[TITLE PAGE]

Group Members: Jenny Goldsher, Noah Harvey, Deandra Martin, Hiroki Sato

Course: Software Engineering

Professor: Dr. Elva

Date: December 16, 2020

**Table of Contents**

SDLC Phases 1

* Planning
* Requirements Engineering
* Design
* Implementation
* Testing
* Deployment

Division of Labor

Lessons Learned

Source Code

**SDLC Phases**

**Planning**

[not sure if the narrative belongs here or not?]

Students in CMS 120 have come to tutoring sessions with several consistent issues. For instance, students write algorithms with incorrect data types for the variables they use. This type of issue comes from how Python is formatted (not making programmers declare data types on assignment/declaration) and obfuscates details from programmers. Although the format of Python is oriented towards making programming easier, beginners do not inherently learn necessary concepts because the language covers up many aspects of programming complexity for the sake of ease of use.

Because CMS 120 is an introductory course, students are not expected to have prior knowledge of these concepts, however, it is good practice to teach students to code defensively and understand how the code they have written works. Since CMS 120 students are beginners coding in python – which allows them to program with more freedom than many other strongly typed languages – they may face more difficulty when they reach upper level CS classes if they never have a chance to develop a proper understanding of key programming concepts. This emphasizes the importance of having beginner students learn effective coding practices early on.

To combat this problem, there is a need for a web application that can read through student code snippets, give feedback, and ask questions. The program will not format the code for students, but it will ask related questions and challenge them to further their understanding of python. This would be a comprehensive way for CMS 120 students to exercise and reflect on their understanding of good defensive coding practice taught in class while they are programming such as being aware of data types of variables, reducing the number of unnecessary declaration and assignment of variables.

**Requirements Engineering**

**Requirements Elicitation**

To formulate the requirements, we gathered information from CS tutors: Jenny and Hiro by facilitating two JAD Sessions. We scheduled meetings and reviewed other related applications (Slack, CodingBat, LeetCode). The tutors first described the project generally independent of a specific platform which allowed us to establish a mutual and solid understanding of what we wanted to achieve.

In the meeting we discussed topics such as:

* The objective of this project/what the client is expecting to get out of this project
* How students should be able to interact with the system
* The types of questions the system should generate
* How the system should respond to student’s input

By holding a JAD session, we were able to pinpoint contradictions, ambiguities, and misinterpretation that could have occurred between clients and developers in real time. Further, we were able to discuss and address those issues. In addition, having both clients and developers in a meeting allowed us to make decisions that both sides could agree on quickly. We discussed whether to allow the system to format the code they received, and we decided that it would not – in keeping with the honor code of Rollins College to avoid unauthorized assistance. We also clarified that the system would not execute code passed by the student as this represents a security vulnerability and depending on its use could also violate Rollin’s Honor Code. We further discussed general types of questions the system would challenge the student with and clarified that they cannot be open-ended.

**Requirements Analysis**

We reviewed the information we had gathered and polished our understanding of what the system needs to be able to do as well as how it should execute these tasks after the JAD session. During our review and discussion within the development team, we realized that students should be able to feed in only a snippet of code instead of the entire Python file. This was the most logical strategy as the file was not getting compiled but parsed, and it would make students practice their coding in real time. Uploading entire files could also be a security vulnerability where the integrity of the system might be compromised by flawed files.

We also briefly discussed what software development model we should use for this project and agreed to use an incremental model. We chose this model based on its increased flexibility that would give us liberty to easily add or remove features. With short development time and a novel environment, this increased flexibility would be important.

