in Q1 and answering "Yes" to Q2.

Success Criterion: If 70% or more users rate the app 4 or higher on trustworthiness and respond "Yes" to Q2, the test passes.

4.2.2 Usability and Humanity Testing

1. **Test for UH-EOU1 Human-Centered Design (HCD)**

   Test ID: UH-EOU1
   Type: Usability, Static, Inspection-Based
   Initial State: The app interface is fully developed, adhering to HCD principles (Harvard Business School, 2024).
   Input/Condition: Inspect the app interface for compliance with the four fundamental HCD principles.
   Output/Result: The user interface must visibly incorporate all four HCD principles.
   How Test Will Be Performed:

   (a) Review the app's design documentation to verify its adherence to the HCD principles: visibility, consistency, user control, and feedback.

   (b) Conduct an inspection-based evaluation of the interface with usability experts, identifying examples of each HCD principle in action.

   Success Criterion: The app passes if all four HCD principles are clearly implemented and documented within the interface.

2. **Test for UH-PI1 Language**

   Test ID: UH-PI1
   Type: Functional, Static, Manual
   Initial State: The app is fully localized with text in Canadian English (en-CA).
   Input/Condition: Inspect all text, labels, and instructions in the app.
   Output/Result: All text should be presented in Canadian English, with no grammatical or spelling errors.
   How Test Will Be Performed:

   (a) Conduct a manual review of all textual elements within the app, ensuring they are correctly localized in Canadian English.

   (b) Use a grammar and spell-check tool to verify accuracy.

Success Criterion: The test is passed if all text is in Canadian English and no errors are found.

3. **Test for UH-L1 Music Theory Familiarity**

   Test ID: UH-L1
   Type: Usability, Dynamic, User Testing
   Initial State: The user is unfamiliar with the app and has no formal music theory background.
   Input/Condition: The user has 10 minutes to explore and use the app without guidance.
   Output/Result: 95% of users without a music theory background are able to generate a score sheet within the allotted time.
   How Test Will Be Performed:

   (a) Recruit a sample of users with no formal music theory background.

   (b) Allow users 10 minutes to familiarize themselves with the interface.

   (c) Observe and document whether each user is able to generate a score sheet within this period.

   Success Criterion: The test is passed if 95% or more of the users complete score generation without assistance.

4. **Test for UH-UP1 Icon Identification**

   Test ID: UH-UP1
   Type: Usability, Dynamic, Survey-Based
   Initial State: The app displays interactive elements with unlabeled icons.
   Input/Condition: Users attempt to identify the function of each interactive icon.
   Output/Result: At least 70% of icons are accurately identified by users.
   How Test Will Be Performed:

   (a) Present a sample of users with the app interface and ask them to identify the function of each interactive icon without clicking or interacting with them.

(b) Record user responses and compare them to the correct icon functions.

Success Criterion: If at least 70% of the icons are correctly identified by the majority of users, the test is passed.

5. **Test for UH-UP2 Information Hiding**

Test ID: UH-UP2
Type: Functional, Static
Initial State: The app is fully developed and ready for inspection.
Input/Condition: Inspect the app interface and accessible areas.
Output/Result: No user-accessible elements reveal implementation-specific or algorithmic details.
How Test Will Be Performed:

(a) Conduct a static inspection of all user-accessible components in the interface, including settings, menus, and tooltips.

(b) Confirm that no elements or descriptions expose the internal implementation or processing details.

Success Criterion: The test is passed if no user-accessible components reveal underlying algorithms or implementation details.

6. **Test for UH-A1 Web Content Accessibility Guidelines (WCAG) Compliance**

Test ID: UH-A1
Type: Accessibility, Static and Dynamic, Automated and Manual
Initial State: The app interface is complete and ready for accessibility testing.
Input/Condition: Perform an accessibility audit using automated tools and manual checks.
Output/Result: The app meets or exceeds WCAG 2.1 Level AA compliance (WCA, 2024).
How Test Will Be Performed:

(a) Run automated accessibility testing tools to identify any WCAG Level AA compliance issues, including alt text for images, colour contrast, keyboard navigability, etc.

(b) Conduct manual testing for areas not covered by automated tools, such as text readability and interaction.

(c) Document any issues and confirm adherence to all WCAG 2.1 Level AA standards.

Success Criterion: The test is passed if the app fully complies with WCAG 2.1 Level AA standards, confirmed by both automated and manual testing.

4.2.3 Performance Testing

1. **Test for PR-SL1 User Interface Response Time**

   Test ID: PR-SL1
   Type: Performance, Dynamic
   Initial State: The application is open and ready for user interaction.
   Input/Condition: Perform multiple interactions, such as menu navigation and input processing.
   Output/Result: The app should respond within 2 seconds for 90% of interactions, with no interaction taking longer than 5 seconds.
   How Test Will Be Performed:

   (a) Conduct a series of interactions across the application, including navigation through menus and processing various inputs.

   (b) Record response times for each interaction.

   (c) Calculate the percentage of interactions that respond within 2 seconds and confirm that no response exceeds 5 seconds.

2. **Test for PR-SL2 File Import and Export Speed**

   Test ID: PR-SL2
   Type: Performance, Dynamic
   Initial State: The app is ready to import and export files.
   Input/Condition: Test with music files up to 100MB in size for both import and export functions.
   Output/Result: The app should complete imports and exports within a reasonable time frame for at least 95% of operations.
   How Test Will Be Performed:

(a) Import and export a range of music files up to 100MB in size, timing each operation. An example music file is given here.

(b) Document the time taken for each operation and calculate the percentage of imports and exports that complete within an acceptable time.

(c) Confirm that 95% of operations meet the expected time frame.

3. **Test for PR-SC1 Epilepsy Safety**

   Test ID: PR-SC1
   Type: Accessibility, Static, Automated
   Initial State: The application is open with all visual elements loaded.
   Input/Condition: Inspect graphical interface elements for compliance with WCAG 2.1 guidelines on flashing content (WCA, 2024).
   Output/Result: No visual elements should flash at a rate of more than 3 flashes per second.
   How Test Will Be Performed:

   (a) Use accessibility tools to scan the interface for flashing content.

