CISC 322/326

Assignment 3

Architectural Enhancement of ScummVM and SCI Engine

Monday, December 2, 2024

Group 21: We Love Josh

Nathan Chow 21nwc1@queensu.ca
Ben Jacoby 22bj@queensu.ca
Savannah Han 21sh113@queensu.ca
Amanda Li 22ayl@queensu.ca
Kristen Lee 21kl52@queensu.ca
Cynthia Wang 21cyw4@queensu.ca

Table of Contents

Abstract

In this report, we propose an enhancement to the ScummVM system—namely, we propose implementing the capability to create desktop shortcuts for games. This would let users launch their games in ScummVM directly without opening the launcher. We analyze two approaches to implementing this. The first involves the direct manipulation of executable files, while the second involves utilizing parameters to create shortcuts. To compare these approaches, we perform a SAAM analysis to see their effects on key stakeholders (namely developers and users) and on the NFRs of ScummVM. We decide to move forward with the parameterized approach.

Next, we see the effects of our approach on the conceptual architecture. We are required to add a new dependency from OSystem API onto ScummVM GUI. In addition, we propose a new subsystem of OSystemAPI to help handle the creation of shortcuts—we call it Shortcut Handler. We then analyze the dependencies that Shortcut Handler is required to have.

Then, we take a broader look to see how our feature will affect the key metrics of performance, evolvability, testability, and maintainability. At a low level, we mention what the affected directories and files are. In addition, we find that no concurrency is needed for our implementation. To show our feature at work, we analyze two key use cases—a user adding a desktop shortcut, and a user launching a game with a desktop shortcut. Finally, we analyze the potential risks of our approach, consider how to test our feature, discuss some limitations we found, and reflect on the lessons we learned.

Introduction

Planning ahead is crucial when extending a complex software system. First, one must consider how the implementation will be done. This involves comparing different approaches to see which one is best. One must analyze which stakeholders are affected and in what ways they are affected, along with looking at the NFRs of the system.

With a plan for how to implement the feature, it is then necessary to analyze in further detail what the effects will be. Namely, looking at how the architecture will change is important—new subsystems and new dependencies may need to be added. In addition, it is important to consider what specific files and directories will be affected, and to see how concurrency may play a role. Analyzing common use cases of the feature is helpful to see how the new subsystems and dependencies interact. Analyzing the risks involved with the change, how to test the new feature, and the feature's limitations are all important as well.

Going through this planning process is exactly the goal of our report, for ScummVM and the new feature of desktop shortcuts.

Proposed Enhancement and Motivation

It has become more and more common in recent times for games to be bundled together with some sort of launcher. Sometimes, a user might even use a launcher to open a game only for it to open yet another launcher. Although this is a minor inconvenience, over time this can be a large source of frustration for a user and might even discourage the user from playing the game as the

extra steps in launching it adds friction in the user experience. ScummVM works the same way; in order to launch a game, the user must either open the ScummVM launcher, select a game, then launch the game, or they must open the command line interface and launch a game by using the correct command line arguments [1]. Our group suggests a solution to this problem by adding a new feature to ScummVM: the ability to create desktop shortcuts for individual games in ScummVM to launch them directly without opening the launcher or command line interface first.

The implementation of this feature would be a good improvement to the user experience in ScummVM, creating a more convenient and smooth experience with the software. User experience is extremely important in a piece of software like ScummVM because the core purpose of the software is to allow a user to play games, not navigate through menus. Software used for commercial or internal purposes can be more unfriendly for users in exchange for flexibility and power since they are used in professional settings and prioritize effectiveness over everything else. But for consumer software like ScummVM, user friendliness should be the highest priority. This is why it is important to remove as many small annoyances as possible. Otherwise, these annoyances will build up over time, leading to major user dissatisfaction that might lead a user to stop using the software altogether.

Approach A

Our first approach to implement desktop shortcuts for individual games is to create new, altered executables for each shortcut to a game. After the user selects what game they want to make a desktop shortcut to, the system will get the full game ID for that game and alter the starting code (entry point) to set the default domain to that game. Once the new executable is compiled, a shortcut will be created on the desktop that links to the new altered executable for that game. Since the altered source code already has the desired game set as the active domain, the launcher will not be opened and the game will launch immediately. In order to implement this approach, we would need to introduce a Pipe and Filter style component to our architecture in order to make a compilation pipeline. This is because this approach involves compiling new executables for each shortcut, which would require ScummVM to add a compiler to create these new executables on demand, which it does not currently support.

