CISC 322/326 Assignment 3: Report

Kodi: Enhancement Proposal

December 5, 2023

Team Torrent (Group 25)

Aselstyne, Alex (alex.aselstyne@queensu.ca)

Dinari, Daniel (20dd29@queensu.ca)

Nagel, Jake (20jn29@queensu.ca)

Peterson, Jack (21jrp10@queensu.ca)

Pleava, Ryan (20rcp5@queensu.ca)

Table Of Contents

Abstract

This paper analyzes the theoretical addition of a subtitle generating feature to the Kodi system. Adding this feature would not only change existing interactions between components but also add new complex components to the system. This paper will analyze a couple of different possible implementations of this feature, apply the SAAM technique to the propositions, explore some use-cases for the new feature, and enumerate the functional and non-functional requirements associated with it.

In the implementation section, two different possible implementations of this feature were discussed. Since both implementations would be encapsulated in the same component, we first outlined the new dependencies of existing components. The "Subtitle Manager" component would depend on the library manager, local file manager, and request manager. The GUI manager and player core would each depend on the subtitle manager.

Within the subtitle manager component, we brainstormed two different implementations. The first implementation would incorporate a machine learning model to generate subtitles on the local machine. This would be a pipe and filter architecture.

The second implementation we derived would move the computation to a remote server. Instead of running the machine learning models on the machine, which may be constricted by the machine's resources, we would send the audio to a remote server and receive back the generated subtitles. This approach would use the client-server architecture.

When analyzing these two methods, we found the local implementation (implementation 1) to best satisfy our NFRs. We also found that two out of the three stakeholders mentioned were better off with this implementation. While there were a couple NFRs and one stakeholder that were better satisfied by implementation 2, implementation 1 was the clear winner.

Two use cases were then explored. These give context to the new components and their interactions with the existing ones. The two use cases identified are uploading subtitles to a central server and generating new subtitles for a video. These diagrams show a mix of new and existing function calls between components, showcasing the deep integration of the new component. Each diagram is also explained in depth to ensure that the use case is well understood by the reader.

We explored testing options for the functional and non-functional requirements. For the functional requirements, we concluded unit testing should be done using the black/white box testing method. Two integration tests should also be performed with respect to the new component. To test the non-functional requirements, we demonstrated a suite of tests to ensure performance, reliability, usability, and security. These tests would be automated and run every time a change is requested.

In doing this assignment, our group learned that the creative process cannot always be performed from scratch. Instead, it is important to supplement creativity with relevant tools. We also learned to be patient during the design process as this determines the ease of completion for the rest of the project.

A reflection and analysis of our conclusions are presented at the end of the report, where further implementation details are also discussed. We additionally reflect on what the team learned about enhancing large-scale software projects.

Introduction and Overview

Kodi is a free and open-source multimedia center designed to be user friendly and all-encompassing [1]. The software is managed by the Kodi Foundation [2], though the bulk of the development is done by Team Kodi which is made up of members of its contributor community. It is available for a multitude of platforms, including iOS, Android, Windows, MacOS and Linux, and supports a variety of input devices, including mouse and keyboard, game controllers, and touch screens. Kodi has support for a variety of video, audio, and image formats.

The Kodi project was originally developed for the Xbox (2001) as a piece of "homebrew software," or software that is not officially endorsed by Microsoft. Users required a modified console to install the software, which was called XBMC (**XB**ox **M**edia **C**entre) at the time of release in 2003. Over the first 10 years following its release, it was ported to many other systems, including Linux, Windows, MacOS, Android, and iOS. In 2014, the software was renamed Kodi, to dissociate the software from its Xbox roots [3].

As a media centre, Kodi does not come bundled with any media, rather the user must provide it themselves. This media can either come from a local disk, another data source on the local network, or a remote server. Add-ons can be used to add additional media sources. These include popular media services such as YouTube, Twitch, Plex, and Spotify. Kodi then handles the entire media playback pipeline, delivering the content to the user for viewing.

This paper outlines the theoretical addition of a subtitle generating component, along with its effects on the system. Even though a feature like this sounds relatively small, we found multiple new interactions and components.

