U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Arquitetura de Sistemas de Software
2023/2024

Homework #06 (part 1)

“Homework 05 Review”

Team 21

Members:
up202008569@edu.fe.up.pt - Ana Rita Carneiro
up202302747@edu.fe.up.pt - Artur Freitas
up202005097@edu.fe.up.pt - Pedro Balazeiro
up202007544@edu.fe.up.pt - Sérgio Carvalhais

Assessment Rubric
Assignment (which architecture): **HW06**

Which team are you reviewing: **T31**

What is your team: **T21**

Team: **T31**

Topic	Strengths and suggestions for the architects
Diagrams	Very Good
Clarity	Very Good. Figure 2 has just bidirectional lines and no legends attached.
Consistency	Good. Apart from the scenario diagrams, the others are very different from each other and they use different shapes to represent similar concepts.
Completeness	Very Good.
Sufficient Level of detail	Good. The text helps to explain, but without it there could be some ambiguities between the diagrams.
Text Description	Excellent
Clarity	Excellent. Very clear.
Consistency	Excellent.
Sufficient?	Very Good. Some parts could be more detailed.
Correctness	Good
Anything missed?	Very Good. How would you deal with security measures and data privacy?
Will it work?	Fair. There may be some problems due to the fact that all libraries are connected to one single controller where all credentials data are stored. What if it fails?
Presentation	Not in our class.
Summary	Very Good.

Explanation:

Diagrams

- Clarity: Are all the figures clear? Is it easy to understand what they mean? Are they laid out in an intuitive way? Is it clear what colors mean? If necessary, is there a legend?
- Consistency: Are the shapes and lines used consistently? Is there anything (a shape or line) that means two different things? Are colors consistent, if used?
- Complete: Is anything missing that would make the diagrams more clear?
- Sufficient level of detail: Do the diagrams give enough detail? For example, are there some components that should be broken down more?

Text Description

- Clarity: How clear is the description?
- Consistency: How consistent is the description with the diagrams?
- Sufficient: Does the text give enough information?

Correctness:

- Did the architects miss anything, such as did they forget a component?
- Will anything not work?

Summary:

- How confident are you that you could implement the design as it is described?

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Arquitetura de Sistemas de Software
2023/2024

Homework #06 (part 2)

“Understand Someone Else’s Architecture”

Team 21

Members:
up202008569@edu.fe.up.pt - Ana Rita Carneiro
up202302747@edu.fe.up.pt - Artur Freitas
up202005097@edu.fe.up.pt - Pedro Balazeiro
up202007544@edu.fe.up.pt - Sérgio Carvalhais

“The Architecture of Open-Source Systems” - Book 1 - Audacity

1.Summary

In this project, we went through a process aimed at uncovering the architecture of Audacity, an open-source audio editing software. We started with a collection of books called "The Architecture of Open-Source Systems" which is a trilogy that provides a description of more than 50 open-source systems. Our team was inspired to look deeper into the open-source scene of Audacity because of its impact and the possibility to learn more about the architectural field in an applied way. Our objective was not only to acquire the architectural principles of Audacity but also to discover and put into practice the engineering concepts and topics from the book on software engineering which will culminate in a deeper understanding of software engineering.

In the first section of our analysis we focused on the evolution of the architecture of Audacity from its inception as a student project to its current status as a widely used open-source audio editor. We emphasize its user-friendly interface and efforts to maintain a unified experience across its various components. Despite limitations imposed by licenses and resource constraints, Audacity continues to evolve, with a shift towards a modular structure enhancing flexibility while presenting challenges in testing and feature expansion. Overall, the ongoing development of Audacity reflects a commitment to improving software architecture and usability, with a focus on balancing flexibility, simplicity, and functionality as we will see later on.

After analyzing its architecture, we went through understanding the diagrams which were well detailed in the book we read but not as well explained. Our focus was to provide a definition about every entity of the diagrams and describe its interactions if not well explicit in the schemas. In this section it was made observations on the structural layers in Audacity, more precisely on how they work in processing the audio inputs and interpreting it.