Approach B

Another approach to implement our new feature is by using parameterized shortcuts. In order to implement this feature, a new subsystem would be added within OSystem API in order to handle the dynamic creation of new parameterized shortcuts whenever a user wants to create a shortcut to launch a different game. This approach leverages the already implemented feature in ScummVM to launch a game without going through the launcher by using the command line interface (CLI) with the game ID. Thus, the parameters for the created shortcuts will be the game ID of the game the user wanted to make a desktop shortcut for, and the shortcut itself will link to the normal ScummVM executable file just like the default launcher. By doing this, opening the shortcut will act as if you executed ScummVM via the CLI with the game ID which bypasses opening the launcher and directly launches a game instead. This approach does not require the introduction of any new architectural styles in the system's architecture, as it only needs to extend the functionality of a few of the existing subsystems to implement the new feature.

SAAM Analysis

The Software Architecture Analysis Method (SAAM) aims to evaluate the impact a newly implemented feature has on the system via comparison. In our SAAM analysis, we are comparing two different implementations of our new feature in terms of major stakeholders we have identified—users and developers—and the non-functional requirements of the system.

To properly analyze the effectiveness of our two proposed implementations, we identified our major stakeholders as the users or players of the ScummVM system, as well as the developers responsible for implementing the change and managing the system.

Stakeholder Analysis

Users

The users are the players that use ScummVM for launching games. For either approach, users may be concerned with non-functional requirements such as performance and usability. Good performance is crucial as poor performance defeats the purpose of adding this shortcut feature for convenience. Users may also be concerned with usability as the feature should be simple to understand and use. Given ScummVM is supported on different platforms, portability may be another concern assuming the user is accessing ScummVM on various platforms such as mobile or desktop. The evolvability of the feature and the shortcuts created in the case of changes or updates to the game being launched is also important for users. Finally, storage efficiency is also a concern as optimally a user would want a single game to take as little space as possible so storage is left for other files.

Developers

The developers are responsible for maintaining the system, which includes ensuring non-functional requirements are still met upon any change. Specifically, developers are concerned with maintainability and testability. Given ScummVM is an open-source project, it is crucial that the maintenance for an new additional feature is not too complex and would not create long-term problems that could convolute or reduce readability of the system for other contributors. For a similar reason, the testability of the feature is also important to ensure it continues to work for any future versions of ScummVM. Reusability may also be a concern as the ability to reuse code for this feature may prove helpful for future development.

Non-functional Requirement (NFR) Analysis

NFR	Approach A: Altered Executables for Shortcuts	Approach B: Parameterized Shortcuts
Performance	**Low:** the user is able to launch games without directly accessing the default launcher. However, given that this approach requires the creation of a new executable file, compiling this new game will take a long time, decreasing performance.	**High:** the user is able to launch games without directly accessing the default launcher, which increases speed and performance.

Usability	**High:** the feature promotes ease of use as the user can simply launch the desired game from their desktop.	**High:** the feature promotes ease of use as the user can simply launch the desired game from their desktop.
Portability	**Low:** the feature is only compatible with desktop platforms.	**Low:** the feature is only compatible with desktop platforms.
Evolvability	**Low:** new altered executables will have to be created upon game updates or changes, reducing convenience.	**High:** given the shortcut accesses the normal executables, the shortcut will still work with game updates or changes.
Storage Efficiency	**Low:** the new executable file created will use more storage space, as essentially a whole new game is created in addition to the original one.	**High:** this approach will not use additional storage space as it is a simple script.
Reusability	**Low:** it is very ineffective as a new game is created with only a few lines changed.	**High:** this approach promotes reusability as it is extending an implemented function.
Maintainability	**Low:** compilers are difficult to maintain, making overall maintenance more complex.	**Medium:** the amount of maintenance will inevitably increase, but not substantially.
Testability	**Low:** this approach does not promote testability as compilers are difficult to test.	**High:** the feature is relatively simple and therefore easy to test.

