Object Oriented Software Engineering based on UML (thanks NYU)

The following is a simplified summary of the process of going from requirements to object-oriented coding, using UML. The following steps are not always performed in this order, and are often iterated many times as the design is refined.

1. Perform a use case analysis to determine product requirements

2. Identify the conceptual (domain) classes that participate in your use cases. These are real-world entities that would be involved in performing the tasks of your use case if it were performed manually, such as customer, database, cash register, statement, etc. Often the secondary actors give rise to classes that are needed to interface with the external entity.

3. Identify any software classes (not real-world) that would help organize the tasks to be performed. These computer software classes may have no real-world counterpart. For example you might include a session manager class to supervise the user interaction, or you may need classes to perform data manipulations.

4. Assign responsibilities to classes. Think about how to assign responsibilities in such a way as to simplify your design. You will need to determine what classes have the responsibility of creating and destroying other classes. Make use of inheritance whenever appropriate.

5. Construct interaction diagrams (sequence diagrams or collaboration diagrams) for your use cases. You may find that it is useful to add additional classes to perform the use cases.

6. Construct a class diagram using the classes you have identified. Include all the associations that are needed for your interaction diagrams to work. Supply any operations and attributes needed to implement your interaction diagrams.

7. Construct state and/or activity diagrams as needed to work out the details of any complex behaviors.

8. Look at your design for possible improvements to the object-oriented design, for example making more use of inheritance, and changing class responsibilities to decrease coupling or increase cohesion. Modify your UML diagrams whenever you add new capabilities that were not in the original diagrams.

9. Generate class definitions (i.e. header files) for the classes you have defined.

10. Implement the methods needed, again making changes to your UML diagrams as you identify new operations and interactions that were not previously in the diagrams.

Case Study: Mini-Moodle

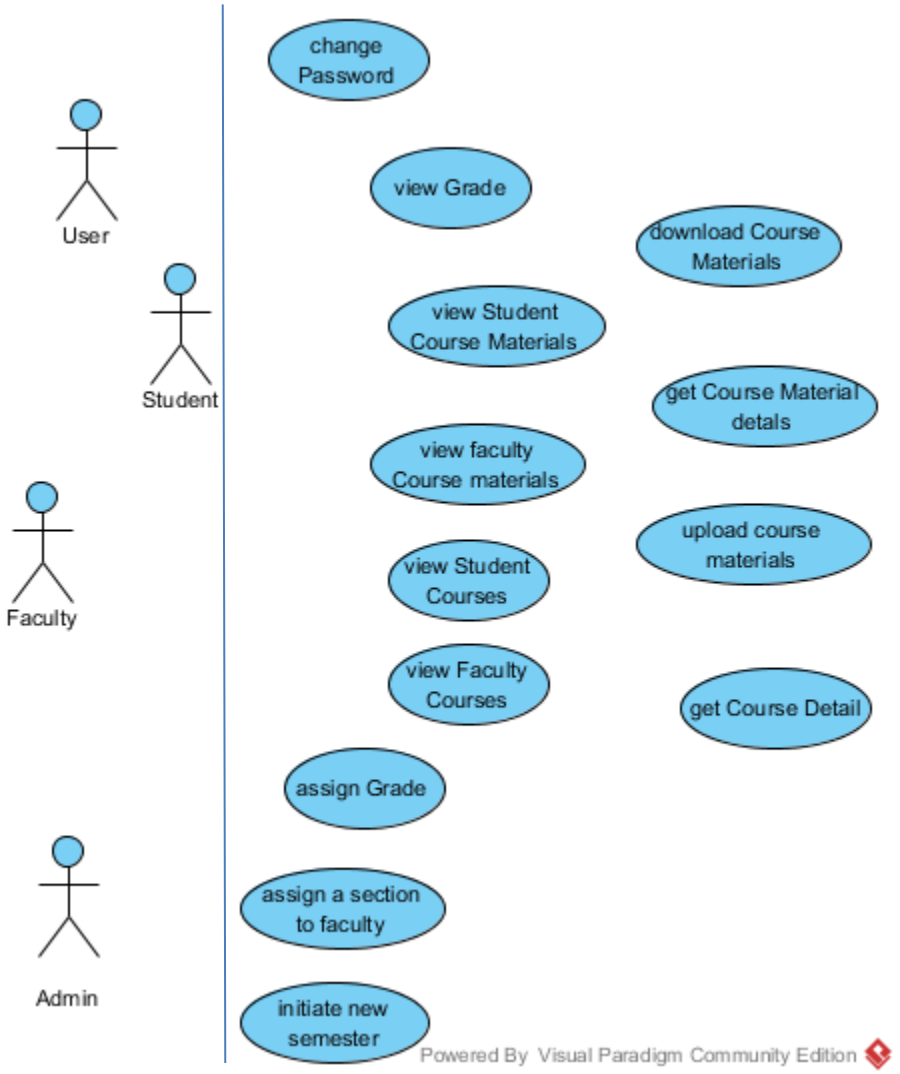
A small educational institution wishes to develop a minimal CMS. In the first iteration they just want a web application where faculty can upload course materials for a course section and a student can download the materials from the website. Assume that there may be more than one course offering (sections) available for any course described in the catalogue [for example; CSCI 437 is a Course in the catalogue and more than one Course Offerings (sections) may be available in fall 2015]. A faculty member may teach many courses (or many sections of the same course) and a student may be enrolled in many courses. There may be many students enrolled in a course section, but only one faculty can teach a course offering at a particular semester. For each course offering, a faculty member may upload more than one course material. A faculty may also enter a grade for a course which a student can see. The system can use the existing data base. The admin need to assign course sections to the faculty.