Following the diagrams analysis we delved into the fundamental architectural patterns shaping Audacity's development. We started with the Model-View-Controller (MVC) pattern, which organizes the application into distinct components responsible not only for data management, but also for user interface presentation and user interaction handling. This separation promotes modularity and flexibility in Audacity's architecture. We then explored the layered architecture pattern, which divides the application into horizontal layers, each handling specific functionalities such as user interface presentation, core application logic, domain modeling, data persistence, and infrastructure support. In fact, the layered structure enhances maintainability, scalability, and testability of Audacity's software components. Additionally, we discussed the pipes and filters pattern, particularly evident in the scripting plugin, which facilitates modular and flexible data processing through a series of components connected by pipes. We ended up touching upon the Repository Pattern observed in the BlockFiles component, emphasizing its structured approach to managing audio data storage and retrieval.

In the last part of this report, named "Quality Attributes", we inferred about the core characteristics defining Audacity's architecture, including its focus on portability, flexibility, maintainability, performance, usability, reliability, extensibility, legality, and security. Our analysis highlighted how these attributes contribute to creating a robust and adaptable audio editing platform.

2.Audacity Architecture

Audacity, an open source audio editor and recorder which was developed in 1999 by Dominic Mazzoni, a student of the Carnegie Mellon University, has become a popular option among users. Initially, the app was oriented toward perfecting audio processing algorithms and it was only later that it became a favorite piece of software. In fact, this project has attracted a community of developers who furthered its development, documentation, and translation into multiple languages.

An advantage of Audacity software is its user interface that is created to be easy to use and investigable. Even though users collaborate with the software via different components without a single unifying style, attempts are made to provide a unified experience to users with different components of the software. It uses libraries such as PortAudio that provide low-level audio device management and wxWidgets that offer platform-independent GUI components among others.

On the flip side, though, the driving forces of designer's decisions do not consist of an exclusive consideration of the usability component. Licenses are a limiting factor in the direction that the tools can take, like for example the VST library. Financially, resource limitations matter a lot, as developers have to emphasize must-have functions and do away with features that require too many resources or consume more developer time than is necessary.

Audacity in fact has a scripting feature though it is not so versatile, having the plugin modules rather than the different scripting languages being directly embedded into the application. This approach aims at the elimination of the complexity and at having a simple and efficient architecture that does not compromise on functionality.

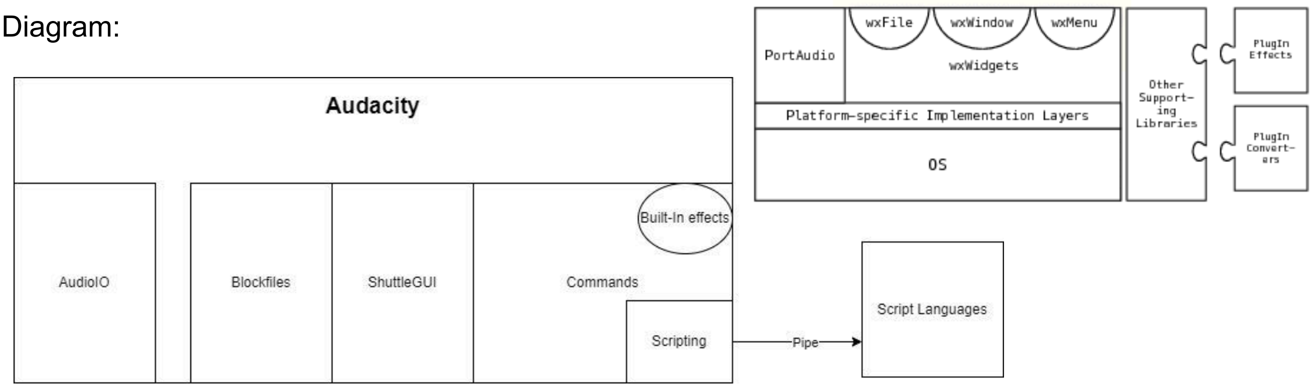
In terms of structure, Audacity has undergone evolution in a natural way throughout time. It started as a monolithic executable but later on adopted the modular structure for Windows, making it more flexible. And while this newness was accompanied by some trade-offs, particularly in terms of the speed of testing while opening up new possibilities for feature expansion, it offered the potential for a new level of efficiency.

In spite of all these hardships, Audacity has not lost its way and it is still improving. The development and addition of essential functionalities like surround sound support indicates a high level of dedication to the improvement of the software architecture and ease of use. Balancing the demands of flexibility, simplicity and functionality is a central issue to be addressed as the development progresses for the Audacity project. It was also taken into account the diagrams on the behavior of the threads and buffers in audio playback and recording and the behavior of the BlockFiles on deleting some of the audio data chunks they hold.

