Project Postmortem Report

Objected-Oriented Design and Analysis - COMPSCI 3307A

December 9th, 2020

Group 18

Group Members:

Hunter Joseph

Brendan Albo

Seyedehnastaran Ghafouriansahraei

Caleb Borwik

Ethan Biswurm

Project Summary

The goal of our project was to create a framework for a smart mirror using C++. The idea was to create a two-way mirror with an inbuilt display behind the glass, so useful information one would like to glance at when getting ready in the morning would be readily available. This display will show various features powered by a raspberry pi to make life more organized. The Mirror Pi will provide widgets and information commonly displayed on devices without the need to check them. This would have allowed users to multitask while also having essential information such as time, weather and calendar events. The software we made runs through a raspberry pi that populates the user's mirror with useful information presented in a heads-up display like manner. The features we incorporated include calendar, greeting message, weather, time, moon phase, and newsfeed. The widgets greeted the user with a welcome message as they started the application. They allowed users to monitor their upcoming calendar events on different time frames. They also allowed for weather updates with various details and metrics. In addition, a newsfeed that frequently updates to keep the user abreast of the world. Lastly, our program will greet the user upon startup for a more engaging and interactive medium. All the while, each widget can be adjusted based on the user's location.

Key Accomplishments

What went right?

In the development of our smart mirror modules, we chose to use QtCreator as our development platform so that we could have additional tools at our disposal when creating portions of our User Interface (UI). Using QtCreator then also allowed us to separate each of our modules into multiple separate windows that each could be repositioned independently. Many of the widgets we intended to develop for our product from the outset of this project were complex or required external data to be gathered and/or measured. Thankfully for our project, there existed publicly available Application Programming Interfaces (API) online that suited our needs and allowed our modules to function as intended. With these obstacles sorted out, we were able to incorporate several features appropriately. These included a weather forecast widget that displayed live weather updates, a lunar cycle widget that presented each night's lunar phase, a news widget, etc. As such, the incorporation of several of our intended features went as planned.

What worked well?

Thanks to our commitment to QtCreator as our platform, we were able to enjoy some additional perks that benefited our development. Since QtCreator had each of our modules running independently in their own windows, we were able to complete another of our user stories, that of incorporating a customizable layout of each of our smart mirror widgets. In addition, our choice to use the Accuweather API for 3 of our modules made it easier to pull the information we needed from the API as each packet was stored in a similar manner.

What was found to be particularly useful?

One of the most useful tools throughout the development process was QtCreator. This tool provided a simple IDE with many examples on how to get started working with GUI's, as before this project, we had not learned how to create and/or work with GUI's. Because of the use of QtCreator, we were able to program the GUI's styling in an external qml file. This allowed for better organization, reduced file length and simplified the design process. Another incredibly useful new tool was API calls, again a new topic for us, but it provided all of the necessary information to populate the GUI and get relevant information. The use of API calls allowed us to get real-time data and pull it straight from a trusted source, as opposed to web scraping. Above all, the most useful thing in the development project was weekly meetings and check-ins. These allowed us to discuss problems we may be facing on our designated module and get creative solutions from the team. As well, these kept us accountable and made sure that what needed to get done was.

What decisions contributed to the success of the project?

A primary reason for our success was using Jira to organize all of our user stories and to house all of our code in one place for any team member to grab and use. When starting the project, it was easy to divide our user stories between partners, which kept us accountable and on track when doing the project. By using Jira, when the initial planning started, we knew approximately how long each module would take based on our user storytime, which allowed us to take the appropriate amount of time before the due date to have everything ready.

Jira was very helpful in designing our project's organization. Still, the use of a Facebook group chat helped us communicate, also majorly contributing to our project's success. The group chat allowed us to keep in constant contact whenever a question arose regarding the project of overall technical difficulties with any of the software needed. We kept in contact with scheduled weekly group calls to get a more personal understanding of where everyone is at.

Code wise, our use of API calls and the use of QTcreator played a significant part in the success of our code design-wise. API calls allowed us to use trusted sources for our location easily, weather and news services rather than us building it all from the ground up ourselves. QTcreator also was a simple, clean service that allowed us to intuitively create each module window with much more each than if we tried to code it all with just a simple C++ IDE.