[The above description is not complete. You may make any valid assumption. But clearly explain your assumptions]

Your tasks (you may do them in different order, iteratively and incrementally). You may work in groups but everyone needs to keep the documents in order. Use a case tool to draw diagrams.

1. Identify actors. Draw usecase diagram
2. Write usecase descriptions; Rank usecases
3. Draw activity diagrams
4. Give important security requirements. Give any other key non-functional requirements.
5. Create Prototypes (may be paper based or GUI based)- use it to refine the above steps (your instructor is the Project Owner)
6. Draw the Analysis level class diagram - Entity classes and their Relationships (associations only)
7. Draw the Analysis level class diagram (complete - with key attributes also)
8. Draw ERD (first iteration) / Create a list of tables
9. Create acceptance test cases
10. Create a database; include sufficient data for acceptance test cases.
11. Create use case realizations (at least for log-in, , upload, download, enter-grade, view-grade)
12. Draw interaction-sequence diagrams
13. Draw collaboration diagrams
14. Draw state transition diagrams
15. Write operation specifications(at least for log-in, view-courses, view-course-materials, upload, download, enter-grade)
16. Write unit test cases. Create integration plan.
17. Code and produce test report
18. Complete first iteration (log-in, upload and download)

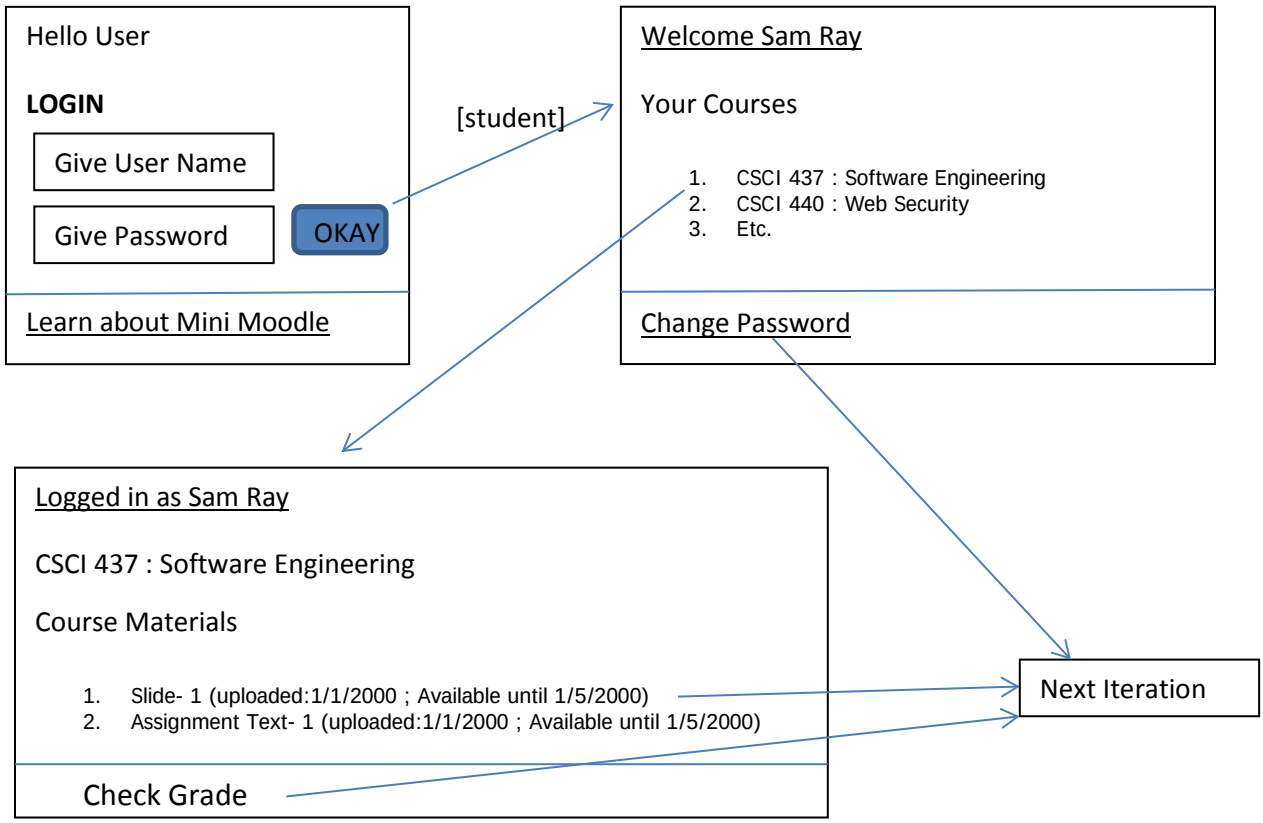
Usecase Diagram



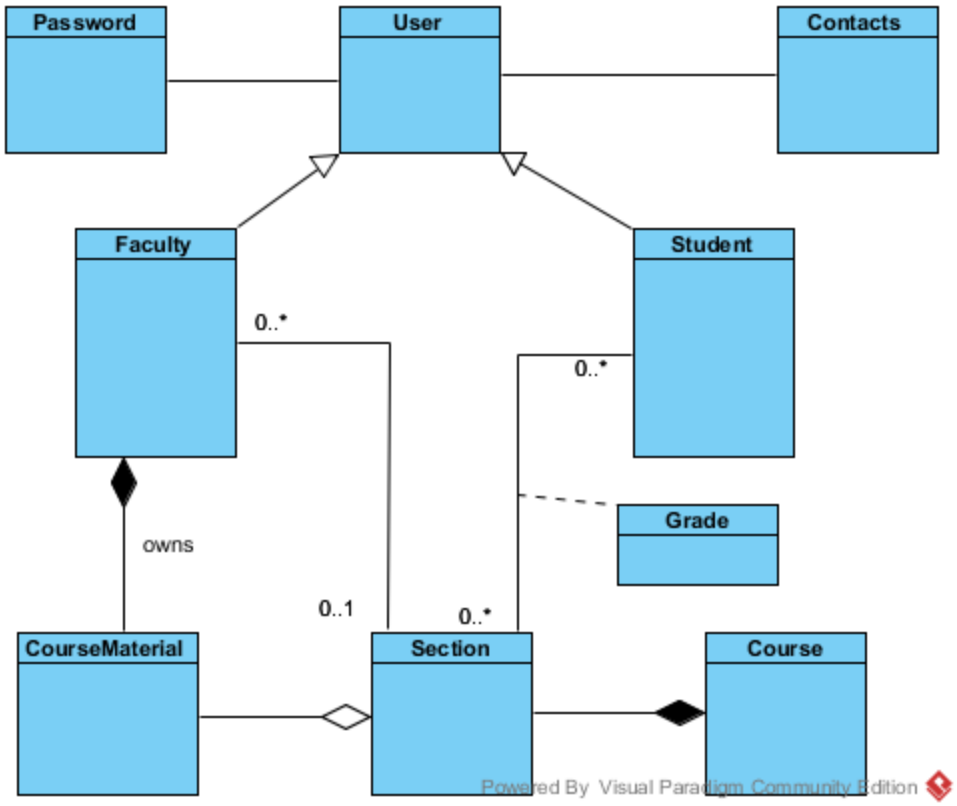
[If you haven't submitted your version before]

