Module Guide for ScoreGen

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

April 4, 2025

1 Revision History

Date	Version	Notes
2025-01-17	0.0	Initial version
2025-02-05	0.1	Issue #210, Issue #211
2025-04-03	1.0	Issue #313
2025-04-04	1.1	Issue #311 TA feedback

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

Symbol	Description
AC	Anticipated Change
API	Application Programming Interface
DAG	Directed Acyclic Graph
DSP	Digital Signal Processing
I/O	Input/Output
M	Module
MG	Module Guide
MIDI	Musical Instrument Digital Interface
MIS	Module Interface Specification
NFR	Non-Functional Requirement
OS	Operating System
PDF	Portable Document Format
R	Requirement
SRS	Software Requirements Specification
UC	Unlikely Change
UI	User Interface
XML	eXtensible Markup Language
ScoreGen	An audio to sheet music generator

Contents

1 Revision History **i**

2 Reference Material **ii**
2.1 Abbreviations and Acronyms ii

3 Introduction **1**

4 Anticipated and Unlikely Changes **2**
4.1 Anticipated Changes 2
4.2 Unlikely Changes 2

5 Module Hierarchy **3**

6 Connection Between Requirements and Design **3**

7 Module Decomposition **4**
7.1 Hardware Hiding Module (M1) 4
7.2 Behaviour-Hiding Modules 4
7.2.1 User Interface Module (M2) 4
7.2.2 Score Generation Module (M3) 5
7.2.3 File Format Conversion Module (M6) 5
7.3 Software Decision Modules 5
7.3.1 Raw Signal Processing Module (M4) 5
7.3.2 Audio Feature Extraction Module (M5) 6
7.3.3 Audio Recording and Playback Module (M7) 6

8 Traceability Matrix **6**

9 Use Hierarchy Between Modules **8**

10 User Interfaces **8**

11 Timeline **9**

Appendix A — Formal Math Specifications **11**

List of Tables

1 Module Hierarchy 3
2 Trace Between Requirements and Modules 7
3 Trace Between Anticipated Changes and Modules 7
4 Task Completion Timeline 9

List of Figures

1 Use Hierarchy Among Modules 8

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between

modules. Section 10 illustrates the design of the user interface through drawings, sketches, and design tool prototyping (e.g. Figma, Marvel, etc.). Finally, section 11 outlines the schedule of tasks related to the development of ScoreGen as well as the development team members responsible for these tasks.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here adheres to the development principle called design for change.

AC1: The specific hardware on which the software is running.

AC2: Musical element extraction and mapping techniques.

AC3: Monophonic audio processing algorithm(s).

AC4: Complex audio processing algorithm(s).

AC5: Performance and latency benchmarks and targets.

Anticipated changes AC3 and AC4 refer to mathematical framework changes that impact core signal processing functionality. Two main methodologies are considered for pitch detection: autocorrelation-based and Fourier-based techniques. For the full mathematical specifications of these methodologies, please refer to Appendix A — Formal Math Specifications.

4.2 Unlikely Changes

The module design aims to be as general as possible. If some design decisions should need to be changed later, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: File format of input data.

UC2: File format of output audio data.

UC3: Post-generation editing and progress saving.

UC4: Choice of sheet music notation.

UC5: User interface features, requirements, usability, styling, etc.

UC6: Non-functional requirements are unlikely to change.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will be implemented.

M1: Hardware-Hiding Module

M2: User Interface Module

M3: Score Generation Module

M4: Raw Signal Processing Module

M5: Audio Feature Extraction Module

M6: File Format Conversion Module

M7: Audio Recording/Playback Module

Level 1	**Level 2**
Hardware-Hiding Module	-
Behaviour-Hiding Module	User Interface Module Score Generation Module File Format Conversion Module
Software Decision Module	Raw Signal Processing Module Audio Feature Extraction Module Audio Recording and Playback Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by (Parnas et al., 1984) . The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *ScoreGen* means the module will be implemented by the ScoreGen software. Only the leaf modules in the hierarchy will be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Module (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Modules

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 User Interface Module (M2)

Secrets: Style and layout standards, user input management, and event- and display-handling.

Services: Provides the user with an intuitive, graphical interface for interacting with the system. Maps user input to appropriate system functions. Dynamically updates the display to reflect user interactions.

Implemented By: ScoreGen.

Type of Module: Interface

7.2.2 Score Generation Module (M3)

Secrets: Musical notation standards and conventions used by the system.

Services: Ends the audio to sheet music transcription pipeline by providing the user with viewable, human-readable sheet music. Maps pre-processed audio data to elements used by viewable or exportable file formats such as MusicXML, PDF, etc. Renders the viewable sheet music for the UI to display to the user.

Implemented By: ScoreGen, libmusicxml.

Type of Module: Abstract Data Type, encapsulates musicxml score file generation.

7.2.3 File Format Conversion Module (M6)

Secrets: Details of reading and writing to files in order to perform conversions from one file format to another.

Services: Converts files to other supported formats (e.g., .wav to .musicxml).

Implemented By: OS, ScoreGen back-end, external libraries.

