Module-3

The Object Oriened Design Process

Object Oriented Design (OOD) has a defined process, to create performance oriented as well as high quality software. The process of Object-Oriented Design (OOD) is as follows:

1. To design classes and their attributes, methods, associations, structure, and even protocol, design axiom is applied. This includes
    - The static UML class diagram is redefined and completed by adding details.
    - Attributes are refined.
    - Protocols and methods are designed by utilizing a UML activity diagram to represent the methods algorithm.
    - If required, redefine associations between classes, and refine class hierarchy and design with inheritance.
    - Iterate and refine again.
2. Design the access layer.
    - Create mirror classes i.e., for every business class identified and created, create one access class.
3. Identify access layer class relationship.
4. Simplify classes and their relationships. The main objective here is to eliminate redundant classes and structures.

- **Redundant Classes:** Programmers should remember to not put two classes that perform similar translate requests and translate results activities. They should simply select one and eliminate the other.
- **Method Classes:** Revisit the classes that consist of only one or two methods, to see if they can be eliminated or combined with the existing classes.

5. Iterate and refine again

6. Design the view layer classes.

    - Design the macro level user interface, while identifying the view layer objects.
    - Design the micro level user interface.
    - Test usability and user satisfaction.
    - Iterate and refine

7. At the end of the process, iterate the whole design. Re-apply the design axioms, and if required repeat the preceding steps again.

Concepts of Object-Oriented Design:

1. In Object Oriented Design (OOD), the technology independent concepts in the analysis model are mapped onto

    - implementing classes,
    - constraints are identified,
    - and the interfaces are designed,

    which results in a model for the solution domain.

2. In short, a detailed description is constructed to specify how the system is to be built on concrete technologies.

3. Moreover, Object Oriented Design (OOD) follows some concepts to achieve these goals, each of which has a specific role and carries a lot of importance in design. These concepts are defined as follows.

- **Encapsulation:** This is a tight coupling or association of data structure with the methods or functions that act on the data. This is basically known as a class, or object (object is often the implementation of a class).
- **Data Protection:** The ability to protect some components of the object from external entities. This is realized by language keywords to enable a variable to be declared as private or protected to the owning class.
- **Inheritance:** This is the ability of a class to extend or override the functionality of another class. This child class or sub class has a whole section that is the parent class or superclass and then it has its own set of functions and data.
- **Interface:** A definition of functions or methods, and their signature that are available for use as well as to manipulate a given instance of an object.
- **Polymorphism:** This is the ability to define different functions or classes as having the same name, but taking different data type.

Domain Modelling:

- Domain modeling is known as conceptual modeling.
- A domain model is a representation of the concepts or objects appearing in the problem domain.
- It also captures the obvious relationships among these objects.

Examples of such conceptual objects are the Book, BookRegister, MemeberRegister, LibraryMember, Librarian etc. for Library information system.

- The recommended strategy is to quickly create a rough conceptual model where the emphasis is in finding the obvious concepts or objects expressed in the requirements while deferring a detailed investigation.
- Later during the development process, the conceptual model is incrementally refined and extended.

Three types of objects that can be identified during domain analysis are as follows

- **Boundary objects**
- **Controller objects**
- **Entity objects**

Boundary objects:

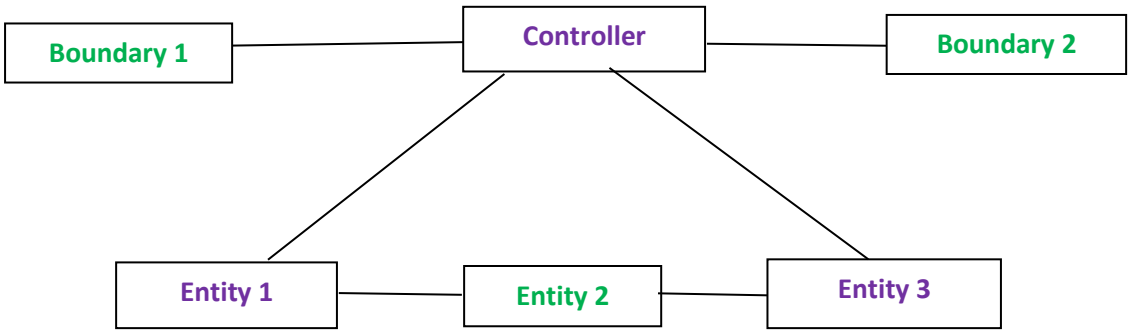
- The boundary objects can be systematically identified from the use case diagram
- The boundary objects are those with which the actors interact. These include *screens, menus, forms, dialogs, etc*.
- The boundary objects are mainly responsible for user interaction. Therefore, they normally do not include any processing logic. However, they may be responsible for validating inputs, formatting, outputs, etc.
- A recommendation for the initial identification of the boundary classes is to define one boundary class per actor/use case pair.

