SYSC3303 Final Report

April 7, 2025

Lab 3 Group 5
Ben Shvetz, 101157858
Akul Ghai,
Jawad Mohammed,
Daniel Kuchanski,
Jonas Hallgrimsson,
Shahrik Amin,

Table of Contents

Breakdown of Responsibilities over Iterations

List of team members and their contributions to Iteration 1:

Akul Ghai
- Wrote Scheduler class code
- Synchronized Scheduler and DroneSubsystem class
- Debugged final draft of code

Ben Shvetz
- Wrote DroneSubsystem class code
- Wrote Zone class code
- Wrote README.txt

Jawad Mohammed
- Wrote Tests related to FireIncident
- Assisted with general system design

Daniel Kuchanski
- Class Diagram and Sequence Diagram UML
- Initial FireIncidentEvent class code
- Helped write Scheduler code

Jonas Hallgrimsson
- Wrote FireIncidentSubSystem Class
- Completed FireIncidentEvent Class
- Wrote FireIncidentEvent Test Class

Shahrik Amin
- Wrote DroneSubsystem class code
- Helped write Zone class code
- Wrote DroneSubsystem Test class code

List of team members and their contributions to Iteration 2:

Akul Ghai
- Worked on new Scheduler logic, Scheduler state machine
- Debugging Scheduler and Drone connections
- Updated on Scheduler state machine diagram

Ben Shvetz
- Worked on Drone state machine diagram
- Worked on DroneStateMachine.java and updating DroneSubsystem
- Updated readme and javadocs
- Worked on SchedulerStateTest and updated SchedulerTest

Jawad Mohammed
- Sequence Diagrams
- Small edits in readme

Daniel Kuchanski

- Worked on Scheduler state machine diagram
- Worked on Scheduler state machine

Jonas Hallgrimsson
- Worked on Drone state machine diagram
- Worked on DroneStateMachine.java and updating DroneSubsystem
- Updated DroneSubSystemTest
- Created DroneStateMachineTest

Shahrik Amin
- Worked on Drone state machine diagram
- Worked on DroneStateMachine.java and updating DroneSubsystem
- Worked on Class Diagram
- Updated readme and javadocs

List of team members and their contributions to Iteration 3:

Akul Ghai
- Worked on UDP implementation for Scheduler Class

Ben Shvetz
- Worked on UDP implementation for DroneSubsystem and DroneStateMachine classes
- Updated drones test files
- Updated readme

Jawad Mohammed
- Sequence Diagram Update
- Updated FireSubsystem Tests

Daniel Kuchanski
- Worked on Scheduler state machine diagram
- Worked on Scheduler state machine
- Created initial Scheduler version and helper methods

Jonas Hallgrimsson
- Updated FireIncidentSubSystem to UDP/RPC calls.
- Updated FireIncidnetSubSystemTest
- Updated FireIncidentEvent

Shahrik Amin
- Worked on UDP implementation for DroneSubsystem and DroneStateMachine classes
- Updated UML Class Diagram

List of team members and their contributions to Iteration 4:

Akul Ghai
- Worked on Scheduler to receive and simulate faults in coordination with drone.
- Worked on scheduler to implement 2 threads, one for receiving fire events and one for drone scheduling.

Ben Shvetz
- Worked on handling of faults in DroneSubsystem and DroneStateMachine
- Worked on new state machine diagram for drone

Jawad Mohammed
- FireIncident Planning
- Fire Incident Tests
- Timing Diagram

Daniel Kuchanski
- Worked on Scheduler to receive and simulate faults in coordination with drone.
- Worked on updated scheduler state machine diagram

Jonas Hallgrimsson
- Updated FireIncidentSubSystem to send new faults.
- Updated FireIncidnetEvent to send new faults.
- Updated event_log to add new faults.

