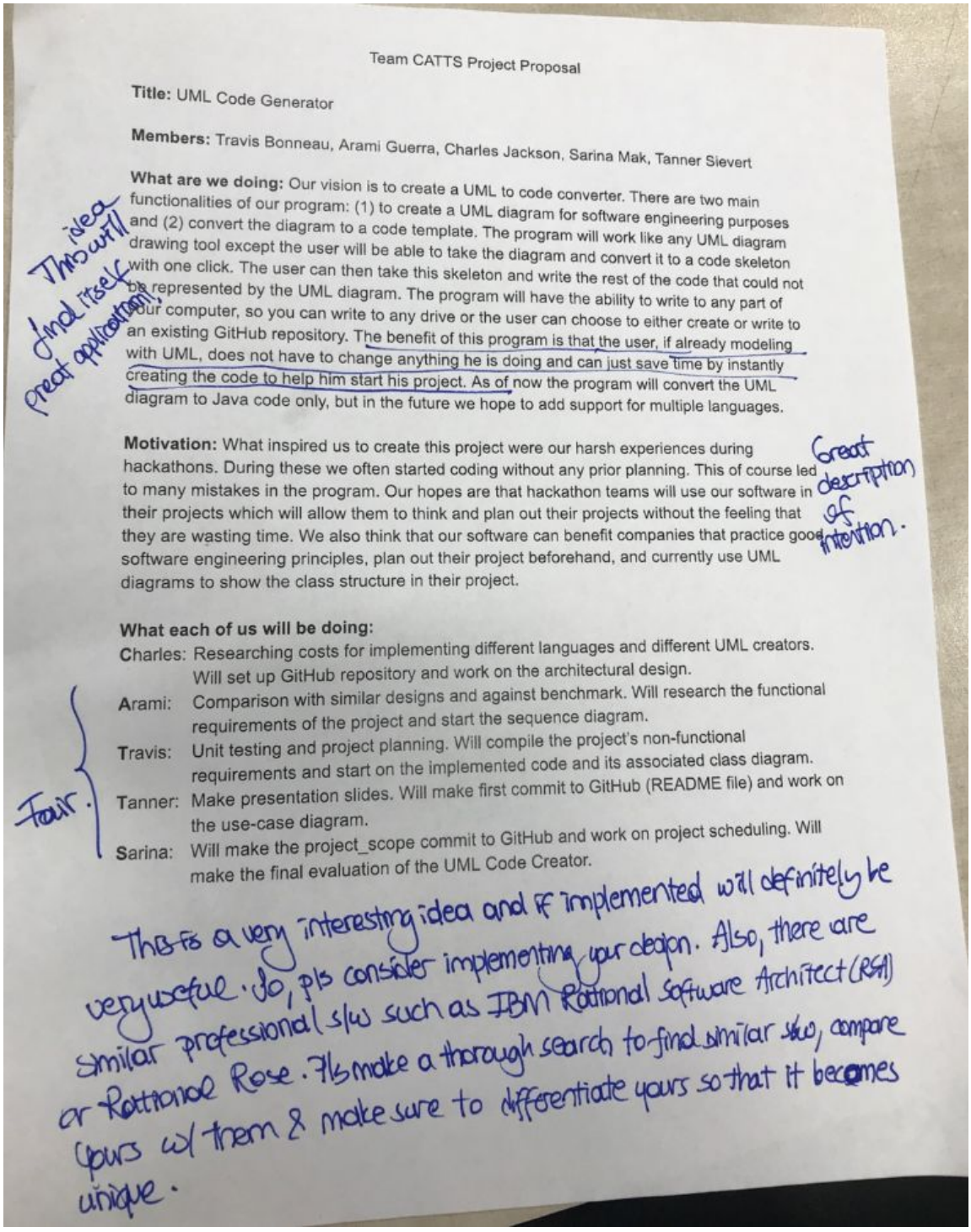
CS 3354 Software Engineering

Final Project Deliverable 1

UMLER
(UML ConvertER)