The first section of this paper begins with an enhancement proposal. This is a high-level description of the proposed feature, subtitle generation. This description does not touch upon the technical side of the software, but rather user experience. This includes its integration within the UI, and a few reasons why it improves the final application

After this, an outline of the functional and non-functional requirements of this new feature is provided. It is always important to examine these requirements before building the software as they can have a large bearing on the architecture.

The current state of the software with respect to the feature was explored next. As subtitles are quite common, and necessary for many people, Kodi has existing functionality related to subtitles. An analysis of how this functionality compares to our feature addition is provided.

Next, two different possible implementations for the subtitle generator component were explored in-depth. Thier strengths and weaknesses were assessed using the SAAM. This explained which implementation might be better for specific stakeholders and weighed the different NFRs to find the best implementation to put into production.

Two use cases for this feature were also dissected, showcasing how different components interact through this feature. Each of these use cases leveraged the new subtitle manager component in a different way. These use cases gave context to the components, providing examples and explanations of their interactions. The two use cases identified are uploading subtitles to the Kodi server and generating new subtitles on the local machine.

After this, testing plans were briefly described. These testing plans provide a method to confirm if the suggested implementation works. Outlining the test cases properly will also aid in the understanding of our feature as a whole. Closely related, the potential risks of implementing this feature were discussed. This includes but is not limited to security, development, and financial risks. By analyzing the risks involved, we seek to understand the potential implications of this feature. We will examine whether the risks are too high and outweigh the benefits of this feature.

We concluded that the introduction of a subtitle generating feature would greatly improve the user experience and accessibility. Through a deep analysis of stakeholders, non-functional and functional requirements, we confirmed that the best way to implement this feature was using a local machine learning model.

Enhancement Proposal

Our proposed enhancement for Kodi is the integration of a machine learning (ML) subsystem to generate subtitles for movies and other forms of media. The feature aims to enhance the user experience by providing subtitles on-demand for any movie in the library ahead of time. Users will be able to right-click on a piece of media from the library and select an action called "generate subtitles", which is particularly beneficial for individuals with hearing difficulties or watching movies in a foreign language. This enhancement not only adds functionality to Kodi but also expands its accessibility and allows it to appeal to a wider audience. Furthermore, the generated subtitles can then be uploaded to a central cloud server from which other users can download and use the subtitles, without needing to do the computational work of generating the file for themselves. Subtitles will be selected from the same menu that is currently used for local subtitles.

Functional Requirements

User Interface: A user-friendly interface that allows for easy activation and interaction with the subtitle feature.

Subtitle Generator: An ML model trained to generate subtitles based on the audio output of the media file.

Subtitle Storage: Capability to store, edit, download and reuse generated subtitles for future viewings.

System Integration: Integration with Kodi's existing architecture, allowing users to activate subtitle generation easily.

Non-Functional Requirements

Performance: High-speed subtitle generation with minimal latency to ensure as little wait time before consuming content.

Scalability: Ability to handle an increasing number of users and simultaneous subtitle generation requests, should a cloud component be involved.

Reliability: Consistent accuracy in subtitle generation across various genres and audio qualities.

Security: Robust security measures to protect user data, especially if any personalization is involved. Other users should not be able to see who created subtitles.

Maintainability: Easy system maintenance for updates and improvements without disrupting Kodi's overall functionality.

Current State of System Prior to Enhancement

Currently, Kodi supports some level of subtitle utilization in multimedia viewing. Natively, Kodi supports media files with either built-in subtitles, or an external subtitle file. For example, MP4 files can contain subtitles directly (hardcoded) or reference dedicated subtitle files, such as an SRT file. Kodi can handle both of these options. Subtitles from these files are displayed on screen during a viewing and can be configured to tailor specific needs.

Kodi also has plug-ins that can be used to download subtitles available online but not already present on the local machine. One of these plug-ins is called “Open Subtitles” [4]. This plug-in has access to a database of 7 million subtitles for popular movies and TV shows (*English subtitles only*). If you own a copy of one of these videos in their database, you can use this plug-in to download its respective subtitles.