Entity objects:

- The entity objects can be systematically identified from the use case diagram
- These normally hold information such as data tables and files that need to outlive use case execution, e.g. Book, BookRegister, LibraryMember, etc.
- Many of the entity objects are “dumb servers”. They are normally responsible for storing data, fetching data, and doing some fundamental kinds of operation that do not change often.

Controller objects:

- The controller objects coordinate the activities of a set of entity objects and interface with the boundary objects to provide the overall behavior of the system. The responsibilities assigned to a controller object are closely related to the realization of a specific use case.
- The controller objects effectively decouple the boundary and entity objects from one another making the system tolerant to changes of the user interface and processing logic.
- The controller objects embody most of the logic involved with the use case realization (this logic may change time to time).
- A typical interaction of a controller object with boundary and entity objects is shown in the following figure.



- Normally, each use case is realized using one controller object. However, some use cases can be realized without using any controller object, i.e. through boundary and entity objects only. This is often true for use cases that achieve only some simple manipulation of the stored information.

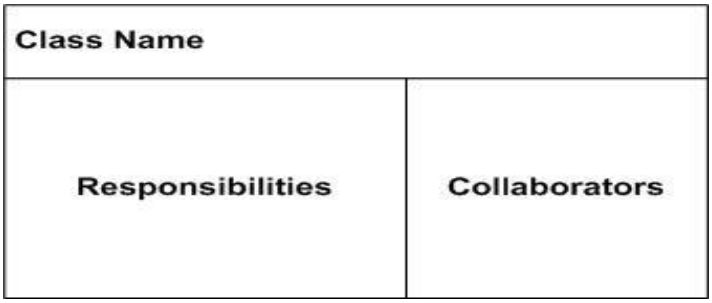
CRC Model

- **CRC** model stands for **C**lass **R**esponsibility **C**ollaborator model
- CRC (Class-Responsibility-Collaborator) technology was pioneered by Ward Cunningham and Kent Becka at the research laboratory of Tektronix at Portland, Oregon, USA.
- It is a tool and method for systems analysis and design
- It is part of the OO development paradigm
- It is highly interactive and human-intensive
- It results in the definition of objects and classes

CRC card

CRC model basically represented by CRC cards or a CRC model is a collection of CRC cards that represent whole or part of an application or problem domain.

CRC card is a standard index card that has been divided into three sections, as shown below



Class:

A **class** represents a collection of similar objects. An object is a person, place, thing, event, concept, screen, or report that is relevant to the system at hand. The name of the class appears across the top of the card.

For example, in a shipping/inventory control system with the classes such as **Inventory Item**, **Order**, **Order Item**, **Customer**, and **Surface Address**

Responsibility

A **responsibility** is anything that a class knows or does. Responsibilities are shown on the left-hand column of a CRC card.

For example, in the shipping/inventory control system customers have names, customer numbers, and phone numbers. These are the things that a

customer knows. Customers also order products, cancel orders, and make payments. These are the things that a customer does. *The things that a class knows and does constitute its responsibilities.*

Collaborator

Sometimes a class will have a responsibility to fulfil, but will not have enough information to do it. When this happens, it has to **collaborate with other classes** to get the job done. The collaborators of a class are shown in the right-hand column of a CRC card.

For example, in the shipping/inventory control system an **Order** object has the responsibility to calculate it's total. Although it knows about the **Order Item** objects that are a part of the order, it doesn't know how many items were ordered (**Order Item** knows this) nor does it know the price of the item (**Inventory Item** knows this). To calculate the order total, the **Order** object collaborates with each **Order Item** object to calculate its own total, and then adds up all the totals to calculate the overall total. For each **Order Item** to calculate its individual total, it has to collaborate with **Inventory Item** to determine the cost of the ordered item, multiplying it by the number ordered (which it does know).