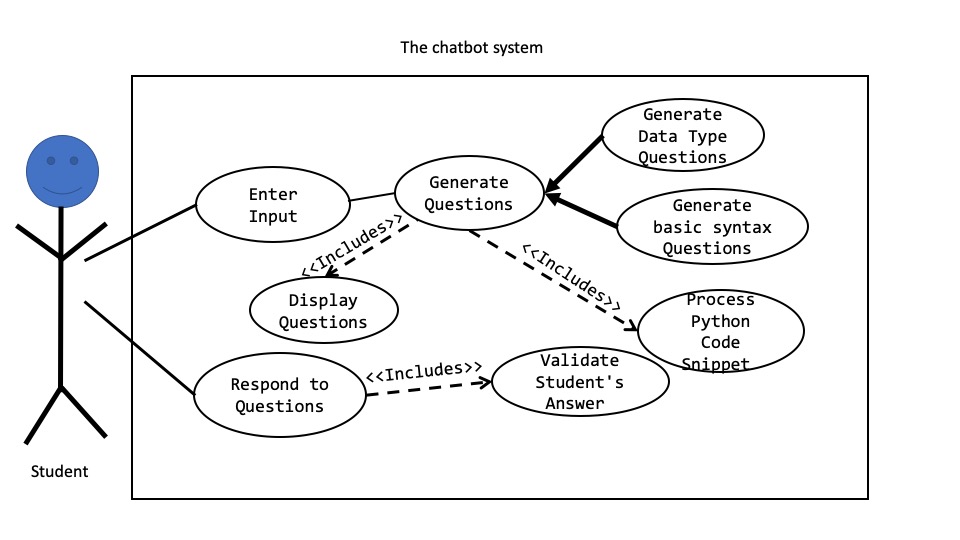
**Requirements Specification**

After meeting with CMS tutors and conducting our analysis, we came up with a list of requirements:

* Functional Requirements:
  + The system should allow users to enter a snippet of Python code.
  + The system should generate a set of questions based on the Python code that the user feeds in.
  + The system is a parser, not a compiler. This program should not execute the Python file and provide its output.
  + The system should provide a link to a page that discusses common error messages and their meaning.
  + The system should generate questions such as:
* Data type questions
* Basic syntax questions
* Questions about functions
* Non-functional requirements
  + The system should be accessible to all students via the internet (Web application)
  + It should be clear to user where they should insert their code snippet
  + The application should be in the form of a chatbot where user can easily interact
  + The system should generate answers but not show the answer until students at least try to answer those questions on their own first.
  + The system does not necessarily have to display all the possible questions, the number of questions can be selected or randomly generated (Default is 7).
  + The system should be implemented on a Linux server and using Python 3.

**Design**

**Use Case Diagram**

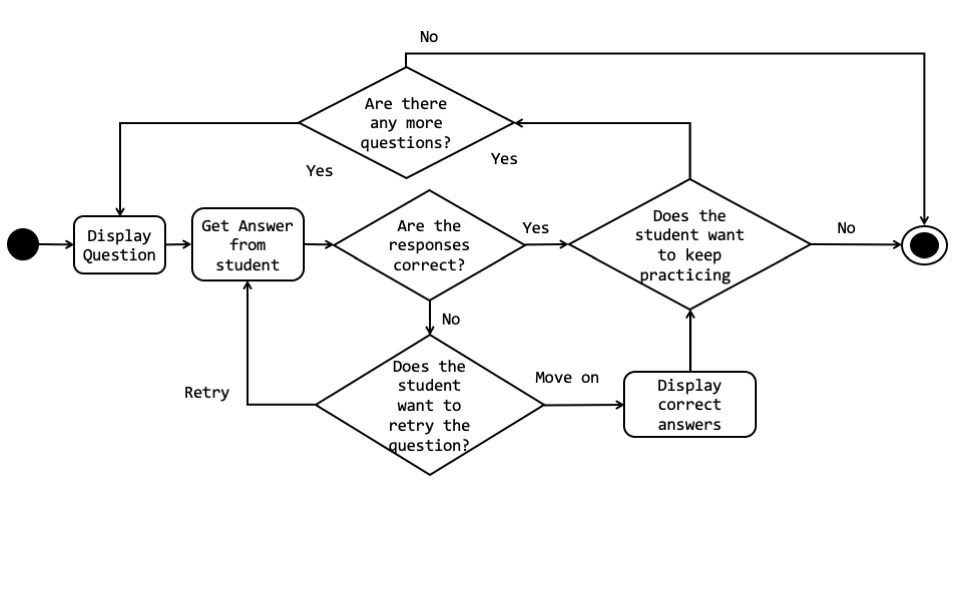


We also modelled our understanding of how students should be able to interact with the system using a Use Case diagram. Users will interact with the system when they insert the snippet of code or when they type in their answers to the questions generated by the system.