Both these options have flaws that the proposed enhancement will resolve. Firstly, finding media with included subtitles can be challenging. If most of the media on your Kodi system is home videos, it is very unlikely that there will be any subtitles. Lots of media found online will not include subtitles either. While some media has subtitles “baked in” to the video itself, this prevents the use of subtitle configuration. Baked in subtitles also cannot be disabled.

Using a plugin to download popular subtitles can be of great use, unless the media does not exist in the database. If the media on your Kodi was produced yourself, it is unlikely that a media database would contain subtitles for it. The only way to produce subtitles for your own media would be to create the subtitle file yourself. Not only is this tedious and time-consuming but it can be extra challenging for non-technical users.

Overall, the current subtitle system in Kodi is useful but lacks 100 percent coverage. Our feature proposal seeks to work alongside the current subtitle system to improve its scope.

Generic Implementation

No matter the specifics of the implementation, there are a few architectural commonalities that must be present in any version of our proposed feature. A new component must be created to accommodate the increased functionality around subtitles, called the “Subtitle Manager”. Primarily, this component encompasses the Machine Learning library used to generate the subtitles and wraps the library’s output into a container file to be stored on disk. This library could either be made in-house which uses a model like Open AI’s Whisper [5] for machine learning, or use a premade, fully built, open-source library like S*ubs AI* [6] This is similar to Kodi’s approach to video encoding/decoding using the open source FFMPEG library. The Subtitle Manager must also communicate with the central Kodi subtitle repository to upload generated files and search for texts generated by other users. Figure 1 below displays this component integrated with the existing architecture.

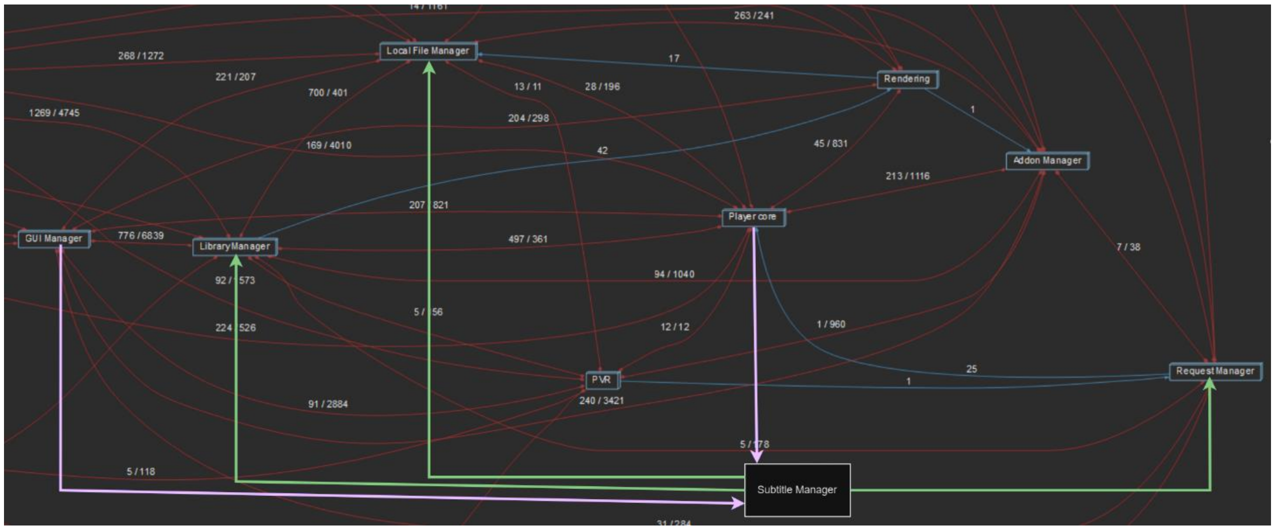


Figure 1: Snippet of dependency diagram from A2, generated in Understand, with the addition of the Subtitle Manager component and dependencies.