Type of Module: Library, contains reusable routines for file format conversions.

7.3 Software Decision Modules

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Raw Signal Processing Module (M4)

Secrets: The underlying mathematical and physical principles that govern sound waves and their transformation into digital representations. Techniques used to transform time domain audio signals into the frequency domain.

Services: Digital signal processing. Takes in raw audio data and transforms them into frequency domain representations. Provides the system with data prepared for subsequent audio feature extraction.

Implemented By: External libraries, ScoreGen back-end.

Type of Module: Abstract Data Type, encapsulates preprocessing.

7.3.2 Audio Feature Extraction Module (M5)

Secrets: The techniques and algorithms chosen to derive pitch, key signature, rhythm, and timing from frequency domain data.

Services: Detection and classification of musical elements such as pitches, key signature, rhythm, and timing. Identifies pitches based on dominant frequencies. Classifies musical characteristics including key signature, rhythm, and timing. Provides the system with data prepared for subsequent score generation.

Implemented By: External libraries, ScoreGen back-end.

Type of Module: Library, contains reusable methods for extracting musical elements from pre-processed data.

7.3.3 Audio Recording and Playback Module (M7)

Secrets: Audio input and output management and related device-specific configurations.

Services: Allows the user to capture audio with a device of their choosing and to listen to their recordings. Sets and configures input and output streams for the device ScoreGen is running on. User-directed recording (i.e. start, stop, playback).

Implemented By: OS, external libraries.

Type of Module: Library, contains reusable routines for accessing and configuring audio I/O.

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR-AI1	M1, M2, M7
FR-AI2	M2, M7
FR-AI3	M2, M4
FR-AI4	M2
FR-SP1	M4
FR-SP2	M5
FR-SP3	M5
FR-SP4	M4
FR-SP5	M4
FR-SP6	M4
FR-SMG1	M3, M5
FR-SMG2	M3, M5
FR-SMG3	M2, M3
FR-UI1	M2, M7
FR-UI2	M2
FR-UI3	M2
FR-UI4	M2
FR-SL1	M2, M6
FR-SL2	M2, M6

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M5
AC3	M4
AC4	M4
AC5	M4, M5

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

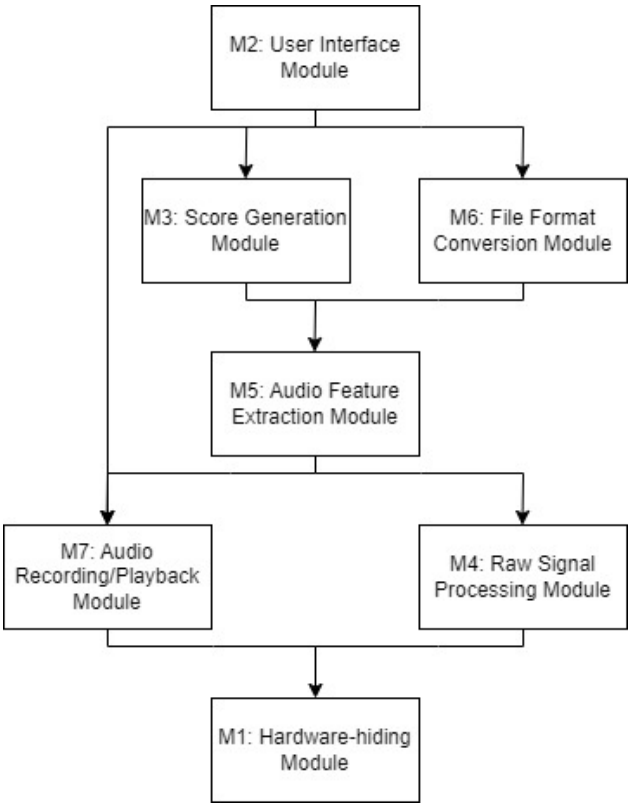

Figure 1: Use Hierarchy Among Modules

10 User Interfaces

- Rough ideas for design in Figma is available at this link.
- Logo design is available at this link.

11 Timeline

The following table breaks down the tasks that must be completed in order to accurately implement all modules of the software.

Task	**Developer**	**Completion Date**
M2 Implementation	Jackson	Jan 20/25
M2 Unit Testing	Jackson	Jan 27/25
M3 Implementation	Ian	Jan 20/25
M3 Unit Testing	Ian	Jan 27/25
M4 Implementation	Emily, Ian	Nov 15/24
M4 Unit Testing	Emily, Ian	Jan 27/25
M5 Implementation	Emily, Ian, Mark, Jackson	Jan 20/25
M5 Unit Testing	Emily, Ian, Mark, Jackson	Jan 27/25
M6 Implementation	Mark	Jan 20/25
M6 Unit Testing	Mark	Jan 27/25
M7 Implementation	Emily	Jan 11/25
M7 Unit Testing	Emily	Jan 27/25
Integration Testing	Emily, Ian, Mark, Jackson	Feb 3/25

Table 4: Task Completion Timeline

References

Simon Dixon. Onset detection. In *Proc. of the 9th Int. Conference on Digital Audio Effects (DAFx'06)*, 2006.