   (b) Verify that all visual effects adhere to the flash rate limitation.

   (c) Document any elements that exceed the flash rate and adjust them to ensure compliance.

4. **Test for PR-SC2 Instrument Input Setup**

   Test ID: PR-SC2
   Type: Functional, Usability
   Initial State: The app is open, with the instrument input configuration tutorial available.
   Input/Condition: Use the tutorial to set up an external instrument or microphone.
   Output/Result: The setup should be completed successfully on the first attempt with a 95% success rate across user testing.
   How Test Will Be Performed:

   (a) Guide a sample group of users through the step-by-step input configuration tutorial.

(b) Track the completion success rate for each user on their first attempt.

(c) Confirm that at least 95% of users complete the setup successfully on their first attempt.

5. **Test for PR-PA1 Pitch Detection Accuracy**

   Test ID: PR-PA1
   Type: Functional, Performance
   Initial State: The app is configured for audio input from diverse instruments.
   Input/Condition: Test pitch detection with a range of instruments and note ranges. Test samples found here.
   Output/Result: The pitch detection accuracy should be within a 1% error margin across all tests.
   How Test Will Be Performed:

   (a) Play or record test audio samples from various instruments covering diverse note ranges.

   (b) Measure pitch detection accuracy by comparing transcribed notes to the actual pitches.

   (c) Calculate the error rate and confirm it remains below 1%.

6. **Test for PR-PA2 Timing Accuracy**

   Test ID: PR-PA2
   Type: Functional, Performance
   Initial State: The app is set to capture audio input.
   Input/Condition: Test with audio samples that have precise note durations and rhythms.
   Output/Result: The app should capture note durations and rhythms with an accuracy tolerance within 100ms.
   How Test Will Be Performed:

   (a) Play or record audio samples with known timing and rhythm.

   (b) Analyze the transcribed timing for each note and compare it to the original timing.

(c) Verify that timing discrepancies are within the 100ms tolerance limit.

7. **Test for PR-RFT1 Reliability (Time Between Failures)**

   Test ID: PR-RFT1
   Type: Performance, Dynamic
   Initial State: The app is running under typical usage conditions.
   Input/Condition: Operate the app continuously for 24 hours.
   Output/Result: The app should function without crashes or failures for the entire 24-hour period in at least 95% of cases.
   How Test Will Be Performed:

   (a) Start the app under standard usage conditions and monitor it for 24 hours.

   (b) Track and log any crashes or failures.

   (c) Confirm that at least 95% of the tests meet the requirement of continuous operation without interruption.

8. **Test for PR-RFT2 Availability (Uptime)**

   Test ID: PR-RFT2
   Type: Performance, Static
   Initial State: The app is installed and operational over an extended period.
   Input/Condition: Monitor app uptime over several weeks.
   Output/Result: The app should demonstrate 99.5% uptime, accounting for any brief planned downtime.
   How Test Will Be Performed:

   (a) Log the app's operational status over an extended period.

   (b) Calculate the percentage of time the app is accessible and operational.

   (c) Verify that uptime meets or exceeds 99.5%.

9. **Test for PR-RFT3 Crash Recovery**

   Test ID: PR-RFT3

Type: Functional, Dynamic
Initial State: The app is open and in active use with an ongoing transcription.
Input/Condition: Simulate an unexpected crash.
Output/Result: The app should automatically save the current session and allow users to recover their work upon restarting, achieving 98% recovery success.
How Test Will Be Performed:

(a) Begin a transcription session and simulate a crash.

(b) Reopen the app and check if the session is restored, including any partially transcribed sheet music.

(c) Confirm that data recovery occurs in at least 98% of test cases.

10. **Test for PR-RFT4 Performance Under Load**

Test ID: PR-RFT4
Type: Performance, Dynamic
Initial State: The app is running and ready to process complex or large inputs.
Input/Condition: Test the app's performance under maximum load conditions (e.g., complex polyphonic inputs, large files).
Output/Result: The app should maintain stable performance with no more than a 30% decrease in processing speed.
How Test Will Be Performed:

(a) Load the app with large or complex audio files that simulate high activity conditions.

(b) Measure processing speed and performance metrics during this test.

(c) Verify that performance remains stable with no significant slowdowns or crashes and that any speed decrease is within 30%.

11. **Test for PR-RFT5 Handling Signal Interruptions**

Test ID: PR-RFT5
Type: Functional, Dynamic

Initial State: The app is receiving audio input from an external instrument or microphone.
Input/Condition: Temporarily disconnect and then reconnect the audio input device.
Output/Result: The app should retain buffered audio data for up to 2 minutes during interruptions and resume transcription seamlessly.
How Test Will Be Performed:

(a) Begin an audio input session and then simulate a signal interruption by disconnecting the audio source.

(b) Wait up to 2 minutes, then reconnect the audio source.

(c) Verify that buffered audio data is retained, and transcription resumes without any data loss in at least 95% of test cases.

12. **Test for PR-RFT6 Graceful Degradation**

Test ID: PR-RFT6
Type: Performance, Dynamic
Initial State: The app is running and under increasing system load.
Input/Condition: Apply stress by loading multiple instruments or large files until performance begins to degrade.
Output/Result: The app should notify the user of delays within 5 seconds of detection and avoid crashing in 99% of test cases.
How Test Will Be Performed:

(a) Gradually increase system load by adding complex inputs or running multiple processes.

(b) Observe if the app notifies the user within 5 seconds when performance slows.

(c) Confirm the app remains operational without crashing in at least 99% of test cases.

13. **Test for PR-RFT7 Automatic Recovery from Software Glitches**

Test ID: PR-RFT7
Type: Functional, Dynamic
Initial State: The app is running and processes files with varied input types.

Input/Condition: Introduce minor software errors (e.g., unexpected input format).
Output/Result: The app should log the error, notify the user, skip the problematic section, and continue without crashing in 90% of cases.
How Test Will Be Performed:

(a) Feed the app corrupted or incompatible files.