1. Complete the above Use case Diagram.
2. Write use case descriptions.
3. Draw Activity Diagrams
4. Rank usecases

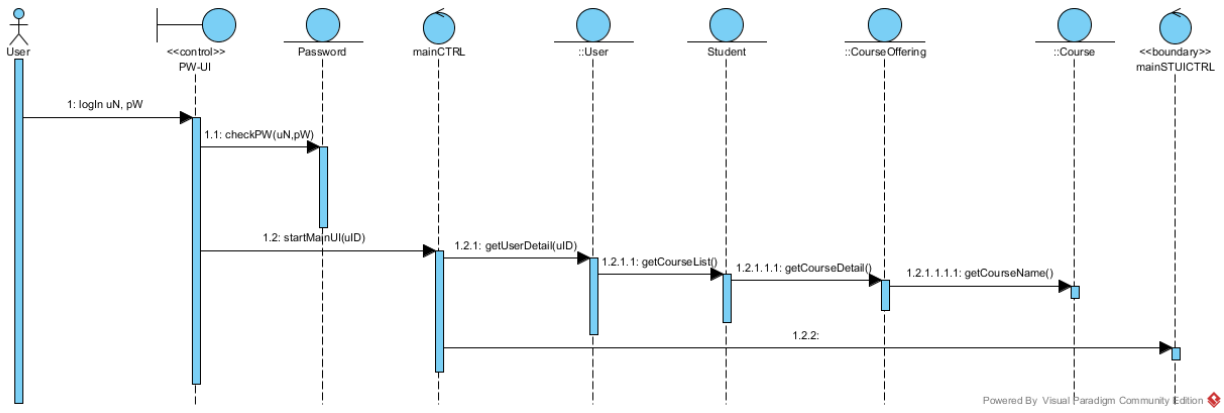
Prototypes for Student interactions



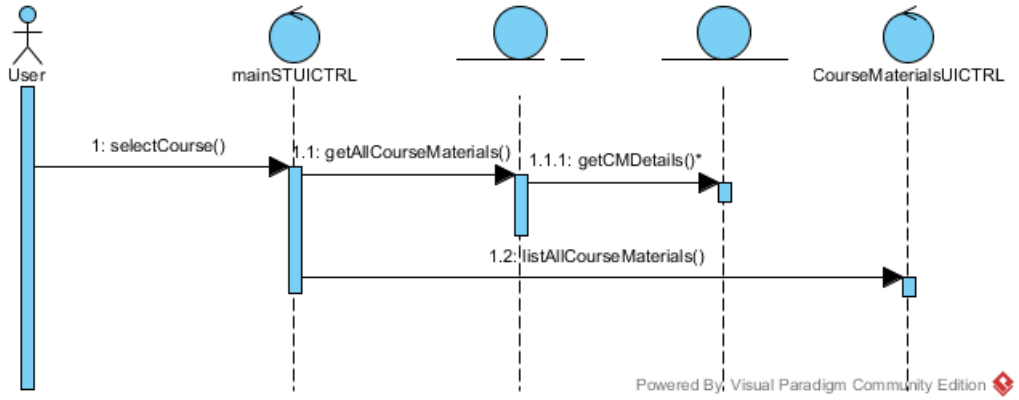
5. Draw prototypes for Faculty & Admin interactions.
6. Complete Class Diagram (insert key attributes)


