Concrete Architecture of KODI

November 19, 2023

Group 23: Cache Me Some Waves
Trevor Withers - 20107953
Eli James - 20177630
Erin Atacan - 20232290
Chengqi Li - 20143719
Joanna Bian - 20149892
Zhihao Nie - 20243603

Table of Contents

1. Abstract

For our assignment, our team will conduct an in-depth analysis of the concrete architecture of KODI, an open-source media player and entertainment system. Using various methods including reflection analysis, we aim to validate the accuracy of the conceptual architecture outlined in our first report, identifying necessary adjustments and refining them. A large part of our study is dedicated to examining a specifically chosen subsystem of KODI (**Data Layer**), observing its components through a detailed reflexion analysis. To assist in the understanding of the technical vocabulary, we have included a Data Dictionary and Naming Conventions section of the study.

We have also included a 'Lessons Learned' segment where we discuss the decision-making process and methodologies employed during our architectural analysis and report writing. This part will aim to shed light on the practical aspects of architectural analysis in software. The concluding section of the report will cumulate our findings, and offer a summary of our discoveries of KODI's concrete architecture. For ease of navigation, links to all these sections along with our cited sources, are organized in the Table of contents. This report aims to present a thorough architectural analysis of KODI but also endeavors to contribute to the broader understanding of software architecture in open-source systems.

2. Introduction and Overview

The aim of this analysis is to develop a more comprehensive understanding of the concrete architecture of Kodi, an open-source media player software that allows you to play and view most videos, music, podcasts, and other digital media files from local and network storage media and the Internet.

A conceptual architecture for Kodi was proposed in our previous analysis A1: Conceptual Architecture, and we have concluded that the architecture style of Kodi is layered. It is thought that the architecture of Kodi consists of three major layers, where the subsystem resides within: The Business Layer, the Presentation Layer, and the Data Layer. Along with the Client Layer, which is the system where the user runs the Kodi player.

This report delved into the source code of Kodi player and documentation, which are used for an understanding of the concrete architecture. The concrete architecture of Kodi is built through derivations on the Kodi source code. The process is achieved using the *Understand* application by SciTools. Further details on the derivation process can be found in the next section.

We revisited the conceptual architecture from A1, by mapping its components with our derivation we have updated the components and dependencies that are missing or inaccurate from the conceptual architecture.

In this report, we will investigate the high-level architecture of Kodi and take a deeper look at the data layer, with a focus on highlighting the differences between its concrete architecture derived from the analysis and the conceptual architecture. The data layer consists of three major parts: Sources, Views and Metadata. Besides comparing against conceptual architecture, The parts will also be examined for their functionalities.

At the end, this report will analyze the concurrency of the concrete architecture, as Kodi player is designed to be a multi-threaded program.. We will provide several use cases of Kodi, with corresponding sequence diagrams.

3. Derivation Process

This analysis derives the concrete architecture of Kodi mainly from examining The official Kodi documentation and source code. The derivation process relies on the *Understand* application, which helps with deriving the architecture and producing a dependency graph through the analysis on source code.

The concrete architecture of Kodi is derived based on the dependency graph built with *Understand*. To build the dependency graph, subfolders and individual source files are examined for comprehending their functionalities and interactions. The files are mapped to corresponding subsystems based on our understanding. The unidentified files are left ungrouped, and a group of files acting as utility functions/componly referenced files are grouped as Utils folder, acting as a 5th subsystem.