(b) Verify that the app logs the error, notifies the user, and skips the problematic section.

(c) Confirm that the app continues functioning without crashing in at least 90% of test cases.

14. **Test for PR-C1 Audio Input Capacity**

Test ID: PR-C1
Type: Performance, Dynamic
Initial State: The app is ready to record audio input.
Input/Condition: Record audio continuously for up to 90 minutes.
Output/Result: The app should process and transcribe the entire duration with no more than a 5% slowdown in speed or accuracy.
How Test Will Be Performed:

(a) Start recording and let the app run continuously for 90 minutes.

(b) Monitor transcription speed and accuracy.

(c) Verify that performance remains consistent and within the 5% slowdown limit.

15. **Test for PR-C2 Simultaneous User Sessions**

Test ID: PR-C2
Type: Performance, Dynamic
Initial State: The app supports simultaneous user sessions.
Input/Condition: Run 10 simultaneous active user sessions.
Output/Result: The app should maintain stable performance, response times, and transcription accuracy in 95% of cases.
How Test Will Be Performed:

(a) Initiate 10 user sessions, each running different tasks.

(b) Monitor performance metrics for each session, including response time and accuracy.

(c) Confirm that performance remains stable with no significant slowdown in 95% of test cases.

16. **Test for PR-SE2 Feature Extensibility**

Test ID: PR-SE2
Type: Structural, Static
Initial State: The app's codebase is available for review.
Input/Condition: Evaluate the app's modular architecture for future extensibility.
Output/Result: The app's design should allow feature additions without major refactoring.
How Test Will Be Performed:

(a) Conduct a code review with development and architecture teams.

(b) Assess the modularity of the codebase and document potential areas for future feature integration.

(c) Confirm that new modules could be added with minimal impact on core functionalities.

17. **Test for PR-SE3 Processing Power for Complex Music Compositions**

Test ID: PR-SE3
Type: Performance, Scalability
Initial State: The app is prepared to process complex musical compositions.
Input/Condition: Process increasingly complex compositions with multiple instrument tracks.
Output/Result: The app should maintain stability and performance as complexity increases.
How Test Will Be Performed:

(a) Gradually add instrument tracks and polyphonic elements to a composition.

(b) Monitor processing speed and stability.

(c) Confirm that the app handles increased complexity without significant performance issues.

18. **Test for PR-L1 Expected Lifetime**

Test ID: PR-L1
Type: Structural, Static
Initial State: The app's development roadmap is complete.
Input/Condition: Review the roadmap for planned updates and feature expansions over the next five years.
Output/Result: The roadmap should ensure that the app remains functional and relevant for at least five years with minor maintenance.
How Test Will Be Performed:

(a) Review the app's development roadmap with the team.

(b) Verify that updates, feature expansions, and maintenance plans are documented.

(c) Confirm that no major rewrites or overhauls are anticipated to keep the app functional and relevant.

4.2.4 Operational and Environmental Requirements Testing

1. **Test for OE-EP1 Operating Environment**

Test ID: OE-EP1
Type: Functional, Dynamic, Environmental
Initial State: The app is running on a personal computer or laptop in a controlled indoor setting.
Input/Condition: The app is operated in various typical indoor environments.
Output/Result: The app should perform consistently without requiring any adjustments based on the environment.
How Test Will Be Performed:

(a) Set up the app in indoor environments such as a home studio, classroom, and office.

(b) Record observations of the app's functionality in each environment, noting any issues related to lighting or ambient noise.

(c) Confirm that the app functions as expected across all tested environments.

2. **Test for OE-EP2 Noise and Audio Input Quality**

   Test ID: OE-EP2
   Type: Functional, Dynamic, Environmental
   Initial State: The app is configured to receive audio input in an environment with moderate background noise(Focus, 2024).
   Input/Condition: The app's transcription accuracy is tested in environments with varying ambient noise levels.
   Output/Result: The app's transcription accuracy should not degrade by more than 5%.
   How Test Will Be Performed:

   (a) Play a controlled background noise source in an environment and start the audio capture process.

   (b) Record the transcription accuracy in the presence of moderate ambient noise.

   (c) Calculate the error rate, confirming it remains within the acceptable threshold of no more than a 5% drop in accuracy.

3. **Test for OE-EP3 Workspace Flexibility**

   Test ID: OE-EP3
   Type: Usability, Dynamic, Environmental
   Initial State: The app is installed on devices with various screen sizes and resolutions, from 13-inch to 27-inch displays.
   Input/Condition: The app is run on screens of varying sizes to check layout flexibility.
   Output/Result: The app's interface should be adaptable to each screen size and resolution, maintaining full functionality.
   How Test Will Be Performed:

   (a) Open the app on devices with screen sizes ranging from 13 to 27 inches.

   (b) Check the layout, navigation, and accessibility of interactive elements on each screen size.

(c) Verify that all interactive elements remain visible and functional across screen sizes.

4. **Test for OE-EP4 Portable Setup Compatibility**

   Test ID: OE-EP4
   Type: Usability, Dynamic, Environmental
   Initial State: The app is installed on a laptop in a temporary workspace, such as a cafe or live performance venue.
   Input/Condition: The app is used in transient environments to evaluate setup and operation without specialized hardware.
   Output/Result: The app should function smoothly on standard laptop hardware, performing effectively in transient conditions.
   How Test Will Be Performed:

   (a) Set up the app on a laptop with standard specifications in a cafe or similar temporary workspace.

   (b) Test the app's functionality, including quick setup and breakdown, ensuring smooth operation.

   (c) Document any usability issues encountered during setup and breakdown in the transient environment.

5. **Test for OE-WE1 General Hardware**

   Test ID: OE-WE1
   Type: Usability, Dynamic, Environmental
   Initial State: The app is installed on devices with varying screen interfaces and sizes.
   Input/Condition: The app is run on screens from 13 to 27 inches with different resolutions.
   Output/Result: The app should be fully functional, maintaining usability across all screen sizes and resolutions.
   How Test Will Be Performed:

   (a) Launch the app on devices with screen sizes ranging from 13 to 27 inches.