Travis Bonneau
Arami Guerra
Charles Jackson
Sarina Mak
Tanner Sievert
Kota Naga Tejaswi LNU

1. Final Project draft description containing instructor feedback.



Team CATTS Project Proposal

Title: UML Code Generator

Members: Travis Bonneau, Arami Guerra, Charles Jackson, Sarina Mak, Tanner Sievert

What are we doing: Our vision is to create a UML to code converter. There are two main functionalities of our program: (1) to create a UML diagram for software engineering purposes and (2) convert the diagram to a code template. The program will work like any UML diagram drawing tool except the user will be able to take the diagram and convert it to a code skeleton with one click. The user can then take this skeleton and write the rest of the code that could not be represented by the UML diagram. The program will have the ability to write to any part of your computer, so you can write to any drive or the user can choose to either create or write to an existing GitHub repository. The benefit of this program is that the user, if already modeling with UML, does not have to change anything he is doing and can just save time by instantly creating the code to help him start his project. As of now the program will convert the UML diagram to Java code only, but in the future we hope to add support for multiple languages.

This will find itself great application.

Motivation: What inspired us to create this project were our harsh experiences during hackathons. During these we often started coding without any prior planning. This of course led to many mistakes in the program. Our hopes are that hackathon teams will use our software in their projects which will allow them to think and plan out their projects without the feeling that they are wasting time. We also think that our software can benefit companies that practice good software engineering principles, plan out their project beforehand, and currently use UML diagrams to show the class structure in their project.

Great description of intention.

What each of us will be doing:

Charles: Researching costs for implementing different languages and different UML creators. Will set up GitHub repository and work on the architectural design.

Arami: Comparison with similar designs and against benchmark. Will research the functional requirements of the project and start the sequence diagram.

Travis: Unit testing and project planning. Will compile the project's non-functional requirements and start on the implemented code and its associated class diagram.

Tanner: Make presentation slides. Will make first commit to GitHub (README file) and work on the use-case diagram.

Sarina: Will make the project_scope commit to GitHub and work on project scheduling. Will make the final evaluation of the UML Code Creator.

Four.

This is a very interesting idea and if implemented will definitely be very useful. So, pls consider implementing your design. Also, there are similar professional s/w such as IBM Rational Software Architect (RSA) or Rational Rose. Pls make a thorough search to find similar s/w, compare yours w/ them & make sure to differentiate yours so that it becomes unique.

Addressing feedback: We will look at other projects similar to ours in order to compare performance, and make sure we have unique features over the others or our performance is better. We looked at Visual Paradigm's UML to Code and Code to UML converter, which can run in either an IDE or a desktop application and supports a wide variety of languages such as C#, Java, DLL, .NET, and Perl. We want to incorporate all of these features into our program, but the difference with ours is that it will be open-source and

free for use by the public. Our target audience includes students who may not be able to afford paid software, like Visual Paradigm, whose cheapest package costs $99. Visual Paradigm also has a Community Edition that is free for non-commercial use, but our program will be free for all uses. We also aim to target object oriented programming languages because they are more popular and can easily be represented by class diagrams.

2. Github Repository

   Delegation of tasks: Who is doing what

   - Sarina - created organization (3354-CATTTS) and initialized our group's repository. Added Charles (charJe) as initial collaborator with owner privileges so that he can add our remaining team members.
   - Charles - added all of the remaining team members and the TA as collaborators
   - Travis and Tanner - made first commit to our repository (README)
   - Arami and Kota - made project_scope commit
   - URL of our repository: https://github.com/3354-cattts/3354-CATTTS

3. **Which software process model is employed in the project and why.** (Ch 2)

   We decided to do an evolutionary process model, specifically Prototyping. We chose prototyping because this is a very user-involved program, so the more prototypes we can get out and start testing, the faster we can see what we need to focus on, what bugs we need to fix, and what features we should add. By doing this, we get to have a feel for the system and we can more easily say what we need to add, remove, or change.

