OBJECT ORIENTED ANALYSIS AND DESIGN

BTech (IT)

Dr. M.P.VANI., Asso.Prof.

UNIT – I

STRUCTURE OF COMPLEX SYSTEMS

Example of complex systems:

1) **The structure of a personal computer. A personal computer is a device of moderate complexity**
2) **The structure of plant and animals**

   Plants are complex multicellular organism it consists of various plant organ system arises, such as complex behaviour as photosynthesis and transpiration.

Animals are multicellular, that exhibit a hierarchical structure similar to that of plants, collection of cells form tissues, tissues work together as organs , cluster of organs define systems and so on.

<u>The structure of matter:</u>

- **Astronomers study galaxies that are arranged in clusters, and stars, planets and various debris are the constituent of galaxy.**
- **Nuclear physics consists of structure of atom which are made up of electrons, protons, neutrons and other particles are formed from more basic components called quarks**

The structure of Social Institutions

- **The structure of social institutions group of people join together to accomplish tasks that cannot be done by individuals .**
- **Multinational corporations contain companies, which in turn are made up of divisions, which in turn contain branches , which in turn contain local offices and so on.**

<u>The five attributes of Complex Systems</u>

1. **Frequently complexity takes the form of hierarchy, whereby a complex system is composed of interrelated subsystem that have in turn their own subsystem and so on, until some lowest <u>level </u>of elementary components is received**

2. **The choice of what components in a system are primitive is relatively artbitary and is largely up to the discretion of the observer system.**
3. **Intracomponent linkages are generally stronger than intercomponent linkages .**
4. **Hierarchy systems are usually composed of only a few different kinds of sybsystems in various combinatons and arrangememnts.**
5. **The complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from a scratch never works and cannot be patched.**

<u>Decomposing complexity :</u>

- **When designing a complex software system it is essential to decompose it into smaller and smaller parts .**
- **Each of these smaller parts may then refine independently.**

<u>Algorithmic Decomposition :</u>

- **It is a top down structural design**
- **We approach decomposition as a simple matter of algorithmic decomposition.**
- **Each module denotes major step.**

COMPLEXITY:

The Inherent complexity of software

- A child learning how to read, blood cells rushing to attack a virus are some of the objects that involve awesome complexity.
- Some software systems are not complex.
- They are largely forgettable applications that are specified, constructed, maintained and used by the same person or professional developer working in isolation
- We are much more interested in challenges of developing of what we call it as industrial strength software.
- It consists of very rich set of behaviour.

- **It is intensively difficult**

Why is software inherently complex?

- **The complexity of software is an essential property not an accidental one**
- **It consists of four elements**

1. **Problem domain**
2. **Difficulty of managing the development process**
3. **Flexibility possible through software**
4. **Problems of characterizing the behaviour of discrete systems.**

1. **Complexity of problem domain**
   - **It involves inescapable complexity.**
   - **It involves contradictory requirement**

     For e.g system of multiengine aircraft, cellular phone switching system or autonomous robot. The functionality of such system is difficult.
2. **Difficult of managing the development process**
   - **The fundamental task of software devolopment team is engineer the illusion of simplicity**

- **We strive to write less code by inventing clever and powerful mechanism that gives the illusion of simplicity, as well as by reusing the existing framework and design.**

3. **Flexibility possible through software:**
   - **A home building company generally does not operate it,s own trees farm which to harvest for lumber**
   - **For software industry such flexibilities are common**
   - **Software development remains a labour intensive business**

4. **Characterizing the behaviour of discrete system**

- **I we toss a ball into air we can predict its path based on the laws of physics**
- **Discrete system by their very nature have finite number of states.**

Organized and disorganized complexity

The canonical form of a complex system:

- **The discovery of common abstractions and mechanisms greatly facilitates our understanding of complex systems**

- E.g with just a few minutes of orientation an experienced pilot can step into multiengine Jet aircraft.

<u>**Bringing order of chaos**</u>

- When desiging a complex software system it is essential to decompose it into smaller and smaller parts each of which may then refine independently

<u>**Algorithmic decomposition**</u>

Top down structural design

- Each module in the system denotes major step in some overall process

Object oriented decomposition

- **Decomposing the system according to the key abstractions in the problem domain rather than decomposing the problem into steps such as Get formatted update and add checksum.**
- **Objects are identified such as master file and check sum which derive directly from the vocabulary of the problem domain.**

<u>Advanatages of Object oriented decomposition</u>

1. **Reduces risk**
2. **Resilent to change**
3. **Reduces the risk of building complex software system**

<u>The Meaning of design</u>

The purpose of design is to careate a clean and relatively internal structure, sometimes also called an Architecture.

