Kodi Software System for Entertainment

Dylan Walker
Schuyler Good
Tim Mah
Qays Ebrahim
Akshay Desale
Lucas Gordon

Table of Contents

Abstract

The following report analyzes the Kodi media player. Kodi is an open-source media playing software. The report analyzes Kodi's architectural design, system evolution, and developer responsibilities.

Kodi was originally known as Xbox Media Player until later evolutions which now make it a powerful software that is supported across multiple operating systems, platforms, and architectures. Kodi is commonly used in home theatres for streaming and playing media files.

Kodi's architecture was designed using a layered, client-server, and repository style. Some of the layers that the software includes are the data layer, business layer, presentation layer, and a client layer. Kodi's control and data flow are connected with an event driven control flow and a data flow. This ensures efficient organization and accessibility of media content. The use of third party addons require the use of client server style architecture in Kodi, where the Addons interface with the web to fetch content through 3rd part API's delivering content over the web to the user. Kodi's architecture also relies on a central database which provides relevant information and content to the various components.

The division of developer responsibilities are modular development, platform-specific support, developer relations, API development, and quality assurance. These developer responsibilities ensure Kodi's software quality and performance.

In this report we explore two common and essential use cases in Kodi. the first being playing media from local sources, this would be the case for most users wanting to play their content. The second use case we examine is adding add-ons and importing content

Kodi's architecture offers excellent insights into the design's functionality and adaptability. There are many complex systems and data flows beneath its user interface.

Introduction & Overview

Kodi is a versatile and open-source media playing software, it is a complex software system that offers a rich and customizable media center experience to users. In this report, we will delve into the architecture, control and data flow, system evolution, concurrency, division of developer responsibilities, and two specific use cases that highlight the functionality and user interaction with Kodi to grasp a better understanding of how the software works.

Kodi originated as the Xbox Media Player (XBMP) in 2002, later rebranding as Kodi in 2014. With continuous development and the addition of features, Kodi has evolved into a powerful media center that supports multiple platforms and architectures. Kodi is commonly used in home theaters, where users would use a PC to stream or play their media files on their big screen.

After analyzing the architecture of Kodi, and researching the Kodi documents online, we have determined it is designed with a layered approach. The different layers we identified were the data layer, business layer, presentation layer, and client layer. These layers work together to comprise the skeleton of Kodis core functionality, and give users features such as access to their media content, the ability to add addons, and seamless media playback. In addition, Kodi has a central database that houses media libraries and metadata, which is drawn upon by the web servers, and media players. This interaction within the architecture is reflective of a repository architecture style as the various components are all accessing one central data block that manages all of the information. Kodi also functions with a client-server architecture, making requests from the client side to a web server in order to fetch other content. The web server component enables users to interact with their media from multiple devices and also allows the ability to stream content over the internet from additional addons.

The control and data flow within Kodi are intricately connected, the control flow is event-driven, where user interactions generate events that are processed by the core engine, creating actions across different components. The data flow involves media import, metadata scraping, database management, user interaction, media playback, and network access. The data flow ensures that the media content and metadata are efficiently organized and accessible for the user.

The division of developer responsibilities is crucial to Kodi's success. Developers are responsible for different aspects, including modular development, platform-specific support, developer relations, API development, quality assurance, and more. Later in the report we delve into further detail on how this impacts Kodi’s core functions, in any case, this division ensures that Kodi remains up to date, user-friendly, and adaptable.

Two use cases provide practical insights into how users interact with Kodi. The first use case explores the process of adding addons to enhance the user's media experience. The second use case demonstrates how users can import and play media from local sources, highlighting the seamless data flow and media playback functionality of Kodi. Each use case has a sequence diagram connected to it where you can see how each component interacts with each other.