3.Diagrams

Structural Layers in Audacity

Diagram:



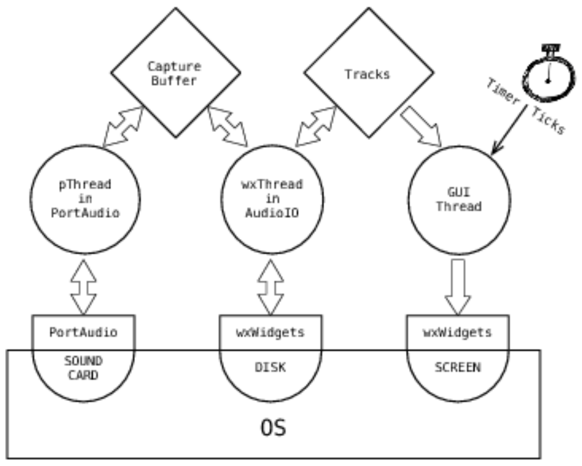
Topics covered:

- **AudioIO**: Manages audio input/output operations.
  - AudioIO ensures seamless interaction with hardware devices for recording, playback, and monitoring.
- **Blockfiles**: Organizes audio data into manageable blocks.
  - Blockfiles provide a structured approach to storing and manipulating audio data within Audacity.
- **ShuttleGUI**: Simplifies GUI element creation and management.
  - ShuttleGUI abstracts complexities, facilitating intuitive interface design and user interaction.
- **Commands**: Executes user-initiated actions within Audacity.
  - Commands enable users to perform various tasks such as editing, effects application, and project management.
- **Built-in effects**: Pre-installed audio processing tools.
  - Built-in effects offer users immediate access to a range of audio manipulation capabilities within Audacity.
- **Script languages pipe scripting**: Allows automation and customization via scripts.
  - Script languages and pipe scripting empower users to automate tasks and extend Audacity's functionality through scripting.
- **PortAudio**: Provides cross-platform audio hardware interaction.
  - PortAudio ensures consistent audio input/output operations across different operating systems in Audacity.
- **wxWidgets**: Cross-platform GUI toolkit for Audacity's interface.
  - wxWidgets facilitates the creation of a unified GUI for Audacity, compatible across various platforms.
- **Other supporting libraries**: Additional functionalities like audio compression support.
  - Supporting libraries enhance Audacity's capabilities with features such as audio compression support.

- **Plug-in Effects and Plugin Converters**: Extend Audacity's features via external plugins.
    - Plug-in Effects and Plugin Converters expand Audacity's functionality with additional audio processing tools and format conversion options.

Threads and Buffers in Playback and Recording

Diagram:

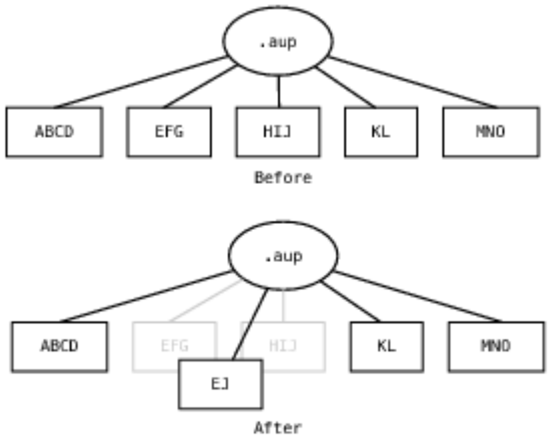


Topics covered:

- **pThread in PortAudio**: Handles audio device interaction in PortAudio.
    - One audio thread is started by PortAudio code and interacts directly with the audio device.
- **PortAudio**: Provides cross-platform audio functionality for Audacity.
    - PortAudio gives Audacity the ability to play and record audio in a cross-platform way.
- **wxThread in AudioIO**: Manages audio data processing in Audacity's AudioIO class.
    - A second thread is started by code in Audacity's class AudioIO.
- **wxWidgets**: Cross-platform GUI toolkit used in Audacity.
    - This main GUI thread is created within wxWidgets rather than by our own code.
- **GUI Thread**: Handles GUI updates and user interactions in Audacity.
    - The update happens in the main GUI thread and is due to a periodic timer that ticks twenty times a second.
- **Timer Ticks**: Periodic events used to trigger GUI updates in Audacity.
    - This timer's tick causes *TrackPanel::OnTimer* to be called, and if updates to the GUI are found to be needed, they are applied.

