System Verification and Validation Plan for ScoreGen

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

October 28, 2024

Revision History

Date	Version	Notes
01/11/2024	0	Initial draft

Contents

List of Tables

List of Figures

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

1 Symbols, Abbreviations, and Acronyms

See the SRS document for additional definitions.

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

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.

2 General Information

2.1 Summary

This Verification and Validation Plan describes the testing process for Score-Gen, 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

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

3.1 Verification and Validation Team

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

3.2 SRS Verification Plan

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

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

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

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

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

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

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

[Create a checklists? —SS]

3.5 Implementation Verification Plan

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

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

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

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

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

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

3.7 Software Validation Plan

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

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

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

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

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

4 System Tests

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

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

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

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   Initial State:

   Input:

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

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

   How test will be performed:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

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

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

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

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

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

4.2.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

Output:

How test will be performed:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

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

5 Unit Test Description

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

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

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

5.1 Unit Testing Scope

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

5.2 Tests for Functional Requirements

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

5.2.1 Module 1

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

1. test-id1

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

   Initial State:

   Input:

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

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

   How test will be performed:

2. test-id2

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

   Initial State:

   Input:

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

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

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

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

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

5.3.1 Module ?

1. test-id1

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

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

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

References

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.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions?

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

Appendix — Reflection

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?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.
4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?