CISC 322 ASSIGNMENT 3

Upon Further Review: Kodi and Letterboxd

Group 2: ArchAngels

December 5th, 2023

Adam Clarke	-	21asc6@queensu.ca	-	20303748
Owen Rocchi	-	19omr6@queensu.ca	-	20223890
Bianca Bucchino	-	20blb@queensu.ca	-	20232319
Aidan Sibley	-	20ajs18@queensu.ca	-	20271383
Omar Ibrahim	-	20omha@queensu.ca	-	20263583
Christian Higham	-	20csdh@queensu.ca	-	20287289

Table of Contents

TODO: Make sure to reload then add spacing when complete (and remove names) ‼️

Executive Summary (Christian)

- Brief overview of the project
- Key findings and recommendations

- WILL DO ONCE REST OF PAPER IS COMPLETE -

Introduction (Christian)

Following from our previous reports, the Kodi media controlling software should be well known at this point. Its many components and subcomponents are layered in a way that allows for optimal video and audio playback, game emulation, and whatever other functions users have created addons for. Primarily though, it is used by movie lovers the world over to watch their favourite locally accessible flicks.

Kodi is not the only piece of software cinephiles flock to, however. For passionate film discourse, there is the movie review website Letterboxd. Boasting over ten million users, Letterboxd allows users to rate and review movies, display your top films on your profile, and catalogue movies based on any criteria you can think of. Noticing the overlap between Kodi and Letterboxd userbases, we wanted to examine how Letterboxd integration into Kodi could possibly work.

This paper explores the intricacies of implementing a new series of one existing software's features into another. This entails determining what features should be added, who would be affected by these features, what requirements these stakeholders would demand, and how the architecture of the software should change to accommodate these new features.

In the case of enhancing Kodi with Letterboxd support, the new features are centred around Kodi's movie-playing aspects. This upgraded Kodi should allow for users to sign into Letterboxd, be recommended movies that can be locally accessed via downloaded files, streaming, or otherwise, and rate and review these movies on Letterboxd from within the Kodi software. The involved parties would be the end users, open-source developers, and Letterboxd itself due to the use of their API. Each of these parties have their own set of nonfunctional requirements. The architectural modifications lie mainly in the domain of Kodi's Media Handling and User Interface components to allow for Letterboxd API usage, though this poses a series of risks that is discussed in the relevant section. Further, we explored two methods of implementation, debated their merits, and determined a theoretically optimal course of action for Letterboxd integration into Kodi.

Current Software System Analysis (Omar)

Kodi's current software system is structured in a repository style, comprising three primary categories: User Interface, Media Handling, and System Integration. These categories interact through a central control and information hub known as the Core Functionality subsystem.

Kodi boasts features such as video playback, retro video game emulation, and addon support. Additionally, it seamlessly communicates with the operating system to manage inputs, events, and the filesystem, ensuring smooth integration between the application and the user's device. The software also includes networking support, facilitating remote control functionality and enabling video/music streaming.

Proposed Enhancement (Adam)

Kodi's user base already consists of, in large part, film enthusiasts. By integrating Letterboxd with Kodi, this will give users the ability to connect with other film enthusiasts, and provides further opportunity for data collection and personalized user movie recommendations.

In today's age, recommendation algorithms are the backbone of both user retention and engagement, and Kodi needs to compete.

Since movies on the Kodi platform are largely user imported, Kodi will need to match user input media to entries in the Letterboxd database. Due to the messiness of string matching, a small, highly specified in-house language model can be employed for this.

Upon implementation, Letterboxd should provide an innovative complement to the Kodi user's browsing and media consumption experience. Users should be able to see a Letterboxd recommendation tab that suggests movies they already own, whether they've imported the video files or connected a service like Netflix. To achieve this, the recommendation system should examine popular addon components for a comprehensive recommendation system. The integration will require heavy reliance on Letterboxd's API. In order for a proper integration, Kodi will need access to:

- Letterboxd's database of media, including ratings, genre, and movie name
- Letterboxd's user authentication, so Kodi users can review movies from the Kodi app
- Letterboxd's recommendation system

The integration of Letterboxd with Kodi needs to be done such that it respects and minimally disrupts the existing architecture, and acts as a complement to it.

