UML and OOD

Sunday, August 13, 2023 9:20 PM

Why Study UML?

1. Mastering a Programming language is not enough
2. Be able to design a software system.
3. Understand and apply OO-principles
4. Describe your design using a Standard language
5. UML - common diagramming techniques
6. Used at many companies and enterprises

Tool

1. Startuml (http://staruml.io) : to create UML design

Development Methodologies

1. Waterfall Model
    a. requires you to have a detailed plan before you start coding.
    b. Linear Model
    c. defines development steps/phases
        i. Collect & analyze requirement:
            1) The expected functionality of the future application must be Clarified with stakeholders
            2) Details must be documented thoroughly
        ii. Architecture definition
            1) What packages or components will form our system?
            2) What are the fundamentals types of each component?
            3) How do these types interact with each other to achieve the required functionality?
            4) How do these types interact with each other to achieve the required functionality?
            5) Is software secure?
            6) How about performance?
            7) How does our s/w respond to errors?
            8) How do we handle edge cases?
            9) Should we extend our system in the future?
            10) What third-party components do we use?
        iii. Implementation
        iv. Verification
            1) Testing based on
                a) Functional
                b) Performance
                c) Security
                d) Usability
        v. Maintenance
            1) Fix defects
            2) Small enhancements
            Note: Avoid making substantial changes during the maintenance phase.
    d. The development process flows in cascades
    e. Each development phase requires the finish of previous one.
    f. Ex: Life-control systems, Medical System, Military System
    g. Use the Waterfall if the requirements are clear and won't change frequently.
2. Agile : Works well where expectation can change rapidly.
    a. Agile values
        i. Individuals and interactions over processes and tools.
        ii. Working s/w over comprehensive documentation.(Document when it provides value)
        iii. Customer collaboration over contract negotiation. (Contract can't be avoided)
        iv. Focus on partnership.
        v. Responding to change over following plan
        vi. Do only as much planning as needed.(Detailed planning not required)
    b. Welcomes changes even at the later part of development cycle.
    c. Idea behind agile is that we can provide functional s/w iteratively instead of delivering the entire project all at once.
    d. The work is broken up into shorter chunks called sprints.
    e. The sprint is usually two to four weeks long.
    f. At the end of each sprint, the team should deliver a version that's an improvement over the previous sprint's outcome.
    g. This interactive approach provides an opportunity to frequently review the product that's being developed.
    h. Stakeholders have a chance to evaluate the software and provide their feedback early on, rather than waiting for the final pr oduct to be delivered.
    i. Works best in situations where the requirements can't be defined upfront.
    j. Higher customer satisfaction.
    k. SCRUM and KANBAN are examples of discrete methodologies that implement the agile approach.
    l. Iterative and incremental development, constant communication and collaboration and adaptability to change.
3. Choosing AGILE or WATERFALL depends on use case
    a. If customer is able to decide what are his requirements then Waterfall model else Agile.

NOTE: None of these systems precisely describe every step of s/w development process

History of Programming

1. Non-structured Programming(1950)
    a. Code consisted of sequentially ordered instructions.
    b. Each statement would go in a new line(Numbered code lines)
    c. Spaghetti code: Complicated, difficult to understand and maintain.
2. Structured Programming(1960)
    a. Breaks code in logical steps.
    b. Relies on subroutines
    c. Ex: C(1969)
    d. Improves code readability.
    e. Reduce development time
3. OOP(1980)
    a. Split apar the program into self-contained objects.
    b. Object: Acts as a program
    c. Operates on its own data.

Object

1. OOP is organized around objects
2. We can describe objects using properties.
3. Objects have their identity.

Class

1. We need a class to create a object.
2. A object is a variable of datatype class.
3. Class is a blueprint of an object.

Abstraction

1. Describe complex problems in simple terms by ignoring details.
2. Focus on essential qualities, discard unimportant ones.

Encapsulation

1. We encapsulate something to protect it and keep its parts together.
2. In OOP, this means packing together our properties and methods in a class.
3. It also means hiding the gears and the levers
4. Object should only reveal the essential features this is called data hiding.