Password
User
Contacts
Faculty
Student
0..*
0..*
Grade
owns
0..1
0..*
CourseMaterial
Section
Course
Powered By Visual Paradigm Community Edition

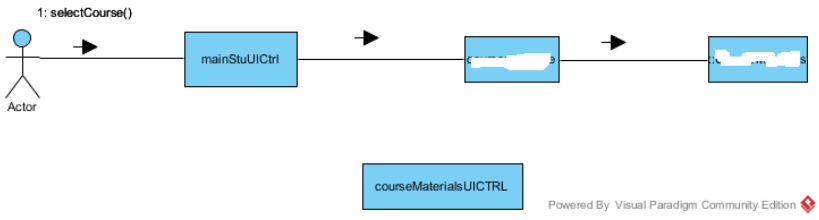
LOGIN/ View-Student-Courses Sequence Diagram



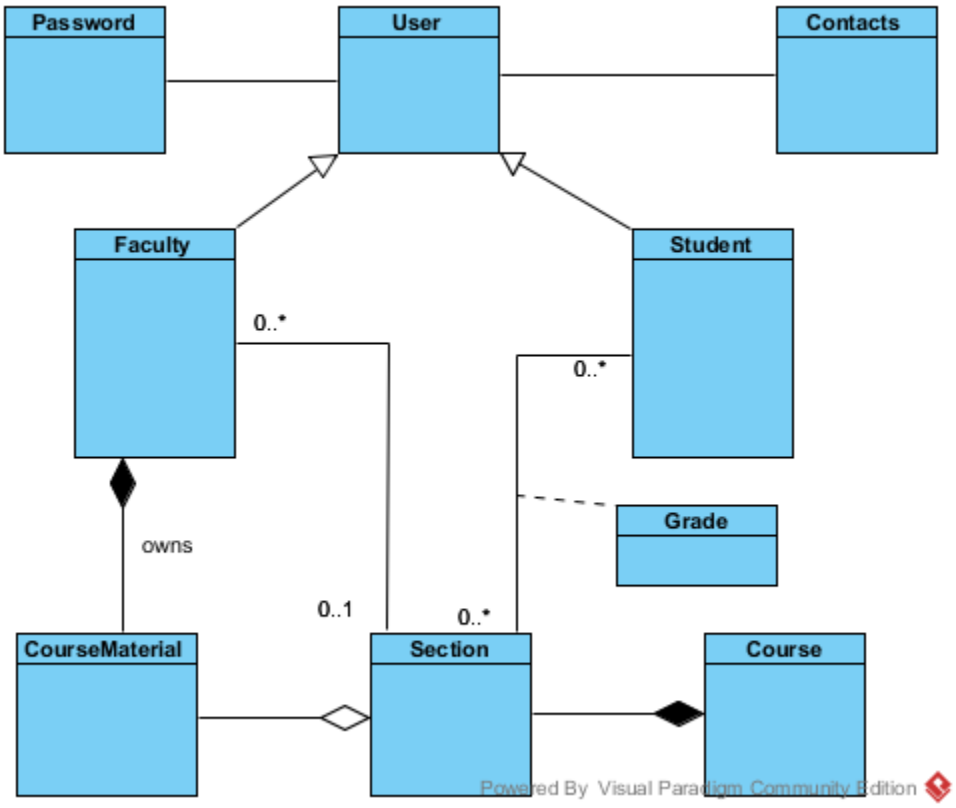
7. Draw Collaboration Diagram for LOGIN/View-Student-Courses (Communication Diagram)
8. Insert suitable class names for the following Sequence Diagram for View-Student-Course-Materials



9. Complete Collaboration Diagram for View-Student-CourseMaterials (Communication Diagram)



10. Draw the relevant sequence diagrams and collaboration diagrams for faculty usecases.
11. Complete Class diagram (insert methods).



12. Faculty needs to a make a course material visible sometime after she upload it. She may later hide it. The material will be archived for four years before deleted. Draw a State Transition Diagram.
13. Write operation specifications.