4. Conceptual Architecture

4.1 Conceptual Architecture

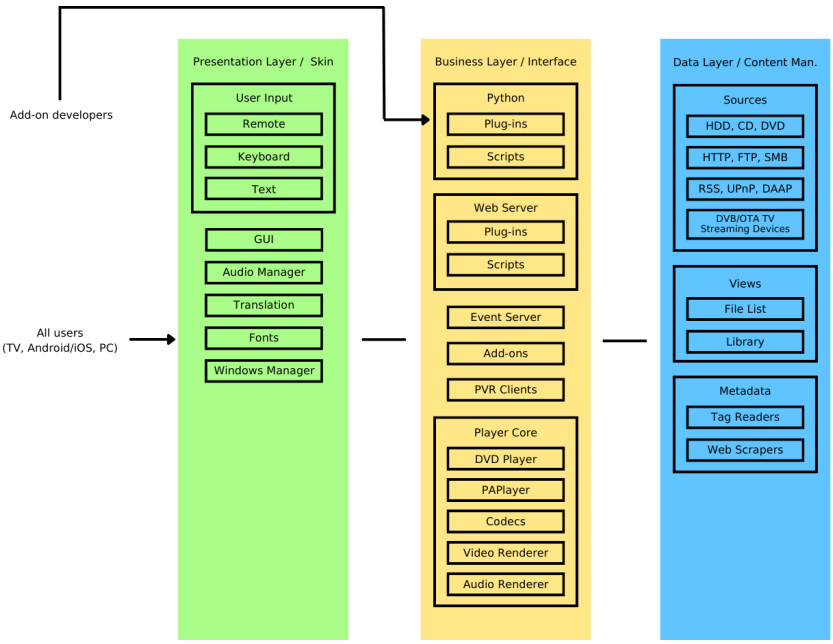

Fig.1

Our original analysis of Kodi led to a proposed conceptual architecture like the one shown above. Simplified to the highest categories, here is what we thought the conceptual architecture was [3]:

4.2 Conceptual architecture, simplified

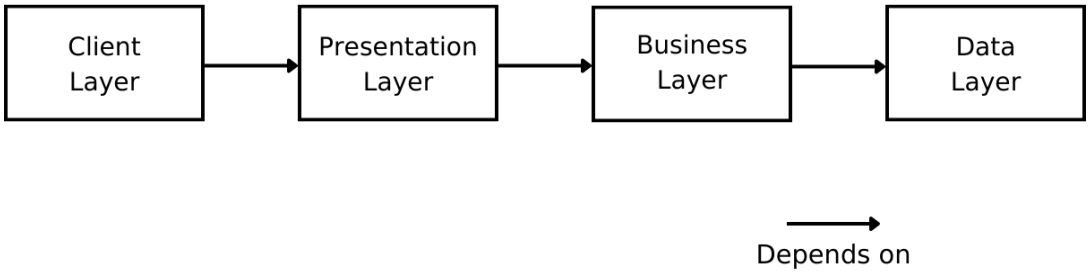

Fig.2

We created this architecture with elements of layered architecture styles in mind, wherein adjacent layers communicate with at most 2 other layers. This offered benefits in understanding the abstracted, high level function of the app, but, as we will see, is fairly disjoint from the concrete architecture. Lastly, it's worth noting that before the group even began constructing the concrete architecture, we acknowledged that real world layered architectures often have connections between layers that are not adjacent, to suit specific performance or functionality needs, which, as we discovered later, is the reality of the concrete architecture we constructed.