Shahrik Amin
- Worked on handling of faults in DroneSubsystem and DroneStateMachine
- Worked on new state machine diagram for drone

List of team members and their contributions to Iteration 5:

Akul Ghai
- Scheduler fixes

Ben Shvetz
- Metrics and drone fixes

Jawad Mohammed
- GUI

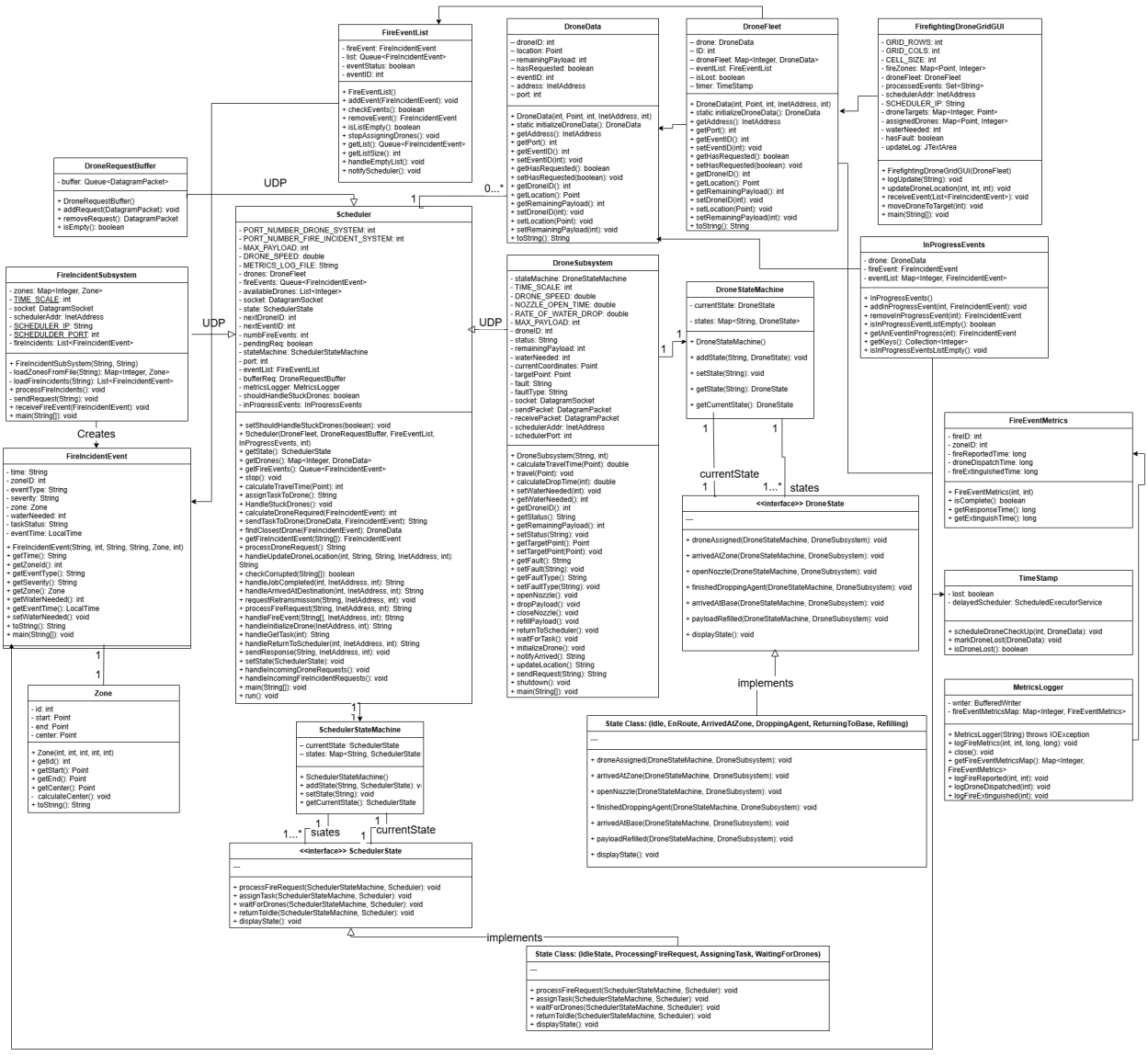
Daniel Kuchanski
- Scheduler fixes

Jonas Hallgrimsson
- GUI

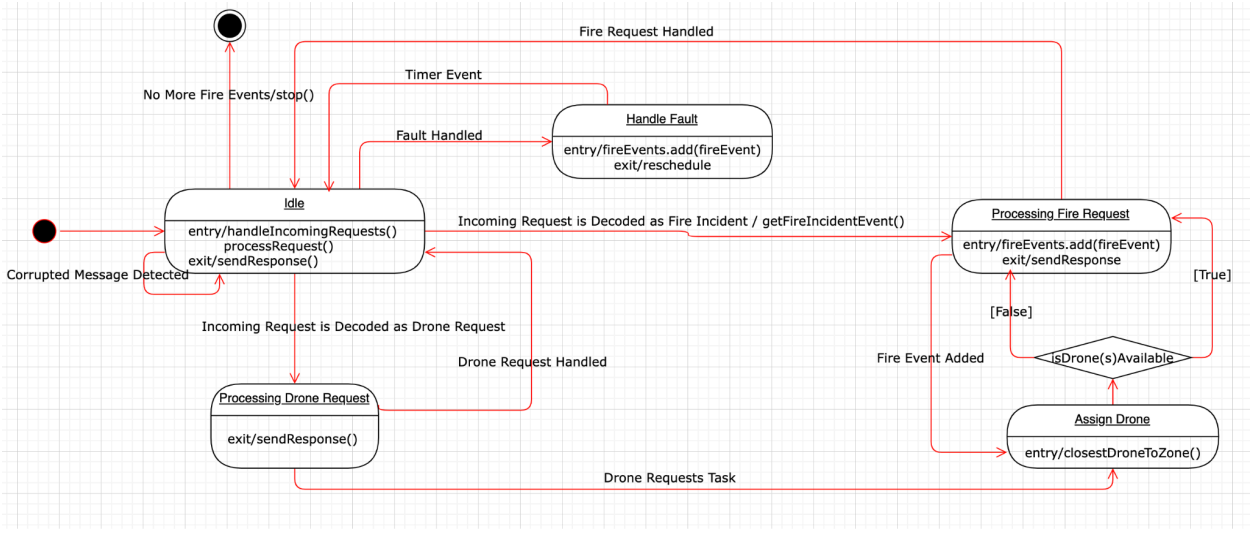
Shahrik Amin
- Metrics and drone fixes

Diagrams

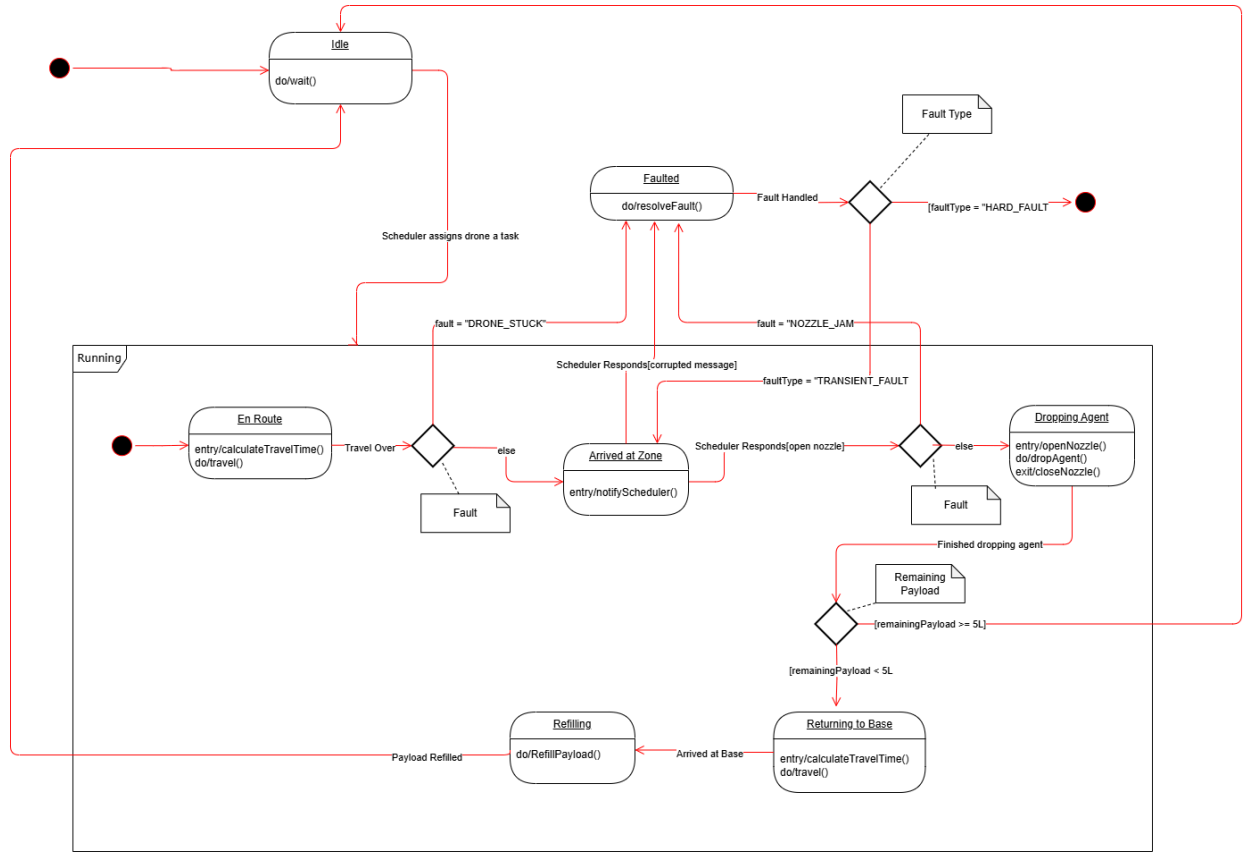
UML Class Diagrams:



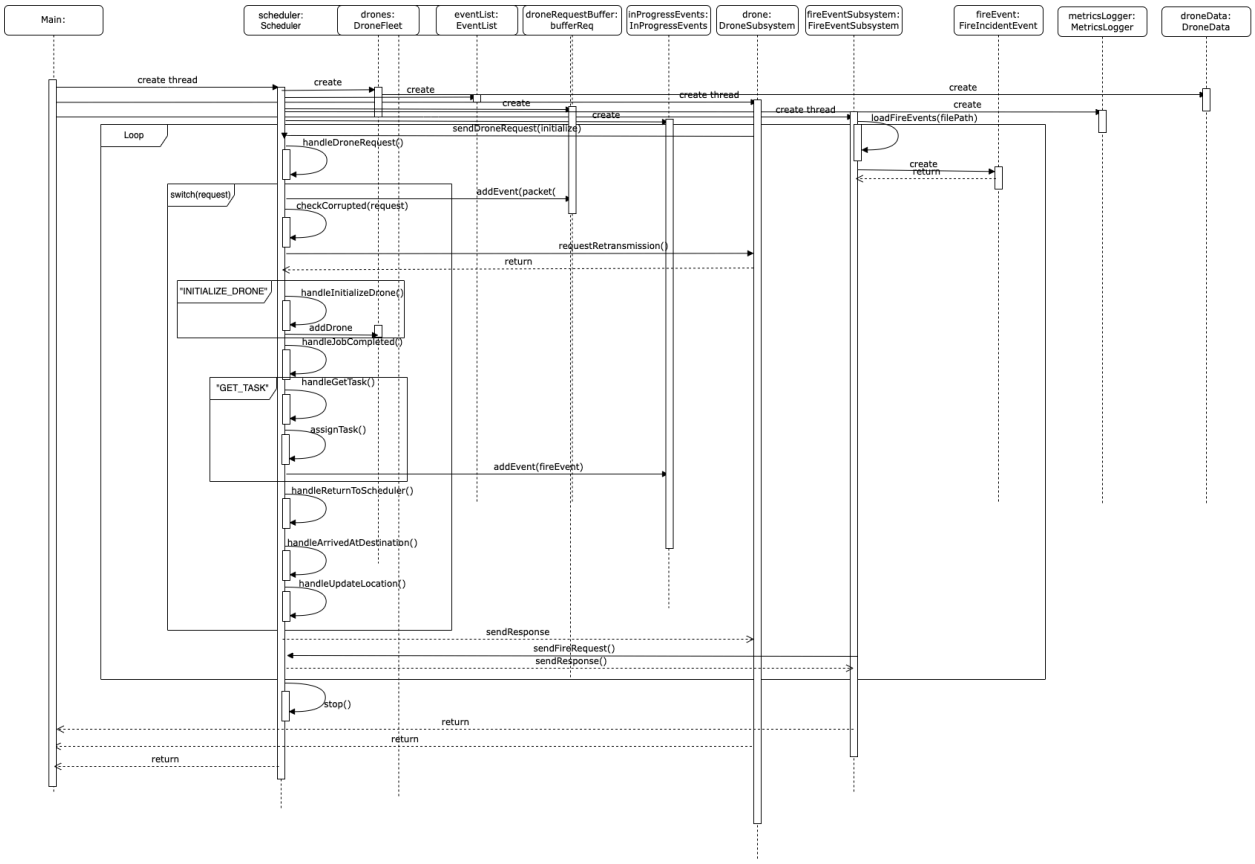
State Machine Diagram Scheduler:



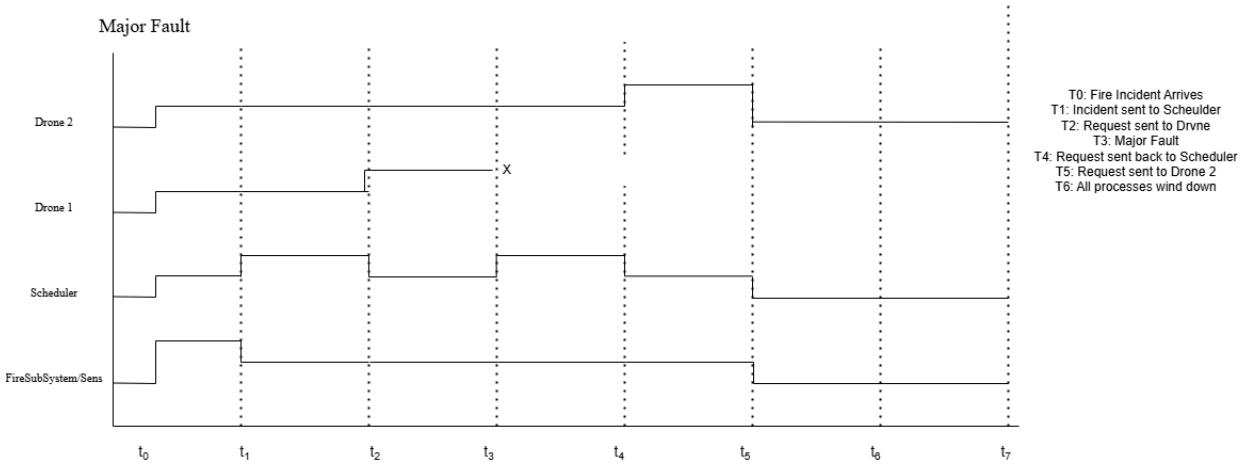
State Machine Diagram Drone:

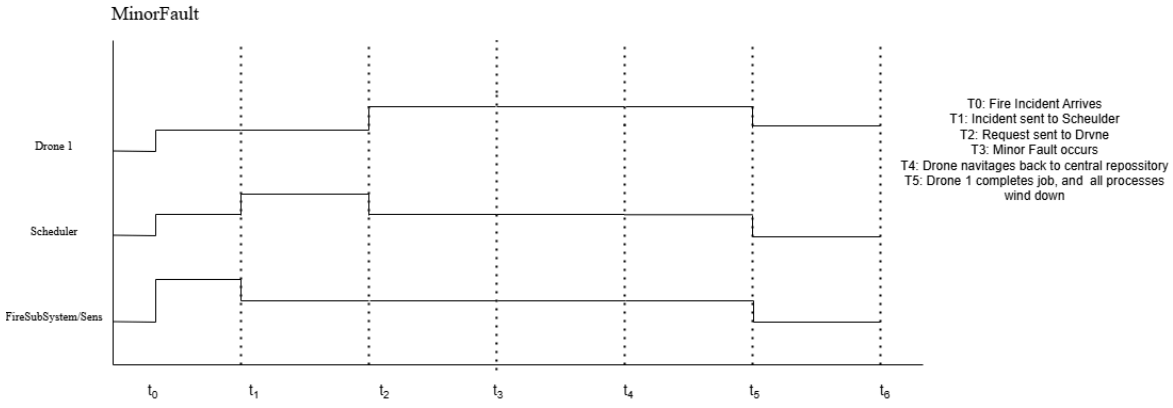


Sequence Diagram:



Timing Diagrams:





Setup and Test Instructions

Setup Instructions

Ensure you have Java JDK installed and an IDE such as IntelliJ IDEA.
Ensure all java files are located in src\main\java directory.
Ensure the event and zone CSV files are located in src\main\resources
Ensure CSV files are in correct format
Edit DroneSubsystem.java run configuration to allow for multiple instances