BlockFiles

Diagram:



(Before deletion, .aup file and BlockFiles hold the sequence ABCDEFGHIJKLMNO. After deletion of FGHI, two BlockFiles are merged.)

Topics Covered:

- **.aup file**: Serves as a master file coordinating various BlockFiles.
  - .aup files act as organizers for Audacity projects, coordinating the management of audio data through the use of BlockFiles.
- **BlockFiles**: Divides audio files into manageable chunks to optimize editing and playback.
  - BlockFiles are smaller, manageable chunks of audio data optimized for efficient editing and playback.

4.Architecture Patterns

The book mentions two architecture patterns directly (“notes”), namely the layers pattern and pipes and filters pattern. However, before delving into those, it introduces another important pattern known as the Model-View-Controller (MVC). This pattern will be discussed first before moving on to the layers and pipes and filters patterns (and others).

Model-View-Controller

In Audacity, the **Model-View-Controller (MVC)** architectural pattern plays a crucial role in organizing the application's structure and functionality. This pattern divides the application into three interconnected components: the Model, the View, and the Controller, each with distinct responsibilities and interactions.

The **Model** represents the underlying data and business logic of the application, such as audio waveforms, tracks, effects, and project settings. It encapsulates operations related to audio processing, file manipulation, undo/redo functionality, and project management. For example, BlockFiles store audio data, while project settings are stored in memory. The Model notifies the View of changes, ensuring that the user interface reflects the current state of the application.

The **View** encompasses all user interface elements responsible for presenting information to the user and capturing user input. This includes graphical components like windows, dialogs, buttons, and sliders. Views, such as the TrackPanel, display audio waveforms, track information, and project settings. They are updated in response to changes in the Model, ensuring that the user interface accurately reflects the state of the underlying audio data and application settings.

The **Controller** acts as an intermediary between the user interface (View) and the underlying data (Model). It handles user input events, interprets user actions, and invokes appropriate operations on the Model to effect changes. For example, when a user interacts with GUI elements like buttons or sliders, the Controller processes these events and triggers corresponding actions in the Model. Controllers also facilitate communication between different parts of the application, ensuring proper coordination and synchronization of activities.

Overall, the MVC pattern in Audacity promotes loose coupling between the Model, View, and Controller, allowing each component to evolve independently without affecting the others. Views observe changes in the Model and update their presentation accordingly, while Controllers handle user input events and invoke operations on the Model. This separation of concerns promotes modularity, maintainability, and flexibility in the development and evolution of Audacity as an audio editing software.

Layers

Another pattern observed in the notes is the **layered architecture pattern**. In a layered architecture, the system is structured into distinct layers, each with its own specific responsibilities and interactions with adjacent layers. This pattern promotes modularity, scalability, and maintainability by separating concerns and establishing clear boundaries between different functionalities of the system.

Audacity effectively employs a layered architecture to organize its software components. The application is divided into multiple horizontal layers, each dedicated to handling specific aspects or functionalities such as audio processing, user interaction, and data management. The texts describe various layers, including the GUI layer (wxWidgets), audio processing layer (PortAudio), and data management layer (BlockFiles). Each layer encapsulates related functionality and communicates with neighboring layers through well-defined interfaces, which enhances modularity and ease of maintenance.

Digging deeper, we can identify several key layers within Audacity's architecture:

- **Presentation Layer**: Also known as the user interface layer, this layer is responsible for presenting information to users and capturing user input. It comprises graphical components like windows, dialogs, buttons, sliders, and menus that enable users to interact with Audacity's audio editing features.
- **Application Layer**: This layer houses the core logic of the Audacity software, orchestrating user actions, invoking appropriate operations on data, and managing

the overall application flow. It handles tasks related to audio processing, file manipulation, undo/redo operations, and project management.