The main risks of this implementation are the possibility of overcomplicating the codebase, difficulty attaining the Letterboxd API, and over-reliance on the Letterboxd API.

Stakeholder Analysis

Identification of major stakeholders

End User

The most obvious stakeholder affected by this change is the end user. The user's experience should be prioritized above all else, so measures must be taken to not only prevent any deterioration but to enhance it.

The recommendation system must seamlessly integrate with Kodi without overly interfering or slowing down the existing interface. Additionally, the number of recommendations should be sufficient, while maintaining relevance and accessibility.

Open Source Developers

As Kodi is open source, developers maintaining and adapting the code-base should also be considered. The architectural and dependency changes must not interfere with the code base's quality and reliability.

Failing to invest in code quality can slow future development in the long term and incurs an invisible debt that grows every time the code is altered.

Letterboxd

If Letterboxd is to provide an API for Kodi developers, they need to ensure it is used appropriately, not needlessly strained, and does not open them up to vulnerabilities. Furthermore, the increase in Letterboxd's user demand must also be considered.

Non-functional requirements (NFRs) for each stakeholder

End User

Response Time

75% of data pulled from Letterboxd must be displayed in less than 2 seconds.
As before-mentioned, the prompt appearance of movie ratings is crucial for maintaining quality user experience.
A delay in the display of information may lead to user complaints or even abandoning the feature entirely.
The 2 second limit, although arbitrary, defines what can be considered a "quick" response time.

Accuracy of Recommendations

Media recommendations must be relevant at least 50% of the time.
. Recommendations that do not align with the user's interests are unacceptable.
It’s understandable that the recommendation algorithm may take some risks in guessing the new media to be recommended, but it is extremely important to stay relevant most of the time. If a user is being shown new content that is relevant to them more often than not, this extension would be working very effectively.

Open Source Developers

Maintainability

Any pushed change to code must update current test cases, and have accompanying test cases included. Testing is extremely important in effective software development, even more so in the context of open source software with many different developers working on the same project. Making it necessary that any pushed changes to code must have accompanying test cases would ensure that every addition is being thoroughly checked for its accuracy. This NFR would ensure success in long term development as it would stop potentially harmful changes to the code from going unnoticed.

Documentation

Maintain documentation explaining at least 95% of code base functionality. This is once again a quality attribute that is specifically useful for the open source context of development. This would ensure that everything about the code base is diligently tracked. This would be specifically useful when examining older additions to code in order to understand the motivation behind them.

Letterboxd

Security

Regularly investigate API for potential vulnerabilities, and address high risk issues within 24 hours. One of the most important things about having an API like this would be keeping it protected. Anybody could be looking to take advantage of or even potentially abuse the API if they found a way. Having the system be regularly checked in would prevent breaches from flying under the radar. Urgency is key as addressing significant issues within 24 hours would prevent any single breach from getting out of hand.

Scalability

Actively monitor server load during times of high activity, ensuring that load stays below 80% maximum capacity. This is important to ensure that there exists enough resources for everyone who wants to use this service on their Kodi system. Keeping an active eye on the load levels will give Letterboxd a better idea of when it may be time to expand on their current infrastructure.

Architectural Analysis of New Subsystem (Aidan)

Analysis of current architecture

Kodi's architecture as outlined in the concrete architecture (A2) report, follows a repository style, deviating from the initially conceptualized layered architecture. This shift represents an increase in flexibility and integration capabilities, at the expense of complexities in separating different contexts. The core subsystems are able to be abstracted to a conceptual level as a result, and we see that there are four of these. **User Interface** provides the dialogs, windowing, and GUI API's,

is the port of user interaction, **Media Handling** is the frontend of the core business logic of Kodi, **System Integration** provides the operating system and hardware integration, and most importantly **Core Functionality** is the real central functional subsystem performing the logic for most functionality. If we take a look at a concrete architectural view we can see that the core functionality subsystem provides Kodi with the necessary adaptability to any potential interactions, be it with the user, or media formats. This adaptability will be instrumental in integrating the Letterboxd module, as it permits the extension and customization to the media playback, and GUI user experience.