<u>The importance of Model building</u>

- **It has a broad acceptance among all engineering disciplines**
- **Model building appeals to the principle of decomposistion, abstraction and hierarchy**

- Each model within a design describes specific aspects of the system under consideration

Elements of analysis and design

- System involve incaremental and iterative process
- Notation: The language for expressing each model
- Process: The activities leading to the orderly construction of the system model
- Tools: The artifacts that eliminate the tediousness of model building and enforcing rules about the models themselves, so that errors and inconsistencies can be expressed.

Typing

- It is a precise characterisation of structural or behavioural properties which a collection of all entities share
- Without type checking the program in most languages can crash at runtime.
- In most systems the edit-compile-debug cycle is so tedious that early error detection is indispensable.
- Type declaration helps to document the program.
- Most compilers can generate more efficient object code if types are declared

e.g of typing:

static and dynamic binding:

Static binding are early binding which means all variables and expressions are fixed.

Dynamic binding means the type of all variables and expressions are not known until runtime.

<u>Concurrency:</u>

Concurrency is the property that distinguishes an active object from one that is not active.

For e.g.

If two active objects try to send messages to a third object we must be certain to use some means of mutual exclusion, so that the state of the object being acted upon is not corrupted.

Persistence:

Persistence is the characteristic of anj object through which it always exists. i.e., even when the creator of an object no longer exists, the object continuous to exist.

Benefits of object model

1. **The main advantage of object oriented system is that you can build on what you already have.**

2. **Reuse code and develop more maintainable systems in a shorter amount of time.**
3. **The use of the object model produces systems that are build upon stable Intermediate form, which are more resilent to change.**

<u>Classification</u>

Classification is the means whereby we order knowledge.

<u>Importance of proper classification:</u>

1. **Classification and object oriented development .**
   - **Identification of classes and objects is the hardest part of object oriented analysis design.**

- Identification involves both discovery and invention.
- Discovery- is used to identify the key abstraction and mechanism that form a vocabulary of a problem domain

  Invention- we develop generalized abstraction as well as new mechanisms that specify how object collaborate.
- Discovery and invention are both problem classification.
- Classification fundamentally a problem of finding sameness
- When we classify we try to group things that have common structure
- Classification helps to identify generalization, specialization, and aggregation hierarchy among classes.
- Classification also helps in making decision about modularization.

2. **Identifying classes and objects**

The three general approaches to classification are

- **Classical categorization**
- **Conceptual clustering**
- **Prototype theory**

- **Classical categorization : here all the entities that have a given property in common, form a category**

  For e.g married people form a category one is either married or not and the value of this property is sufficient to decide which group a particular person belongs.

- **Classical approach uses related properties.**

- **Conceptual clustering: It is the most modern variation of the classical approach.**

- **It is used to explain how knowledge is represented.**
- **It represents more of probabilistic clustering of objects.**
- **It is classified by focussing on best fit.**

3. **Prototype theory: in prototype theory we group things according to the degree of relationship to concrete prototype**

- **Class of object is represented by a prototypical object and the object is considered to be a member of this class if and only if it resembles this prototype.**

OBJECT ORIENTED ANALYSIS AND DESIGN

To model the world is done by discovering the classes and objects that form the vocabulary of the problem domain.

Classical approaches: The number of methodologies have proposed various sources of classes and objects. They derive primarily from the principles of classical categorization.

- **Tangible things- Cars, pressure sensors**
- **Roles- mother, teacher, politician**

- **Events- landing, interrupt, request**
- **Interaction- loan, meeting, intersection**

Behaviour analysis: These approaches are more related to conceptual clustering

- **We form a class based upon groups that have similar behaviour.**
- **This approach deals with analysing what takes place in the system these are the system behaviours.**
- **System behaviour is closely related to function point.**
- **A function point is a outwardly visible and testable behaviour of the system**
- **It can include output, inquiry, input, file or interfaces.**

Domain analysis:

- it is an attempt to identify the objects, operations and relationship needed
- Use Case analysis, CRC cards, formal English description.

Structured analysis:

- It is a front end to object oriented design
- We start with the essential model of the system
- It is described by DFD diagram
- It consists of external entities, data stores, control stores, control transformation
- Candidate class derive from two sources data flows and control flows.

Object: it represents entity the entities are as follows:

- Physical entity
- Conceptual entity
- Software entity
- An object ids an entity with well defined boundary and identify that encapsulate state and behaviour state is represented by attribute relationship .
- Behaviour is represented by operations, methods and state machine.