   (b) Confirm that all UI elements are accessible, functional, and scale correctly on each screen size.

(c) Record any deviations in layout or usability and confirm functionality across all specified screen configurations.

6. **Test for OE-IA1 Audio Input Devices**

   Test ID: OE-IA1
   Type: Functional, Dynamic
   Initial State: The app is installed and configured to receive audio input.
   Input/Condition: Connect standard audio input devices (USB microphone, instrument pickup, built-in microphone).
   Output/Result: The app should support audio input from USB Audio Class 1.0 and higher devices.
   How Test Will Be Performed:

   (a) Connect various standard audio input devices, including USB microphones, instrument pickups, and built-in microphones.

   (b) Record audio using each device and monitor the app's performance.

   (c) Verify that audio is captured successfully from all tested devices without connectivity issues.

7. **Test for OE-IA2 Audio File Import and Export**

   Test ID: OE-IA2
   Type: Functional, Dynamic
   Initial State: The app is running and ready to import and export audio files.
   Input/Condition: Use sample audio files in WAV format for import and export.
   Output/Result: The app should successfully import and export audio files in WAV format.
   How Test Will Be Performed:

   (a) Import WAV file into the app and verify playback or analysis accuracy.

   (b) Export audio files in WAV formats, ensuring they are playable in standard audio players.

(c) Confirm that imported and exported files retain their quality and format specifications.

8. **Test for OE-IA3 Music Notation Software Integration**

   Test ID: OE-IA3
   Type: Functional, Dynamic
   Initial State: The app has completed audio processing and is ready to export sheet music.
   Input/Condition: Generate sheet music and export in MusicXML format.
   Output/Result: The exported files should be compatible with popular music notation software.
   How Test Will Be Performed:

   (a) Convert an audio sample to sheet music within the app.

   (b) Export the generated sheet music as a MusicXML file.

   (c) Test the exported files in another music notation software (TBD) to ensure compatibility and accuracy.

9. **Test for OE-P1 Distribution**

   Test ID: OE-P1
   Type: Functional, Static
   Initial State: The app is packaged and ready for distribution.
   Input/Condition: Package the app as a downloadable installer.
   Output/Result: The installer should be downloadable from a website or repository without additional dependencies.
   How Test Will Be Performed:

   (a) Package the app into an executable installer.

   (b) Upload the installer to a hosting site (TBD) and download it on test devices.

   (c) Verify that installation completes successfully with no dependencies required.

10. **Test for OE-P2 Installation Process**

Test ID: OE-P2
Type: Usability, Functional
Initial State: The app installer is ready for use.
Input/Condition: Begin installation with minimal technical guidance.
Output/Result: The installation should be simple and guide users effectively.
How Test Will Be Performed:

(a) Run the installer and proceed through the installation process.

(b) Confirm that instructions are clear and that users can install the app with minimal input.

(c) Verify that the app installs correctly and launches without issues.

11. **Test for OE-P3 Size and Compatibility**

Test ID: OE-P3
Type: Functional, Static
Initial State: The app installation file is prepared.
Input/Condition: Check the installation file size and install it on various systems.
Output/Result: The installation file should be 500MB or less, and the app should be compatible with different system setups.
How Test Will Be Performed:

(a) Confirm the installation file size does not exceed 500MB.

(b) Install the app on multiple devices with different operating systems and hardware specifications.

(c) Ensure the app operates smoothly across all tested systems.

12. **Test for OE-P4 Post-Installation Configuration**

Test ID: OE-P4
Type: Usability, Functional
Initial State: The app is installed and launched for the first time.
Input/Condition: Access the settings configuration panel.
Output/Result: Users should be able to modify settings from the

configuration panel without editing files manually.
How Test Will Be Performed:

(a) Launch the app and navigate to the settings configuration panel.

(b) Verify that users can adjust the input source and output format through the panel.

(c) Confirm that all settings save correctly and can be accessed again upon reopening.

13. **Test for OE-R1 Initial Release**

Test ID: OE-R1
Type: Functional, Static
Initial State: The app has completed development and quality assurance.
Input/Condition: Conduct a final testing phase to ensure all core functionalities are operational.
Output/Result: Core features should be functional with no major bugs, and ready for initial release.
How Test Will Be Performed:

(a) Perform a full quality assurance test on the app, focusing on core functionality such as audio-to-sheet music conversion in a production environment.

(b) Document any bugs or issues and resolve them before release.

(c) Verify that the app is stable and ready for general use by conducting a final user acceptance test.

4.2.5 Maintainability and Support Requirements Tests

1. **Test for MS-M1 Version Releases**

Test ID: MS-M1
Type: Functional, Dynamic
Initial State: A new version of the app has just been released.
Input/Condition: The user opens the app with an internet connection after a new release.
Output/Result: The user should receive a notification to install the

new version.
How Test Will Be Performed:

(a) Release a test version update while the app is active on a connected device.

(b) Open the app on the test device and verify that the user receives a prompt to install the new version.

(c) Document any delays or failures in notification.

2. **Test for MS-M2 System Crash**

Test ID: MS-M2
Type: Functional, Dynamic
Initial State: The application is running and unexpectedly shuts down.
Input/Condition: Force a non-user-prompted shutdown of the application.
Output/Result: The app should reopen within 1 minute and recover data from the previous session.
How Test Will Be Performed:

(a) Simulate an unexpected shutdown (e.g., force-close the app).

(b) Verify that the app automatically reboots within 1 minute.

(c) Confirm that data from the last session is preserved and accessible upon reopening.

3. **Test for MS-S1 Software Bugs**

Test ID: MS-S1
Type: Functional, Dynamic
Initial State: The user encounters a bug and accesses the reporting feature.
Input/Condition: Submit a bug report via the app's GUI.
Output/Result: The development team should be notified within 1 hour of report submission.
How Test Will Be Performed:

(a) Use the app's bug reporting feature to submit a test report.