Kodi is a powerful and effective open source media center software with a complex architecture, seamless control and data flow, and a wide range of developer responsibilities. Understanding its inner workings and user interactions provides valuable insights into its functionality and adaptability. By taking the time to analyze its software architecture, we have embarked on the journey of unraveling and understanding the complex systems and data flows that are hidden beneath the UI.

Architecture

Conceptual Architecture

The Kodi application was constructed using a layered architecture that is separated into a data layer, a business layer, a presentation layer and a client layer. The data layer accesses the file system to retrieve media from the hard drive or other external drives such as DVD, CD, etc. Additionally, it scrapes metadata from files or databases to provide key information about the media content it is showing.

The module layer includes several key components, including one for managing add-ons extending the user's capabilities within the app. It also features a webserver module that enables users to manage content remotely. The various player modules play a central role in managing, processing, controlling, and playing multimedia content.

The presentation layer focuses on the GUI of the application. This includes the visual appearance of the app, including windows, buttons, dialog boxes, and more. It also encompasses the Windows manager, which helps organize the layout and appearance of windows. The translation module facilitates the translation of the app into different languages. Meanwhile, the fonts module allows users to switch between different fonts for text within the application. Lastly, the Audio manager permits users to adjust the volume, configure audio devices, and change audio outputs.

The devices client layer is mainly responsible for interacting with the Kodi server, it includes any input devices or devices that Kodi is being run on. The client layer is the user-facing part of the architecture and it focuses on giving the user a polished and intuitive experience.

Kodi can also be modeled as a client-server application. It includes previously discussed client-facing aspects within the client layer and presentation layer of the application that connect to a web server which holds all of the user's movies, tv shows, photos and music. The server manages all media titles for easy access. When a user wants to access one of their videos, they make a request that sends a client to retrieve the content from the server, which ensures a responsive and organized user experience.

Additionally Kodi was also created using the repository model. Kodi incorporates a central database that stores all of the media content accessible to each user. Within the database various component managers access relevant pieces of media the user wishes to access and it is transferred to them over the server resulting in an easy and seamless transition.

System Breakdown

Kodi is a media player. It is compatible with most common media formats and will play users' files. It can be broken down in a few main components that interact to provide an enjoyable user experience.

The database component is the foundational component in this architecture. It stores all of the media in each user's library, providing easy access each time a user wants to view it. It allows users to quickly sort their files by criteria such as including title, genre, actor, director, release year, and more. The database also stores summaries and metadata about each of the media files that it houses. It uses a web scraper to obtain information from websites about the media files, and it also uses a tag reader to extract data straight from the files themselves.

A large portion of what makes Kodi unique is its ability to stream shows from third party sources. Kodi has permission to access these outlets through add-ons. This includes streaming services like Netflix, and TV channels such as PBS and many more. In order to use these add-ons Kodi has scripts to retrieve them from their sources and send the relevant information to the corresponding player so whichever video you choose can be played.

The web server component plays a pivotal role in allowing users to manage their content through their browser or from other applications. It also enables multiple users on a local network to access each other's content making it easy for people to use multiple devices to access content simultaneously. The server acts as the connection between the data layer and the rest of the application. It draws content from add-ons and the database and feeds it to the media players or the GUI depending on the user's intended action.

To allow for playback of content, Kodi has a series of data players that can parse the media and convert it to a viewable format to output to the user. This includes a video player, an audio player, a music player and a photo renderer. These components work with the hardware to synchronize the content and create a smooth and polished product for users to enjoy. They interact with the database directly to render content, and also interact with the web server to render add-on content. The media is outputted to the GUI to be displayed in an enjoyable and immersive way.

System Evolution

Kodi started out as xbox media player (XBMP) in 2002 by 2 developers who merged their separate media player projects. The open source software allowed pictures, movies, and music to be played from an Xbox console using the harddrive, CD drive, or over the internet. By the end of the year, they merged with another media player, Yet Another Media Player (YAMP), to release XBMP 2.0. By December of 2002, the team released bug fixes, support for AC3 5.1 (lossy audio format), and a volume normalizer and amplifier feature. They also released playlist features Xbox game launcher dashboard, language support, and Windows media streaming.