4. **Software Requirements**

   a. **Functional requirements.** To simplify your design, please keep your functional requirements in the range minimum 5 (five) to maximum 7 (seven). (Ch 4)

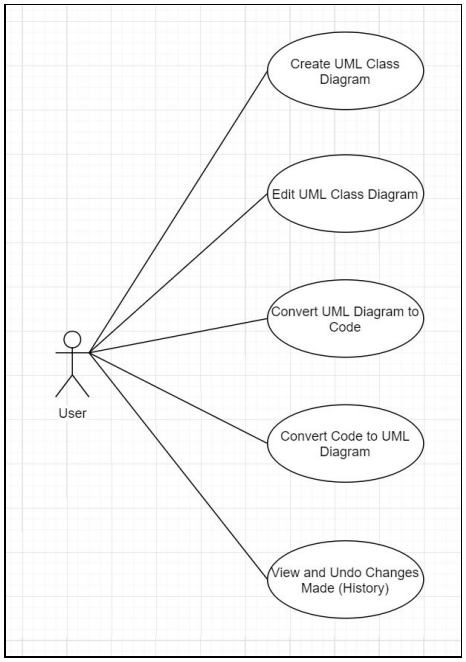
      1. Have code skeleton be well integrated and usable
      2. Have code skeleton be easy to read and understand
      3. Have the ability to incorporate multiple programming languages
      4. Allow creation of UML diagram in software
      5. Allow minor error correction and trouble-shooting options
      6. Allow for history of UML and code to review previous versions

   b. **Non-functional requirements** (use all non-functional requirement types listed in Figure 4.3 - Ch 4)

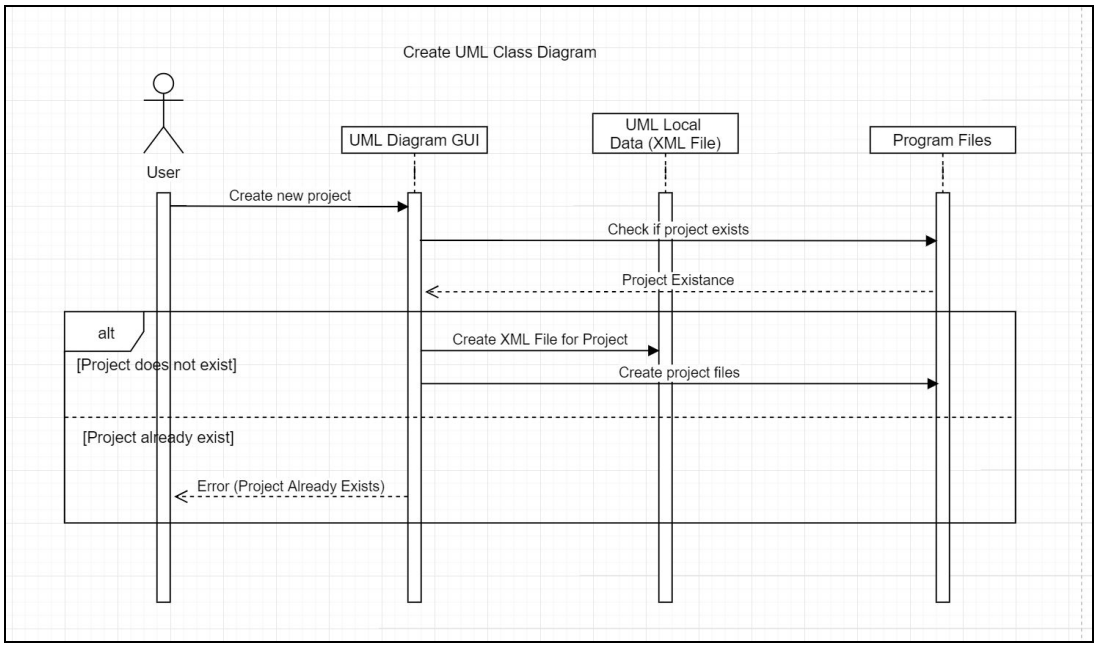
      - Product Requirements
        - (Usability Requirements) Be able to work on a variety of operating systems (e.g. Mac, Windows, Linux)
        - Efficiency Requirements
          - (Space Requirements) Program should allocate memory efficiently
          - (Performance Requirements) Return code skeleton in 10 seconds or less
        - (Security Requirements) Program should ensure no major security flaws in skeleton code, and use private variables and methods where possible.