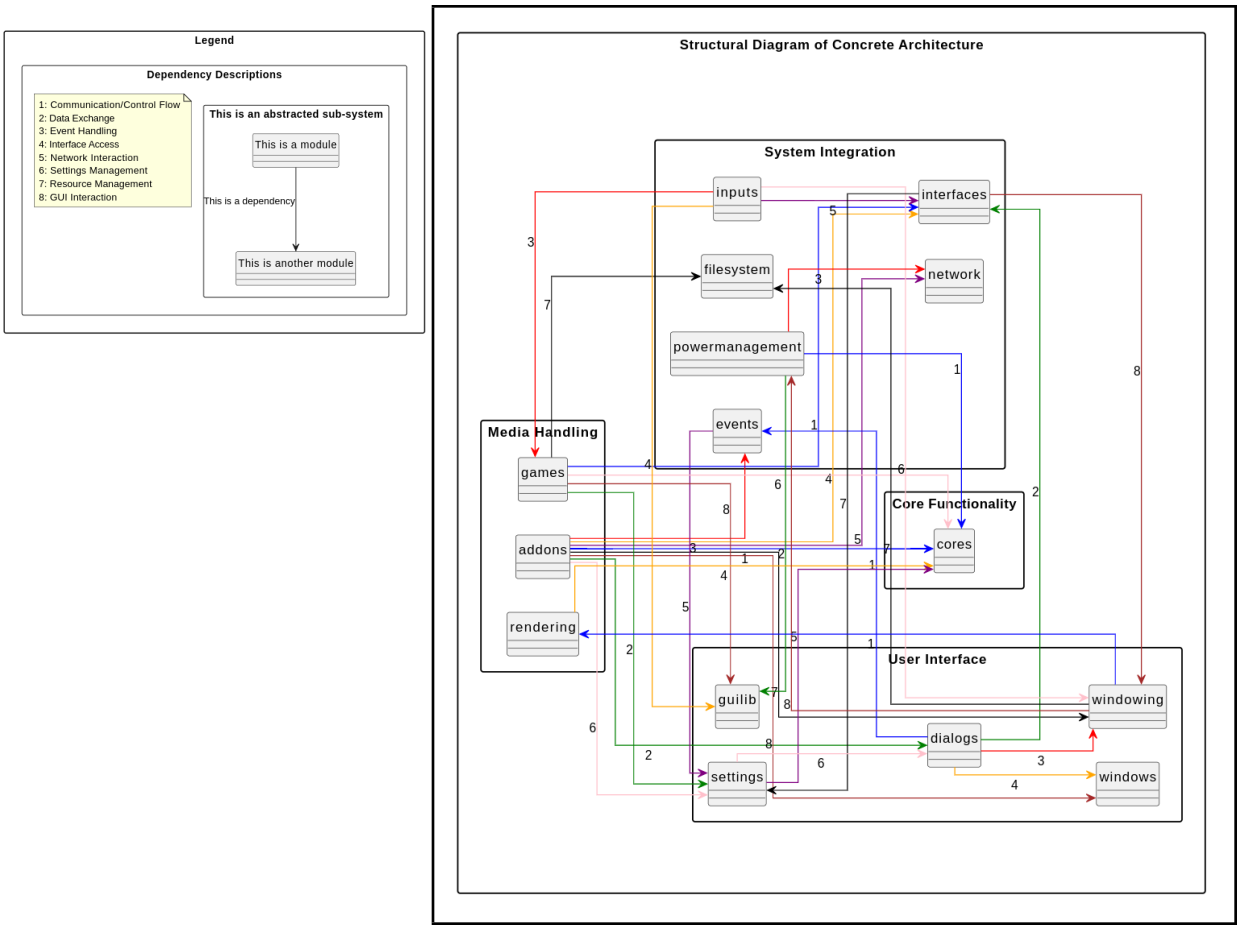

Figure 1 - Kodi Concrete Architecture

The deviation from a strict modular approach to a more intricate interdependency and integration within the system highlights the complex nature of media handling in Kodi. The architecture's evolution reflects a balance between theoretical principles and pragmatic software development needs, accommodating evolving functionalities and user requirements.