By 2003, the group launched a beta version of the successor to XBMP, Xbox Media Center (XBMC). It was able to load and play most audio, video, and picture formats. They added performance features, such as multiple CPU core support and user customization features. In 2004, they released Version 1.0.0 with Python support for user scripts. They also released karaoke support and more file support. In 2006, XBMC added more media player support, which had features like audio crossfade.

In 2007, work started on porting XBMC to Linux. They replaced DirectX (Which was prominent in the Xbox version) with OpenGL and SDL on Unix operating systems. In 2010, they split the Xbox version off, and the main XBMC no longer supported Xbox. They subsequently moved their repository to Github in 2011. The name changed to Kodi in 2014.

Kodi's renders using a game loop rendering system, which was how the original Xbox rendered their games. This is very intensive, as it renders whenever it can, even if it doesn't have to change the screen. The developers are working to change it to an event-driven rendering system, where it only renders when necessary. Kodi supports all major architectures (x86, PowerPC, ARM).

Control and Data Flow

Kodi's architecture involves a complex flow of control and data among its various components, which work together seamlessly to provide users with a rich and highly customizable media center experience. Similarly, the event-driven architecture ensures a responsive and interactive user experience. It allows for the smooth navigation of menus, control of media playback, and customization of settings.

Control Flow:

The control flow within Kodi is primarily event-driven. When a user interacts with the graphical user interface (GUI) through a remote control, keyboard, or other input devices, events are generated. These events trigger specific actions within Kodi's core engine. For instance, when a user selects a media item in the GUI, an event is generated, and Kodi's core engine processes this event to initiate the playback of the selected media file. Similarly, when a user adjusts the volume, Kodi's control flow handles the volume control event.

The core engine acts as the central controller in Kodi's architecture. It manages all user interactions and orchestrates various components to respond to these events. The control flow can be summarized as follows:

1. **User Interaction:** User input is received through input devices, such as a remote control or keyboard.

2. **Event Generation:** User actions generate events. For example, clicking on a menu item or pressing a play button generates events.

3. **Event Processing:** Kodi's core engine processes these events, determining the appropriate actions to take.

4. **Component Interaction:** Depending on the event, the core engine may interact with various components, such as the GUI, player, add-ons, and the database.

5. **Feedback to User:** The GUI is updated to reflect the actions initiated in response to the user's input.

Data Flow:

Kodi's data flow is equally crucial to its functionality. It revolves around managing and organizing media content, along with metadata, within the system. Here is how data flows through Kodi's architecture:

1. **Media Import:** When you add media files to Kodi, it scans and imports them into the library. This process involves identifying media types, such as movies, TV shows, music, and videos.

2. **Metadata Scraping:** Kodi uses scrapers to fetch metadata for imported media. Scrapers are configured to extract information from online databases and websites, such as movie titles, actors, genres, and cover art. This metadata is then associated with the media files in the library.

3. **Database Management:** Kodi maintains a database to organize and store metadata. The database acts as a central repository for information about media content. It helps in efficient searching, sorting, and retrieval of media.

4. **User Interaction:** When a user interacts with the GUI, it queries the database to display information about the available media content. The data flow includes displaying lists of movies, TV show episodes, music albums, and more.

5. **Media Playback:** Data is passed to the media player component when a user initiates playback. The player handles the actual rendering of audio and video content.

6. **Streaming and Network Access:** Kodi can access media content from local and network sources, including shared network folders and internet streaming services. The data flow includes the retrieval and streaming of media content from these sources.

Kodi's data flow ensures that media content is organized, searchable, and accessible to users. It also facilitates the ability to maintain a personalized library of media content with rich metadata. Additionally, the flow of data allows users to seamlessly navigate and enjoy their media, whether it's locally stored or streamed from the internet.