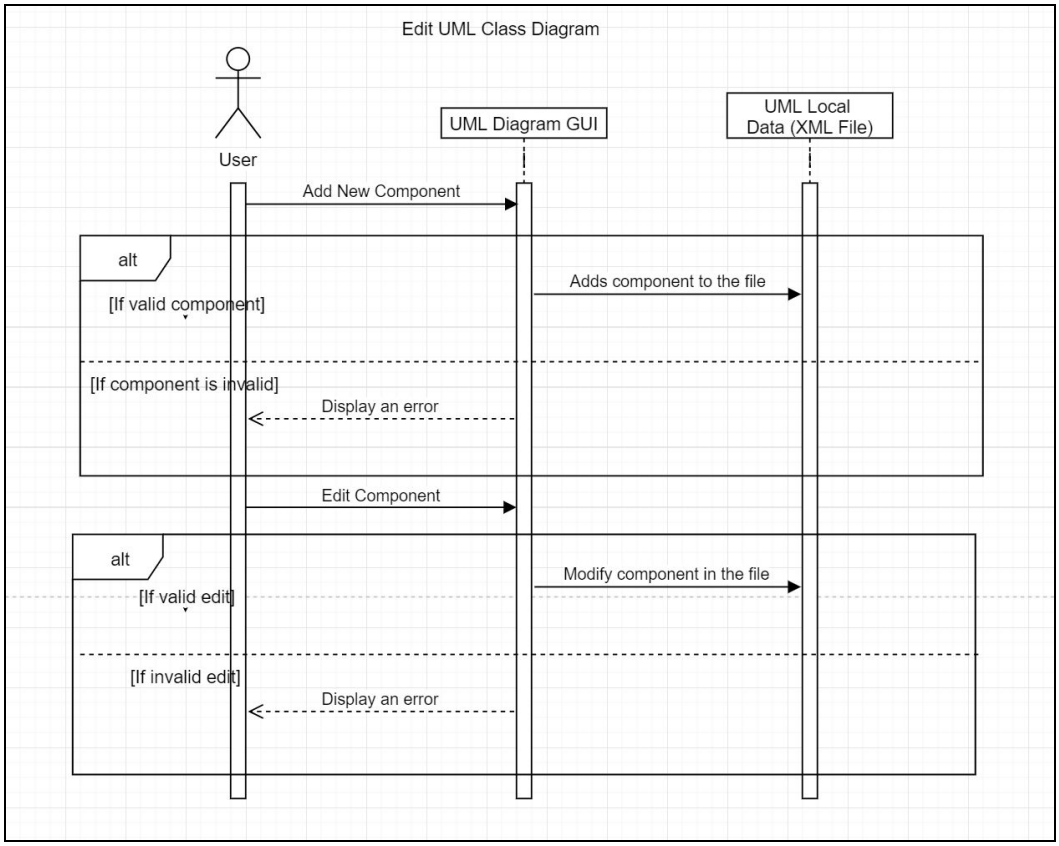
    - (Dependability Requirements) Program should recover from minor errors and throw/catch exceptions when possible
- Organizational Requirements
    - (Environmental Requirements - assuming it means where we are developing the program) Program will be developed on a Linux Ubuntu 18.04 environment
    - (Developmental Requirements) Program should be developed and structured in a simplistic manner to aid in quick and efficient development
    - (Operational Requirements) User will need to have access to a computer that is connected to the internet in order to easily push updates
- External Requirements
    - (Regulatory Requirements) Program should have a clear terms and conditions page and also state which open-source license it is under, specifying the grounds for its use and distribution
    - (Ethical Requirements) Program should not be used in the development of any system meant to create harm (e.g. weapons system, spyware)
    - Legislative Requirements
        - (Safety/Security) Program should offer “safe mode” where skeleton code is encrypted using RSA encryption.
        - (Accounting Requirements) Program should not be too costly, in monetary terms, to develop or maintain, as the final program will be free to all users. This is also a small project, so a group of 6 people should be able to develop and maintain the project.