Inheritance

1. Inheritance means code reuse i.e. reusing an existing class implementation in a new class.

Polymorphism

1. The condition of occurring in several different forms
2. Method Overriding
    a. Subclasses can provide a specialized implementation of a method defined in the superclass.

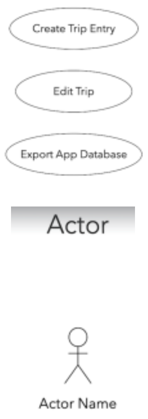
Object Oriented Analysis and Design

1. Collect Requirements(A thing that is needed or wanted): Define what the systems needs to do. Identify constraints and boundar ies.
    a. Identify the problems we want to solve.
    b. Clarify the functionality required to solve the problems
    c. Document important decisions.

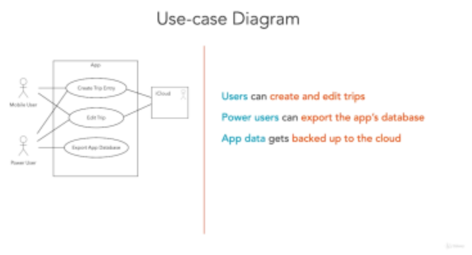
d. The features of system have
    i. Functional requirements
        1) Represents the features
        2) Define how to react to an input.
        3) Determine the expected behavior.
    ii. Non-functional requirements
        1) Not directly related to the features of the system
        2) Ex: Performance, Scalability, Reliability, Legal requirements, Documentation.
e. How to collect requirements:
    i. Write them down.
    ii. Tools/Systems
f. Ex: The app must store travel expenses organized by trips
    i. Functional Requirements
        1) Each trip must have a home currency.
        2) The default currency is fetched from the phone's settings.
        3) User setting need to override the default home currency.
        4) Expenses can be entered in any of the supported currencies.
        5) The app must automatically convert the amounts to the home currency.
    ii. Non-functional Requirements
        1) The app must run on IOS 9 and newer versions.
        2) The app must avoid unnecessary network roundtrips to reduce data roaming fees and preserve battery.
        3) The app must include the support email and the link to the app's website.

2. Describe the software system
    a. Describe the system from the user's point of view. One way of doing this is
        i. Use Case: Avoid technical details
            1) Title: Short, Descriptive use case title.
            2) Actor: User
            3) Scenario: Explain how the software works in this scenario.
                a) Ex: Title: Create New Trip
                    Actor: Mobile User.
                    Scenario:
                    - The user can initiate the creation of a new trip from the main screen.
                    - The title is mandatory. All the other settings are optional.
                    - Optionally, the user can write a short description and set a start and end date for the trip.
                    - The app assigns a default home currency based on the phone's settings. Users can override the default home currency with any of the supported - currencies.
                    - The app allows setting a budget for the trip. This setting is optional
                    - Also the user can assign a custom thumbnail to a trip.
                    - The user can save the trip or cancel the trip creation process.
        ii. User Story: Very brief description of a feature.
                a) Format: As a <type of user>, I want <some goal> so that <some reason>.
                    i) Ex: 1. As a user, I want to add notes to my expenses, so that I can identify them later on.
                        2. As a power user, I want to retrieve the app's database file, so that I can inspect it on any computer.
            1) If you can't describe a user stories in one or two sentences you may need to split it into multiple smaller user stories. This larger user stories are known as Epic.
            2) Epic describe a bigger chunk of functionality. Should be split into smaller user stories.
                a) Ex: Epic: As a traveler, I want to track my expenses while abroad, so that I don't exceed my budget.
                    User Story #1 As a user, I want to create new trips, so that I can track each of my travel separately.
                    User Story #2 As a business traveler, I want to tag my business trips, so that I can separate them from my private travels.
            3) Use user stories to ignite discussions instead of describing details.
    a. Create wireframes and prototypes if needed.
3. Identify the classes
4. Create diagrams
    a. Class diagram
    b. Sequence diagram
    c. UML diagram: Graphical notation used to describe object-oriented systems
        i. Use-case diagram
        ii. Class diagram
        iii. Sequence diagram