4.3 Understand Dependency Diagram

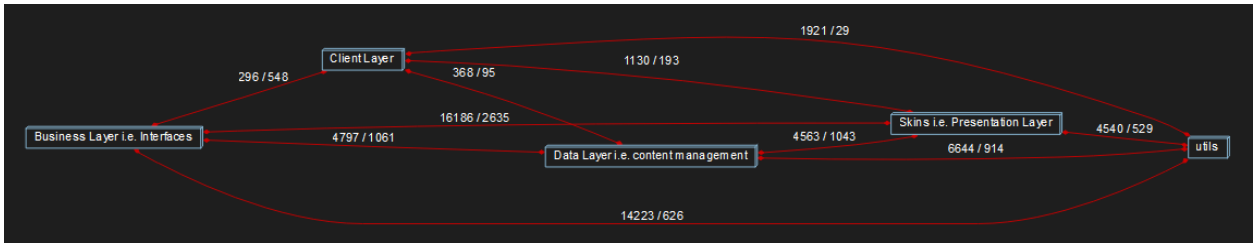

Fig.3

4.4 Concrete Architecture

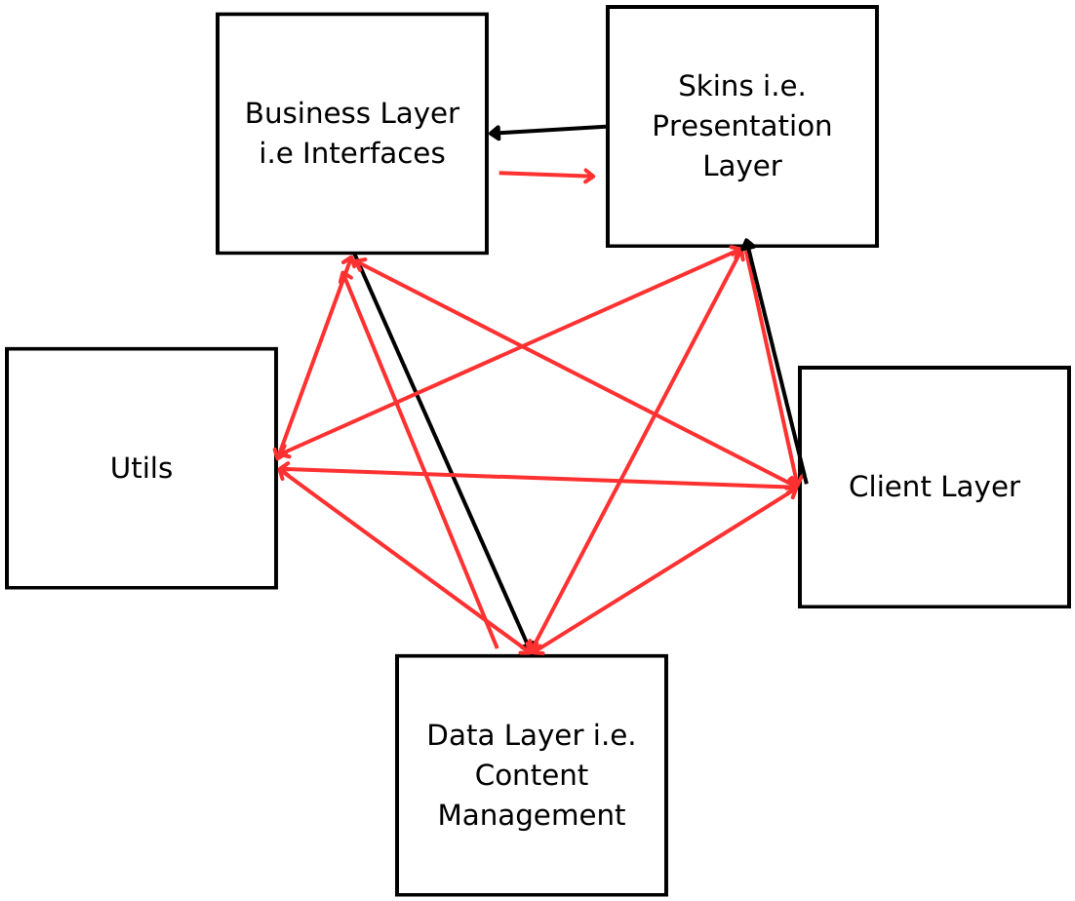

Fig.4

Fig.5

Shown above is the concrete architecture of Kodi. It is composed of 5 subsystems, 4 of which were used in the conceptual diagram. We discovered emergent dependencies using Understand, as well as a 5th subsystem, which we called Utils.

Client Layer

The client layer relates to the platform upon which the user is accessing Kodi from.

Skins/Presentation Layer

The skins/presentation layer is responsible for user input as well as displaying information for the user, such as font, language packs and managing views.

Business/Interfaces Layer

The business/interfaces layer contains the python module, web server module and player core module, which handle server access, libraries/add-ons, scripts for connections to steaming servers, and reading/playing of video and audio. [1]

Data Layer

The data layer contains the files for multimedia functions, specifically for sourcing and viewing multimedia elements, as well as storing, creating and processing metadata about the multimedia [2].

5. Reflexion Analysis

The concrete architecture shown earlier demonstrates numerous differences compared to the conceptual architecture. In this section, we will discuss the 5th unexpected subsystem, as well as the divergences between the conceptual/concrete architecture.

New Subsystem: Utils

When constructing our concrete architecture using Understand, we discovered a directory of commonly used scripts upon which every other subsystem depended, so we decided to describe a 5th, unexpected subsystem called utils that acts as a repository for commonly used functions and processes throughout the other 4 subsystems.

Divergences

There were a number of divergences between the conceptual and concrete architectures, which we will try to rationalize in this section. Firstly, all of the “1 way” connections between layers became bidirectional, turning the sequential layer model with data flowing from one layer to the next into a network of interconnected subsystems, wherein each layer has a bidirectional connection with each other layer, notably including the added Utils layer. The reader can examine this change in figure 4.4, and see each black (1 way) connection turn into a red (bidirectional connection). Secondly, before a rationale for each divergence is given, we will explain the suspected reason for all 4 subsystems connecting to Utils being simply that the scripts contained in that directory are commonly used functions used throughout the entire project by Kodi’s open source community.