(b) Verify that the report is logged in the development team's system within 1 hour.

(c) Document any delays or issues in the notification process.

4. **Test for MS-S2 Operating System**

   Test ID: MS-S2
   Type: Functional, Static
   Initial State: The app is ready to be installed on various Windows versions.
   Input/Condition: Install and run the app on devices with Windows 10 and 11.
   Output/Result: The app should perform all expected functionalities on both Windows 10 and Windows 11.
   How Test Will Be Performed:

   (a) Install the app on devices with Windows 10 and Windows 11.

   (b) Confirm that all core functionalities (e.g., audio capture, transcription) work as expected on each operating system.

   (c) Document any compatibility issues or functional limitations on each OS version.

5. **Test for MS-A1 Internet Connection**

   Test ID: MS-A1
   Type: Functional, Dynamic
   Initial State: The app is running on a device with intermittent internet connectivity.
   Input/Condition: Test the app's functionality both online and offline.
   Output/Result: The app should provide a consistent user experience regardless of internet connection status.
   How Test Will Be Performed:

   (a) Launch the app and test core functionalities (e.g., audio processing, and navigation) with a stable internet connection.

   (b) Disconnect the internet and continue using the app, verifying that the user experience remains consistent.

(c) Document any discrepancies in functionality when offline.

6. **Test for MS-A2 GUI Navigation**

   Test ID: MS-A2
   Type: Usability, Functional
   Initial State: The app's GUI is displayed with a connected keyboard and without a mouse.
   Input/Condition: Navigate the app's interface solely using keyboard shortcuts.
   Output/Result: The app should remain fully functional and navigable without a mouse.
   How Test Will Be Performed:

   (a) Disconnect any mouse from the test device and launch the app.

   (b) Use keyboard shortcuts to navigate through each major feature and menu within the app.

   (c) Confirm that all functionalities are accessible and that navigation is smooth using only the keyboard.

4.2.6 Security Requirement Tests

1. **Test for S-A1 User Authentication**

   Test ID: S-A1
   Type: Functional, Static
   Initial State: The application is closed.
   Input/Condition: Open the application as a standard user without any authentication.
   Output/Result: The application should allow access without requiring login credentials or password input.
   How Test Will Be Performed:

   (a) Launch the application and verify that the user is granted access without needing to enter any login details.

   (b) Document whether any authentication prompt appears unexpectedly.

(c) Confirm that the user can freely access all functionalities without an authentication barrier.

2. **Test for S-P1 Data Storage**

   Test ID: S-P1
   Type: Functional, Static
   Initial State: The application is running, and the user has chosen a specific location for file storage.
   Input/Condition: Save an output file in the user-selected directory.
   Output/Result: The output file should be saved in the specified directory with full read and write permissions for the user.
   How Test Will Be Performed:

   (a) Configure the application to store output files in a user-defined location.

   (b) Generate and save an output file, then verify that the file appears in the specified directory.

   (c) Check that the user has read and write permissions for the saved file.

3. **Test for S-P2 PII**

   Test ID: S-P2
   Type: Functional, Static
   Initial State: The application is running.
   Input/Condition: Use the application, observing all interactions for data input requests.
   Output/Result: The application should not request any Personal Identifiable Information (PII) from the user.
   How Test Will Be Performed:

   (a) Launch the application and monitor for any prompts requesting personal data.

   (b) Interact with all features of the app, verifying that no PII requests (e.g., name, address, or contact information) are made.

   (c) Document any instance where PII might be requested, and confirm the app remains free of such prompts.

4. **Test for S-P3 Input Data**

   Test ID: S-P3
   Type: Functional, Dynamic
   Initial State: The application has processed an audio input file.
   Input/Condition: Complete an audio processing session.
   Output/Result: The application should clear all caches and temporary files associated with the processed audio upon completion.
   How Test Will Be Performed:

   (a) Process an audio file within the application.

   (b) After processing, check the system's temporary file storage to ensure no audio data or temporary files remain.

   (c) Confirm that all caches are cleared, and no residual audio data is left on the system.

4.2.7 Cultural and Compliance Requirements Tests

1. **Test for CR-CR1 Musical Convention**

   Test ID: CR-CR1
   Type: Functional, Static
   Initial State: The application is open, ready to display and create sheet music.
   Input/Condition: Use the app to create and display sheet music, verifying adherence to Western music notation (University, 2024).
   Output/Result: The generated and displayed sheet music should accurately follow Western music notation standards.
   How Test Will Be Performed:

   (a) Generate a sample sheet music file using the app, incorporating various standard Western notation elements (e.g., treble and bass clefs, notes, rests, time signatures).

   (b) Compare the sheet music against standard Western notation guidelines, ensuring proper symbol usage and layout.

   (c) Confirm that all notation adheres to Western music conventions without deviations or omissions.

2. **Test for CR-CR2 Expected Music Theory Level of Users**

   Test ID: CR-CR2
   Type: Usability, Dynamic
   Initial State: The application is open and available to users with varying levels of music theory knowledge.
   Input/Condition: Users with basic to intermediate knowledge of music theory interact with the app.
   Output/Result: The app should be intuitive and accessible, allowing users to navigate and use core features comfortably.
   How Test Will Be Performed:

   (a) Recruit a sample group of users with basic to intermediate music theory knowledge.

   (b) Have users complete a task (e.g., creating a simple sheet of music) and observe their interaction with the app.

   (c) Provide tutorials or guides as needed and gather feedback on their clarity and usefulness.

   (d) Verify that users can successfully navigate and use the application without advanced music theory knowledge, meeting usability expectations.

3. **Test for CR-LR1 Copyright Issues**

   Test ID: CR-LR1
   Type: Functional, Static
   Initial State: The application is open, ready for user interaction.
   Input/Condition: Observe the app's interface for the presence of a copyright disclaimer.
   Output/Result: The app should display a clear disclaimer advising users on copyright compliance related to audio input.
   How Test Will Be Performed:

   (a) Open the application and navigate to any sections that mention user responsibilities or usage terms.