In summary, Kodi's architecture relies on a well-defined control and data flow to provide a versatile and user-friendly media center experience. The control flow manages user interactions and responses, while the data flow ensures that media content and associated metadata are efficiently organized and accessible to users. This combination of control and data flow is at the core of Kodi's success as an open-source media player and entertainment hub.

Concurrency

A common practice in large projects, such as Kodi, to increase efficiency of all the moving parts involved is to design the architecture in a way that allows for concurrency. Concurrency is, in general, a collection of techniques and mechanisms that allow the program to execute multiple commands at the same time. Since Kodi best aligns with the architecture styles repository and client-server, there is certainly room to improve the program with the use of concurrency.

It is worth noting that the typical meaning of concurrency aligns best with the understanding of operating systems, and as a specific example, multithreading and applications that use that technique to streamline the code. While Kodi does indeed use concurrency, it is not in the form previously described as Kodi is more of a single user experience that waits for a response from the user. This means that Kodi does not really need to rely on making it faster and thusly does not take too much advantage of concurrency

Some examples of where Kodi does use concurrency is to optimize the user experience and make it as smooth as possible. Kodi is able to do this by utilizing the separation of the layers in the architecture to its advantage. For example, when downloading an add-on, the user is able to continue to browse and interact with the various parts of the system - even being able to view downloaded media at the same time.

Another way Kodi utilizes the concurrency technique is by taking advantage of the client-server aspect of itself. To make use of the client-server architecture, Kodi can send a request to the server while performing other actions, and will continue to be able to perform other actions.

Division of Developer Responsibilities

Taking a look at Kodi's development style, there are several implications related to the division of responsibilities among development staff and engineers. The responsibilities of developers can be divided into the following five categories, Modular Development, Cross-Platform Support, Developer Relations, API Development, and Quality Assurance. By exploring these five developer responsibilities we can grasp a better understanding of Kodi's architecture overall.

In terms of modular development, Kodi is organized into several modules including the user interface, add-ons, content management, and file sharing. All of these modules can be handled by different developers and engineers, which will enable them to focus on different aspects of the software. Some developers may be broken down for focus areas such as front-end, full-stack, back-end, data, etc. The advantages of using a modular development style is elevated levels of scalability and reusability, as well as being easier to debug problems in your code.

Another responsibility for the developers to uphold is maintaining cross-platform support, Kodi can run across multiple operating systems such as Windows OS, iOS, Android, and Linux. As such, it requires the developers to be tasked with different responsibilities related to a specified operating system. This ensures that Kodi runs smoothly across different operating systems and has a uniform experience for the user. This specialization ensures that platform-specific issues and features are handled by developers familiar with the respective platforms.

Some developers can also be tasked with maintaining Kodi. These developers will monitor bug tracking, user feedback, and feature requests. They will also prepare any documentation for APIs for those who wish to use Kodi in a technical regard. The specific interaction with end users will allow for faster bug fixes and ensures that the product is tailored to the customer's needs.

A crucial feature in Kodi's architecture is its API compatibility, ensuring that Kodi has the ability for third-party developers to create addons for their software is essential for the user experience. These developers have the responsibility of ensuring their add-ons work seamlessly with Kodi and offer value to the users.

Some developers are responsible for testing the software for quality assurance and performance issues. Responsibilities include writing test cases, performing tests, and documenting the results.

