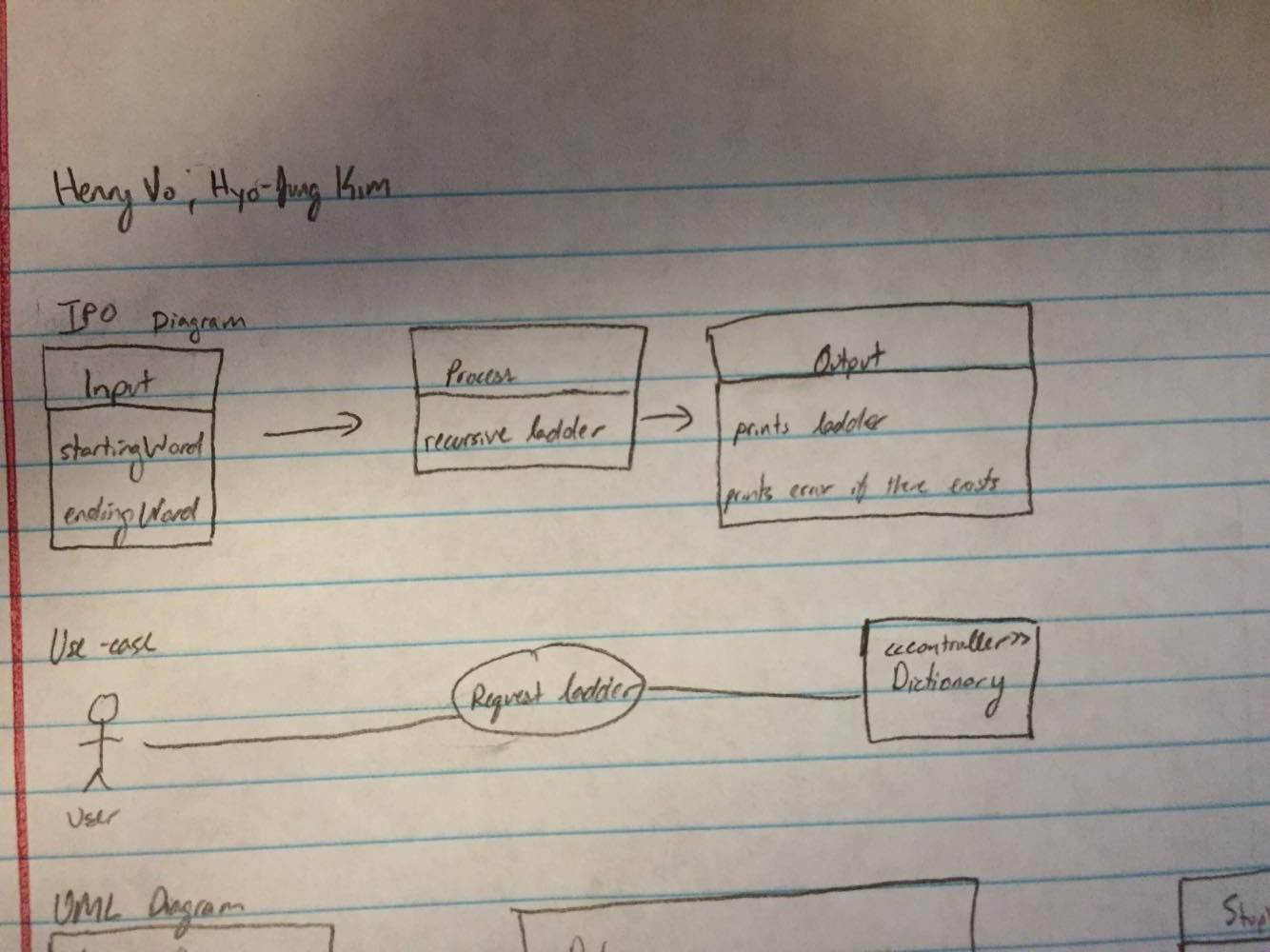
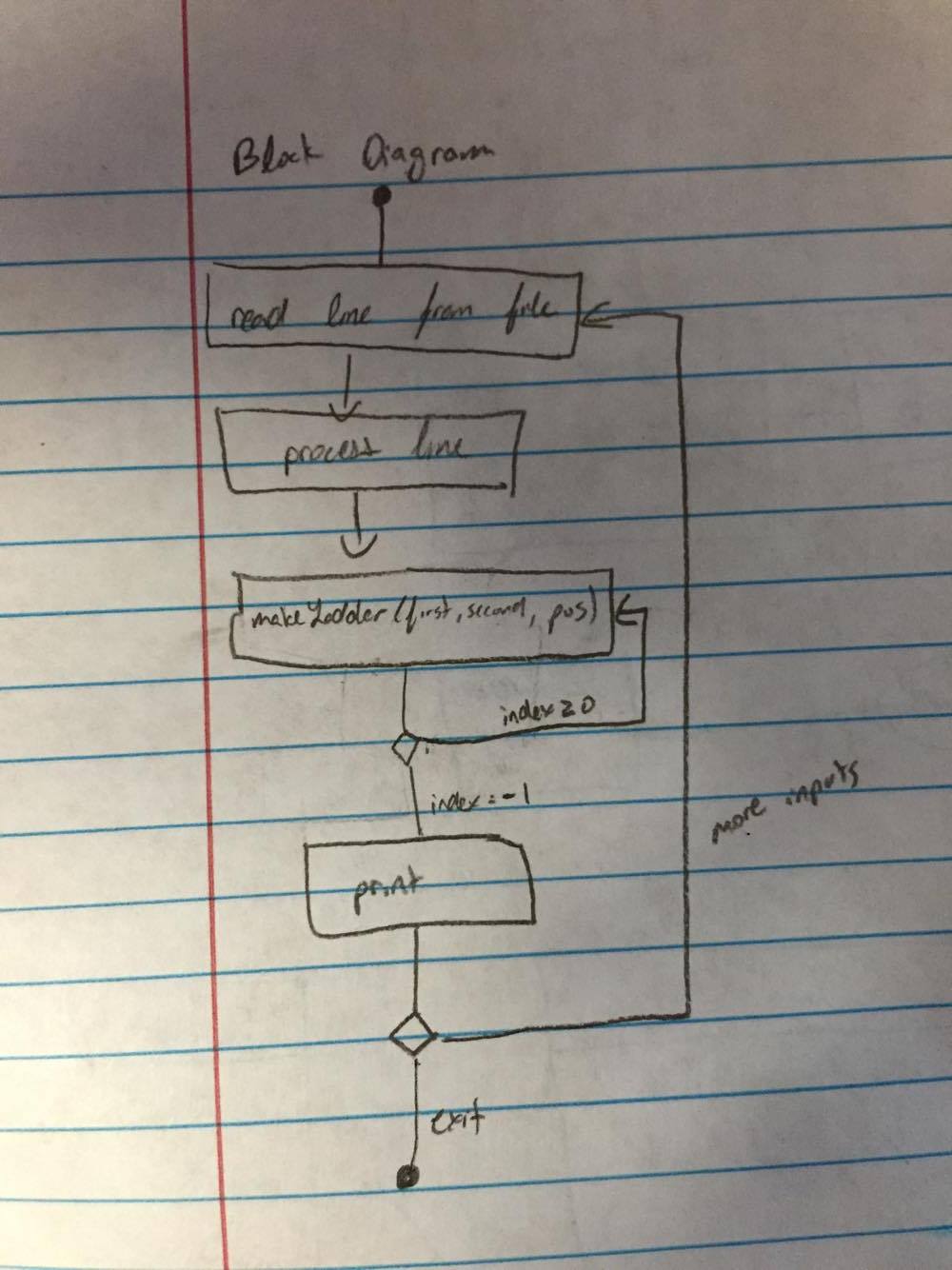