Based on the above analyses of our two suggested approaches, our team concluded that Approach B is the best implementation of this feature, as it is better than Approach A in every aspect except for usability and portability. Though, even for these two non-functional requirement cases, the two approaches are tied.

In terms of the users, Approach B is better. Most importantly, Approach B will have better performance than Approach A, as compiling may take up a significant amount of time—slower speed completely defeats the purpose of user convenience for this feature. Approach B is also more evolvable to changes or updates to the original game being launched, as this implementation links the shortcut to the original executables and encompasses any changes to them. In comparison, Approach A will require newly altered executables that encompasses these new changes to be created for the shortcut, which makes it far less efficient and convenient. The creation of a new executable file of the game for Approach A would also use more storage space than Approach B's use of a script.

In terms of the developers, Approach B is also superior as it promotes usability, maintainability, and testability given its simplistic implementation and the fact it extends on an existing function in ScummVM. In comparison, the use of a compiler for Approach A complicates all three of these non-functional requirements.

Effects of the Enhancement

Current State of the System

The current conceptual architecture for ScummVM retains the same architectural style and subsystems as we defined in A2 and A1. The system will remain a hybrid layered and interpreter architectural style with the same layers and groups as our proposed conceptual architecture, with SCI VM remaining as the interpreter. Our proposed feature will not be implemented using a new subsystem; rather, it will be implemented within OSystem API, so our subsystems remain unchanged in our conceptual architecture. However, our enhancement requires one new dependency, which is highlighted in red in the figure below.