Proposed architectural changes to support the enhancement

The proposed architectural changes involve the integration of the Letterboxd Module into Kodi's existing structure. This module will create a bridge between Kodi's Media Handling subsystem and the Letterboxd service, enhancing the system's capability to offer personalized movie recommendations. The User Interface subsystem will also be updated to display these recommendations effectively.

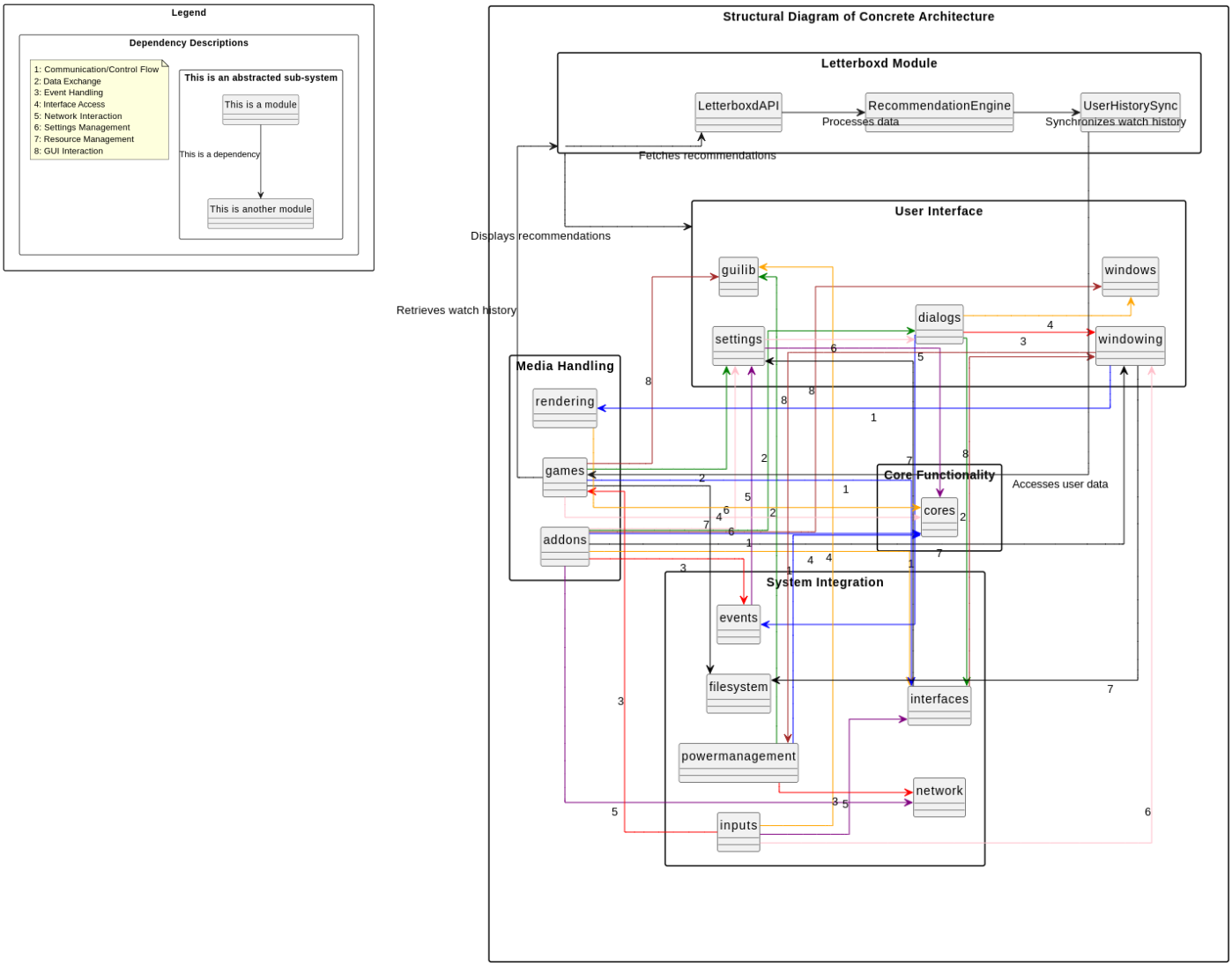

Figure 2 - Kodi Concrete Architecture with Letterboxd Integration

The integration of the Letterboxd Module into Kodi's existing architecture, as depicted in the UML diagram, represents a significant enhancement in terms of personalization and user experience. The Letterboxd Module, a new subsystem within Kodi's structure, is designed to

interact seamlessly with the Media Handling and User Interface subsystems. Its primary function involves data synchronization and processing, where it retrieves the user's watch history from the Media Handling subsystem. This data is then processed and utilized to generate personalized movie recommendations through the Letterboxd API.

The User Interface subsystem will undergo updates to ensure these recommendations are displayed in a user-friendly and engaging manner. This integration aims to enrich the user experience by providing tailored content suggestions, thereby increasing user engagement and satisfaction with Kodi.

maintaining the integrity and performance of the existing Kodi architecture while adding significant value to the user experience. ?

To accommodate the Letterboxd Module, the Media Handling subsystem will be enhanced to provide the necessary data regarding user watch history. This modification is crucial for the Letterboxd Module to function effectively. Additionally, new interfaces and data flows will be established between the Letterboxd Module and Kodi's existing subsystems. These interfaces are designed to ensure efficient and coherent data flow, maintaining the functionality and responsiveness of the system.

Discussion of the impact of these changes and potential risks