The subtitle manager has 3 dependencies. It relies on the Local File Manager to store the subtitle files it generates, the Library Manager to pass in the audio files from the videos for subtitle generation, and the Request Manager to pull in subtitles generated by other community members and then uploaded to the central Kodi subtitle repository. The Player Core relies on the subtitle manager to supply subtitles, and the GUI Manager must make function calls to the Subtitle Manager. These components operate in an object-oriented style, where the implementation and underlying data of each component is hidden away from the others. They only communicate through public method calls.

Where the specific implementations differ is in the internals of the subtitle manager, and the ways in which it interacts with other components. The simple binary existence of the dependencies does not change, however. The difference between the two implementations lies in the subtitle generation. In implementation #1, the subtitles are generated on the user’s local device, while in implementation #2 they are generated on a server machine.

Affected Files

All files in the Library Manager, Request Manager, and Local File Manager will remain unaffected, as the Subtitle Manager will simply make calls to existing functions. The GUI Manager and Player Core, however, will be affected as follows:

- *Kodi\xbmc\cores\VideoPlayer\VideoPlayerSubtitle.cpp* (Add additional subtitle source)
- *Kodi\xbmc\video\dialogs\GUIDialogSubtitleSettings.cpp* (Add additional options for generation in subtitle dialog)
- *Kodi\xbmc\video\dialogs\GUIDialogSubtitles.cpp* (Allow selection of generated subtitles)
- *Kodi\xbmc\cores\VideoPlayer\DVDSubtitles\DVDFactorySubtitle.cpp* (Allow subtitles to be generated for DVD files; add additional source)
- *Kodi\xbmc\settings\SubtitlesSettings.cpp* (Need additional settings for uploading or downloading settings from the Kodi Server

Possible Specific Implementation #1

Under implementation #1, the user's machine does all computation necessary to generate the subtitles. This means that the ML model will need to do all computations on whatever hardware the user has. Under this implementation, a Pipe-and-Filter architectural style is used to generate the subtitles. As explained on an Open AI blog post [7], machine learning text generation is typically done through streaming the audio data through a series of lower-level components (filters).

Possible Specific Implementation #2

In implementation #2, the audio data from the video file is sent to a remote server to be processed through the ML model. The output of the model is then sent back to the local machine to be saved in a local file, as outlined in the generic implementation. Under this implementation, the primary architectural style is Client-Server. The Pipe-and-Filter style is abstracted away from the Kodi code base and is therefore no longer relevant.

In this implementation, the Subtitle Manager relies on the Request Manager to connect to the external server for subtitle generation. The output file can then be saved to the Kodi repository by the cloud generator, to avoid sending the data through an extra trip through the user's system.

SAAM

The Software Architecture Analysis Method (SAAM) was applied to the above possible implementations to generate the following analysis. Table 1 details who the stakeholders are and which NFRs are most important to each of them, while Table 2 explores each NFR in depth.

Table 1: Important stakeholders and their associated primary Non-Functional Requirements (NFRs).

Stakeholders	Primary NFRs
Users	**Performance –** Users want subtitles to be generated as quickly as possible. **Cost –** The cost to the user should be as low as possible, if not zero, since Kodi is currently free software. **Accuracy** – The subtitles generated need to be as accurate as possible to be useful to the user.
Kodi Developers	**Scalability –** Currently, Kodi is highly scalable because it is all locally computed on the user's machine. Developers would prefer to keep it that way to keep complexity down. **Maintainability** – Developers want the software to be as easy to maintain as possible. **Testability** – Ease of testing is positive for developers, to produce bug-free code.
Kodi Foundation	**Cost –** The Kodi Foundation is funded by donations from users, and as such wants to keep their costs very low. **Maintainability** – Keeping the software maintainable will help attract developers from the open-source community. **Security –** Security is a high priority for the Kodi Foundation, both to maintain their image and integrity as well as protect their assets.

Table 2: Comparison of each NFR paired with each potential implementation. Testability is comparable between the two implementations and as such was not included.