In addition to that, we agreed that we should use a combination of incremental approach and integration and configuration approach.

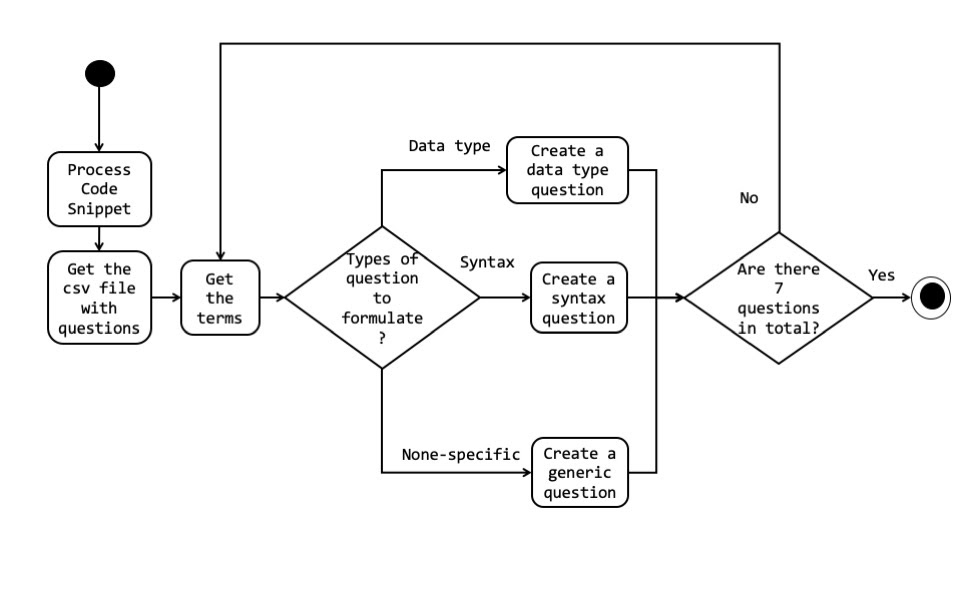
**Activity Diagrams**

**Activity Diagram for Validate Student’s Input**



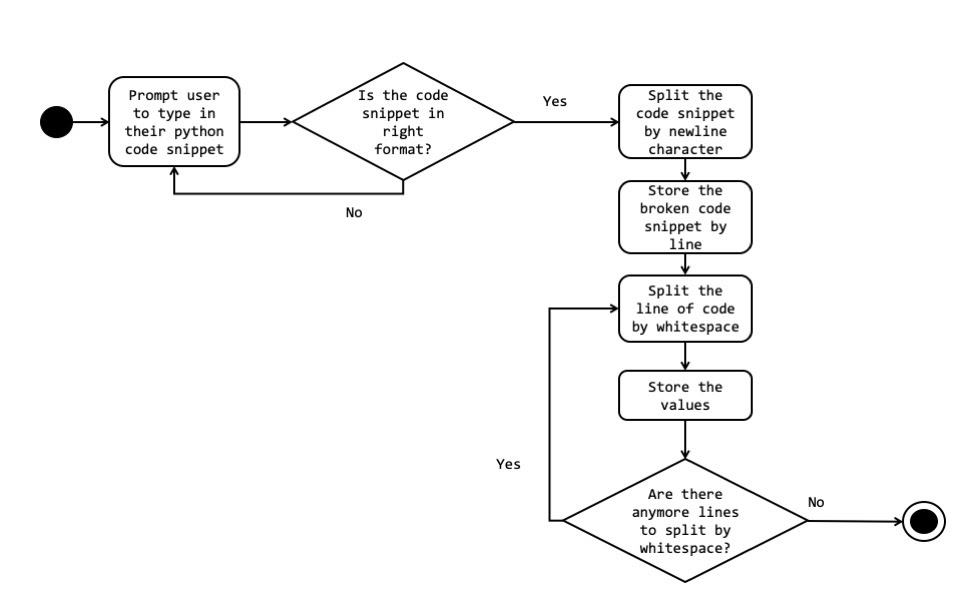
The tutor bot should display the default 7 random questions and prompt the student to respond. If their responses are correct, the student is given the option to keep practicing - which would display 7 more questions if available - or not. However, if at least one of the responses is incorrect, the student is given the option to retry and re-enter the correct answer or move on and display the correct answers. In this case, the student is free to keep practicing or not.

