Project Postmortem Report

Computer Science 3307A: Object-Oriented Design and Analysis

Group 52 ("G52")

Gerrid La Sala
John Buckle (251039665)
Patrick Carter (250832347)
Puneet Shokar (250964258)
Robert Foster (251042289)

8 December 2020

Table of Contents

Project Summary

G52 is a scheduling assistant with a focus on User-Defined Configurations. Users will be able to create their own unique calendars, To-Do Lists, Notifications and generate Personalized Agendas that incorporate highly customizable settings to suit individual need. Central to the G52 experience is “Activities” Activities will encompass everything from discrete events like meetings to tasks such as Assignments. Activities will be highly customizable. G52 will also implement a Notification System, which will customize both the Delivery Method and Persistence Level. Different Delivery Methods will include Email, Text Message Phone Call. The Persistence Level will allow G52 will push Notifications via different Delivery Methods at varying frequencies depending on User preference. For example, a Low Persistence Notification would send a Notification once and require no further action, but a Critical Notification will persist until the User Acknowledges it. Users will have the ability to create “Buckets” for different Activities. For example, a student might set their Buckets as “Assignments”, “Group Meetings”, “Exams” and “Quizzes”. An Office Manager may have Buckets such as “Meetings”, “Presentations”, “Lunches”, and “Deadlines”. Users will have the ability to assign each Bucket a default Priority Level. Setting the Priority Level means that when a user has created their schedule, G52 will be able to auto-generate personalized To-Do Lists based on their Activities by using the Priority System. The Priority System lets a User create customized To-Do Lists based on filters such as having a list that only includes Meetings and Deadlines but ignores any Lunches. The final goal is that Users will be able to have a unique Personal Assistant that can organize all of their Activities by using the simple customizations that G52 offers. This program will be written in C++.

Key Accomplishments

What went right?

Throughout the semester, our group, G52, worked meticulously and formed a great bond. Our key accomplishments from our endeavours include great organization and regular group communication, including weekly meetings. This helped by building confidence throughout the team, providing a motivating environment to help us accomplish tasks. Our team structure and organization also proved beneficial with the delegation of tasks as all members were informed and ambitious. Additionally, the course tools, such as BitBucket, Trello, and Jira boards, helped coordinate our group efforts and regularly shared updates. This was significantly more usefully coupled with our software's modular nature as we were able to easily delegate specific tasks(classes) to individuals while keeping everybody in the group updated on what's currently being worked on..

What worked well?

Communication was vital to our success as a group throughout this term. Zoom was a good start for the group as we used it to schedule weekly meetings. This kept us organized and on track at the beginning of the term, but as the pace started to pick up, we found other software more friendly for our use case. The use of discord and Facebook messenger kept the group in regular contact and allowed us to stay up to date during vital work periods. During the later stages of development, we would also use discord voice channels to hold “work periods” to communicate whilst developing.

What was found to be particularly useful?

Like with any team, organization is essential for success. The G52 team acknowledges the importance of Bitbucket, Jira, and the Trello Board -- these tools allowed everyone in the team to stay on track and up to date with ongoing tasks and updates.

Despite Jira and the Trello Board being useful for task assignment and checking task status, Bitbucket was the most useful tool. Moreover, given the nature of Bitbucket, we took advantage of the branching, committing, pushing, and master functionalities to enhance the efficiency and toggling of multi-member pushes. This allowed the team to review code pushes, and to ensure that these pushes did not have errors. Additionally, the feature of Bitbucket which kept track of all commits allowed team members to check prior commits to see what was changed or added if necessary (this was something that we leveraged tremendously to fix errors when attempting to debug).

What design decisions contributed to the success of the project?

There were two (2) main factors related to the design decisions that contributed to the success of the project, namely, (1) the modular design of the program and (2) the detailed comments throughout all the classes.

The modular design enabled the team to "partition" the program to test certain functionality at any given point. This enabled us to ensure that certain components of the program we're working on before implementing another feature. Looking back, despite leveraging this method of feature testing, if we were to work together again in the future, we would do this more often and test certain implementations more rigorously to ensure "bullet-proof" feature implementation.

Having comments throughout the program made the lives of everyone on the team much easier. If person1 added an object (.h and .cpp files -- fully functional) that was critical for person2's tasks, if person2 was having an issue, person2 would be able to check the comments of person1's implemented object class to ensure that they were manipulating the correct object and data.

Key Problem Areas

Things can't go smoothly all the time, and every road has a pothole or two along the way. In this section, you discuss some of the issues encountered in working on your project. In particular, you should address the following questions:

What went wrong?

Our project's modularity initially seemed like a great idea as it allows us to swap in and out parts as we please. However, the result was much code interconnected with each other that made debugging very difficult. Many of the classes were dependent on each other, so debugging them individually was not easy. Trying to run the entire project at once led to more problems as now you have to deal with the errors of 20+ files of code, and it became difficult to see exactly where major errors were happening.