- **Domain Layer**: Representing the underlying domain model of Audacity, this layer defines data structures, business rules, and behavior associated with audio data manipulation. It encapsulates concepts such as audio waveforms, tracks, effects, project settings, and other domain entities.
- **Persistence Layer**: Responsible for storing and retrieving application data from permanent storage, such as files or databases, this layer manages tasks related to reading and writing audio files, project settings, preferences, and other persistent data.
- **Infrastructure Layer**: Providing support services and utilities used by higher-level layers, the infrastructure layer includes components for logging, error handling, configuration management, and other cross-cutting concerns necessary for the application's proper functioning.

The layered architecture of Audacity offers several advantages:

- **Modularity**: Each layer encapsulates a specific set of functionalities, simplifying understanding, maintenance, and extension of individual components without affecting other parts of the system.
- **Scalability**: The layered structure enables Audacity to scale effectively by adding new features, modifying existing functionality, or integrating with external systems without disrupting the overall system architecture.
- **Flexibility**: Clear separation of concerns allows Audacity to adapt to changing requirements, technologies, and environments more easily, as developers can replace or upgrade individual layers without impacting the rest of the system.
- **Testability**: The modular nature of the architecture facilitates comprehensive testing of individual layers, ensuring overall quality and reliability of the Audacity software.

In summary, the layered architecture of Audacity enhances maintainability, extensibility, and reliability, making it a robust and flexible platform for audio editing and processing.

Pipes and filters

The "**pipes and filters**" pattern is a fundamental architectural pattern present in Audacity's development, especially in the implementation of the scripting plugin.

In the context of this pattern, data flows through a series of components, or filters, connected by pipes, where each component performs a specific operation on the data. This pattern is analogous to a data processing pipeline, where data is passed through a sequence of processing stages.

In Audacity, the scripting plugin serves as a conduit for the flow of textual commands and responses between the scripting environment and the Audacity application. This plugin acts as the "**pipe**" through which data (commands and responses) flows between the scripting environment and Audacity, similar to how data flows through pipes in the traditional pipes and filters pattern.

The scripting environment sends textual commands to Audacity through the plugin, which are then processed by Audacity according to its internal logic. Similarly, Audacity sends textual responses back to the scripting environment through the plugin, completing the feedback loop.

By leveraging the pipes and filters pattern, Audacity's architecture achieves a modular and flexible design, where components can be added, removed, or modified independently, allowing for easy extensibility and customization. This pattern also promotes separation of concerns, as each **filter** (or component) is responsible for a specific operation, making the overall system easier to understand, maintain, and evolve.

Overall, the presence of the pipes and filters pattern in Audacity's development underscores its adherence to proven architectural principles and design patterns, contributing to the robustness and effectiveness of the software.

Repository

The Repository Pattern is indeed evident in Audacity's architecture, particularly in the implementation of the BlockFiles component.

In the context of the Repository Pattern, the BlockFiles component serves as a structured mechanism for managing audio data storage and retrieval. Here's how it embodies the key characteristics of the repository pattern:

1. **Structured Storage**: BlockFiles provide a structured way to store audio data blocks. Instead of storing the entire audio recording in a single file, Audacity divides it into smaller blocks stored in separate files. This structured approach allows for more efficient manipulation and retrieval of audio data.
2. **Centralized Access**: BlockFiles act as a centralized access point for managing audio data. They provide methods and operations for storing, retrieving, and manipulating audio data blocks. This centralized access ensures consistency and uniformity in how audio data is managed throughout the application.
3. **Abstraction of Data Access**: BlockFiles abstract away the details of data storage and retrieval from the rest of the application. They encapsulate the logic for interacting with audio data blocks, shielding the higher-level components of Audacity from the complexities of file management and disk I/O operations.
4. **Persistence**: BlockFiles facilitate the persistence of audio recordings by storing them in a durable format on disk. They ensure that audio data remains available even after the application is closed or restarted, providing a reliable storage solution for audio projects.
5. **Efficient Retrieval**: By organizing audio data into smaller blocks, BlockFiles enable efficient retrieval of specific segments of audio recordings. This allows Audacity to access and manipulate audio data quickly and effectively, enhancing the overall performance of the application.

Overall, the use of the Repository Pattern in the BlockFiles component enhances the modularity, flexibility, and maintainability of Audacity's architecture. It provides a robust and structured approach to managing audio data storage and retrieval, contributing to the overall effectiveness and reliability of the software.