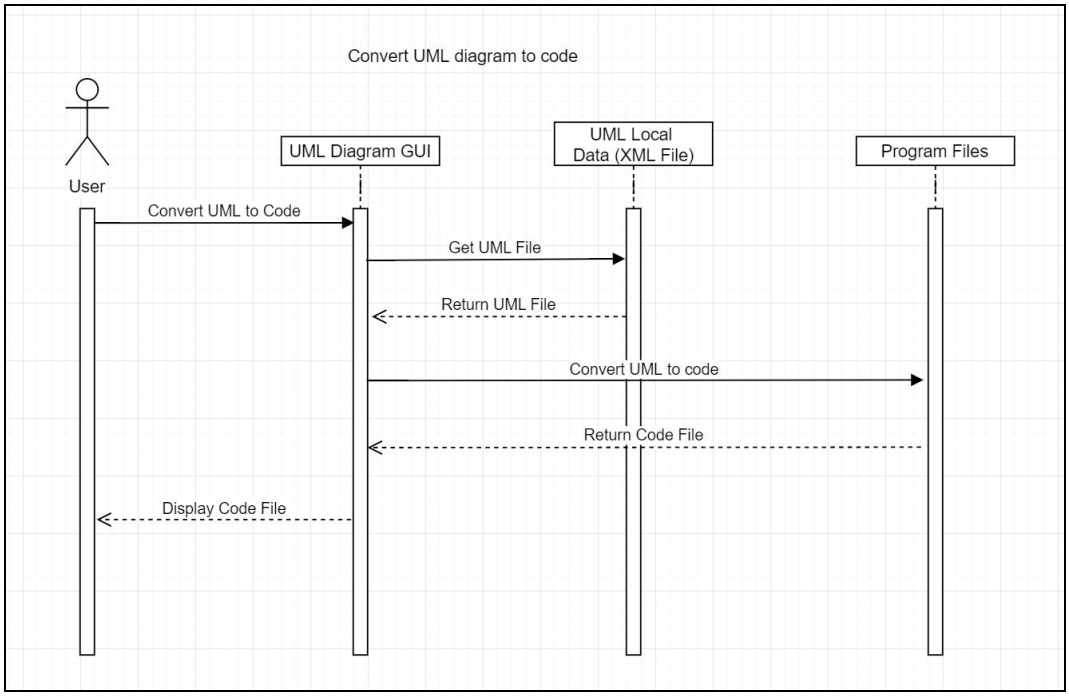
5. **Use case diagram** – Provide a use case diagram (similar to Figure 5.5) for your project. Please note than there can be more than one use case diagrams as your project might be very comprehensive. (Ch 5 and Ch 7)

