**Individual Report, Self and Team Evaluation**

1. **Name:** Jacob Hillebrand
2. **Team Names:** Jacob Hillebrand, Jack Hildebrandt
3. **Date Period:** From: 09/27/2019 To: 10/01/2019
4. **Project or Assignment Name:** Assignment 2 PART 1 (mini-Shell)
5. **Summary of Activities:**

|  |  |  |
| --- | --- | --- |
| **Task/Event/Job/Action** | **Date** | **Time** |
| Group meeting 1 | 09/27/2019 | 11:30 P.M to 3:00 P.M |
| Requirements, Methods and Roles definition | 09/27/2019 | 11:30 P.M to 12:30 P.M |
| Worked on 1 (individually or in group) | 09/27/2019 | 12:30 P.M to 3:00 A.M |
| Group meeting 2 | 09/28/2019 | 1:00 P.M to 5:30 P.M |
| Worked on 1 (individually or in group) | 09/28/2019 | 1:00 P.M. to 4:30 P.M. |
| Tested my implementation (unit testing) | 09/28/2019 | 4:30 P.M to 5:00 P.M |
| Testing Part 1 | 09/28/2019 | 5:00 P.M to 5:30 P.M |
| Submission | 10/01/2019 | 10:30 A.M. |
| Total hours |  | 8 hours |

**Requirements & Analysis Phase:**

When looking at the design requirements for part 1, we wanted to split the program up into objects for key pieces of functionality. We eventually decided it would be best to split the first part of the program up into two classes; a logger class and a commandspawner class. The logger object would be created upon shell creation and would store information about the shell history. The commandspawner object would be created in the main() function, and would handle calling the logger methods and calling other necessary methods to run programs.

# Design Phase:

During the design phase, we began deciding how to create our various classes. I (Jacob) determined that the logger class would have a constructor that would load a vector with history items from a file, a method to add a history item to both the vector and the file, a method to retrieve the last history item, and a method to print all history items. Jack determined that the commandspawner class would have a constructor that would get the user’s current working directory, a method to grab a command from user input and run it, a method to tokenize an input, a method to print the terminal history, and a method to attempt to run additional commands.

**Implementation Phase:**

During implementation phase, we implemented the design described above. I was successfully able to create my history class, and Jack was successfully able to create his commandspawner class. We were then both successfully able to use both classes as objects in our main program and get a successful “shell” run to complete the part 1 objectives. Jack even had time to make the working directory output colored :).

**\*\*\*\*\*\*Challenges**

1: Tokenizing proved to be a bit tricky, despite our prior knowledge of it.

2: I attempted to use the dup2 command to utilize cout and cin for writing to the history file (unsuccessfully).

3:

\*\*\*\*\*\*\*Things learned

I learned a lot about using c++ during this part of the project. I learned more about the limitations of passing variables to children processes in an object, and also learned about using dup and dup2 a lot more. And finally, I have finally figured out how to program without “using namespace std” :).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Teammate Evaluations (Confidential)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Member 1: Jack Hildebrandt

* Assigned Tasks/Work: Creating the CommandSpawner class
* Strength: Creating the overall project structure, persistence through adversity
* Weaknesses: Overenthusiasm
* Recommendation: Don’t completely lose it, but learn to reel back the ambition in the face of a deadline

Member 2: Jacob Hillebrand

* Assigned Tasks/Work: Creating the logger class
* Strength: Clear and concise code design, simple and functional code
* Weaknesses: Difficulty in pursuing an objective when stumbling blocks occur
* Recommendation: Learn to take a breather, and keep going when challenges arise