NFR	Implementation 1 (Local)	Implementation 2 (Cloud)
Performance	Highly dependent on the characteristics of the end user's machine. Faster computers will produce subtitles much faster.	Quick results can consistently be computed regardless of the user's machine. Internet speed is the only factor, as audio needs to be uploaded.
Cost	Free for both parties, aside from development time (which does not differ between the two implementations).	One party will have to pay for server time to generate the subtitles. Either the Kodi Foundation will absorb it, or it will be passed down to the end user as a "premium" option.
Accuracy	Different models may need to be selected for slower machines, which produce less accurate results.	Likely very high, assuming high performance cloud architecture is used.
Scalability	High. There is no change compared to current Kodi versions.	Limited. Additional server resources will have to be purchased if more users begin using the service.
Maintainability	Somewhat diminished, as the library used for subtitle generation will have to be periodically updated.	Greatly diminished. Not only will the library need to be updated, but server infrastructure must be maintained and monitored.
Security	Unchanged. No additional data is sent from the user to Kodi.	Potential risks. Audio files sent to the subtitle generation server may contain malicious code.

SAAM Discussion and Conclusion

Upon considering the above tables, implementation 1 was selected as the best option for the subtitle generator. Although the performance and accuracy are potentially diminished by selecting this implementation, all other NFRs are better satisfied through option 2. Of all listed NFRs, performance and accuracy may be some of the least important. They only affect the user, and in a very "soft" way. Even if the subtitles take a long time to generate or are not perfectly accurate, many users will still choose to use the feature because it's better than no subtitles at all.

Possibly the most important NFR to consider is the cost. One of Kodi's key pillars is that it is *free* and open-source software. Since the Kodi Foundation is donation-supported, it would be very unlikely that they could support paying for the subtitle generation server-time out of pocket. Therefore, they would have to pass this cost down to the user. By creating a tier that requires users to pay, they are violating this fundamental ideology. Cost alone may force the selection of implementation 1.

When considering the three stakeholders, the users are the only group who benefit from implementation 2. Both developers and the Kodi Foundation wholly benefit from the first option. It is easy to understand, then, how some users may favour implementation 2, but ultimately the best choice for all involved parties is implementation 1.

Use Case #1

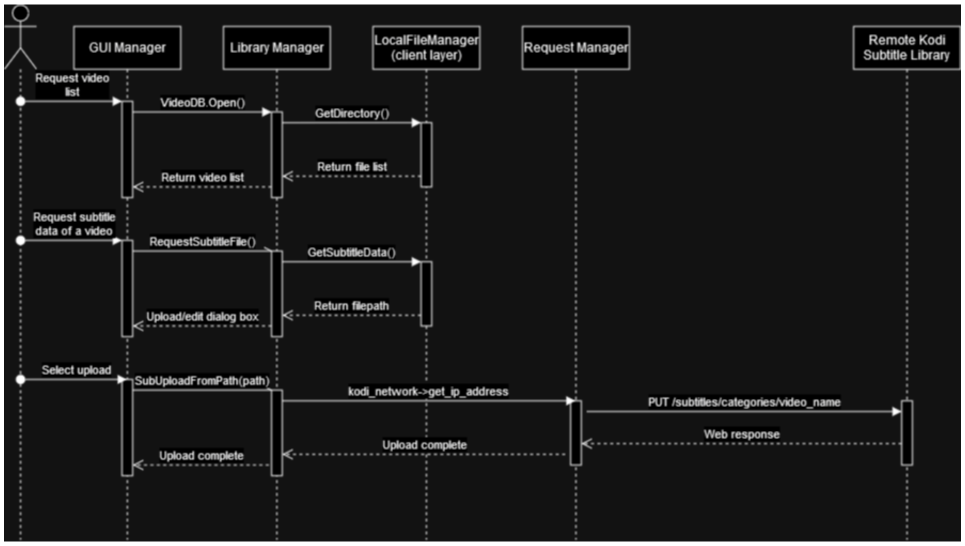


Figure 2: Sequence diagram of use case 1 – a user uploads subtitles to a central server for user sharing.

In use case 1, a user opens a list of video files, selects a video file and hits the 'subtitle' options in a drop down right click menu. The user then selects the upload subtitles button. This will upload their custom subtitles to a remote subtitle library where other users can download the subtitles.