   (b) Confirm that a disclaimer is present, informing users of copyright restrictions and advising compliance.

(c) Verify that instructions are clear, guiding users on ensuring their audio inputs do not violate copyright laws.

4. **Test for CR-SCR1 Technological Standards**

   Test ID: CR-SCR1
   Type: Functional, Static
   Initial State: The application has a completed sheet music file ready for export.
   Input/Condition: Export sheet music in MusicXML format and verify the use of third-party libraries.
   Output/Result: The app should export error-free MusicXML files and have documentation of all third-party libraries.
   How Test Will Be Performed:

   (a) Generate a sheet music file within the app and export it in MusicXML format.

   (b) Open the exported file in compatible notation software (TBD) to confirm it is error-free and follows the MusicXML standard.

   (c) Review the app's documentation to verify that all third-party libraries are listed with their licensing terms for compliance purposes.

4.3 Traceability Between Test Cases and Requirements

For ease of traceability, test cases have been named such that each ID is identical to the ID of the functional or nonfunctional requirement it is intended to verify and validate.

Table 3: Requirements Traceability Matrix

SRS Location	Requirement ID	Test IDs
Section 9.1	FR-AR1	FR-AR1-3-1 FR-AR1-3-2
	FR-AR2	FR-AR2
	FR-AR3	FR-AR1-3-1 FR-AR1-3-2

SRS Location	Requirement ID	Test IDs
Section 9.2	FR-SP1	FR-SP1
	FR-SP2	FR-SP2
	FR-SP3	FR-SP3
	FR-SP4	FR-SP4
	FR-SP5	FR-SP2
Section 9.3	FR-SMG1	FR-SMG1
	FR-SMG2	FR-SMG2
	FR-SMG3	FR-SMG3
Section 9.4	FR-UI1	FR-UI1
	FR-UI2	FR-UI2
	FR-UI3	FR-UI3
Section 9.5	FR-SL1	FR-SL1
	FR-SL2	FR-SL2
Section 10.1	LF-A1	LF-A1
	LF-A2	LF-A2
	LF-A3	LF-A3
Section 10.2	LF-S1	LF-S1
Section 11.1	UH-EOU1	UH-EOU1
Section 11.2	UH-PI1	UH-PI1
Section 11.3	UH-L1	UH-L1
Section 11.4	UH-UP1	UH-UP1
	UH-UP2	UH-UP2
Section 11.5	UH-A1	UH-A1
Section 12.1	PR-SL1	PR-SL1
	PR-SL2	PR-SL2
Section 12.2	PR-SC1	PR-SC1
	PR-SC2	PR-SC2
Section 12.3	PR-PA1	PR-PA1
	PR-PA2	PR-PA2
	PR-RFT1	PR-RFT1

Section 12.4

SRS Location	Requirement ID	Test IDs
	PR-RFT2	PR-RFT2
	PR-RFT3	PR-RFT3
	PR-RFT3	PR-RFT3
	PR-RFT3	PR-RFT3
	PR-RFT3	PR-RFT3
	PR-RFT3	PR-RFT3
Section 12.5	PR-C1	PR-C1
	PR-C2	PR-C2
Section 12.6	PR-SE1	Omitted
	PR-SE2	PR-SE2
	PR-SE3	PR-SE3
Section 12.7	PR-L1	PR-L1
Section 13.1	OE-EP1	OE-EP1
	OE-EP2	OE-EP2
	OE-EP3	OE-EP3
	OE-EP4	OE-EP4
Section 13.2	OE-WE1	OE-WE1
Section 13.3	OE-IA1	OE-IA1
	OE-IA2	OE-IA2
	OE-IA3	OE-IA3
Section 13.4	OE-P1	OE-P1
	OE-P2	OE-P2
	OE-P3	OE-P3
	OE-P4	OE-P4
Section 13.5	OE-R1	OE-R1
Section 14.1	MS-M1	MS-M1
	MS-M2	MS-M2
Section 14.2	MS-S1	MS-S1
	MS-S2	MS-S2
Section 14.3	MS-A1	MS-A1

SRS Location	Requirement ID	Test IDs
	MS-A2	MS-A2
Section 15.1	S-A1	S-A1
Section 15.2	N/A	N/A
Section 15.3	S-P1	S-P1
	S-P2	S-P2
	S-P3	S-P3
Section 15.4	N/A	N/A
Section 15.5	N/A	N/A
Section 16.1	CR-CR1	CR-CR1
	CR-CR1	CR-CR1
Section 17.1	CR-LR1	CR-LR1
Section 17.2	CR-SCR1	CR-SCR1

5 Unit Test Description

This section will remain blank for revision 0 of the VnV plan, as the design document has not been completed yet.

5.1 Unit Testing Scope

N/A.

5.2 Tests for Functional Requirements

N/A.

5.3 Traceability Between Test Cases and Modules for Unit Tests

N/A.

Appendices

A Symbolic Parameters

For Revision 0 we do not have any implementation details of the symbolic parameters for the tests, however, this section will be filled out as we progress. audio file

B Usability Survey Questions

The following usability survey will be provided to user testers alongside this VnV plan and the SRS document.