Skins/Presentation → Client

The presentation and client layers have a bidirectional connection. The client layer contains the platform specific code files in a repository which process the display of content on different devices. This platform repository has a connection to the windows manager and GUI elements folders within the presentation layer, which are both bidirectional connections. The previously assumed one way connection is sensical as the platform code depended on GUI support from the presentation layer, but we now see how various windowing functions and windows managers could depend on the platform code, explaining the transition from one way to 2 way connection between the two.

Business/Interfaces → Skins/Presentation

Another emergent difference in dependencies is the reliance upon the interface layer by the interfaces layer. In the concrete architecture, this is explained as the webserver, interfaces and addons folders within the interfaces layer are depended upon by the user input directory within the presentation layer.

Data/Content Management → Business/Interfaces

In the conceptual architecture, the interfaces layer depends on the content management layer, but in the concrete architecture, this dependency goes both ways. The web server and event server directories within the interfaces layer exhibits a bidirectional dependency on the metadata available within the content management layer, as well as the views directory within the content management layer. This is likely due to metadata being used to select the correct function in the event server.

Data/Content Management ←→ Client

These layers are not connected in the conceptual diagram, but have a bidirectional connection in the concrete architecture. The platform specific code in the client layer depends on and is depended upon by the metadata available in the data layer. This could potentially be a redundancy: platform specific metadata could very well be stored specifically in the client layer, but instead Kodi contains it all in the content management layer, thus connecting these layers in both directions in the real world code.

Data/Content Management ←→ Skins/Presentation

These layers are also only connected in the concrete architecture, and bidirectionally at that. This is due to the 2-way dependency between the sources directory and windows manager directory of the data and skins layers respectively. Furthermore, the sources directory of the skins layer is bidirectionally dependent on the GUI elements of the skins layer. Lastly, the metadata and views directories of the data layer are bidirectionally connected to the GUI elements, windows manager and user input directories of the skins layer. These connections seem to favor the data layer side, that is the data layer depends on the skins layer more in terms of number of dependencies by almost a factor of 2 times, yet the two way dependence is still notable.

Business/Interfaces ←→ Client

Lastly, every repository within the interfaces layer has a two-way dependency on the platform specific code in the client layer. This is sensical as the addons, player core, web server and interfaces directories would all have platform specific functionality, deployment and appearance thus needing a connection with the client layer's platform repository.

6. Subsystem Analysis (Business Layer i.e. Interface)

We will describe, analyze and compare the Business Layer subsystem. This subsystem handles the interface between the user's requests to the system and the system itself.

6.1. Conceptual View

From our original conceptual view of Kodi, the Business Layer view was as follows:

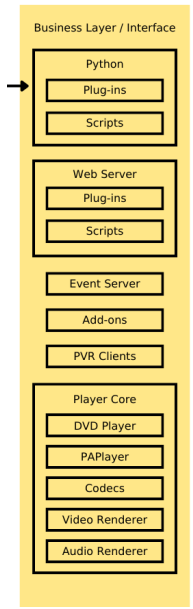

Fig.6

6.2 Concrete View

Using Understand, we generated a diagram with all dependencies between subsystems. The diagram below has a much more concise display of the concrete view. We have added the unexpected dependencies in red.