Key Problem Areas

What went wrong?

Throughout the development period, we ran into many issues, some we were able to overcome, some we were not; the main problems that we were not able to solve were; Compatibility with IP-API was simply impossible due to QtCreator not working with curl. This meant that we were not able to find the user's location automatically through a GPS check of their IP. In the end, we had the user enter in a text field and select their location from a list of returned cities retrieved by an API request. Our profile class lost significant functionality and purpose due to the extremely independent nature of our code. If we were working with a physical device, the code would have all started with the profile class housing instances of the modules with common elements shared between them. This would have allowed the user to have a more personalized device and closer to what our original vision was. The music class ended up becoming so problematic we were unable to incorporate it as a feature. We attempted to have the current music playing from either Spotify or apple music displayed. Apple music simply does not allow for this as they are very restrictive on the use of their software. Spotify, on the other hand, has an API specifically for this, but in order to use it, a user must first get a key from a website externally. This process was unable to be automated as it has captcha checks, and therefore the user would have to retrieve it themselves. In the end, we decided that this was not an achievable feature, primarily through testing and changing the user.

What project processes didn't work well?

As a team, we discovered that if we had had a physical object to code each Widget on, they would have been far more integrated. Rather than having exclusive windows, this common physical space would have produced a far more polished final product with a singular common window for the GUIs.

A process that we experienced a lot of difficulty with was the issues related to syncing with Google Calendar. While possible to complete in theory, utilizing C++ to sync with Google Calendar for the implementation of our Calendar Widget is extremely difficult to accomplish. Google publishes a library for its current API version in Java, leaving the API libraries for other languages out of date. There were possible workarounds that we came across in our research; however, given the time constraints associated with the project, it was not feasible to move forward with this wishlist feature.

Another process that presented significant challenges was the Location Widget. Similar to some of the other Widgets, we were able to use the Accuweather API to allow the user to set their location. However, getting the API to work correctly proved to be more challenging than initially anticipated. Multiple members of our team worked on this particular class to get it working, and while we were able to get the class working, it was not to the level that we were hoping to accomplish. The initial plan was to select the correct city location from a produced list using GPS coordinates; our final class required the user to choose the correct location manually.

What technical challenges did you encounter?

Due to the circumstances of our coding environment, having to be simulated and entirely virtual, our group ran into several issues with being able to prototype and test our product throughout development. One such issue was that the virtual environment that simulated a Raspberry Pi would often mean we would have to use inconvenient intermediaries during development. It would be transferring files between our primary machine and the virtual environment or moving the same domain to another machine. In addition, the poor processing power of this virtual environment often made it difficult to program or debug our project during development due to the small display window it showed and the inability to run a web browser for research alongside other programs. Being required to use a virtual box also leads to many issues such as the program crashing as well as a wrong internal clock within the virtual environment, leading to frequent problems with compiling our code.

When development would progress without the obstacles mentioned so far, we encountered challenges in applying our existing knowledge of C programming with the QtCreator IDE conventions. Due to our lack of prior knowledge or experience with Qt, we would often need extensive research to solve issues of incorporating C libraries, establishing network connections, and class cross dependency. Sometimes this research would examine other options and lead to us leaving issues like cURL implementation and cross dependent widgets unresolved.

Finally, the additional tools we were required to use for project tracking and version control provided some unexpected obstacles. Jira would sometimes not allow for us to delete accidentally created user stories and also did not provide any fields for input of story length. Bitbucket on the other hand was unreliable as it's usage had led to object corruption on occasion.

What design decisions made it more difficult to succeed in your project?

One of the biggest obstacles we faced as a group was how we designed each class independent of each other. During these times of COVID, it forced us to have to do work separately and on our own time schedules which did not allow us to wait for a group member to finish their class on a single document before the next member started theirs.

This isolation from the group resulted in the benefit of code independence , as in if one module of the smart mirror stopped working, the others would still continue to work. This independence came at a cost of difficulty in coordinating our project as one whole cohesive unit. By having individual modules, it resulted in our qt applications being unable to combine. Which, without the use of multithreading, made it hard to link each module to another.