To do this, the user first opens their video database within Kodi using the GUI manager to get a list of their videos. The GUI manager must interact with the Library Manager to get the videos directory to pull the video and its metadata. The Library Manager will call the Local File Manager to open the machine file manager, which will return a list of all saved videos in the Kodi library folder. This information is then returned to the user by displaying the file list in the GUI.

The user will then right click and access their subtitle options, where they will select a saved subtitle package for the video. There is an option to upload the subtitles to the Kodi Subtitle Library for other users with the same video. The Library Manager will send a request through the Request Manager to upload the subtitle data. The Request Manager will upload the file using the PUT call, which will upload the subtitle data in the corresponding video folder on the Kodi Subtitle Library. The user will then receive a prompt from the GUI showing that the upload was successful.

Use Case #2

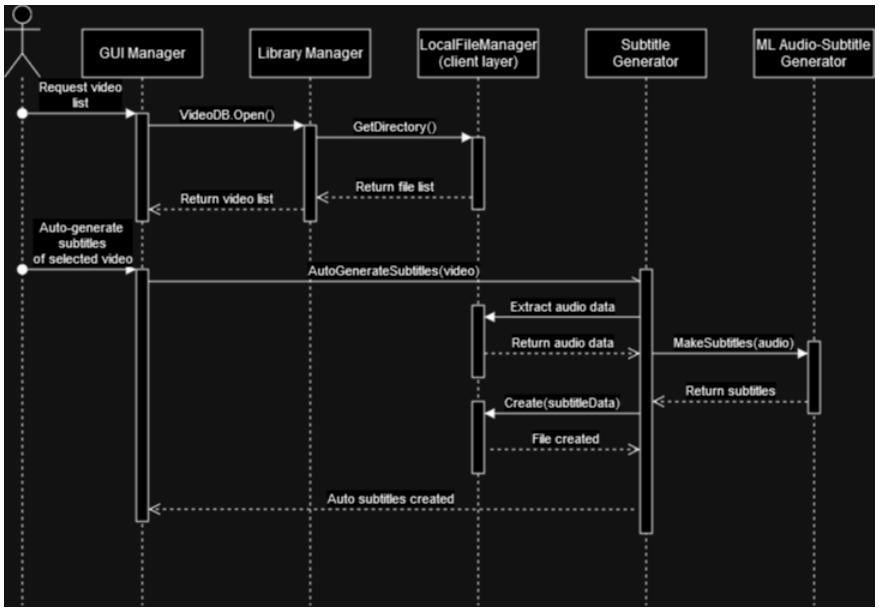


Figure 3: Sequence diagram for use case 2 – a user automatically generates subtitles for a video using the built-in subtitle ML subtitle generator.

In use case 2, a user browses a list of their saved videos and selects a video to automatically generate subtitles for. The user will select “subtitle options” in a drop down right click and hits “automatically generate subtitles”. The subtitle generator will then create subtitles for the video from its audio data.

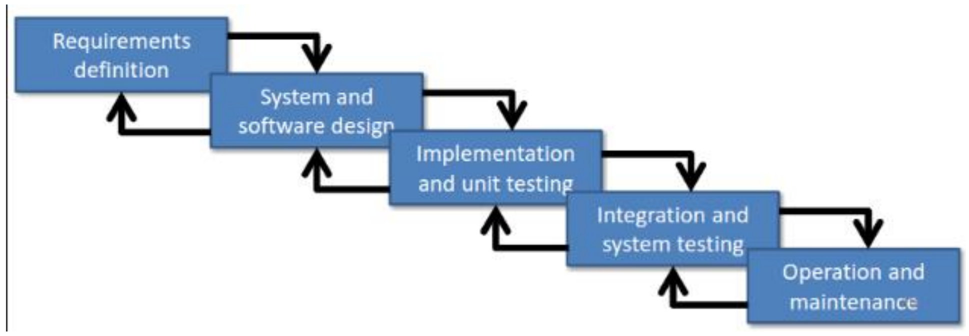
To do this task, the user must first acquire their video list to display in the GUI. This is done through the Library Manager to get the user’s video directory. The Library Manager will call the Local File Manager to get the contents in the video directory. This information is then returned to the user by displaying the file list within the GUI.