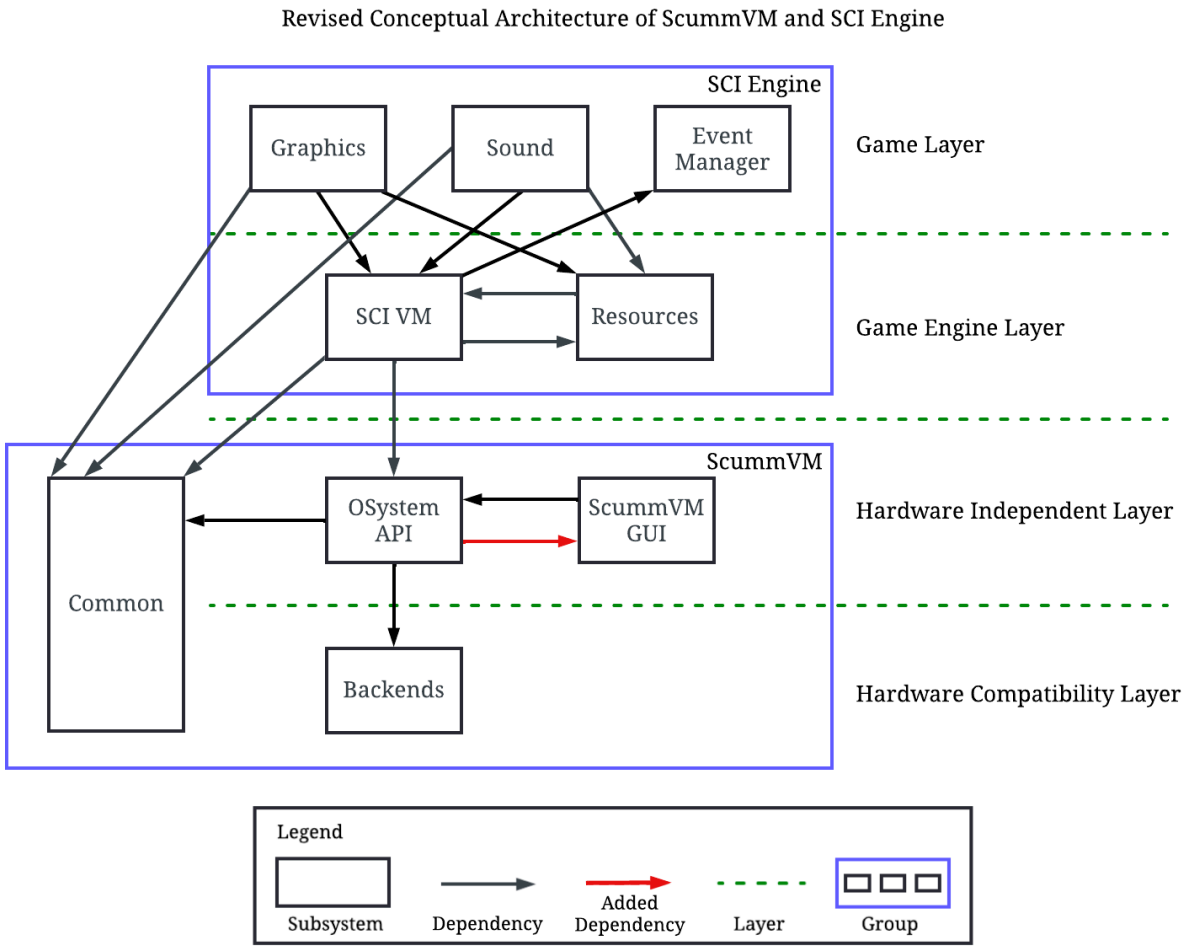

Figure 1: Revised Conceptual Architecture of ScummVM and SCI Engine

Our conceptual architecture will be modified to add a new dependency so that our new feature can be implemented. In particular, we are adding the **OSystem API → ScummVM GUI** dependency so that OSystem API, which we are using to launch games from desktop shortcuts, can use ScummVM GUI, which is where the original game launcher is defined. Using ScummVM GUI, our subsystem can detect installed games and fetch their game ID to input into the command line interface.

Effect On the High and Low Level Architecture

The high-level architecture of ScummVM and SCI Engine will be slightly altered to add a dependency from OSystem API to ScummVM GUI to implement our feature using Approach B.

The low-level architecture of OSystem API will be altered to add a subsystem to handle parameterized shortcut creation. We will call this subsystem Shortcut Handler. This subsystem would utilize OSystem API's command line interface to launch a game directly from a desktop shortcut.

Interactions With Other Features and Subsystems

Shortcut Handler, which is defined as a subsystem within OSystem API, would use other subsystems within OSystem API, Backends, Common, and ScummVM GUI to function.

The command line interface as well as all the commands a user is able to use is defined in OSystem API. Thus, Shortcut Handler would use the CLI to directly launch a game using a desktop shortcut. As outlined in Approach B, this is because we want the launch of a desktop shortcut to act as if the user opened the game using the CLI and the game ID. Additionally, since OSystem API contains the main loop of the program in a subsystem called Base, Shortcut Handler would depend on Base to ensure the game starts and runs properly.

In order to implement our feature for multiple desktop platforms such as Windows or Mac, Shortcut Handler would depend on Backends. Since Backends implements OSystem API for various platforms, the interaction between Shortcut Handler and Backends would allow our feature to be implemented for different computer platforms. Backends would also be used to make a system call to create a shortcut with the game ID as the parameter.

Since Common contains utility classes used by OSystem API, Shortcut Handler would need to depend on Common as well. In particular, our feature would use the text console, tokenizer, and String classes defined in Common in order to use the CLI to launch a game.

Finally, our feature depends on ScummVM GUI. As mentioned in previous reports, ScummVM GUI contains the game launcher, options dialogue, and global main menu. Although we will be bypassing the game launcher, we will still need to use the global main menu, since the main menu should be accessible to the user even while they are playing a game. ScummVM is also able to detect the user's installed games, which Shortcut Handler needs to function.

Performance, Evolvability, Testability, and Maintainability

Performance: The overall performance of the system will increase, since we are bypassing a step when launching a game. Normally, before a game can actually be launched, the user would need to either open the launcher and select their game or open the command line interface and input the appropriate command. However, our new feature allows a game to be launched by simply clicking on the desktop shortcut, which would increase performance since the launcher or CLI would not need to be opened.

Evolvability: Our feature can easily adapt to new game changes, because if a game is changed then our desktop shortcut would simply launch the new version of the game instead. Our feature would also be able to adapt to system changes if those changes are made for computer platforms. If the changes are made for platforms like mobile or console, our feature will not be able to adapt because it is only implemented for desktop.

Testability: The amount of testing will increase slightly in order to make sure the desktop shortcuts launch the games properly. Even so, our feature is relatively simple and would therefore be easy to test.

Maintainability: Our feature will increase maintenance because new code is being added. However, the increase will not be too significant because we are only adding one subsystem in OSystem API and one dependency.

