A class note on

Introduction to Object Oriented Analysis and Design

Definition

In general, **analysis** emphasizes an investigation of the problem and requirements of the domain, rather than a solution. Whereas, **design** emphasizes a conceptual solution that fulfils the requirements, rather than its implementation.

In **OO analysis**, emphasis is given on finding and describing the objects or concepts in the problem domain. But in **OO design**, emphasis is given on defining software objects and their collaboration for fulfilling the requirements.

**OO Concept and Principles**

- Objects exist in nature and thus OO technology reflects Real world scenario
- OO thinking is in abstraction level (should not be intoxicated with technological implementation)
- OO leads to reuse of components/sub-systems (Component Based Development model), which leads to a faster development as well as high quality programs.

**Classes and Objects**

- A Class is an OO concept that encapsulates the data and procedural abstractions required to describe the content and behaviour of some real world entity.
- The data abstraction is implemented as attributes and the procedural abstraction as operations or methods or services.
- Methods and objects are cohesive (i.e. methods can change some of the attributes of objects) whereas classes tend be decoupled from other elements of the system (classes can be communicated by other elements through methods only).

**Attributes**

- Description of classes and objects are attributes.
- Real life entities are described with words that indicate stable features (e.g. shape, weight, colour, date or birth, name etc.) [Here, features = values of domain]
**Operations, Methods, and Services**

- Algorithms encapsulated in classes or objects, which process the data, are called operations or methods or services, which are termed in conventional sense as modules.
- Each operation represents one of the behaviours of a class or an object.
- Objects are designed in such a way to be able to get a stimulus (often called message) upon receiving which they start to behave through operations.

**Messages**

- Objects interact through messages.
- A message has i) destination, ii) operation and iii) parameters.
- Messages tie an object-oriented system together.

**Encapsulation, Inheritance and Polymorphism**

- Encapsulation helps achieve information hiding, in which the internal implementation details of data and procedures are hidden from the outside world.
- Any object that sends a message to another object does not bother about the detail of internal data structures of the object thus by reducing the propagation of side effects.
- Class hierarchy maintains the mechanism such that any change in data or operations contained within a super class is immediately inherited by all subclasses. Multiple inheritances could be achieved by restructuring or overriding the existing classes.
- Polymorphism allows a number of different operations to have the same name, which in turn decouples objects from one another – making each object independent.
- Overloading concept is used for the purpose. For example the following design

```plaintext
case of graphtype:
    If graphtype = linegraph then DrawLineGraph (data);
    If graphtype = piegraph then DrawPieGraph (data);
    If graphtype = xygraph then DrawXyGraph (data);
    If graphtype = bargraph then DrawBarGraph (data);
end case;
```

can be reduced to

```plaintext
graphtype draw
```
The OO Paradigm

- OO is evolutionary (incremental, spiral, concurrent development paradigms), not revolutionary.
- Iterative process with resembling to Spiral model.
- It emphasizes the creation of classes
- The Unified Software Development Process is adopted – Rational Unified Process and UML are the software industries’ best practices nowadays.

Software Complexity

- Software might be referred to as simple or complex - depending upon their functionality and behaviour. Generally, industrial-strength software are more complex than those developed individually or by amateur developers.
- Time and space are considered to be general complexities. Additionally, maintaining integrity of hundreds of thousands of records while allowing concurrent updates and queries; or managing command and control of real-world entities like air traffic are also examples of complex systems.

Why is Software inherently complex?

The complexity can be derived from following four elements:

1. **The complexity of the Problem domain**
   - Requirements – both functional and non-functional (like usability, performance, cost, survivability, reliability etc.) - give rise to complexity.
   - Impedance mismatch between the users of a system and its developers in understanding and expressing the problem domain also increases the level of complexity
   - Change in requirements during the development phase makes the system more complicated.