A decision which also made it difficult to succeed, was our choice of incorporating certain widgets into our project. We were unaware of the difficulty of incorporating certain aspects such as displaying music, and linking with your google account to sync your calendar. By choosing these without prior knowledge of C++'s capabilities, it resulted in lots of headaches and frustration as C++ is not well suited for it.

What were the effects/impact of these problem areas?

One of the most problematic results of our project was the lack of a physical device to work on. The main impact this had on the code was then individual modules that run independently of the others. Resulting in each module needing to be run from its own terminal call and have individual GUI's for each module than can be dragged and placed at will. Rather than a single GUI with each module appearing as a element on the GUI with a position that can be changed from the profile class. Due to the lack of a physical device our code was developed very independently which resulted in a major impact of the final product.

What corrective actions did you take to resolve the problems?

User stories were written in a way that allowed each story to be assigned to a member and be implemented independently. Therefore, we implemented each module such that they utilize minimal resources and that each module can run in parallel. We decided this is the most efficient solution since the project had to be done virtually and group members could not meet and build a physical smart mirror using a raspberry pi.

Effective communication and keeping group members accountable was a major challenge since every aspect of the virtual nature of this project. To tackle this problem, we set up weekly group meetings where members updated each other on the progress they made with their assigned modules. The meetings also gave members a chance to ask questions, have discussions about different possible solutions and overcome the technical challenges of coding the modules. Instant messaging apps such as Facebook messaging and Discord were used for code and screen sharing to easily discuss and resolve the issues. Moreover, utilizing Jira and updating the progress of each user story helped us coordinate our team efforts more effectively.

Lessons Learned

We learned that communication and knowledge sharing leads to more effective teamwork and better project outcome. While each module was assigned to individual members, it was still important to have discussions on the different problems faced and come up with solutions together as a group. The key to successful project outcome is effective communication and teamwork.

After reflection, things that we would do differently involve the use of the c++ language, as our project would have been better suited with Java or Python. We did have aspects that worked very well, such as the location searches, lunar phases, clock, and news all using API calls effortlessly. If we had designed our project beforehand using widgets that only involved these API calls, rather than trying to integrate google, it would have gone smoother.

Throughout the development of this project many of us were exposed to many new sides and facets of programming software from the ground up. As such we developed processes to handle these new circumstances. One such process was how to perform API calls within QtCreator/C in a manner that would minimize network usage and allow for the information pulled to be formatted correctly. This involved establishing manager methods that would handle pinging the API host and output the resultant data file, often in JSON format, and then parsing that data file for the data relevant to our module. In

using QtCreator a lot of our work was also focused on the GUI of our product. As each module had its own window as a base for development, our chosen best practice for module creation was to prioritize visibility of changing information and minimize window size to more efficiently use display space.

Particular areas that could see improvement in our project in its current state involve the code and visual presentation. The code in its current state runs completely independently of the other code, a massive improvement would be to change this. Ideally we could have written the code onto a physical device, that is passed from member to member. This would mean that the modules are written in conjunction with one another and work together, which would drastically improve both the code and the connectivity between classes. One of the main advantages that this would provide would be the qml GUI would be able to house every widget rather than each module requiring its own terminal call and terminal window.

In addition another major improvement would be the use of the profile class. As it stands it does not serve as much purpose as we would like. Ideally it would have its own GUI where a user inputs their desired modules and preferences of those modules. With the main GUI housing all modules selected could appear from. This would make the code significantly more connected and allow preferences of modules to be changed substantially easier. This class would have stored all the attributes and objects of each module type inorder to make a profile unique and customizable.

Implementing another class specifically for our API calls would be another improvement we would have liked to add. This would have reduced the number of API calls in our classes as well as guarantee that the information displayed is from the exact same timeframe and source. It would also reduce the number of calls we make which, as our API restricts the number of calls we can make to it, would be a large improvement.

Overall, our team found this to be a refreshing experience and the end result was quite enjoyable. However, the steep learning curve that presented itself with the use of GUIs and API calls made the project very intimidating, especially in the early stages. With the knowledge we have accumulated, we believe those early stages would be far more manageable which would enable us to push our creativity to another level. As a team, we have a collective interest in developing a similar project in the future. Given more time and the increased knowledge we now hold regarding the fundamental concepts, we would be able to improve the Widgets that we were able to implement and would embrace the opportunity to add a number of more unique Widgets as well.