The integration of the Letterboxd Module into Kodi's architecture primarily aims to enhance the user experience through personalized movie recommendations. This feature promises to elevate user engagement by tailoring content suggestions based on individual watch histories. Simultaneously, updates to the User Interface are designed to display these recommendations in an intuitive and accessible manner, further augmenting Kodi's usability. However, this integration also introduces challenges in terms of system performance and non-functional requirements. The additional processing required for data synchronization and handling API responses could potentially impact Kodi's performance, particularly if not optimized effectively. Moreover, the reliance on Letterboxd's API introduces a new dependency, making Kodi's functionality subject to network connectivity and external service availability.

One significant risk involves the dependency on Letterboxd's external services. The module's functionality hinges on the API's availability and reliability, making it vulnerable to service outages or changes in API structure. To mitigate this, implementing robust error handling and fallback mechanisms within the API integration is essential. Furthermore, integrating a complex new module like Letterboxd within Kodi's existing architecture could lead to system performance degradation, manifesting as slower response times or increased system load. This necessitates careful optimization of the module to minimize its performance impact, along with comprehensive testing under various scenarios. Additionally, the introduction of new features into the user interface poses the risk of increased cluttering, potentially having an impact on ease of use. This being said, it would be necessary to implement thoughtful UI and UX within the GUI.

Finally, the integration's complexity presents challenges in system maintainability, potentially complicating the already unorganized nature of the concrete architecture. This underscores the need for clear documentation. There is also the risk that user acceptance may be shallow; new features might not align with the preferences or expectations of familiar and frequent users. *(Yes Spotify we are looking at you).*

In summary, while the integration of the Letterboxd Module into Kodi offers significant benefits in terms of personalized user experience, it also introduces various challenges and risks. Properly addressing these through robust design, comprehensive testing, and ongoing user engagement will be crucial to the successful implementation and adoption of this enhancement.

Evaluation of Architectural Options (Omar)

The implementation of this enhancement can be applied in two main ways: using a modular or integral approach.