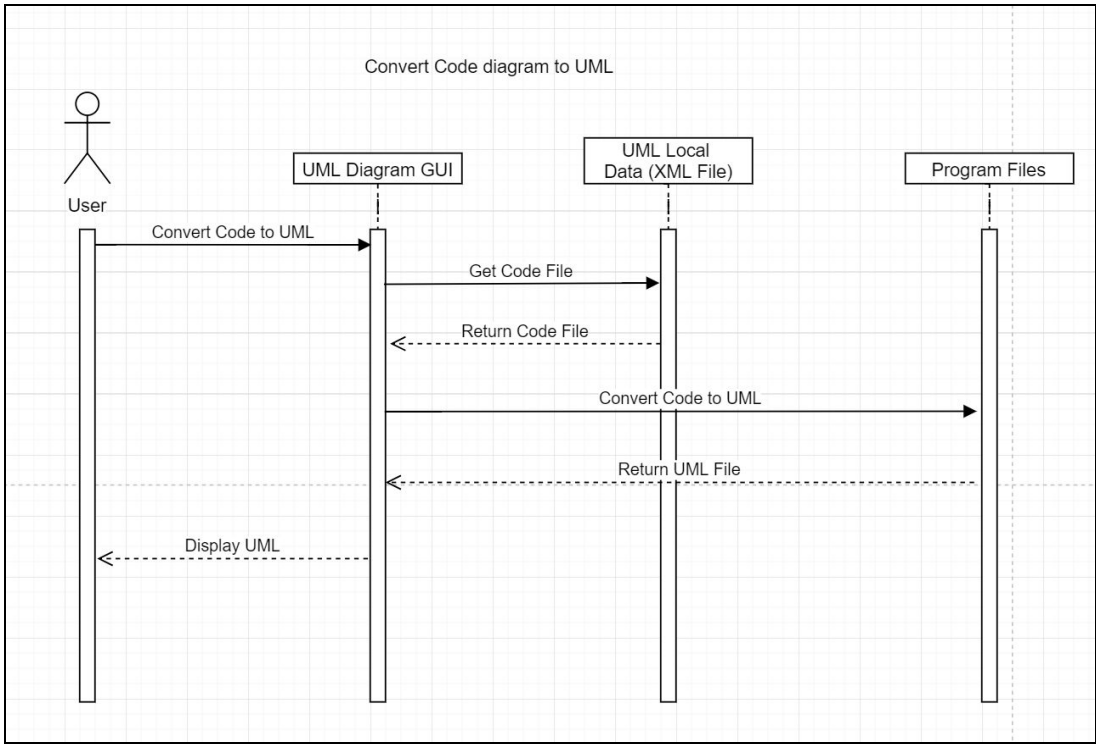


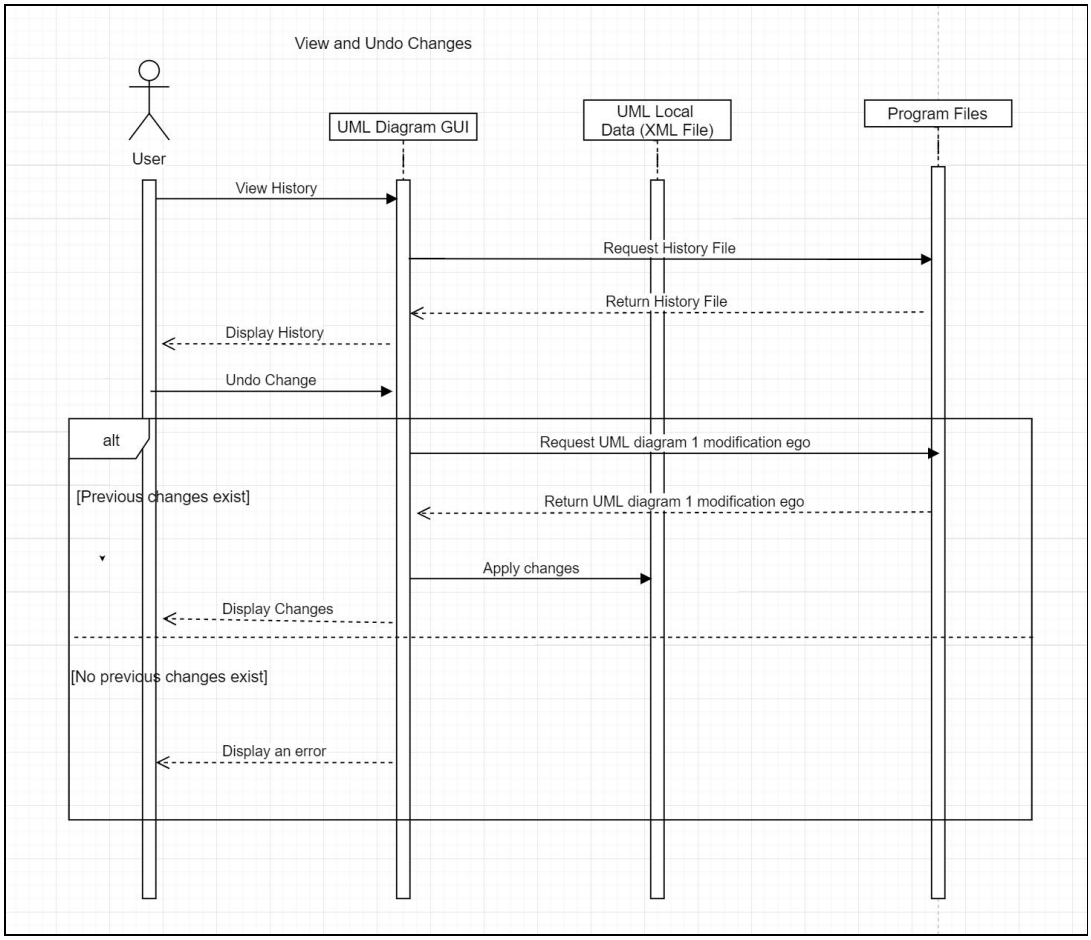
6. **Sequence diagram** – Provide sequence diagrams (similar to Figure 5.6 and Figure 5.7) for each use case of your project. Please note that there should be an individual sequence diagram for each use case of your project. (Ch 5 and Ch 7)