2. **The difficulty of managing the developing process**
   - Engineering the illusion of simplicity is challengingly difficult and tricky too. Systems with a millions of lines of codes are huge in terms of space and even decomposing them might produce hundreds or thousands of modules, which are hard for any developer or understand fully.
o Establishment of a team of software developers requires complex communication and difficult coordination particularly if they are geographically dispersed. There also is a challenge of team building as well as maintaining the unity and integrity of the design.

3. **The Flexibility possible through software**
   o The highest possible flexibility of software enables developers to express any level of abstraction, which turns out the software development work to be labour intensive work because users expect higher level abstraction be built on top of primitive building blocks.

4. **The problems of characterizing the behaviour of discrete systems**
   o In a continuous (analog) system, a state is defined by a continuous function. In contrast, a discrete system has a finite number of possible states.
   o The present state of an application is depicted by a number of variables, their current values and also by the current address and calling stack of each process.
   o In a large discrete system, there is possibility of combinatorial explosions which makes the finite number of possible states very large.
   o An event external to a system has the potential to place the system in a new state and thus making the mapping from state to state non deterministic – in worst case, creating a corruption in the system too.
   o There is no mathematical model nor do we have intellectual capacity to model complete behaviours of large discrete system.

**Attributes of a Complex System**

1. **Hierarchical**: There exist a hierarchy of systems and interrelated subsystems ultimately going down to a certain level of their own sub systems.
2. **Different level of abstraction**: The primitive components of a system are relatively arbitrary and may be seen as a higher level of abstraction to another observer.
3. **Decomposable**: A system can be broken down into definite numbers of identifiable parts, which are not completely independent. There are intra-component and inter-component interactions, which leads to the relative isolation of each component.
4. **Reuse of common patterns**: A complex system is made up of only a few kinds of different sub systems in various combinations and arrangements. Many of them have common patterns and are re-used to make a complex system.
5. **Evolutionary**: A complex system evolved over time from a number of simple working systems. Making from scratch, a complex system can never be built successfully.

**An example of Complex System: Structure of a Personal Computer**

- A device of moderate complexity, which has been evolved from small working sub systems like gate architecture, for instance.
- Composed of the same major elements – CPU, monitor, keyboard, secondary storage
- An element can be taken and decomposed further and can be studied separately.
- The collaborative activity of each major part (hierarchically arranged) causes a computer system to function.
- The hierarchy of the computer system also represents different levels of abstraction – each built upon another and in each level of abstraction, a collection of devices are found to be collaborating one another to provide functions to higher layers.
Managing Complexity

• Organised complexity can be managed more easily than disorganised complexity. If complex systems are in a standard (canonical) form i.e., includes five attributes and class-object architecture, then the complexity is said to be organised.
• There is yet another factor in complexity management – limited human capacity in dealing with the complexity.
• There are three (3) ways to resolve the complexity – decomposition, abstraction and hierarchy.

Role of Decomposition

• Divide and rule: Only a few parts need to be taken care of at a time.
• Ways of decomposing - Algorithmic and Object Oriented
  o Algorithmic Decomposition: Process and steps are considered. Functional components are structurally designed with their relationships. It mostly deal with “how” not “what”.
  o Object Oriented Decomposition: Carries out the key abstraction in problem domain by identifying objects from the vocabulary of the domain. It concerns mainly with “what”; and all “how”’s found in the vocabulary become operations of related objects.

Role of Abstraction

• Multiple levels of abstraction exist. Also, different abstractions are possible from several dimensions.
• When a higher level abstraction fails to succeed, we ignore inessential details and focus on more generalised parts.

Role of Hierarchy

• By recognising the class and object hierarchies, the study of collaboration / interaction in and among levels are made simpler.
• Discovery of common or dissimilar patterns of many objects are simplified once the hierarchy is exposed and thus the identification of object behaviours too.

Sources:

• Roger S. Pressman, Software Engineering: A Practitioner’s Approach
• Grady Booch, Object-oriented analysis and design with applications
• Craig Larman, Applying UMN and Patterns