The user will then hit the ‘automatically generate subtitles’ option, which is preprocessed by right clicking a specific video. To generate the subtitles, the user, using the GUI, makes a request to the subtitle generator. The subtitle generator will interact with the Local File Manager to extract the audio data from the video file. Then, the subtitle generator will call its ML Audio-Subtitle subcomponent. The ML Audio-Subtitle subcomponent will then translate the audio data into subtitle data, the subtitle data is sent back to the Subtitle Generator, which will then create the file containing the subtitle data using the Local File Manager. The user will receive a prompt after this process has been completed.

Testing Plans

It can be assumed that the Kodi developers have opted for the iterative waterfall model for Kodi's development. The iterative waterfall method is the oldest and most stable software development approach, which is deemed highly suitable for such a complex system like Kodi.

Within the iterative waterfall model, implementing a new feature such as subtitle generation falls into the "implementation and unit testing" phase. In the software maintenance model, specifically categorized as perfective maintenance, the addition of novel features is addressed. During this phase, the focus is not only on crafting unit tests to evaluate the functionality of the new feature but also on conducting regression testing. This kind of testing includes rerunning of both functional and non-functional tests established based on the developers' original specifications and requirements. The goal is to seamlessly integrate the subtitle generator into the existing system while validating its functionality.


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Functional Testing:

A major part of functional testing falls into unit testing, which consists of black/white box testing, to test specifications/functionality of the feature. Black box testing is applied for assessing external behavior, while white box testing examines internal functions.

- **Example Black Box Test (Output Coverage):** On a specific media file, for example the John Wick 2014 movie, have the subtitle generation enabled. Then run through the movie and compare the displayed subtitles vs full transcript of the movie. If the subtitles displayed match +99% of the transcript then flag this test as pass.
- **Example Black Box Test (Input Coverage) - Remote Code Execution Security Test:** Ensure that all search bars and human input are interpreted properly and that no code additional code or commands are executed than expected. Example, if Kodi used SQL

behind the scenes, use SQL profiler to ensure that no additional queries are being run besides those that are called directly by the system. Completion criteria is that each search form adequately abstracts its input enough so that malicious code cannot be executed in the deeper layers of the system.

- **Example White Box Test (Decision Mutation):** Consider a decision point that determines the language selection based on user preferences to start the generation of subtitle data. If such language is not stated but the generator is turned on, it might fall back English as the language. To test the fallback feature, we can intentionally turn the generator on but not feed any language preference. This test will pass if English language generation is performed and ultimately displayed on the said media.
- **Example Integration test:** Compare debug output of the generated subtitles vs the output actually displayed on the screen. If debug output 100% matches the displayed subtitles, then this test is flagged as a pass. This test shows the accuracy of the integration of the ML Audio-Subtitle Generator along with Subtitle generator and GUI manager.

Non-Functional Testing:

Non-functional testing comprises performance, reliability, usability, and security tests. These ensure a comprehensive evaluation of the subtitle generation feature.

All of the mentioned functional and non-functional tests are fully automated. Additionally, automatic report generation is implemented to seamlessly facilitate unit and integration tests into the system testing suite, enabling future regression testing.

As mentioned before, integration testing is conducted after white/black box testing to ensure the seamless incorporation of the subtitle generation component with all other components. Regression testing involves the systematic rerun of existing and previous functional and non-functional tests. This iterative process ensures the continued functionality of the system, validating that newly implemented features, such as the subtitle generation, do not adversely impact the existing functionalities.

Potential Risks

1. **Security Flaw:** Injection Attacks in User-Uploaded Subtitle Files

If the subtitle manager fails to adequately sanitize and validate user subtitle data before it is sent to Kodi's server, malicious users might embed harmful code within subtitle files posing a risk to Kodi's server and the clients themselves. The injected code could exploit vulnerabilities that enable remote code execution (RCE) when an end client has the harmful subtitle pulled and ran on their machine. The solution to this would be to implement stringent input validation and sanitization procedures within the subtitle generation component on user's machines.