Part 1: User Demographics

1. How old are you?
    - 0-11
    - 12-16
    - 17-24
    - 24+
2. What is your level of expertise in reading sheet music?
    - No knowledge whatsoever
    - Beginner
    - Intermediate
    - Advanced
    - Expert
3. How frequently do you use music transcription or audio analysis software?
    - Daily
    - Weekly
    - Monthly
    - Rarely

- Never

4. What is your primary purpose for using ScoreGen?

    - Personal music practice
    - Music transcription
    - Composing original songs
    - Academic research
    - Other (please specify): _____

Part 2: Ease of Use

5. How intuitive was the user interface for ScoreGen?

    - Very intuitive
    - Somewhat intuitive
    - Neutral
    - Somewhat unintuitive
    - Very unintuitive

6. Did you find the user guide and any in-application instructions sufficient for using the software?

    - More than sufficient
    - Sufficient
    - Neutral
    - Somewhat insufficient
    - Very insufficient

7. How easy was it to generate sheet music from an audio file?

    - Very easy
    - Somewhat easy
    - Neutral
    - Somewhat difficult

- Very difficult

8. How easy was it to generate sheet music from live input (i.e., through your microphone or audio port)?

    - Very easy
    - Somewhat easy
    - Neutral
    - Somewhat difficult
    - Very difficult

Part 3: Performance and Accuracy

9. How accurate was the transcribed sheet music to the provided audio input?

    - Very accurate
    - Mostly accurate
    - Somewhat inaccurate
    - Very inaccurate
    - I can't read sheet music

10. Did the VnV plan cover sufficient cases for addressing the accuracy of a variety of audio types (e.g., different instruments, polyphonic vs. monophonic sounds)?

    - More than sufficient
    - Sufficient
    - Neutral
    - Somewhat insufficient
    - Very insufficient

11. Were there any audio types or formats where the system performed noticeable poorly?

    - No
    - Yes (please specify): _____

12. How satisfied were you with the speed of the software (e.g., start-up time, length of time between audio input and score generation)?

    - Very satisfied
    - Satisfied
    - Neutral
    - Dissatisfied
    - Very dissatisfied

Part 4: Overall Satisfaction

13. Ho satisfied are you with ScoreGen as a whole?

    - Very satisfied
    - Satisfied
    - Neutral
    - Dissatisfied
    - Very dissatisfied

14. Would you recommend this software to others?

    - Definitely
    - Probably
    - Not sure
    - Probably not
    - Definitely not

15. If comfortable, please provide your favourite part of this experience.

    ---

16. If comfortable, please provide your least favourite part of this experience.

    ---

17. Any additional comments or suggestions?

C Supplementary Testing Information

C.1 Ommitted Testing

While the majority of the functional and nonfunctional requirements have associated test cases, some were omitted for various reasons. This section explains their omission from the document.

1. FR-SP4 Processing Limitations
   Appears under SRS section 9.2 Signal Processing.
   This functional requirement does not have a corresponding test case. The nature of this functional requirement demands more technical knowledge about implementation details that have not yet been decided. The exact definition of 'overly complex audio input' shall remain ambiguous at this stage to prevent the overextension (or underextension) of the development team's capabilities during the project's irregular time constraints.

2. PR-SE1 User Base Growth
   Appears under SRS section 12.6
   This nonfunctional requirement will not be tested as the scope of having 1000 users simultaneously using the application doesn't make sense for our use case.

C.2 Testing Data and Files

Requirement test cases may have input that overlap with other test cases, as such, the testing data and files have not specifically been assigned to any one test case. Testing data is available for any test case that may require them. Test files and datasets are available under the test folder in the project's GitHub repository (Gut, 2020; Recordings, 2024a,b; digifishmusic, 2010; Hooktheory, 2024; Judge, 2023; PianoMother, 2024).

References

Web content accessibility guidelines. https://www.w3.org/TR/WCAG21/, 2024. Accessed: 2024-10-11.

BEM Community. Bem – block element modifier methodology. https://getbem.com/, 2024. Accessed: 2024-10-30.

digifishmusic. Pack: Scales musical. https://freesound.org/people/digifishmusic/packs/6116/, 2010. Accessed: 2024-10-31.

Sample Focus. Long cinematic fx. https://samplefocus.com/samples/long-cinematic-fx-riser-pan, 2024. Accessed: 2024-10-31.

Google. Google's c++ style guide. https://google.github.io/styleguide/cppguide.html, 2024. Accessed: 2024-10-30.

Kevin Gut. Empty files. https://cable.ayra.ch/empty/, 2020. Accessed: 2024-10-31.

Harvard Business School. What is human-centered design. https://online.hbs.edu/blog/post/what-is-human-centered-design, 2024. Accessed: 2024-10-31.

Hooktheory. C major cheat sheet. https://www.hooktheory.com/cheat-sheet/key/c/major, 2024. Accessed: 2024-10-31.

Jon Judge. C major scale. https://www.basicmusictheory.com/c-major-scale, 2023. Accessed: 2024-10-31.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. Development plan. https://github.com/emilyperica/ScoreGen/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2024a.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. Hazard analysis. https://github.com/emilyperica/ScoreGen/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf, 2024b.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. Module guide. https://github.com/emilyperica/ScoreGen/blob/main/docs/Design/SoftArchitecture/MG.pdf, 2024c.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. Module interface specification. https://github.com/emilyperica/ScoreGen/blob/main/docs/Design/SoftDetailedDes/MIS.pdf, 2024d.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. Problem statement and goals. https://github.com/emilyperica/ScoreGen/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2024e.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. System requirements specification. https://github.com/emilyperica/ScoreGen/blob/main/docs/SRS-Volere/SRS.pdf, 2024f.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. User guide. https://github.com/emilyperica/ScoreGen/blob/main/docs/UserGuide/UserGuide.pdf, 2024g.

Emily Perica, Ian Algenio, Jackson Lippert, and Mark Kogan. Verification and validation report. https://github.com/emilyperica/ScoreGen/blob/main/docs/VnVReport/VnVReport.pdf, 2024h.

PianoMother. Hot cross buns. https://www.pianomother.com/Hot_Cross_Buns.html, 2024. Accessed: 2024-10-31.

Python Software Foundation. Pep 8 – style guide for python code. https://peps.python.org/pep-0008/, 2024. Accessed: 2024-10-30.

YourAccompianist Piano Recordings. Reference scales. https://www.youraccompanist.com/free-scales-and-warm-ups/reference-scales, 2024a. Accessed: 2024-10-31.

YourAccompianist Piano Recordings. Reference scales. https://www.youraccompanist.com/free-scales-and-warm-ups/reference-scales, 2024b. Accessed: 2024-10-31.

Indiana University. Music notation style guide. https://blogs.iu.edu/jsomcomposition/music-notation-style-guide/, 2024. Accessed: 2024-10-31.