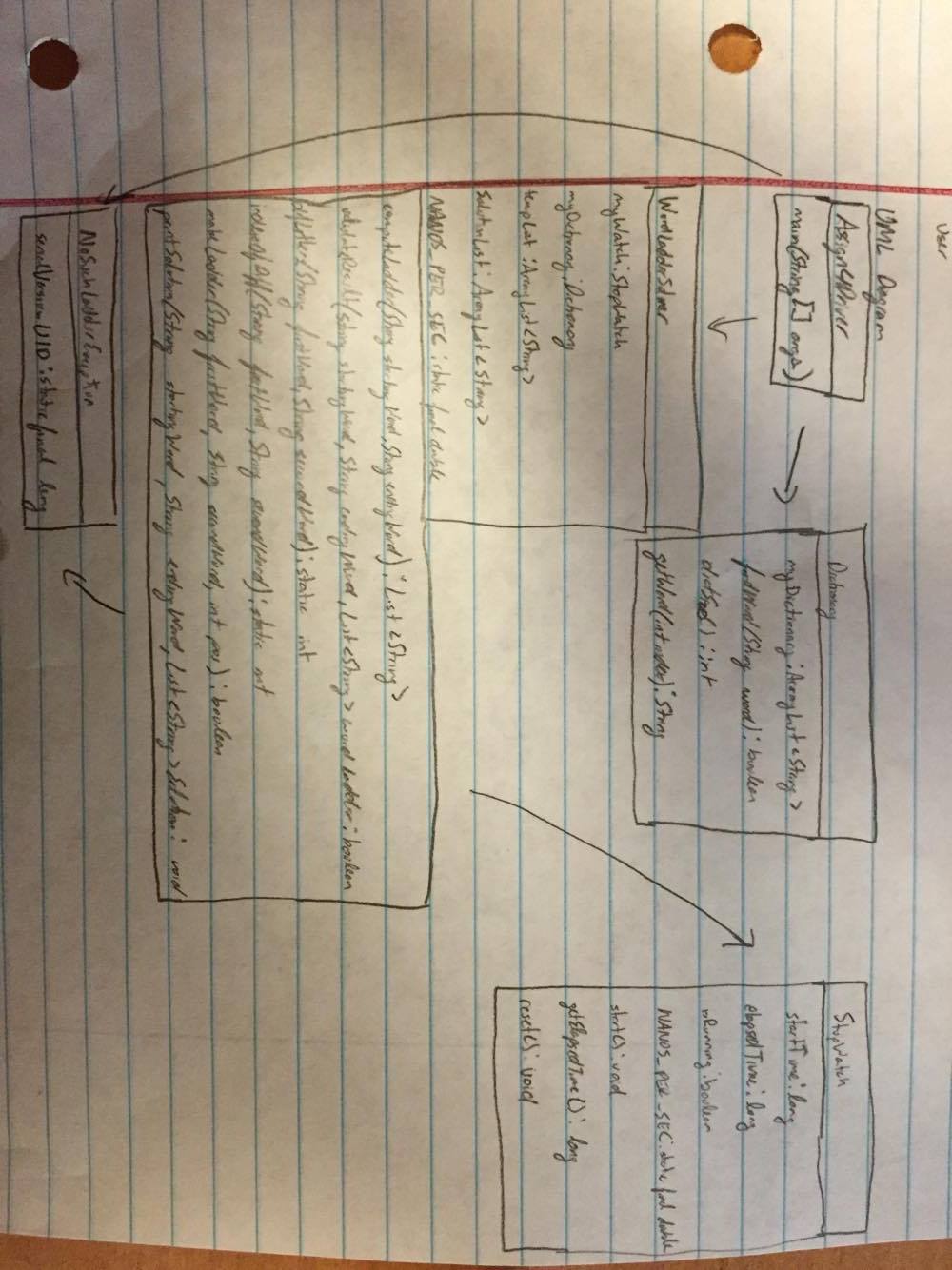
IPO and Use-case Diagram