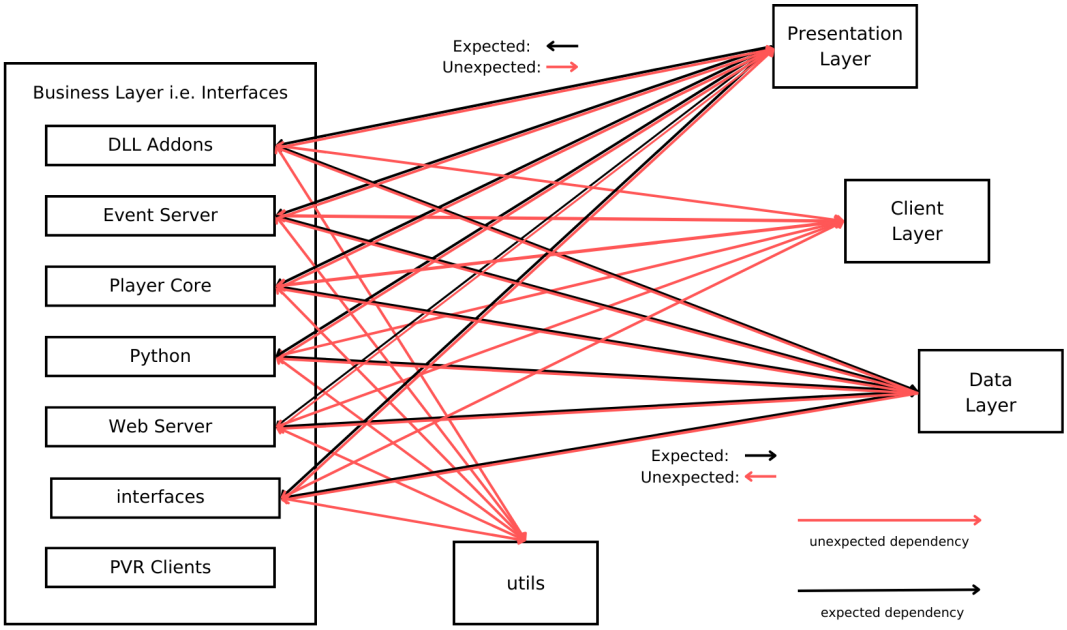

Fig.7

6.3 Reflexion Analysis

From analyzing our source code, we found that the original proposed unilateral architecture for the Business layer lacks several dependencies, including a new subsystem: utils. We found 4 main divergences from our conceptual architecture: Client Layer → Business Layer, Data Layer → Business Layer, Business Layer → Skins Layer and Business Layer → Utils.

Client Layer → Business Layer

Using Understand, we found that the Client Layer has around 400 dependencies on the Business Layer. Specifically on the DLL Addons, Event Server, Player Core, Interfaces and Web Server.

Data Layer → Business Layer

The Data Layer gained several hundred dependencies on the Business Layer. Specifically, towards the Event Server, the Player Core, DLL Addons, Interfaces and Web Server.

Business Layer → Skins Layer

The Business Layer gains several thousands of dependencies on the Presentation Layer. DLL Addons depend on dialogs within the GUI Elements subsystem and Windows Manager. Also, User Input was also depended on.

Business Layer → Utils

The final major divergence in our conceptual architecture was the addition of the utils subsystem. With dependencies with every other sub system, utils provides the entire Kodi software with repeatedly used files. This is important since so many layers need access to similar content at the same time.

7. USE CASES & DIAGRAMS

1) User watch and record live TV shows

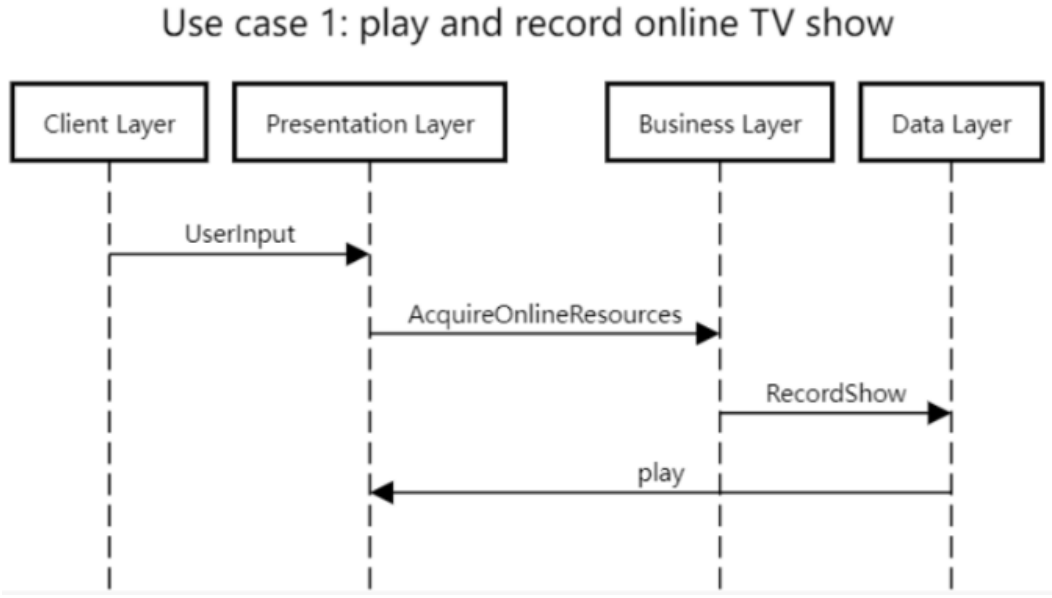

Fig.8

2) play music from CD:

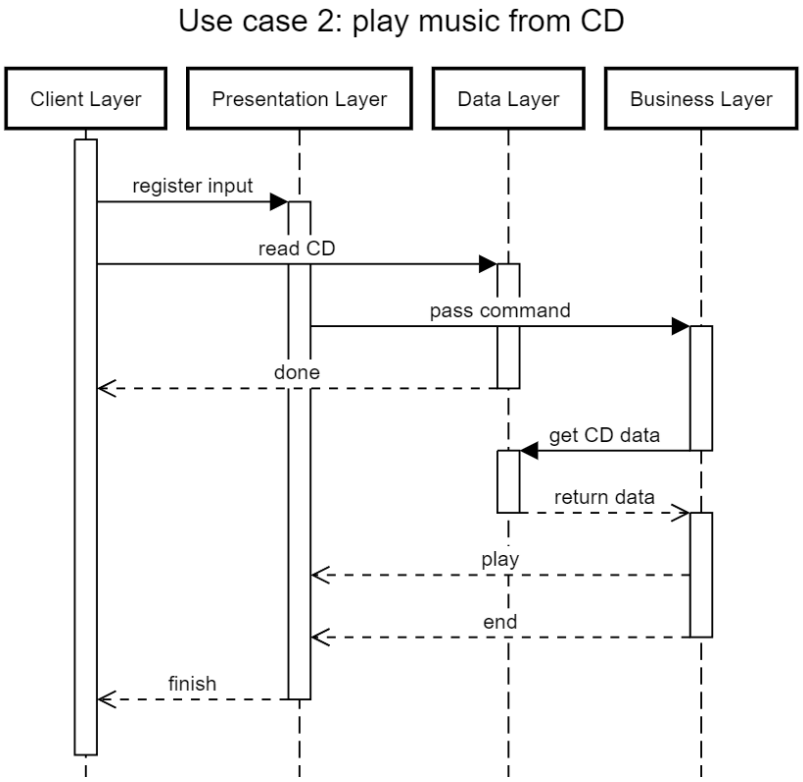

Fig.9

8. DATA DICTIONARY

Subsystem: a component of a larger system
Architecture: the overall design of the system
Multi-threaded: Ability for a program or system to enable more than one user at a time without requiring multiple copies of the program
Source code: group of instructions a programmer writes using computer programming languages
Conceptual Architecture: the architecture designed during conceptual phase, could be much different in development
Concrete Architecture: the architecture built in development
Layer: components of the architecture that are related or similar are placed in the same layer
Concurrency: the execution of a set of multiple instruction sequences at the same time
Metadata: the data providing information about other data
Dynamic linking library (DLL): a module that contains functions and data that can be used by another module
Graphic user interface (GUI): digital interface in which a user interacts with graphical components such as icons or buttons

9. LESSONS LEARNED

When developing a system, taking it from its conceptual architecture to its concrete one, we find a few neat emergent properties. Firstly, whole new subsystems can be exhibited in the real world project which were not incorporated in the conceptual architecture of a project at inception. Secondly, the interconnectedness of subsystems with a project's concrete architecture exhibit more dependencies than foreseen in the conceptual counterparts. Next, we see that a one way dependency can become a two way dependency between subsystems when taking an architecture from conceptual to concrete. Lastly, we got a glimpse at the difficulty of "casting" a real codebase's architecture onto a conceptual architecture's subsystems, showing how the code practices of a large group of contributors can muddy the waters as a project comes to fruition.

10. CONCLUSION

To conclude, through utilizing Scitools Understand on source code and analysis on the official architecture documentation of the Kodi player, as a team, we delved into these subsystems, examining their functions and interrelations. Furthermore, our report acknowledges that the initial conceptual architecture was not flawless; we have identified six unexpected dependencies and one new subsystem: utils. Additionally, a reflection analysis was performed. To deepen our understanding of the Kodi player architecture, we incorporated two use cases, illustrating subsystem interactions in playing TV shows and CDs. Overall, our collaborative effort in breaking down and analyzing the concrete architecture of Kodi proved successful.

11. REFERENCES

Architecture. Architecture - Official Kodi Wiki. (2023, July 10). https://kodi.wiki/view/Architecture [1]

Dreef, K., van der Reek, M., Schaper, K., & Steinfort, M. (2015, April 23). *Architecting software to keep the lazy ones on the couch*. Kodi. https://delftswa.github.io/chapters/kodi/ [2]

Kodi (software). Wikipedia. https://en.wikipedia.org/wiki/Kodi_(software)#Architecture [3]