Object model:

Abstraction: it is the essential characteristics of an entity it distinguishes from all other kinds of entity.

- It defines the boundary relative to the perspective viewer.
- It is not a concrete- denotes the ideal essence of something.

  e.g. student, course, professor

Encapsulation:

- Hides implementation from clients. Clients depends on interface.
- It is achieved through information binding.

Modularity:

- Breaks up something complex into a manageable pieces
- Helps people understand complex systems

  e.g. course registration system, billing system, students management system.

Hierarchy

- It is a ranking or ordering of elements

  e.g. single inheritance, multiple inheritance

Object oriented programming

Object oriented programming is a method of implementation in which programs are organized as a co-operative collection of objects, each of which represent instance of some class and whose classes are all members.

<u>Object:</u> An object is a real world element each object has

- **Identity – that distinguishes from other objects in the system**
- **State – It determines characteristic properties of an object.**
- **Behaviour – that represents externally visible activities**

State: The state of an object includes all of the properties of the object pluscurrent value of each these properties

e.g.

structure of personal record

```
struct personalrecord
{
 Char name[100];
Int id;
Char dept[10];
```

```
 Cloat salary;
 };
```

Encapsulate the state of an object

```
Class personalrecord{
Public:
Char* empname()const;
Int empid() const;
Char*empdept() const;
Protected:

Char name[100];

Int id;

Char dept[10];

Float salary;

};
```

Behaviour : behaviouris how an object acts and reacts in terms of its state changes and message passing.

```
Class que{

Public:
```

```
Que()
Que(const que&);
Virtual que();
Virtual que& operator=(const que&);
Virtual int operator=(const que&)const;
Int operator !=(const que&)const;
Virtual void clear();
Virtual void append(const void*);
Virtual void pop();
Virtual void remove(int at);
Virtual int length()const;
Virtual int ISEpty() const;
Virtual Iconst void* front()const;
Virtual int location(const void*);
Protected:
…..
……
};
```

- In the above e.g class uses the common C idiom of setting and getting items via void*
- It provides an albstraction of heterogeneous que

- The clients can append objects of any class to a que object but this jis not safe because the client must remember the class of the object placed in the que.
- The use of void* means we cannot rely upon the action of ques destructor (*que()) to destroy the elements in the que.
- Since declaration represents a class not an object we have to declare the instances that clients can manipulate

```
Que a,b,c,d;

a.append(&ram);

a.append(&sita);

a.append(&shyam);

b=a;
```

a.pop();

after executing these statement we may safely pop b three more times but may be safely poped only two times.

Unified Process:

The unified process is also called unified software development
*it is the creation of pioneers of the OOAD i.e. BOOCH, JACOSON,RAMBAUGH
*They merged together the best of their individual.

The unified process has two basic characteristics

- It is component based
- The components are interconnected via interfaces
- Unfied process extensively uses UML terminology and notations

USE CASE DRIVEN:

It is a set of actions that the users of the system perform in order to get desired work done.

- **A use case represents functional requirements .**
- **A use case model is the diagrammatic representation of the overall system requirements.**
- **The term use case driven indicates that the unified process starts with the definition of use cases, then they are transformed into design and code, and finally validation during the testing phase**

ARCHITECTURE CENTRIC:

- **It consists of operating system, hardware platform, networking protocols to be used, users development technology to be used, functional and non-functional requirements and so on.**

ITERATIVE and INCREMENTAL

- Big and complex projects can last several months.
- The whole software system is divided into smaller sub-projects.
- Each iteration takes into account two important aspects of software development additional use cases that extend the use of the system and the risks that can cause problem later on.

- The iteration take the software system as it existed in the previous iteration as the input and transform it into better software system.
- Iteration may not necessarily always add code. Instead it may **replace code.**

UNIFIED PROCESS LIFE CYCLE

1. **Inception**
2. **Elaboration**
3. **Construction**
4. **Transition**

1. **Inception: it is the first phase it seeds idea for the development for entering into Elaboration phase.**

2. **Elaboration: it is the second phase of the process. The system requirement range from general vision statement, precise** evaluation criteria, each specifying functional and non- functional behaviour and basis for testing.

3. Construction: It is the third phase , it is a software brought from an executable architectural base line ready to be transitioned to the user community, here it is constantly re-examined against the business needs of the project.

4. Transition: It is the fourth phase , here the software is given in the hands of the user community, rarely the software process end here,

during this phase the system is continuously improved , bugs are eradicated, some features which did not exist earlier are added.