Option 1: Modular

First and foremost, realizing this enhancement in a modular fashion would involve creating a completely new Letterboxd module and connecting it to the existing architecture.

The new module would communicate with the media handling module, which would then pass all critical information to the core functionality module. Of course, the user interface module would also have to also be considered to handle windowing. This will primarily provide the sign in box, recommendations menu, and movie ratings sheet.

All critical information including the API, recommendation engine, and user history will also be stored in this new module. The user history will be synchronized with Letterboxd via the user history sync module. Subsequently, existing movies in the user's library will be string-matched with the Letterboxd database to provide recommendation and rating information.

This information will be displayed to the screen when any media is selected for playback. The technical implementation of windowing and handling user events will be handled by the existing user interface module.

This implementation would be ideal for developers as it integrates logically with the already-existing infrastructure. It also builds upon Kodi's composability and modularity.

However, the modular approach comes with a few drawbacks for Kodi's userbase. Firstly, the integrability of this new enhancement is inherently insecure. While the ability to add features anywhere is convenient and composable, it also makes it easier for security flaws to be exploited by bad actors. Additionally, the ease of integration might impact performance, as memory is not directly accessed and handled by the system, but instead must go through a proxy component

first. The potential impact on Kodi users remains uncertain, requiring further experimentation and research.

Option 2: Native

Alternatively, directly integrating this feature into the already-existing modules would be another viable approach. The recommendation system would likely become part of the media handling component and have to manage its own user interface.

This would establish the module more organically, as an integral component of Kodi rather than acting as an addon.

The recommendation engine and API would engage directly with the existing library and autonomously handle all required operations without the need of interacting with any proxies. However, this implementation could pose significant challenges for future developers maintaining the codebase. The fundamental integration with Kodi's architecture would introduce complexity. The lack of encapsulation could potentially allow bugs introduces by this feature to affect the broader Kodi system.

Despite these concerns, if executed correctly, this integration has the potential to enhance long-term performance and user experience. It would reduce overhead during module calls and mitigate memory access issues, contributing to improved user security and data protection.

Overall, although the second option's higher potential for usability and performance may be enticing, option 1 is deemed more realistic and balanced. Despite security concerns involved with a modular approach, adherence to best practice methods can mitigate risks.

Additionally, the anticipated performance gain with a native approach is likely to be minimal. Option 2 would only be feasible if an overhaul of the entire codebase were already planned, an improbable scenario.

Ultimately, considering all factors, the first option emerges as the most likely means to realize a recommendation system, as detailed in the preceding sections.

Use Case and Sequence Diagram (Aidan)

- Presentation of at least one use case
- Sequence diagram explaining the flow through the proposed architectural modifications

Conclusion

In conclusion, after deeply analyzing Kodi's system, we have outlined its conceptual and concrete architecture. These analyses have allowed us to identify flaws, possible enhancements and add-ons in the Kodi system. Our proposed integration of Letterboxd into the Kodi media controlling software presents an opportunity to enhance the user experience for film enthusiasts, who are prime users of Kodi. Allowing users to sign into Letterboxd, receive personalized movie recommendations and rate and review films directly within the Kodi interface creates a connection between movie playback and passionate film discussions.

The analysis conducted in this report dives into the intricacies of implementing such an enhancement. We have examined many aspects, from the current software system analysis to the proposed architectural changes. The stakeholder analysis specifically identifies the various interests of end users, open-source developers, and Letterboxd, along with their specific non-functional requirements.

The architectural analysis provides an intricate overview of Kodi's current structure and the proposed changes required to accommodate the Letterboxd module. We discuss the potential impact of our proposed changes, including the risks associated with dependencies on Letterboxd's API and system performance challenges. We recognize that the success of this integration relies on addressing these challenges through careful design, extensive testing and continuous use engagement. Overall, while there are risks associated with the integration of Letterboxd, the benefits of its integration are substantial for an enhanced user experience.

References

- List of all sources cited in the report