**Activity Diagram for Generate Questions**

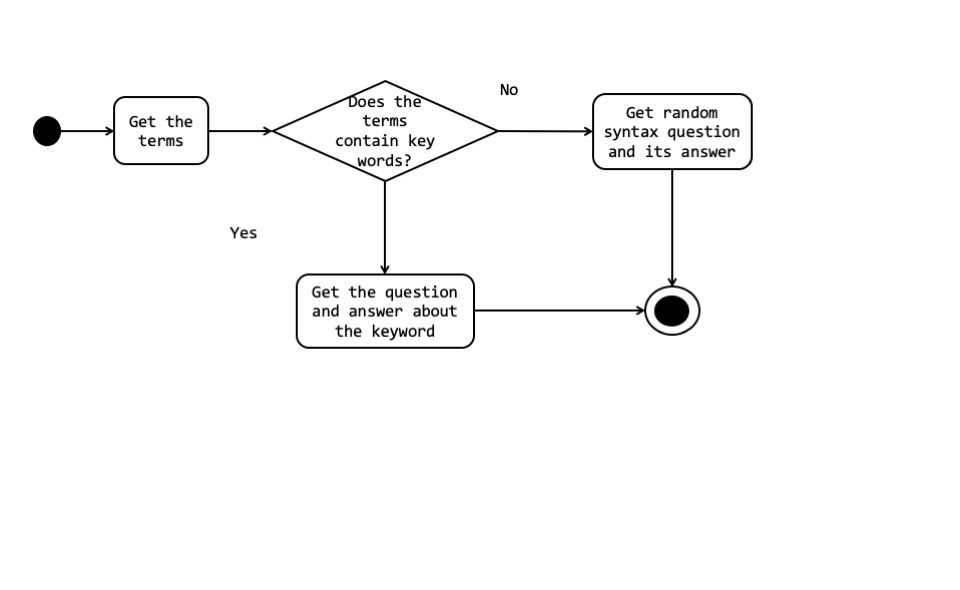


The system is going to process the code snippet created by the user and split it into tokens. We also get the .csv file with questions and based on the tokens we are going to create questions about either data type, python programming syntax or generic questions. Generic questions are not specific to syntax or data type but are about python or programming in general. This route will be taken by the system when we do not have enough code snippets provided by the user.

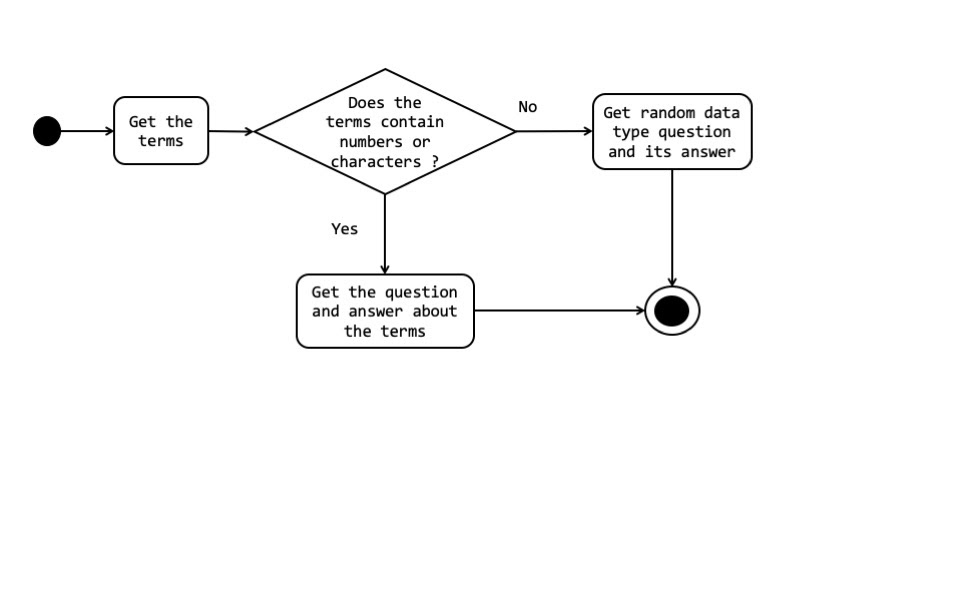
**Activity Diagram for Process Code Snippet**