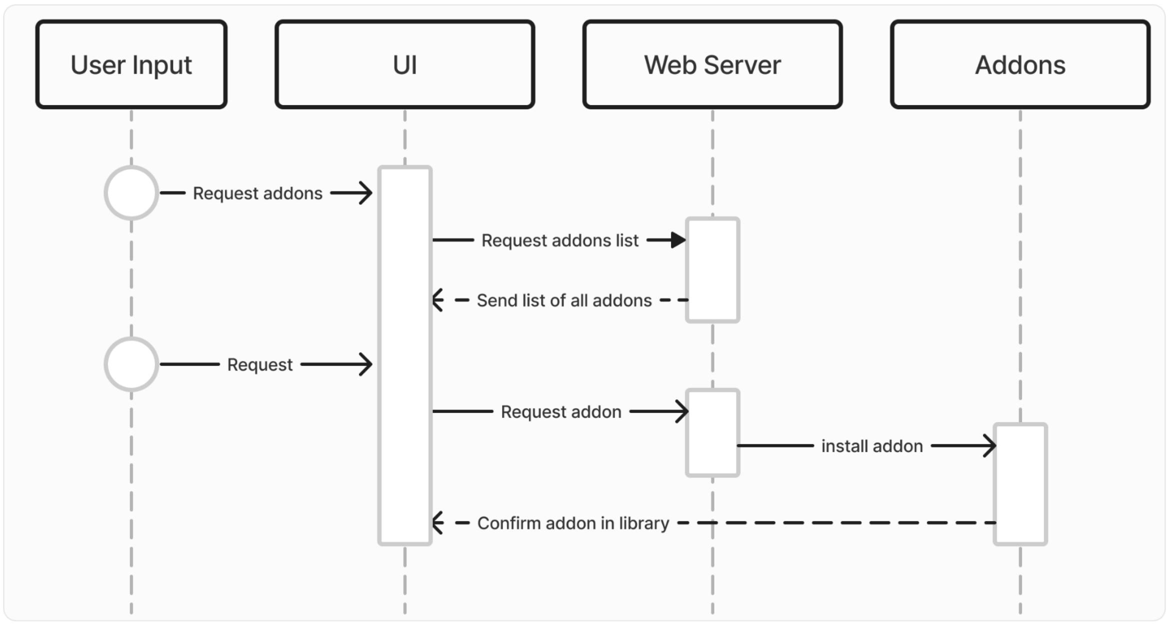
Relevant Use Cases

We will be exploring two applicable use cases, the first use case is the ability for the user to add additional addons. In Kodi, addons are extra ways for users to enjoy media over the web. For example a user could download a news addon to be able to stream news live from the internet. The other use case we are examining is importing and playing custom media where the user can add video, music, or any other files and play it or view it through Kodi.

Adding addons

In this use case, the user is attempting to add an addon from the list of addons displayed in the addon menu, and add it to their addons, such that they can use them to view content that isn't locally. First the user requests the list of all addons available in the User Interface Component. The request is then passed to the Web Server Component, where it fetches all the available addons that the user has the ability to add. The Web Server Component then returns the data (list of all addons) to the User Interface Component, where the user can see it. Then the user selects an addon from the list and the User Interface Component sends a request to add the specific addon to the Web Server Component. The Web Server Component indexes for that addon, then adds it to the Addons Component which keeps track of what addons the user has added. The last step in the process is to return to the UI that the addon has been downloaded, by showing it in the library (this step may also be more complicated, where a loader showing the download time would show up).