Impacted Directories and Files

The directory **scummvm/base** would be modified to add our new subsystem. As well, the files **scummvm/base/commandLine.h** and **scummvm/base/commandLine.cpp** would be modified to accommodate our new feature [2].

Concurrency

No new concurrency will be added to the architecture because synchronization is not necessary for adding or launching games using desktop shortcuts.

Use Cases

ScummVM and its available game engines are most commonly used to play text-based or point-and-click graphic adventure games. Thus, we propose two use cases demonstrating desktop shortcut functionality within ScummVM.

Use Case #1: Adding a desktop shortcut

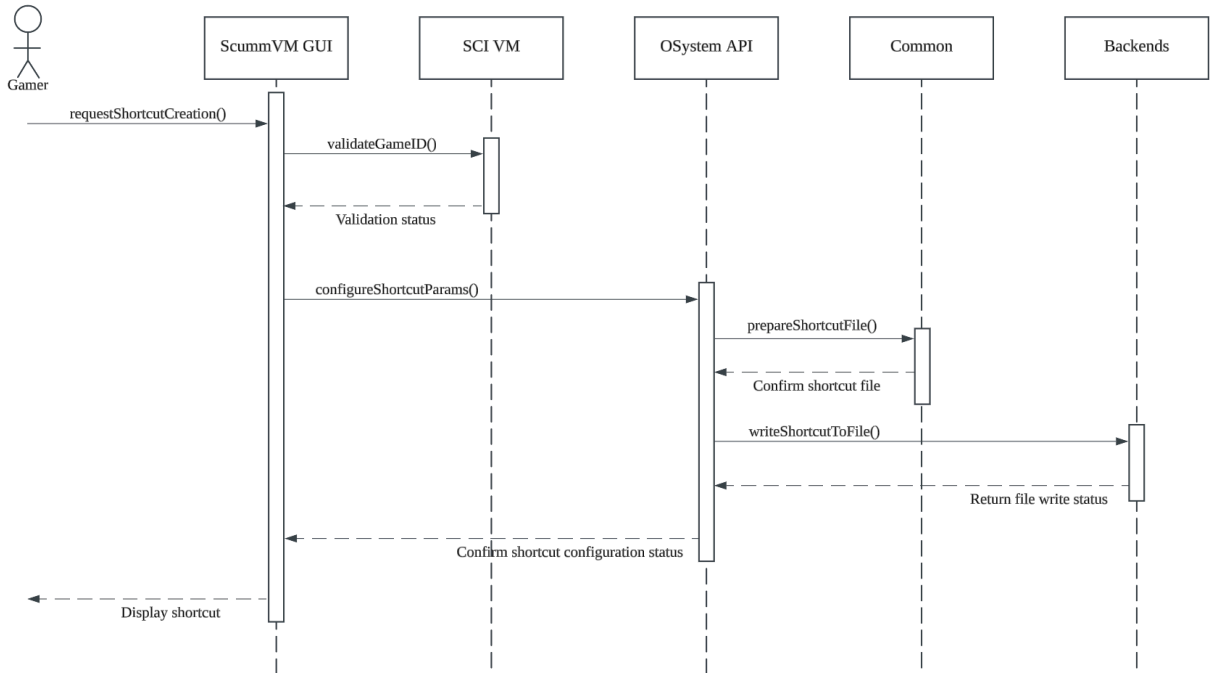

Figure 2: Sequence diagram for use case of adding a desktop shortcut

This use case involves adding a desktop shortcut for a game in ScummVM, allowing the user to launch their game directly from the desktop without opening the ScummVM launcher. This feature enhances usability and accessibility by providing quick access to individual games.

The process begins when the user selects the "Create Desktop Shortcut" option in the ScummVM GUI. The GUI initiates the process by calling requestShortcutCreation() and passes the game ID to the SCI VM for validation. The SCI VM verifies the game ID using validateGameID() and ensures it corresponds to a game in the user's library. Upon successful validation, the ScummVM GUI calls configureShortcutParams() through the OSystem API, directing the Shortcut Handler subsystem to prepare metadata for the shortcut, including paths and icons. Shortcut Handler then calls prepareShortcutFile() in the Common subsystem to structure the shortcut data and sends it back for further processing. The prepared data is forwarded to the Backends subsystem through the OSystem API using writeShortcutToFile(), which interacts with the user's OS to create the shortcut file on the desktop. Once the shortcut is successfully created, the Shortcut Handler confirms the result to the ScummVM GUI, which displays the newly created shortcut to the user.