The bot will prompt students to enter their line of code in Python. The bot should determine if the snippet is in the right format - if it is, the code is split by a new line character and the broken code is stored by line. The line of code is then split by whitespace and stored as values. If there are more lines, the code is split and stored until there are no more lines to process. Should the format of the entered code snippet be incorrect, the bot will prompt the user again to re-enter their snippet.

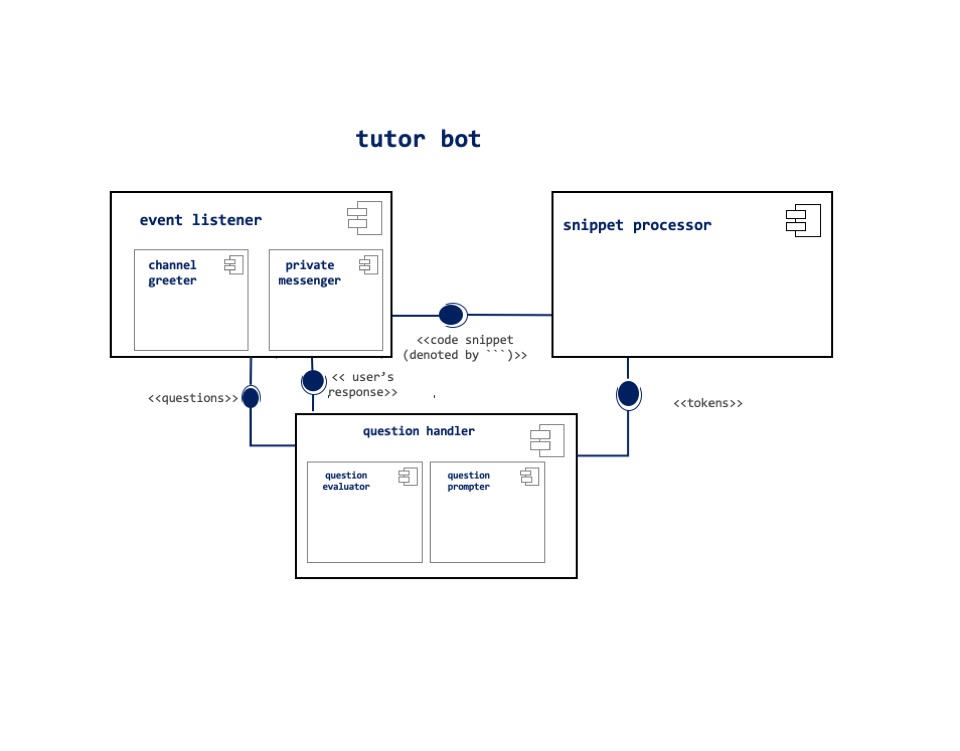
**Generate Basic Syntax Question**

If the code snippet provided contained tokens that matched the keywords for syntax type questions, then it should generate a question about the keyword. Otherwise, it should generate a random syntax question.

**Generate Data type question**

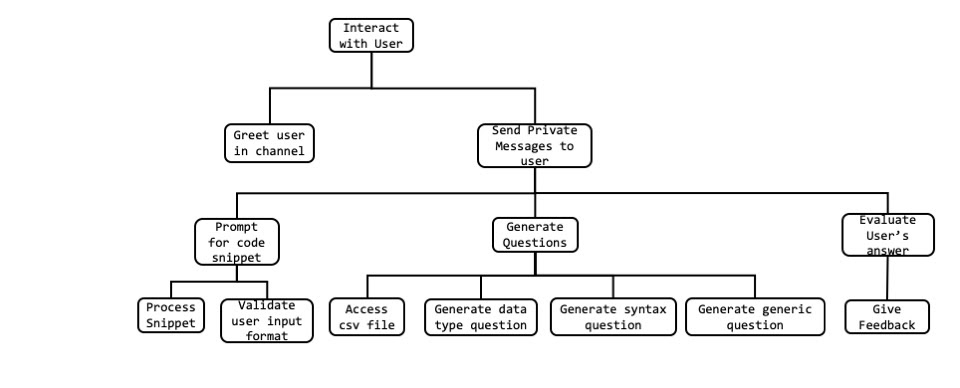
If the code snippets provided were related to data types such as variable declaration with value, and functions with return statements with values then the system will create a question about the data type related to the values in the code snippet. Otherwise, a random question relating to data types will be generated.