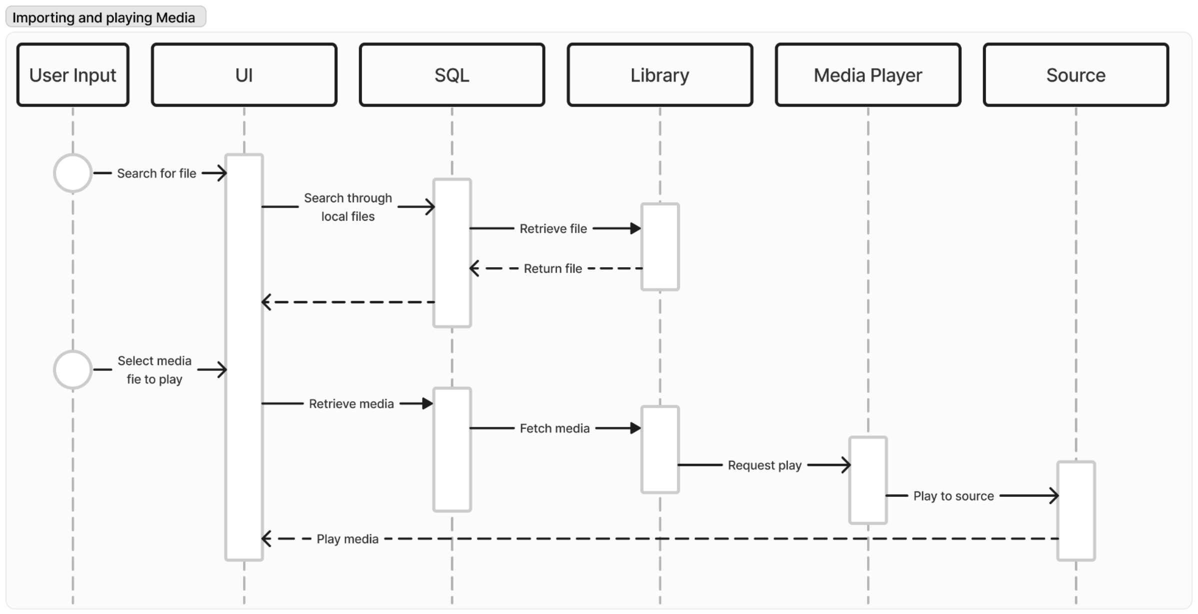
Request Addon



Importing and Playing Custom Media

In this use case we are importing and playing media from a local drive source. This process involves two big steps: Firstly, the user will Search for a file using the User Interface

Component, then it queries the SQL Component to search through the system's local files, and retrieve it from the Library Component. The Library Component then returns the file, back to the SQL Component which in turn returns it to the User Interface. Secondly, the next step is to send that file to be played, we do so by retrieving the media from the SQL Component and Library Component then requesting it to play in the Media Player Component. Then we send the file to be played in the Source Component. Then it Plays the media in the User Interface Component.



Lessons Learned

Undertaking the task of delving into the architecture of Kodi has provided an invaluable understanding of the inner workings of a major software project like Kodi. In terms of architecture, seeing a realized version of the layered, client-server, and repository architecture styles gives a deeper understanding of those styles in particular but also a better understanding of architecture styles in general. In particular, learning about how data passes through the system, both in how it is received and how it is used, was a useful experience for our future projects as data management is potentially a hard thing to keep track of in a project if handled incorrectly.

Glossary

ARM - A reduced instruction set architecture for efficient computer processors.
Architecture - Overarching design of the system being built. Usually follows a pre-established format.
Audio Amplifier - Amplifies low power signals
Audio Crossfade - Smooth transition between two audio clips
Business Layer - Architecture layer that includes the server, and media players, that bridges the gap between the user interface and the repository.
Client Layer - Architecture layer including user input and devices.
Client-Server - A style of architecture that utilizes a web based server that hosts information that the client(s) send to and from the server.
Concurrency - A collection of techniques and mechanisms that allow the program to execute multiple commands at the same time
Dashboard - User interface that displays constantly updating information
Data Layer - lowest level of the architecture that stores all of the data and information used by the application.
DirectX - Collection of programming interfaces for game or video development.
Event-Driven Rendering - Updates frame when needed, or certain regions of the frame
Game Loop Rendering - Rendering and redrawing frames as the computer commutes it.
Layered - Architecture style built on the idea of utilizing layers in the design.
Layers - Conceptual sections of the code base that interact mainly with other parts of the code in its own 'section'.
Linux - A popular open source operating system
Media - General term for data used
Media Player - Software that allows playback of video or audio files
OpenGL - Cross-platform and cross-language interface for 2D and 3D graphics
Presentation Layer - Architecture layer consisting of the UI and styling elements.
Repository - In relation to data, a repository is the place where it is stored. As an architecture style, it represents a style where the code is built on top of a repository of data which it directly references
SDL - Cross-platform abstraction layer for media hardware components
SQL - Structured Query Language is a domain-specific language used in programming and designed for managing data.
Scripts - Manipulate, customize, or add on to existing software
UI - User Interface, an area in a program where the user interacts with the program and provides user input and output