Reflections

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?

   Mark This deliverable went very well. I feel that our group was able to work separately on different sections of the document, yet when the changes were merged together, the results fit well together. This suggests we operate together smoothly without need of excessive feedback.

   Ian The effort and work we put into our initial SRS paid off very well for this deliverable, especially when it came to functional requirements and the structure that was used in the SRS. This structure and breakdown naturally fit well into the breakdown required by the creation of test cases for specific testing areas. Additionally, I think our team communicated very well while writing this deliverable, both individually for parts that were dependent on multiple members and team-wide.

   Emily Communication within the team for this deliverable was the best it has been to date. It had been lacking during our last deliverable, which led to a less cohesive document and team-wide project issues not being fully adressed. This was a good wake-up call for us, and as a result we set explicit roles for all aspects of this deliverable, removing the concept of 'group' issues to make sure we don't get stuck in the cycle of assuming someone else is getting things done.

Jackson I think the best part of this deliverable was how clear the requirements were. I feel as though as a group we are getting used to what we need to deliver to achieve success in each milestone, and in this document in particular I found that we were easily able to divide up the work and complete it efficiently. Additionally, trying to get the main content done a day before the submission deadline helped us to have a buffer of time to fix things up.

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

   Mark There were no particular pain points in this deliverable, though there was significant volume of writing out each section, and there had to be significant recall of previous documents to reference them where necessary.

   Ian The largest pain point for me was creating explicit differences between many test cases that share not only the same input, but relatively similar expected results/output. Many of the tests could cover multiple functional aspects but in order to keep the approximate 1:1 ratio of requirements to test cases, this required deeper thought into how the test cases can be effectively differentiated. Ultimately this was beneficial as it further separates functionalities that often depend on one another due to the nature of the project.

   Emily My biggest pain point was getting over the concept of writing a verification and validation plan before writing any code. I find it difficult to conceptualize abstract tests that are not implementation-specific, so this document was good practice in creating system-wide tests, as opposed to the unit tests I'm more comfortable with writing.

   Jackson The biggest pain point in my opinion was the high coupling between the NFR and FR sections. This caused me to have to wait on creating some sections for the non-functional requirements because the performance requirements specifically were very similar to the functional requirements. Additionally, we had to make sure there were no merge conflicts when adding the mapping chart between test IDs and requirements.

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 knowl-

edge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

Proper verification and validation of this project will require the team to gain both technical and procedural knowledge on performing effectives tests. Gaining proficiency in testing frameworks such as Google Test will allow the team to design and execute a comprehensive test suite, as frameworks will become the base of our unit and integration tests, as well as in our implementation of a regression test suite. The team will need to become comfortable with YAML syntax, as well as its uses in configuration management, to ensure we use GitHub Actions for automated testing to the fullest of its ability. Usage of YAMLs in testing can greatly improve modularity and will make the process of changing test data very easy, if this is ever necessary. Static testing knowledge is a vital aspect of VnV, and all team members will aim to increase their static testing capabilities, focusing on code walkthroughs, inspections, and manual code reviews. Creating a VnV plan is useless without thorough documentation for the VnV report, and thus a vital skill to this aspect of the project is being able to write thorough documentation and taking detailed notes at every step of the process.

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?

   Proficiency in testing frameworks

   APPROACH 1: Apply a framework such as Google Test to some example project, and practice creating unit, integration, and system tests.
   APPROACH 2: Enroll in and complete an online course covering the usage of a specific test framework.

   Mark I would prefer to practice some basic implementation, but begin applying the skills on the actual project ASAP and learn as I go. Overall for the initial learning I'd rather use approach 1.

   Emily I often prefer learning by doing, rather than reading from a textbook, so I will pursue approach 1.

   Ian Due to the tight timing, approach 1 is more attractive. I will pursue this approach since I'm already enrolled in enough courses for the

semester.

Jackson Likely approach 1, I'd rather be hands-on.

YAML Syntax

APPROACH 1: Practice writing YAML syntax by creating GHA workflows.
APPROACH 2: Review YAML files of public projects, such as those from previous years of the capstone course or large open-source projects found on GitHub.

Mark I would prefer approach 2 where needed. It is faster for learning in the short term and I can get the practical experience I need when working on crucial workflows for the project.

Emily Approach 1 is more hands-on and thus applies more to my learning style than approach 2, so this will be my preferred way of gaining YAML knowledge.

Ian I will pursue approach 2 as it can provide me with a gauge of what is expected for this course in terms of GHA and YAML files, while also allowing me to view them in a variety of context-specific applications.

Jackson Approach 2 is more attractive to me, I do like reading reference material if I know it's correct.

Static testing knowledge

APPROACH 1: Identify various tools used for static testing (ex. SonarQube, ESLint) and incorporate them in existing or example projects to get comfortable with their usage.
APPROACH 2: Research and go over existing literature outlining static testing methodologies and best practices.

Mark I would prefer Approach 2 to get a strong overview of the best principles to follow before applying the skills on our capstone project.

Emily I will pursue approach 1, as this method is very similar to how we will use static testing within our project and can help me to get comfortable with static testing tools in a relevant environment.

Ian Approach 1 makes more sense to me for the purposes of this course and the rate we should be improving our static testing knowledge. Practice will accelerate this process.

Jackson Approach 1 is better in the case of our project, and I'd rather do that than go through all that documentation about 'best practices'.

Documentation

APPROACH 1: Write up a variety of reports in different styles using dummy test results.
APPROACH 2: Review course notes from SFWRENG 3RA3 as well as all standards described in Dr. Smith's lectures.

Mark I will choose approach 2 as it is much quicker and it is an effective way to grasp a lot of information quickly.

Emily I will pursue approach 2, as the ease with which I can access the necessary knowledge sources will make it simple for me to learn proper documentation guidelines.

Ian I'll choose approach 2 in this case as it's more specific to the expectations of the course.

Jackson Since I already took 3RA3, I believe I'd rather review my course notes as a starting point.