UML Basics and Fundamental Diagram Type
1. Graphical notation used to communicate the design of software systems.
2. UML has many Diagrams
    a. Use Case Diagram: Describe the functional model of the system i.e. the functionality of system from user's point of view.
        i. Visualize Functional Requirements of the system
        ii. Use case diagrams show groups of related use cases.
        iii. Result is an overview of the system that may include several written use cases.
        iv. To represent a use case draw oval in the middle of the screen and put the title of the use case in it.
        v. Eg: Example of use cases from Travel expense app
            1) Create Trip Entry
            2) Edit trip
            3) Export App Database



            4) We usually draw the primary actors on the left side and the secondary ones on the right side of the Use Case diagram.
            5) Use case diagrams provide a clear way to communicate the high-level features and the scope of the system.
            6) You can quickly tell what our system does just by looking at this Use Case diagram.
            7) The UML Use Case diagram includes other artifacts and relationships between use cases.
            8) Provide an easy-to-understand overview of the features of our system.
            9) Use Case diagrams are not a replacement for written use-case descriptions.
            10) Use case descriptions include more information to ensure that we don't miss any important details or requirements.



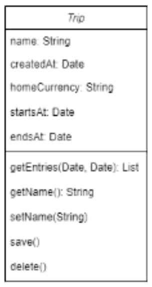
Use-case Diagram

Users can create and edit trips

Power users can export the app's database

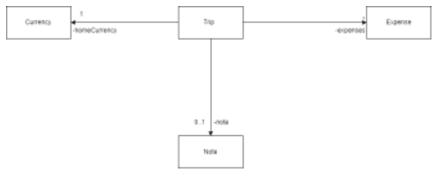
App data gets backed up to the cloud

b. Structural Diagram: To decide the structure of the system.
   i. Class Diagram
      1) Used to describe the structure of the system in terms of objects, attributes, operations and relations.
      2) After identifying the entities that form our system, we start creating class diagrams for each of them.
      3) A class is represented as a rectangle with three compartments.
         a) First compartment contains class. Class Naming convention: Should be a noun in singular. Starts with uppercase letter. If Mul tiple words present in name then UpperCamelCase should be used.
         b) Second compartment contains the attributes. Attributes should follow lowerCamelCase. Specify the type of attribute
         c) Third compartment contains the methods. Method names should be in lowerCamelCase. Specify method arguments. Add colon after c losing parenthesis followed by return type



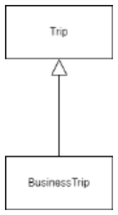
      4) Visibility(Who can access data and methods)-> Expose only as much as needed and hide everything else.
         a) (+) Public -> can be also used by code outside of the object.
         b) (-) Private -> can only be accessed within the defining class.
         c) (#) Protected -> accessible from defining and child classes
         d) (~) Package -> available within its enclosing package.

      5) Relationship between classes
         a) Association (has-a)
            i) Solid line that ends with an open arrowhead represent association.
               Ex: Functional Requirements
                  As a traveler, I want to track my expenses while abroad, so that I don't exceed my budget.
                  A trip will usually have multiple expenses.
               i. We can represent the multiplicity of associated objects as follows.
               ii. (*): A trip can have zero or more expenses.
               iii. 1 : A trip must have exactly one home currency.
               iv. 0 to 1: A trip may or may not have a single note.
               v. Associations can show multiplicities at both ends of the line.
               vi. The default multiplicity is one.

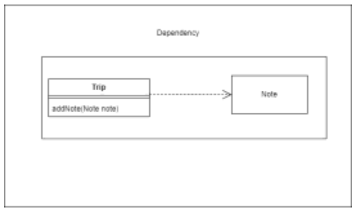


            ii) Association tells us that the classes refer to each other.
            iii) With the help of association we show that only one of the classes refers to the other one.
            iv) The arrow points to the class that's referred to by the other class.

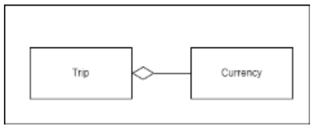
         b) Generalization(is-a)
            i) Used to express that one model element is based on another model element.
            ii) Represented as a solid line with a hollow arrowhead that points to the parent.