**Component Diagram**



The tutor bot system will have three main sub-components, the event listener, the snippet processor, and the question handler. The event listener will handle communications with slack via the Slack API. It will be able to post messages or greetings to the tutor bot channel on slack. It will also be able to private message students for tutoring. The event listener will provide the snippet processor with code snippets sent in private messages. The question handler will provide the event listener with prompts and questions. It will also evaluate answers provided by the event listener and if needed offer feedback documentation.

**Function Decomposition**



The tutor bot’s should be able to interact with the user by greeting in the channel or by sending private messages. System will prompt the user to enter a code snippet and receive it from the user and validate whether it is in the right format or not. If it was valid then the system will process the code snippet and should be able to create questions based on the lists of terms generated after the processing. Finally the private message will have functionality to evaluate the answers given by students and provide feedback and documentation links if needed.

**Implementation**

Before we start our coding, we’ve had a brief meeting with all group member to discuss more details of our software in coding specific way. The major components included communication through Slack’s API, Python for question processing(parsing) and generation, and a correctly formatted question CSV file. These key components would be translated into code using the component diagram we had previously created during the design stage. We decided that the best interface type between the Parser and Question handler was a two dimensional Python list data structure. Once the question was correctly parsed, the output would be passed to the question handler Python code which would access the CSV file to generate the content to be presented to the user. This content would then be passed back with the Slack API to the user interface.

**Testing**

We utilized Pythons unit testing suite to test our individual python files/functions. Each component of our program implemented in Python had its own test suite that accommodated for normal and abnormal outputs. After completing the unit testing for each Python component, we integrated the components that interacted with each other. Since the question handler took input from the parser, we tested that those two functions worked correctly together. Additionally, we had to make sure that the parser could take and process string input from the Slack API. We noticed that the string input from Slack did not include tab characters (‘\t’), so we altered our \*\*/code to count the number of spaces that started a line rather than the number of tabs.

**Deployment**

**Division of Labor**

**Lessons Learned**

This project showed us the importance of testing, clear communication in the form of documentation, and appropriate implementation time. As we have discussed in class, testing is a critical part of the SDLC that occurs throughout multiple phases. Specifically, testing during implementation is key in making timely fixes and aiming for exhaustive testing. Because we had a shortened implementation time, we did not have the time we would have liked to ensure that our test cases were complete.

In relation to communication, our project required us to work with multiple platforms (Python IDE and Slack API) which necessitated that we have clear communication and documentation for how these technologies worked and interacted with each other. These interactions included the passing of parameters between functions/platforms, reading from a CSV file, and taking user input. Having clear documentation of the expected input/output of each function allowed different programmers to code interrelated functions without ambiguity. Clear documentation also enabled us to easily translate our diagrams (component and use case) and functional requirements into code. As we learned about in class, the outputs of the previous phases of the SDLC are meant to facilitate future phases, and we benefitted from that by creating and using our documentation during implementation.

The final major takeaway was the importance of allowing enough time to have a thorough implementation process. As previously discussed, we would have liked to have been able to have more time to conduct more testing during implementation. More thorough testing resolves issues in real-time before they can become more costly further in the SDLC. Additionally, if we had more time for implementation we would have more easily been able to execute all of the intended features of our software. We also may have been able to create a larger set of questions to diversify the potential content for students.

Overall, this project embodied many of the key concepts we had discussed in class. Having the opportunity to apply these concepts to a project reinforced the concepts of testing, communication, documentation, and time constraints. We also experienced the challenges of system heterogeneity, shortened development time, and integrating components from different programmers. Because of the content we had covered in class, we were aware that these are common difficulties during software engineering. This allowed us to be conscious of these possible constraints and attempt to work around them.

**Source Code**