Use Case #2: Launching a game from a desktop shortcut

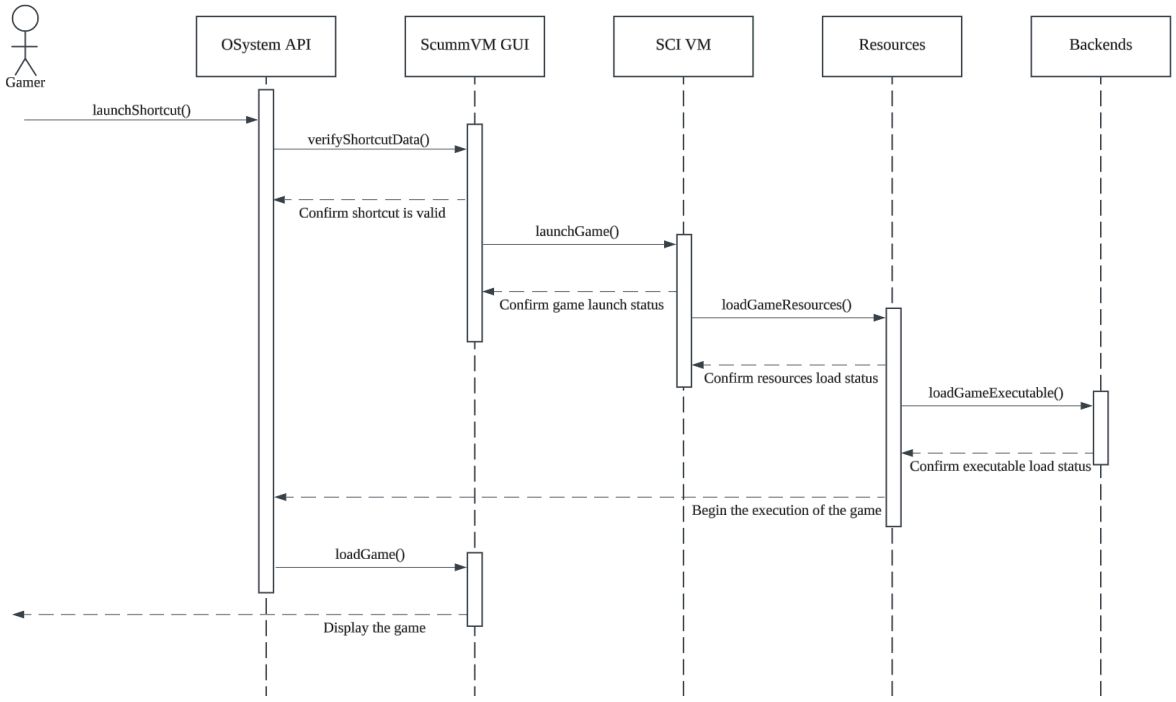

Figure 3: Sequence diagram for use case of launching a game from a desktop shortcut

This use case involves launching a game directly from a desktop shortcut, bypassing the ScummVM launcher. This feature streamlines gameplay by allowing users to start their selected game with a single action.

The process begins when the user double-clicks a desktop shortcut. This action triggers the Shortcut Handler within the OSystem API to call verifyShortcutData(), ensuring the shortcut points to a valid game. The Shortcut Handler then calls initialize() to set up the ScummVM environment. Once initialized, the Shortcut Handler notifies the ScummVM GUI to call launchGame(), passing the game ID to the SCI VM. The SCI VM then invokes loadGameResources() in the Resources subsystem to retrieve all necessary assets for the game. Resources further interacts with Backends by calling loadGameExecutable(), which handles platform-specific operations for loading the game. After successful loading, the game launches, and the user is seamlessly taken into gameplay.

Potential Risks

While the implementation of the Shortcut Handler within OSystem API should be relatively simple, we have identified some potential risks with its addition.

Maintainability

Potentially, there is a case where ScummVM may change the required parameters from the game ID for its command line interface (CLI) to the older version, causing the created desktop

shortcuts to fail to work. In this case, more maintenance may be required to account for changes to the required CLI parameters.

