



---

## Table of Contents

---

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Team Member 1: Henri Bai .....</b>    | <b>3</b> |
| 1.1      | What went Wrong.....                     | 3        |
| 1.2      | What went RiGHT.....                     | 3        |
| 1.3      | Lessons learnt .....                     | 3        |
| <b>2</b> | <b>Team Member 2: Andrew Dorman.....</b> | <b>4</b> |
| 2.1      | What went Wrong.....                     | 4        |
| 2.2      | What went RiGHT.....                     | 4        |
| 2.3      | Lessons learnt .....                     | 4        |
| <b>3</b> | <b>Team Member 3: Matt Herbst.....</b>   | <b>5</b> |
| 3.1      | What went Wrong.....                     | 5        |
| 3.2      | What went RiGHT.....                     | 5        |
| 3.3      | Lessons learnt .....                     | 5        |
| <b>4</b> | <b>Team Member 4: Bryan McCoy.....</b>   | <b>6</b> |
| 4.1      | What went Wrong.....                     | 6        |
| 4.2      | What went RiGHT.....                     | 6        |
| 4.3      | Lessons learnt .....                     | 6        |

# 1 Team Member 1: Henri Bai

## 1.1 WHAT WENT WRONG

Many things could have gone wrong and they were avoided. Examples of things going wrong were the wrong technologies used to develop the software. We chose python and java for we had the most experience with the languages, and there exist a stable and easy to use interface between java and python. The only thing that has gone wrong is the fact that we were not able to develop code collaboratively like we have at the end of the semester. We could have developed and implemented many more features if we coded together earlier.

## 1.2 WHAT WENT RIGHT

Collaborative programming. We adopted the methodology of one developer sitting at the computer programming while another developer watched over the shoulder critiquing the typer. This improved the quality of the code developed which reduced a massive amount of debugging time. This enabled us to implement features efficiently. The communication and integration problem has been solved because everyone is in the same room, and we can make sure that other people's code will integrated easily. We also avoided project failures by managing our time well, and preventing feature creep. We decided on a set of features and analyzed their complexity and time required to make them, and set those to stone. No features to be added.

## 1.3 LESSONS LEARNT

Although I have had an internship before and know the basic structure of software development, this class enabled me to explore all parts of software development and not just testing and integration (which was my role in my internship). Knowing how developers are able to analyze, model, and implement features enables me to better understand my role in a product cycle (such as testing). This enables me to anticipate any inconsistencies encountered when a project is moved from one part of the phase to another, and know how to solve these problems easily. Learning the terminology will also reduce the time required to learn the standard practices companies will adopt.

## **2 Team Member 2: Andrew Dorman**

Overall, I enjoyed this class a lot. I think it was fun working on a single project for the whole semester, because once it's complete at the end, you have something really cool that you can be proud of.

### **2.1 WHAT WENT WRONG**

I think the set up of the class assignments kind of dictated our coding schedule. I don't think we really started doing very much actual programming until most all of the documents were complete. Obviously, in a professional setting, not much implementation would take place until the design stages are over, but since this was a classroom setting, I think starting earlier would have been beneficial.

### **2.2 WHAT WENT RIGHT**

Key events I remember are the initial decision making with our group about what to do for our project. Then the development stages were very key – determining requirements and features; really what the project is going to be and do. The actual coding stage obviously stands out to me, as it resulted in several late nights, and was also the most enjoyable part of the project.

### **2.3 LESSONS LEARNT**

Although I think the project went very well, and I really enjoyed working with the group I was in, I suppose if I were to do it all again, the main thing I would do differently is to start doing code work a lot earlier. We did end up completing our project on time, but there was a lot of scrambling to get it done that last week before it was due. I think had we started coding earlier, we could have been doing with the core requirements with time to spare, and been able to add additional neat features that would have really spiced up our final product.

The class gave me the valuable experience of working on a large project in a team, something that will be helpful in a couple years when I enter the workforce. In general, I think the class went very well, and is definitely one of my favorites I've taken so far at Iowa State.

## **3 Team Member 3: Matt Herbst**

### **3.1 WHAT WENT WRONG**

A lot of the coding was done towards the end of the semester. I think this happened to most groups, but luckily we had started the essential parts (communication, getting Jython working) early and were able to piece in the rest independently of each other. It would have been nice to have started our simultaneous coding sessions (see next section) about a month ago instead of closer to finals so that we'd have been able to add even more features marked optional in the SRS. We had a lot of trouble automating tests (which was a requirement for the project). Although a lot of the GUI couldn't easily be tested this way, it would have been nice to have considered the automated tests early on. That way, instead of building the system and having a lot of trouble testing the parts buried under a GUI, we could have done regression testing throughout on the parts that didn't have a GUI controlling them until later.

### **3.2 WHAT WENT RIGHT**

Coding together in a single room boosted our productivity. Before we started doing this, if one of us had a problem or didn't know how to start, we would wait for the next meeting or send out an e-mail. This takes time and causes that member to lose momentum in their work. Sitting in the same room and coding allowed us to ask each other questions about how to proceed or how a particular piece of code works right away. Additionally, members could check on the progress of the others and give them feedback if something doesn't look right.

Splitting up the work and setting deadlines helped somewhat. When we finished or had most of it done by the deadlines we set arbitrarily, it seemed to keep things moving. The problem was enforcing the deadlines, since they were set by us.

### **3.3 LESSONS LEARNT**

In future programming projects, I'll probably try to convince the team to start early and code together rather than relying on people to occasionally make submissions to the repository. And, although we tested throughout, we didn't keep automated testing in mind until we remember it was required for the project. It would have been nice to have done that early so that we'd have regression tests to run (and, of course, get a good grade).

## **4 Team Member 4: Bryan McCoy**

### **4.1 WHAT WENT WRONG**

As for what went wrong I think most of it had to do with the way our code sessions didn't sync with our documents. Early in the semester we were coding way ahead of our documents and were basically writing them based off of our code. Around the DD1 however, we were writing it without code to base it from and it suffered as a consequence. Also we spent a long time planning with no real action to speak of. I think this was due in part to the need for the rest of the team to figure problems out. In future projects more team work meetings will be made as opposed to team planning meetings.

### **4.2 WHAT WENT RIGHT**

What went right was my group and the way we came together and made amazing progress in a very short amount of time. We all also had unique skills and were able to specialize in some areas where others would have taken much more time. I also think that our initial design went right as it allowed us to do what we wanted without any major changes.

### **4.3 LESSONS LEARNT**

The lessons I learned were to communicate well and get along with the group. I learned a lot about Swing which I had never used before and how java communicates through sockets. I learned how the software development process is done and how much work is involved outside of coding. Finally I learned how to think like a developer and an end user as at all times. I had to be thinking of "how do I want this to come out?" and "how do I implement it?" Keeping both of those ideas in mind when coding can save you lots of time later.