Maybe other patterns:

In Audacity, several design patterns are employed to enhance the organization, flexibility, and efficiency of the software:

Observer Pattern:

Audacity leverages the observer pattern to ensure that changes in audio data or user interactions are propagated to various components of the system. For instance, when there's a modification in audio waveform data, this triggers updates in the TrackPanel for visualization purposes, as well as in audio effects for real-time processing and audio file management components for saving changes.

Facade Pattern:

The ShuttleGui class acts as a facade in Audacity, offering a simplified interface for interacting with the underlying wxWidgets GUI library. By encapsulating the complexities involved in GUI creation and manipulation, ShuttleGui promotes code reuse, readability, and maintainability, making it easier for developers to work with GUI elements.

Batch Processing:

While not explicitly mentioned, Audacity supports batch processing, representing a form of the batch processing architectural pattern. This feature allows users to apply a sequence of audio processing operations to multiple audio files simultaneously, automating repetitive tasks and enhancing productivity. By enabling users to process audio files in batches, Audacity streamlines workflows and improves efficiency in audio editing tasks.

5.Quality Attributes

Let's delve into the important quality attributes of the Audacity system based on the provided book:

- **Portability**: Audacity's architecture relies heavily on wxWidgets, a cross-platform GUI library, and PortAudio, a cross-platform audio I/O library. This design choice enables Audacity to be portable across different operating systems, such as Windows, macOS, and Linux. Portability is crucial for ensuring that users can run Audacity on their preferred platform without encountering compatibility issues.
- **Flexibility**: The use of modular architecture and third-party libraries allows Audacity to be flexible in terms of feature development and experimentation. By leveraging modular components and libraries like wxWidgets, Audacity can easily incorporate new features and adapt to changing requirements without requiring significant rewrites or redesigns.
- **Maintainability**: Audacity's architecture emphasizes the importance of clean code and modular design. For example, the introduction of ShuttleGui and other abstractions helps improve code readability and maintainability by reducing redundancy and complexity. This focus on maintainability ensures that the Audacity codebase remains comprehensible and manageable, even as it continues to evolve.
- **Performance**: While Audacity prioritizes portability and flexibility, it also recognizes the importance of performance, especially in audio processing tasks. Efforts such as on-demand loading and background rendering aim to improve performance by

optimizing resource utilization and reducing latency. Additionally, considerations for real-time effects and playback highlight the system's focus on delivering responsive and efficient audio processing capabilities.

- **Usability**: The Audacity user interface is designed to be intuitive and user-friendly, catering to both novice and advanced users. Features like ShuttleGui and batch chains simplify complex tasks and streamline the user experience. However, there are challenges, such as managing large numbers of BlockFiles, that can affect usability, highlighting the need for ongoing refinement and optimization.
- **Reliability**: Audacity prioritizes reliability in audio recording, playback, and editing tasks. The use of robust third-party libraries like PortAudio ensures stable audio I/O operations across different platforms. However, challenges related to BlockFiles management and scripting plugin security underscore the importance of continuously monitoring and addressing potential reliability issues.
- **Extensibility**: The support for scripting languages and plugin development reflects Audacity's commitment to extensibility. By providing APIs and extension points, Audacity empowers users and developers to customize and extend the functionality of the software according to their specific needs. This extensibility contributes to Audacity's longevity and relevance in the audio editing domain.
- **Legality**: Legal concerns with the licensing of other projects has limited what the developers know would result in a better product. Some options had to be made!
- **Security**: Cutting network capability gets rid of many security concerns. Avoiding TCP/IP was a way to improve security.

Overall, Audacity's architecture demonstrates a balance between various quality attributes, prioritizing portability, flexibility, maintainability, performance, security, legality, usability, reliability, and extensibility to create a powerful and accessible audio editing platform.

6.Conclusion

To sum up, the architecture of Audacity perfectly captures the flexibility and robustness of open-source software. Flexibility, usability, and maintainability have always been at the forefront of Audacity's architecture, despite obstacles like resource and licensing limitations. By carefully implementing architectural patterns such as Model-View-Controller, layered design and many more, Audacity manages to strike a careful balance between practical utility and structural integrity. Furthermore, its emphasis on high-quality features like portability, flexibility, and dependability highlights how committed it is to providing a powerful and intuitive audio editing experience. The architecture of Audacity is still a monument to the enduring value of careful design and the cooperative spirit of the open-source community, even as it advances and innovates.