Security

Bypassing ScummVM's built-in launcher and allowing the desktop shortcut to directly link into the original executables may pose some security risks, assuming that ScummVM's default launcher may have included some security checks to prevent malicious command injection into the CLI or unintended changes to the original executables that Shortcut Handler may not have.

Testing

Testing is important to ensure the newly introduced interactions work between our proposed feature, Shortcut Handler, and other features already present in the system. This testing can be done via the white-box testing methods of integration testing and unit testing. Integration testing requires the observation and testing of interactions between multiple modules, while unit testing involves testing each method present in the system individually. Given the implementation of Shortcut Handler introduces a new dependency between the OSystem API and ScummVM GUI, we can observe how this dependency impacts these two subsystems via integration testing. Unit testing can be done directly for our proposed feature to ensure it correctly fetches the game ID and is able to access the CLI as intended.

Lessons Learned and Limitations

Since we created this idea ourselves, one of the challenges we encountered was the ambiguity around its successful implementation. Nevertheless, this assignment provided a valuable opportunity for us to put our knowledge of software architecture into practice by applying it to a real-world software project.

Since this was the first time we used SAAM, we learned how added features can impact the stakeholders in a project. Before discovering SAAM, we thought that just the software would be impacted by adding a new feature, but in practice, other aspects like evolvability are also impacted.

Another lesson we learned is to ensure everyone understands the details of how the new feature is implemented, since this time, there was slightly more confusion and a need for clarification.

Conclusion

In this report, we explored how to implement desktop shortcuts for games in ScummVM. First, we compared two approaches for how to implement the feature—namely one involving modified executable files, and one involving parameters. We did this with a SAAM analysis, where we saw how each change would impact key shareholders and the NFRs of the system. In the end, we decided to use parameterized shortcuts.

Second, we saw how the conceptual architecture of ScummVM would need to change to accommodate desktop shortcuts. We found that OSystemAPI would have to depend on ScummVM GUI, which was a dependency that did not originally exist in our conceptual

architecture of the system. We also proposed adding a new low-level subsystem called Shortcut Handler within OSystemAPI, and explored what dependencies it would have.

Third, we saw how the performance, evolvability, testability, and maintainability of ScummVM may change. We explored what specific directories and files would be impacted, and saw that the concurrency of the system does not change. From there, we analyzed two important use cases of the feature—namely, adding a shortcut and using a shortcut.

Finally, we saw what the possible risks of the change were, specifically in regards to maintainability and security. We ended off by exploring some of the limitations of our approach, and discussing the lessons we learned in our work.

Data Dictionary

Conceptual Architecture: A theoretical high-level overview of the system's structure, illustrating how various components and modules are organized and interact with one another, without detailing the actual implementation.
Dependencies: A relation between components or resources, where one component uses functions or resources from the other.
Interpreter architecture style: A software design pattern where code is interpreted and executed directly, line by line, at runtime, allowing for flexible updates and modifications without the need for recompilation.
Layered architecture style: A structural framework that organizes system components into distinct layers, where each layer has specific responsibilities and interacts with adjacent layers, enhancing modularity and ease of maintenance.
Open-source: Software that is made publicly available for anyone to use, modify, and distribute, typically fostering collaboration and community-driven development.
Subsystem: An independent component within the overall architecture that encapsulates specific functionalities, enabling collaboration with other subsystems to achieve the system's goals.

Naming Conventions

API: Application Programming Interface
CLI: Command Line Interface
GUI: Graphical User Interface
NFR: Non-Functional Requirement
SAAM: Software Architecture Analysis Method
ScummVM: Script Creation Utility for Maniac Mansion Virtual Machine
SCI: Sierra Creative Interpreter
VM: Virtual Machine

References

[1] "Command line interface — ScummVM Documentation documentation," *Scummvm.org*, 2020. https://docs.scummvm.org/en/v2.8.0/advanced_topics/command_line.html (accessed Nov. 30, 2024).
[2] scummvm, "GitHub - scummvm/scummvm: ScummVM main repository," *GitHub*, Mar. 31, 2024. https://github.com/scummvm/scummvm (accessed Nov. 17, 2024).