1. Open project in an IDE such as IntelliJ IDEA
2. Ensure src\main\java directory is marked as Source Root directory
3. Run the Scheduler.java class
4. Run as many instances of DroneSubsystem.java class as you need. (Ex. 5 instances for 5 drones)
5. Run the FireIncidentSubSystem.java class
6. Run FireFightingDroneGridGUI

Test Instructions

Follow Setup Instructions
Additionally,
Ensure all test files are located in src\test\java

1. Open project in an IDE such as IntelliJ IDEA
2. Ensure src\tests\java directory is marked as Test Resources Root directory
3. Right-click src\tests\java directory and select "Run all tests"

Measurement Results

In our drone simulation, we measured two metrics, response time and extinguish time. The response time is the time it took to assign a drone to a fire from when it was first reported. The extinguish time is the time it took to fully extinguish the fire from when it was first reported. In order to calculate these metrics, we logged the system time when a fire enters the system, the time when a drone is assigned to the fire, and the time when the fire is fully serviced.

For a simulation of 6 fire events here are the measurement results:

Fire ID	Zone ID	Response Time(ms)	Extinguish Time(ms)
1	1	17	130730
2	3	1	130350
3	4	1	130200
4	2	1	200480
5	1	1	169330
6	3	143980	150280

Looking at the above results, we can see that the scheduler is fast to assign drones unless there is no drones available and has to wait for one to return. The extinguish time takes a few minutes and varies since the drone is flying to different zones and dropping different amounts of firefighting agent.

Conclusion(Reflection)

This project was a fascinating look into developing software with a large scope and a large team of programmers. The Fire Prevention Drone System presented a challenging set of problems that we all had to work collaboratively to solve. The first aspect that stood out to us was the sheer size of the project. The twelve-page project specification outlined a system built with concurrency at its core, a technology none of us had a particularly significant amount of experience with. In fact, some of us were a little out of practice with Java, which didn't exactly inspire confidence. Yet, as the class progressed and the actual deadlines approached, we found the iterations to be well within reach. Overall, the project was a worthwhile learning experience, and we enjoyed working on it as a team.

What parts do you like:

Due to the drone system having so many parts, we all had the opportunity to work on something that intellectually challenged us. This included the ever-expanding scheduler and core program logic, the UDP interfacing between various components of the program, and the UI, which was developed during quite a time crunch at the end of the semester.q

In FireIncidentSubSystem, we appreciated the class encapsulating the complex functionality of detecting, parsing and processing fire incidents in a modular way. Its design simplifies the transformation of raw input data into actionable fire events that can be managed easily by the rest of the system.

We appreciated how the Scheduler class seamlessly orchestrates the interaction between the fire incident subsystem and the drone fleet. Its design incorporates a robust state machine that allows for smooth transitions between various operational states, ensuring that tasks are efficiently assigned to available drones while handling faults gracefully. We also liked its use of multithreading and asynchronous message handling via UDP, which provides real-time responsiveness and scalability.

We also handled intergroup dynamics particularly well. There were no major moments where group members butted heads. The team atmosphere was very calm and collected. Everyone submitted deliverables on time, and we were all open to assisting each other when someone needed help.

What parts should be redone:

Looking back on this project, many of our issues were rooted more in planning problems than any lack of coding skill. Over the semester, our task delegation process was as follows: we would organize a big group meeting, hash out what was needed for the next iteration at a subsystem level, then split off into teams who worked on their subsystems independently. Usually, this was the Fire Reporting/Drone Request team, the Scheduler team, and the Drone team.

Near the submission date, we would meet again and make sure the code was compatible, fixing any bugs as they presented themselves. This was what worked best for us due to a variety of reasons, including conflicting time schedules and different workloads for different subsystems through the various iterations.

Unfortunately, since we were working in silos, as the codebase grew more complicated, we ran into interfacing problems multiple times. While our individual systems worked, when we connected them to each other a bug would present itself. Such as [a time when this happened]. These were often short fixes, but we did have to do large logic overhauls once or twice. With iteration deadlines looming, this led to a few tense crush periods. [Such as when.. If we can think of an example thats great otherwise we can get rid of this line] This also resulted in a certain amount of spaghetti code, aspects of the project that are perfectly functional and meet the requirements, but could have been done more efficiently if we had planned more thoroughly.