Another issue was how certain features were prioritized over others. Our initial scope of the application was pretty wide and had much functionality; however, what that ended up causing was wishlist or otherwise non-essential features were being worked on and pushed aspects that were required/essential to our application to the back burner. This led to not enough time to complete and debug the core backbone of the application, and as a result, our overall productivity was not as efficient as it could have been.

What project processes didn't work well?

Jira wasn’t as useful in the later stages of the project as many aspects of our project changed as we went on. The main issue was that the board was created when we were still in the design stage and wasn’t actively updated as we changed how certain features would be implemented. It could be useful in a work setting with a project manager, but it was much easier to keep track of what we needed to do/work in group meetings in this project.

What technical challenges did you encounter?

One of the biggest problems of trying to complete a project this big was the lack of continuity of devices between all our members, resulting in code that would otherwise compile on one machine and cause an error on another. Virtualization is the easy solution to this problem, but especially during crunch time, it is much preferred to work on a machine you’re familiar with, so making sure everyone is compiling on the same platform is not always guaranteed. A better way to handle this would be to ensure that everyone has the same virtual machine setup before starting coding to prevent hardware compatibility issues.

Another issue was with the compatibility with many external API's. Our project initially planned to integrate with many other existing services, which sounds great on paper, as it allows practically any user to get functionality out of this application. However, many of these API's did not play well with C++, which is the language of this application. For example, Google no longer provided a C++ API, so any integration of Google accounts would require a lot more work than initially anticipated. Twitter also does not provide a native C++ API, but they provide support for cURL, which allowed us to work around that. However, their API is currently in the process of an update, and we were only able to get access to the beta version of the new API, which does not have support for the features we need for our application. Instagram does not have a C++ or cURL API, so any attempt to integrate with Instagram would have to be done purely through HTTP requests, which again created problems in terms of handling authentication.

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

The high level of modularity allowed us to simplify our project, but it also led to many complications because of the high degree of dependability between different classes. Debugging especially became difficult, especially with different people working on classes that depend on each other, as one change could break compatibility with the entire project. The modularity also set up roadblocks in working on other aspects of the project, as pretty much every part required everything else also to be working to successfully debug and figure out where the errors are coming from.

As a whole, the total scope of our project might have been an overreach. During initial brainstorming, we were creating all these different features we wished to implement and led to an overall project that became difficult to completely grasp because of all the different moving parts.

What were the effects/impact of these problem areas?

These problems created a large bottleneck in the development of the program. High coupling meant that any single bug was difficult to trace down. For example, one class error would cascade through any other class that called it, each giving their own error. Trying to trace down which class originated what bug was very time consuming because even if one class were fixed, new errors would show up. Because this was a group effort, these bottlenecks made it difficult to work separately. This was because, to fix anyone's bug, it meant that multiple classes also needed to be repaired. Having multiple people repair multiple classes separately resulted in in-group members uploading vastly different versions of classes to Bitbucket.

What corrective actions did you take to resolve the problems?

The main corrective action that was taken to correct these problems was tediously debugging each file while coordinating. The Group communicated using discord and worked together to solve as many of the bugs as they could. Patrick oversaw all Bitbucket mergers to master; this way, a single person knew what was happening with every merger so that the master was not being corrupted with multiple disjointed updates. At times the group would collaborate by sharing one person's screen while the rest of the group worked together to get things working. Other times, certain parts of the code were changed entirely to make it work.

Lessons Learned

The key lessons that we learned were mostly related to time management, project scope, and collaboration. While we established excellent communication within our group, we struggled with writing the code collaboratively. Because all of our schedules were so different, it was hard for many of us to be working on the code simultaneously. This caused large problems (especially towards the end) with pull requests having conflicts on Bitbucket. In addition to this, we were too eager to merge in new code that caused both compile and run-time errors, which held us back from debugging other parts of the program. In terms of design decisions, we overestimated how much of the code was required to perform basic tasks. Because of this, we spent too much time developing features that ultimately weren't implemented due to rushing to finish core features. Regarding time management, we left the bulk of the project until too close to the due date to give us enough time to implement the above features and fully debug the program. The project was an excellent experience in learning how to collaborate with others with their own unique coding styles and problem-solving processes. We learned that the flow of code needs to be planned out to prevent bottlenecking. Although our additional features were very modular, the core functions were coupled closely together. Because all of us were working on the core code, the bottleneck was amplified greatly. As we now have a better understanding of design patterns, we would ensure that our project was less coupled. This could be achieved by taking a closer look at the UML Diagram and deciding what really needed to be coupled. Lastly, we would be more careful about starting from a small section of running code and adding to it as needed rather than developing so much on our own and finding it didn't work when we put it all together, making debugging next to impossible.

In conclusion, we all really enjoyed the project, the ability to learn C++ interactively, and especially the opportunity to work together. We would all like to work on another project like this in the future, and in fact, many of us plan to work as a group in 4483B.