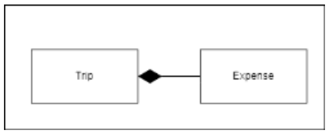
         c) Dependency(references)
            i) We talk about dependency relationship if changes in one of the classes may cause changes to the other.
            ii) Represented as dashed line that ends with an open arrowhead.
            iii) Arrow points to the dependency.
            iv) Dependency is directed relationship.
            v) Difference between Association and Dependency
               i. Association indicates that a class has attributes of the other class's type
               ii. Dependency is created when the class receives a reference to the other class, for instance, through a member function parameter.



d) Aggregation(has-a)
   i) Represents a part-whole relationship and it's drawn as a solid line with a hollow diamond at the owner's end.
   ii) Considered as redundant as it expresses the same thing as the association.



e) Composition(part-of)
   i) Stronger form of association.
   ii) Implies ownership
   iii) When the owning object is destroyed, the contained objects will be destroyed, too.
   iv) Represented as a filled diamond on the owner's end connected with a solid line with the contained class.
      i. Ex: Expenses of trip can't exist without the trip.



f) Realization(implementation behavior)
   i) Indicates that a class implements the behavior specified by another model element.
   ii) Represent a hollow triangle on the interface end connected with dashed lines to the implementer classes.
   iii) We could specify an interface to ensure that all current and upcoming trip classes provide a common set of methods.
   iv) Allows polymorphic behavior.

c. Dynamic Behaviour: Describe the system's functionality focusing on what happens and the interactions between objects.
d. Independent of any programming language.
e. Used to create a detailed blueprint of a system(Needed in waterfall model)

3. Sequence Diagram
   a. Use case and class diagrams are static diagrams for representing structure of system.
   b. UML provides dynamic diagrams to represent how objects communicate with each other.
   c. We use Sequence diagram (commonly used dynamic diagram) to describe the flow of logic in one particular scenario.
   d. Provides an overview of what is going on in particular scenario.
   e. Focus is on the most important interactions between objects.

4. Activity Diagram
   a. Used to model the flow of activities in a system and the decisions that are made along the way.
   b. Used to model workflows or business processes.
   c. Made up of nodes and flows or edges.
   d. Nodes represent actions whereas flow line show the flow of control between actions.
   e. Can also express conditional logic
5. State Chart Diagram
   a. Used to model the object's states and state transitions over its lifetime
   b. Made up of three main elements: States, Transitions and events.
   c. Represent current condition of an object.

Assignment

Start with the requirements collection phase, then write some use cases. Put together the use-case diagrams then model the static structure using class diagrams.Then, think about some of the behavioral aspects that could be modeled using sequence, activity and state chart diagrams.

If you're on a Mac, I'd suggest you check out *MindNote*. It's a great application that makes it easy to collect and organize your ideas.

Questions for this assignment

a. **Have you collected some requirements?**
   You'll need at least some draft list of requirements before you can start with the next phase.

b. **Did you map the requirements to written use-cases? Feel free to choose epics/user stories.**
   Once you have a list of requirements, you should come up with a formal list of use-cases. These short, written descriptions help you understand the requirements better. It's surprising how much you can learn by trying to consolidate your thoughts.
   If I Had More Time, I Would Have Written a Shorter Letter
   - a modern day translation of Blaise Pascal's statement.

c. **Next, create some use-case diagrams based on the written use cases.**
   i. **Focus on the main actors and use cases.**

d. **Think about the main classes you'll need for your application. Draw your class diagrams and figure out the relations between your classes.**
   i. **Check out the videos from the UML section if you need some guidance. Also, feel free to post your questions in the forum.**

e. **Use sequence, activity and state chart diagrams to express the dynamic behavior of your app.**
   i. **You don't need to cover all the details. Pick a particular scenario and describe the main objects and their interactions a sequence diagram. Then choose another workflow and represent it using activity diagrams.**
   ii. **Any objects that have multiple states? Start drawing a state chart diagram, figure out the states and the transitions.**

f. **Put it all together**
   i. **You can now create a draft design document. You should use a text editor that lets you copy and paste images.**
   ii. **Then start copying your requirements, the written use cases and the various UML diagrams. In the end, you'll have a nice booklet, an initial version of a design document.**