inVøke

Project Plan

James Kang, Owen McEvoy, Mason Sayyadi, Sam Lundquist, Zachary Hoffman

Table of Contents

Management Plan

To evenly distribute the workload, our group made the decision to split up into two teams. The first team, the back-end team, consisted of Zach, James, and Owen. The other team, the front-end team, consisted of Mason and Sam.

For the back-end, different modules were assigned to each programmer so that they may be developed in parallel. Owen worked on modules relating to file operations, such as importing and exporting rosters. Zach worked on the modules relating to the data structures and their functionality, as well as creating complex tests for the system. James worked on maintaining the randomness of the queue while also developing the command line interface.

On the front-end team, Mason was responsible for designing key aspects of the User Interface and writing wrappers to make the C++ code functional in Xcode. Sam was responsible for implementing the functionality of the UI by writing Swift to call the corresponding C++ functions. This was by far the biggest challenge our group faced and proved to be far more difficult than we initially predicted. Due to this and the lack of time, our group decided to pivot to a command-line interface.

Our team met every Monday at 3:30 pm and Friday at 10:00 am for the past four weeks. In meetings, we discussed the progress we were making, what we planned to have done by the next meeting, and the challenges we were anticipating. Apart from these meetings, our team communicated daily through Discord. Our frequent communication ensured that we were all on the same page when making key decisions about our system.

Task Breakdown

Legend: Owen = OM, Mason = MS, James = JK, Zach = ZH, Sam = SL

Task	Completed By	Date Assigned	Time Spent	Date Completed
Complete Student and Class Structs	ZH	1/10/2020	8 hour	1/13/2020
Complete v1 SDS, SRS, and Project Plan	All	1/12/2020	5 hours	1/17/2020
Client Meeting	All	1/17/2020	30 minutes	1/21/2020

Complete RosterIE module	OM	1/21/2020	4 hours	1/25/2020
Create Xcode workspace	SL	1/21/2020	3 hours	1/26/2020
Complete UI design	MS	1/21/2020	8 hours	1/26/2020
Write C++ containers for swift	MS	1/26/2020	9 hours	1/28/2020
Complete State Persistence module	OM	1/25/2020	9 hours	1/29/2020
Learn how to use Xcode and work with swift	SL	1/26/2020	8 hours	1/29/2020
Complete ColdCallQueue module	JK, ZH	1/13/2020	16 hours	1/27/2020
Complete Reporting module	OM	1/25/2020	4 hours	1/29/2020
Create test cases	ZH	1/27/2020	6 hours	1/30/2020
Create system directories	OM	1/29/2020	1.5 hours	2/01/2020
Evaluate and test the backend prototype	JK, OM , ZH	1/27/20	7 hours	2/01/2020

Set up all front end windows to be used in the program	MS, SL	1/28/20	10 hours	1/30/2020
Combine back end to front end	MS, SL	1/30/20	22 hours	Could not complete
Debug back-end code	JK, OM, ZH	2/1/20	13 hours	2/3/2020
Command-line interface implementation	JK, OM, ZH	1/30/20	26 hours	2/3/2020
Finish Documentation	ALL	2/3/20	15 hours	2/3/2020
Presentation preparation	ALL	2/3/20	tba	Not Finished

Monitoring and Reporting

We utilized the Task and Assignment breakdown template posted on Professor Hornof's website to keep track of our progress. We imported it to Google Sheets so the latest version would be accessible. When a new task was assigned, an edit to the document would be made and the person would be notified via our Discord channel. Once that person completed their task, they were responsible for letting their teammates know so that they may confirm its completion.

Another way progress was tracked was through our repository on Bitbucket. While creating a separate branch for each module and with every push to that branch was a pull request. The pull request notified group members of the commit and required each group member to approve it before it could be merged into the master branch. This proved to be an effective way to share our progress with one another.

Build Plan

We decided to write our project in C++ since it is significantly faster than Python and since it is far more manageable than C. Furthermore, we also believed that C++ would be Xcode compatible which would give us freedom to make a simple, yet aesthetically pleasing user interface. After we completed our initial documents in week 2, the back-end team began writing code to achieve the system's functionality while the front-end team began familiarizing themselves with Xcode and developed the design for the UI.

During week 3 we had completed the modules for importing and exporting the rosters, storing the class data, and calling on students from classes. Owen worked primarily on the state persistence module, Zach worked on the class list module, and James implemented the aspect of randomness into the queue. While these aspects of the back-end were being worked on, the front end team had successfully created a UI. Mason worked on developing the design while Sam implemented functionality such as customizable keystrokes which prompted the application to be displayed/hidden.

In week 4, the bulk of the back-end was complete and ready for testing. Zach developed 5 test cases, testing each use case and also the randomness of the queue. The tests proved to

be essential to the success of the system by revealing a few bugs in the state persistence module. While these issues were being handled, the front-end team ran into an issue. Despite what we initially thought, the use of C++ came with an array of problems in Xcode. To overcome these obstacles Mason developed a wrapper to put the C++ code in which could then be called on by the Swift code Sam was writing. Unfortunately, Swift ended up being incompatible with some of the complex return items of the C++ code. This meant that our group had to pivot from our unique user interface to a command-line interface. Due to the time crunch, the objective was to create a simple interface that encapsulates the functionality of the back-end.

Build Plan Breakdown

Despite some hardships, our group made the right decision by creating the system in C++. The system is really fast, has no memory issues, and does not rely on any abstract libraries ensuring that it will remain functional for years to come. One obvious shortcoming from our planning was the lack of research about the C++ and Xcode compatibility issues. This is what ultimately prevented us from achieving the user-friendly interface which we initially sought out to create.

Nonetheless, dividing the group into two teams proved to be very effective. It allowed us to make progress on both fronts concurrently, optimizing the productivity of our group under the given time constraints.