Module Interface Specification for Software Engineering

Team #22, TeleHealth Insights
Mitchell Weingust
Parisha Nizam
Promish Kandel
Jasmine Sun-Hu

January 15, 2025

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of Media Processing Module

6.1 Module

MediaProcessingModule

6.2 Uses

VideoProcessingModule, AudioProcessingModule

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
processMedia	mediaFile: str, mediaType: str, assessmentID: str	report: MediaAnalysisReport	MediaProcessingException

6.4 Semantics

6.4.1 State Variables

- processedMedia: Map(assessmentID, MediaAnalysisReport) - stores combined results from video and audio analysis.

6.4.2 Environment Variables

N/A

6.4.3 Assumptions

- mediaType specifies whether the file is video or audio (e.g., ”video”, ”audio”).
- Delegates processing to VideoProcessingModule or AudioProcessingModule based on mediaType.

6.4.4 Access Routine Semantics

processMedia():

- transition:
  - If mediaType is ”video”, calls processVideo from VideoProcessingModule.
  - If mediaType is ”audio”, calls processAudio from AudioProcessingModule.
  - Combines results into a single MediaAnalysisReport per assessment completed.
- output: Returns a MediaAnalysisReport with details from video and audio analyses.
- exception: Throws MediaProcessingException if the file cannot be processed or delegated.

6.4.5 Local Functions

N/A

7 MIS of Video Processing Module

7.1 Module

VideoProcessingModule

7.2 Uses

N/A MediaProcessingModule

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
processVideo	videoFile: str, assessmentID: str	report: VideoAnalysisReport	VideoProcessingException

7.4 Semantics

7.4.1 State Variables

- processedVideos: Map(assessmentID, VideoAnalysisReport) - stores results of processed videos.

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

- Video files are in a supported format (e.g., MP4, AVI).
- Video processing is done within a time threshold for real-time feedback.

7.4.4 Access Routine Semantics

processVideo():

- transition: Analyzes the video to identify any disturbances, bias, or cheating patterns.
- output: Returns a detailed VideoAnalysisReport containing flagged events and metrics.

- exception: Throws VideoProcessingException if the file cannot be processed or analyzed.

7.4.5 Local Functions

N/A

8 MIS of Audio Processing Module

8.1 Module

AudioProcessingModule

8.2 Uses

N/A MediaProcessingModule

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
processAudio	audioFile: str, assessmentID: str	report: AudioAnalysisReport	AudioProcessingException

8.4 Semantics

8.4.1 State Variables

- processedAudio: Map(assessmentID, AudioAnalysisReport) - stores results of processed audio.

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

- Audio files are in a supported format (e.g., WAV, MP3).
- Background noise levels are detectable and quantifiable.

8.4.4 Access Routine Semantics

processAudio():

- transition: Analyzes the audio for disturbances such as background noise or interruptions.
- output: Returns a detailed AudioAnalysisReport with flagged issues.
- exception: Throws AudioProcessingException if the file cannot be processed or analyzed.

8.4.5 Local Functions

N/A

9 MIS of Logging Module

9.1 Module

LoggingModule

9.2 Uses

N/A

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
logEvent	eventType: str, message: str	status: bool	LoggingException
fetchLogs	logType: str, timeRange: TimeRange	logData: List(LogEntry)	LogFetchException
clearLogs	logType: str, timeRange: TimeRange	status: bool	LogClearException

9.4 Semantics

9.4.1 State Variables

- logs: Map(logType, List(LogEntry)) - stores all logs categorized by type.

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

- Logs are categorized by their type and timestamp for easy retrieval.
- Each log entry includes metadata such as the timestamp, severity, and module of origin.

9.4.4 Types of Logs

Event Logs

Tracks application activities, such as user logins, file uploads, and media processing events. Example: "User 'parent1' logged in successfully at 14:32:00." Error Logs

Captures unexpected behaviors, exceptions, or failures in the application. Example: "VideoProcessingException: Unable to analyze video file 'session123.mp4' due to corrupted data." Audit Logs

Records critical changes and actions for accountability, such as user role changes or log clearances. Example: "Admin user 'clinician1' updated parent access rights at 16:45:00." Performance Logs

Monitors application performance metrics like response times, memory usage, and processing durations. Example: "Media processing for 'session456' completed in 3.2 seconds with 200MB memory usage." Debug Logs

Includes detailed information for troubleshooting during development or maintenance. Example: "Entering function 'processVideo' with input file 'session789.mp4'." Security Logs

Tracks security-related events such as failed logins, access violations, or token expirations. Example: "Security alert: Failed login attempt for user 'parent2' at 18:12:30."

9.4.5 Access Routine Semantics

logEvent():

- transition: Adds a new log entry to the logs map under the appropriate eventType.
- output: Returns true if the log is successfully recorded.
- exception: Throws LoggingException if the log entry cannot be added.

fetchLogs():

- transition: Retrieves all logs of the specified logType within the provided timeRange.
- output: Returns a list of LogEntry objects matching the criteria.
- exception: Throws LogFetchException if no logs are found or retrieval fails.

clearLogs():

- transition: Removes all logs of the specified logType within the provided timeRange.
- output: Returns true if logs are successfully cleared.
- exception: Throws LogClearException if logs cannot be cleared.

9.4.6 Local Functions

N/A

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

10 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

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. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?
5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)
6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)