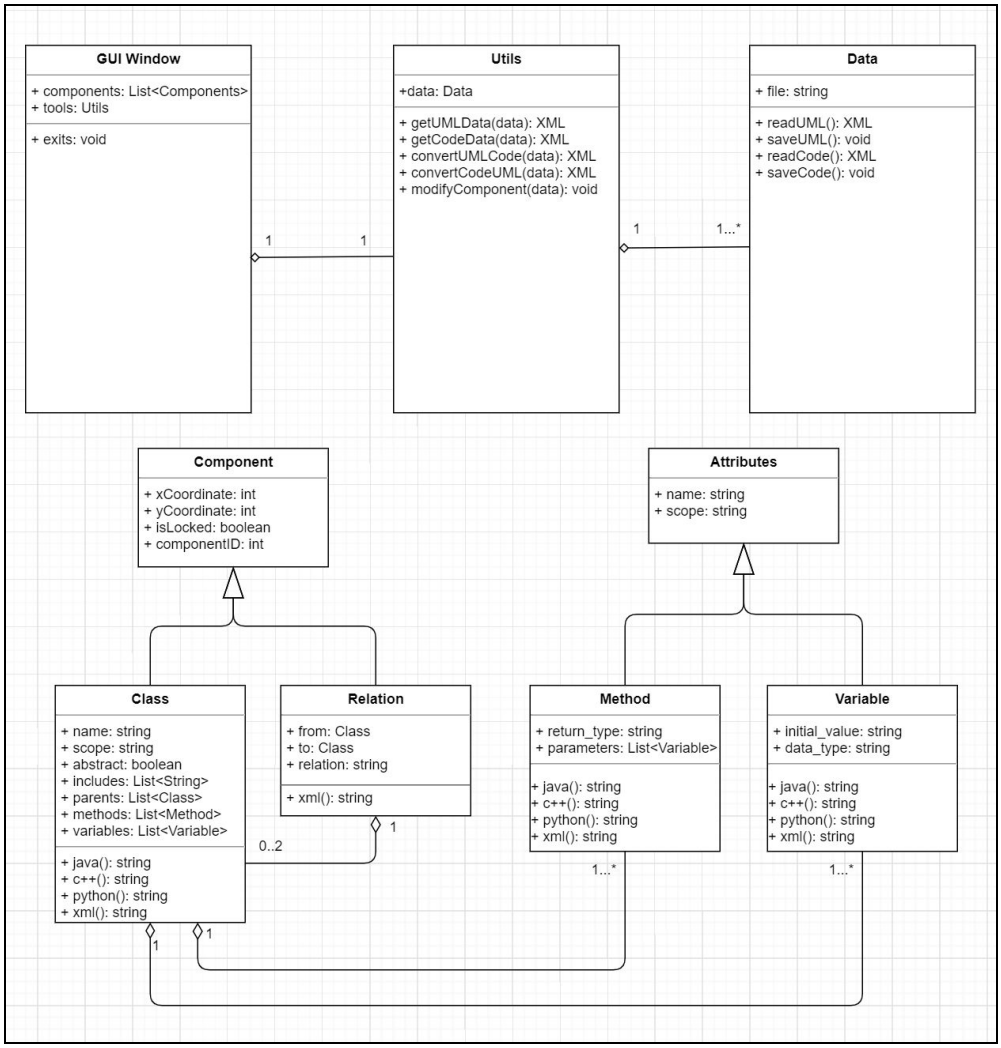

Edit UML Class Diagram
User
UML Diagram GUI
UML Local Data (XML File)
Add New Component
alt
[If valid component]
Adds component to the file
[If component is invalid]
Display an error
Edit Component
alt
[If valid edit]
Modify component in the file
[If invalid edit]
Display an error


Convert UML diagram to code
User
UML Diagram GUI
UML Local Data (XML File)
Program Files
Convert UML to Code
Get UML File
Return UML File
Convert UML to code
Return Code File
Display Code File


Convert Code diagram to UML
User
UML Diagram GUI
UML Local Data (XML File)
Program Files
Convert Code to UML
Get Code File
Return Code File
Convert Code to UML
Return UML File
Display UML


View and Undo Changes
User
UML Diagram GUI
UML Local Data (XML File)
Program Files
View History
Request History File
Return History File
Display History
Undo Change
alt
Request UML diagram 1 modification ego
[Previous changes exist]
Return UML diagram 1 modification ego
Apply changes
Display Changes
[No previous changes exist]
Display an error

7. **Class diagram** – Provide a class diagram (similar to Figure 5.9) of your project. The class diagram should be unique (only one) and should include all classes of your project. Please make sure to include cardinalities, and relationship types (such as generalization and aggregation) between classes in your class diagram. Also make sure that each class has class name, attributes, and methods named (Ch 5).



8. **Architectural design**

We choose the Model-View-Controller (MVC) pattern because it is the most effective architecture pattern for this project. Originally, we had decided upon the Layered pattern but after further analysis, now believe that the MVC would be a better fit. This architectural design fits perfectly with our program, because the view corresponds to the GUI, the model corresponds to the local data files, and lastly the control layer corresponds to the program files. It is also described as a system that should be "used when there are multiple ways to view and interact with data" which is true - we want to be able to manipulate the UML and be able to see its effects on the code, and vice versa. The Repository structure would not work for our project because we do not need a central data source. The Client-Server pattern also would not be relevant to our project because we will not need our clients to access services on servers, unless we make our project into

a web application. The Pipe and Filter pattern also would not work for this project because we are not performing data processing. In general, our project will require a lot of of user interaction, which the Pipe and Filter pattern does not support. The Layered pattern would be a good pattern for our project but it is more focused on creating a secure system, which is not a priority for our project.