2. **Performance Risk:** Resource hogging during ML subtitle generation

When users click "generate subtitle," the process consumes substantial computing resources, potentially causing delays and slowdowns on the client's machine. This may result in an unsatisfactory user experience and impact overall system performance. A solution could be implementation of asynchronous generation during inactive device times like overnight.

Lessons Learned

The process of proposing an enhancement to the Kodi software gave our group a chance to apply the lessons we learned from the last deliverable, while also highlighting for us the importance of creativity and reasoning in problem solving.

The first challenge we encountered was during the brainstorming process. Many of the different features we came up with were either already incorporated into Kodi, or simply did not offer enough utility to warrant moving forward with them. We found that by taking a step back and simply using the application, it became much more natural to find a feature that we felt was missing. For our group, this step in the process was a lesson that the creative process cannot always be performed from scratch, and that it is important to supplement our creativity with relevant tools such as the application itself and the internet.

Furthermore, another lesson learned by our group was not to be too hasty in our design process. After establishing that our enhancement would be subtitle generation, we had to make sure that the architectures we chose were suitable and determine how a different architecture would differently depend on other components. With each new rough sketch of our use case diagrams, the number of our dependencies grew. Sometimes, we would be satisfied with our work, come back to it the next day and realize that another subcomponent interaction was missing. After refining the use cases diagrams and intuiting the added interaction within the system, we concluded that it was better that we waited a little longer before starting the new dependency diagram, as we would've missed many important details.

In the end, the process of creating this enhancement taught us the importance of cultivating our creativity through the utilization of available tools and being patient during the design process. Next time we are faced with a similar problem, we have this experience to fall back on and gain insight from.

Conclusion

In summary, our enhancement proposal for the Kodi software was the introduction of a subsystem responsible for generating subtitles for any media file, through the use of an ML model. Alongside this, users who have generated quality subtitles for a file can upload them to a cloud Kodi server, from which other users can access and download. One of our implementation designs made use of a pipe & filter architectural style, while the other used a client-server architecture.

Furthermore, the dependency graph provides context on the interfacing patterns of the new module and allows to see 'where' it would fit in the system. We determined these dependencies through intuition, heuristics, and an analysis of how pre-existing modules that were like this one interacted with the system.

Through a SAAM analysis across the 2 different implementations, we found that the first implementation was able to offer an easier integration process with the rest of the system, meaning that in a real-world setting, it is more likely that this is the way in which a system of this kind would be implemented.

The sequence diagrams created in this assignment provide information on how a specific function would interface the system to complete its job. Our sequence diagrams represent two use cases, the first being a user submitting their already-generated subtitles to a central Kodi server for other users to use, and the second being a user pre-generating the subtitles for a given media file. Like the A2 report, our use case diagram contains specific function calls to highlight the specific pieces of a module or subsystem that were being called.

In conclusion, our enhancement aims to help those who are hard of hearing, those who may not be completely fluent in the respective language of the movie, or even those who want to watch in a different language altogether. The enhancement would require a complete integration into the current system, and after careful analysis, the first implementation serves the general role of the enhancement better.

References

[1] "About," Kodi, https://kodi.tv/about/ (accessed Dec. 5, 2023).

[2] "Kodi Foundation: About," Kodi, https://kodi.tv/about/foundation/ (accessed Dec. 5, 2023).

[3] "History of Kodi," History of Kodi - Official Kodi Wiki, https://kodi.wiki/view/History_of_Kodi (accessed Dec. 5, 2023).

[4] "OpenSubtitles - Homepage," OpenSubtitles, https://www.opensubtitles.org/en/en%20 (accessed Dec. 5, 2023).

[5] OpenAI, "OpenAI/whisper: Robust speech recognition via large-scale weak supervision," GitHub - Whisper, https://github.com/openai/whisper (accessed Dec. 5, 2023).

[6] Abdeladim-S, "Abdeladim-S/subsai: 🎞 subtitles generation tool (web-UI + CLI + python package) powered by OpenAI's whisper and its variants 🎞," GitHub, https://github.com/abdeladim-s/subsai (accessed Dec. 5, 2023).

[7] OpenAI, Introducing Whisper, https://openai.com/research/whisper/ (accessed Dec. 5, 2023).