Unix - Operating system Linux and MacOS is based on
Volume Normalizer - Change the overall volume of the audio to a set average amount
Web Scraper - A software application that extracts data from websites.

Conclusions

In conclusion, the Kodi media player has a vastly unique and powerful overarching architecture that can be broken down into three main underlying architectures: a layered architecture, a client-server architecture, and a repository style architecture. The layers are further broken down into four major parts: data, modules, GUI, and client layers. The data layer retrieves and decodes media from internal or external connected devices. This serves as a main repository within the application which all of the other modules access data from, this is where we derived the repository style architecture subsystem from. The module layer includes several key modules, including those for managing add-ons extending the user's capabilities within the app. It also features a webserver module that enables users to manage content remotely, this is where the client-server architectural style is housed. It uses 3rd party addons to interface with web API's to deliver content from the web to the user. The various player modules play a central role in managing, processing, controlling, and playing multimedia content. The presentation layer focuses on the GUI of the application. This includes the visual appearance of the app, including windows, buttons, dialog boxes, and more. It also encompasses the Windows manager, which helps organize the layout and appearance of windows. The translation module facilitates the translation of the app into different languages. Meanwhile, the fonts module allows users to switch between different fonts for text within the application. Lastly, the Audio manager permits users to adjust the volume, configure audio devices, and change audio outputs. The devices client layer is mainly responsible for interacting with the Kodi server, it includes any input devices or devices that Kodi is being run on. The client layer is the user-facing part of the architecture and it focuses on giving the user a polished and intuitive experience.

Utilizing this architecture, the data is largely moved with event-based calls, and relies on the user to control what happens with the majority of data stored in the system. This data system is accomplished by having the data come from third party add-ons, which is sent directly to the storage device where it waits to be used.

The technique enabling the event-based system to be largely operated by the user is known as concurrency. Concurrency is a collection of techniques compiled with the intention of allowing code and the systems built on top of it to execute multiple commands at the same time. In Kodi, this is most visible when downloading and using addons as the data can be downloaded or used while other operations, such as navigating the system, can be done.

The success of Kodi can be largely attributed to the division in responsibility the developers at Kodi take on. At Kodi, there are unique and highly specialized teams dedicated to tasks such as maintenance, cross-platform support, and API support and generation.

References

Architecture. Architecture - Official Kodi Wiki. (n.d.). https://kodi.wiki/view/Architecture

Kidman, A. (2020, December 3). *History of Boxee: And Boxee was born, slowly*. Gizmodo Australia. http://www.gizmodo.com.au/2010/10/history-of-boxee-and-boxee-was-born-slowly/

Kodi Wiki. Official Kodi Wiki. (n.d.). https://kodi.wiki/view/Main_Page

Kodi. GitHub. (n.d.). https://github.com/xbmc/xbmc

Stevens, T. (2019, July 19). *XBMC arm port teased, will manage HD playback from pocket-sized beagleboard (video)*. Engadget. https://www.engadget.com/2009-11-02-xbmc-arm-port-teased-will-manage-hd-playback-from-pocket-sized.html

Timothy. (2010, May 28). XBMC Discontinues Xbox Support. http://hardware.slashdot.org/story/10/05/28/043243/XBMC-Discontinues-Xbox-Support?art_pos=1

Xbox Media Center, The new (ad)venture. Xbox Media Center. (n.d.). https://web.archive.org/web/20031002022404/http://www.xboxmediacenter.com/