Fredric J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. In *Proceedings of the IEEE*, volume 66, 1978.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

Lawrence Rabiner and Robert W Schafer. On the use of autocorrelation analysis for pitch detection. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(1):24–33, 1977.

Lawrence R. Rabiner and Robert W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.

Ruohua Zhou and Joshua D. Reiss. Music onset detection. In Wenwu Wang, editor, *Machine Audition: Principles, Algorithms and Systems*, pages 297–316. IGI Global, 2010. ISBN 978-1-61520-919-4. doi: 10.4018/978-1-61520-919-4.ch012.

Appendix A — Formal Math Specifications

This appendix provides detailed mathematical formulations for two pitch detection methodologies that may be employed for this project's signal processing: autocorrelation-based and Fourier-based techniques. These specifications are general and intended to be used as a simplified overview of the algorithms. The actual implementation may vary based on emprical performance.

A.1 Autocorrelation-based Pitch Detection

Description: The autocorrelation method is a time-domain approach that analyzes the periodicity of a signal to determine its fundamental frequency. It does so by computing the correlation between the signal and time-shifted versions of itself over a range of lags. The lag corresponding to the highest normalized correlation is identified as the pitch period, from which the fundamental frequency is derived.
The formulation below is based on the autocorrelation technique for pitch detection as detailed in Rabiner and Schafer (1977) and further discussed in Rabiner and Schafer (1978).

Mathematical Formulation: Let $x[n]$ be a discrete-time signal defined for $n = 0, 1, \ldots, N-1$. The total energy of the signal is computed as:

$$r_0 = \sum_{n=0}^{N-1} x[n]^2$$

For a given lag ℓ, the autocorrelation function is defined as:

$$r(\ell) = \sum_{n=0}^{N-\ell-1} x[n]\, x[n+\ell]$$

The normalized autocorrelation is then given by:

$$R(\ell) = \frac{r(\ell)}{r_0}$$

The algorithm searches for the lag ℓ^* within a predefined range $[\ell_{\min}, \ell_{\max}]$ that maximizes $R(\ell)$:

$$\ell^* = \underset{\ell \in [\ell_{\min}, \ell_{\max}]}{\arg\max}\ R(\ell)$$

If the maximum normalized autocorrelation $R(\ell^*)$ exceeds a threshold (e.g., 0.5), the fundamental frequency f_0 is estimated by:

$$f_0 = \frac{f_s}{\ell^*}$$

where f_s is the sampling rate. If $R(\ell^*) < 0.5$, no clear pitch is detected (i.e., $f_0 = 0$).

Parameter Constraints: The search range for ℓ is derived from the desired frequency bounds:

$$\ell_{\max} = \min\left(N-1,\ \left\lfloor \frac{f_s}{f_{\min}} \right\rfloor\right), \quad \ell_{\min} = \max\left(1,\ \left\lfloor \frac{f_s}{f_{\max}} \right\rfloor\right)$$

with typical values $f_{\min} = 100\,\text{Hz}$ and $f_{\max} = 2000\,\text{Hz}$.

Relevant Details: To reduce computational load, the lag ℓ may be incremented in steps (e.g., by 2 samples). The procedure returns the fundamental frequency f_0 if a valid peak is found; otherwise, it indicates the absence of a clear pitch by returning 0.

A.2 Fourier-based Pitch Detection

Description: The Fourier-based method analyzes the frequency content of a signal using the Fast Fourier Transform (FFT). It computes the frequency spectrum of the signal, identifying the peak frequency as the pitch.

Mathematical Formulation: The following describes the three major mathematical components that are considered for this project's signal processing framework:

1. *The Input Signal:* Let $x : \mathbb{Z} \to \mathbb{R}$ be a discrete-time signal with finite energy:

$$E = \sum_{n=-\infty}^{\infty} |x[n]|^2 < \infty$$

   This formalizes the representation of our raw audio data.

2. *The Window Function:* To localize the Fourier analysis to specific segments and reduce spectral leakage, a window function is applied Harris (1978). For a window length $N \in \mathbb{Z}^+$ (with $N \geq 1$), we define the Hanning window $w : \{0, 1, \ldots, N-1\} \to \mathbb{R}$ as:

$$w(n) = \begin{cases} 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right), & 0 \leq n \leq N-1, \\ 0, & \text{otherwise} \end{cases}$$

3. *The Short-Time Fourier Transform (STFT):* The STFT converts segments of the time-domain signal into the frequency domain, enabling the analysis of evolving spectral content Dixon (2006); Zhou and Reiss (2010). Given a hop size $H \in \mathbb{Z}^+$, each frame of the signal is defined as:

$$\{x[mH + n] : n = 0, 1, \ldots, N-1\}$$

   where $m \in \mathbb{Z}$ is the frame index. The Fourier transform of the windowed frame is computed by:

$$X(m,k) = \sum_{n=0}^{N-1} x[mH+n]\, w(n)\, e^{-j\frac{2\pi}{N}kn}$$

   with $k \in \{0, 1, \ldots, N-1\}$ representing the frequency bin index.