Block Diagram



UML Diagram



**Driver Algorithm**

Start

Open File

New Line?

Get starting word

Get ending word

Solve ladder

Exit

YES

NO

Test Plan

1. Black box testing
   1. Run the program with any example starting word and ending word, then check the output if it fits with requirements
      1. If all the words in the solution list changes by 1 letter each time
      2. If any of the words in the solution list are in the dictionary
      3. If any of the words in the solution list are 5 letter words
      4. If any of the words in the solution list are repeated
   2. Check if the input test files are not given
      1. Dictionary or test inputs are not given or corrupted
2. White box testing
   1. Test all the path
      1. Try one example which the starting word is not 5 word letter long
      2. Try one example which the ending word is not 5 word letter long
      3. Try one example which the starting word is not in the dictionary
      4. Ending word is not in the dictionary
      5. Try one example which doesn’t have any solution
      6. Try one example which the ending word and starting words are same

6. A paragraph describing the rationale behind your design. This would include:

a) How does your OOD reflect the interaction and behavior of the real-world objects that it models?

Just like the way human finds the possible way to solve the problem, our program just follows it.

1. Check if inputs are validated (Starting and ending words, Dictionary)
2. Find next possible words from dictionary that is one letter different from the given word
3. Check if it is the ending words
4. Repeat 2 - 3 until it finds ending words.

As we constructed our program as object oriented design, it encapsulated data and procedures within each classes.

b) What alternatives did you consider? What were the advantages/disadvantages of each alternative both from a programming perspective and a **user perspective**?

The searching algorithm we could use was either Breadth-First search (BFS) or Depth-First search (DFS). We chose DFS to solve the problem, since DFS uses lower memory requirements than BFS since it is not required to save all of the next pointers(next possible words) at each level it go through. However, BFS also has advantages over DFS, since it can construct a shortest path from point starting word to ending word. Also, BFS doesn’t have to worry about infinite loop in the database.

As a user perspective, BFS is more commonly used for the searching method, since it doesn’t have to deal with recursion and infinite loop in the system. However, we decided to use DFS because we figured out the way to deal with infinite loop and we were concerning more about memory of the system.

c) What are some expansions or possible flexibilities that your design offers for future enhancements?

We could build a map for the dictionary. For example, we could build adjacency matrix or adjacency lists for the dictionary at the beginning of the system, so we do not have to check next words through the dictionary, every time we need one. The other possible expansion we could make is the probability system. The system currently we built, run the dictionary brute-force method to find the next possible words. We are currently using, DFS, but if we can order possible choices of words by probability that it may find the closest path, then the running time of the system will be reduced significantly.

d) How does your design adhere to principles of good design: OOD, cohesion, coupling, info hiding, etc?

As our program is constructed as Object Oriented Design, Our program encapsulates data and procedures grouped together within the classes.