The following is an example CRC model for a shipping/inventory control system, showing the CRC cards as they would be placed on a deskor work table.

Note the placement of the cards: Cards that collaborate with one another are close to each other, cards that don't collaborate are not near each other

Inventory Item	
Item number Name Description Unit Price Give price	

Order	
Order number Date ordered Date shipped Order items Calculate order total Print invoice Cancel	Order Item Customer

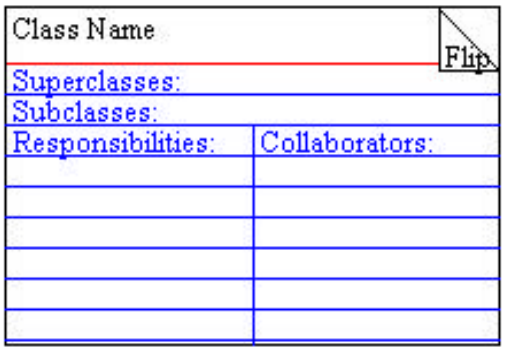
Order Item	
Quantity Inventory item Calculate total	Inventory item

Customer	
Name Phone number Customer number Make order Cancel order Make payment	Order Surface Address

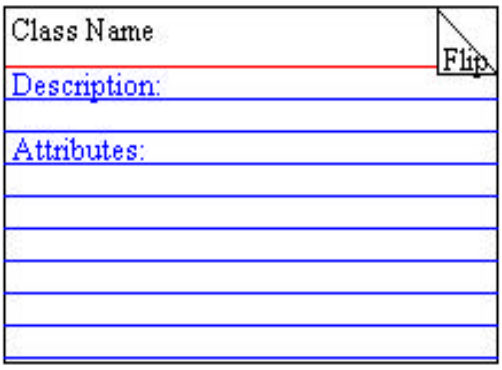
Surface Address	
Street City State Zip Print label	

(**A CRC model for a simple shipping/inventory control system)**

Sample CRC Card (Front & Back)



Front Side of CRC Card



Back Side of CRC Card

CRC card Designing and Processing

CRC Card Session

- CRC cards are index cards that are prepared one per each class.
- Cards should be physical cards, not virtual cards with size 3x5 or 4x6 inch
- CRC cards are usually developed in small group sessions
- The ideal size for the CRC card team: 5 or 6 people
- The team should be composed of
    - One or two domain experts
    - two analysts
    - an experienced OO designer and
    - one group's leader/facilitator

During the Session

- All the group members are responsible for holding, moving and annotating one or more cards as messages fly around the system.
- Group members create, supplement, stack, and wave cards during the walk-through of scenarios.
- A session scribe writes the scenarios.

Processing

1. Brainstorming
    - One useful tool is to find all of the nouns and verbs in the problem statement.
2. Class Identification
    - The list of classes will grow and then shrink as the group filters out the good ones.
3. Scenario execution (Role play)
    - The heart of the CRC card session

CRC card Strenngths or Advantages

- The cards are informal
- It provides a good environment for working and learning.
- Spreading domain knowledge
- Spreading OO design expertise
- It provides Implicit design reviews and Live prototyping
- Allow the participants to experience first-hand how the system will work
- Useful tool for teaching people the object-oriented paradigm
- It is also used to Identify holes in the requirements

Limitations or Disadvantages

- Provide only limited help in the aspects of design.
- Do not have enough notational power to document all the necessary components of a system.
- Do not specify implementation specifics.
- Cannot provide view of the states through which objects transition during their life cycle.

Further Reading (More about CRC Cards)

Example of CRC card:

Case study: *A small technical library system for an R&D organization*

- Requirement Statement
- Participants (Who? Why?)
- Creating Classes
- The CRC Card Sessions
    - scenario execution

Finding Classes

•Look for anything that interacts with the system, or is part of the system

•Class names are singular nouns

• Suggested Classes for Library System are
  - Library, Librarian, User, Borrower, Article, Material, Item, Due Date, Fine, Lendable, Book, Video, and Journal
• Classes after filtering
  - Librarian, Lendable, Book, Video, Journal,Date, Borrower and User
• Assigning Cards
  - A CRC Card per Class, put name & description of the class

Scenario execution

• Scenario executions/Role Plays (For what?)

- Filter and test identified classes
- Identify additional classes
- Identify responsibilities and collaborators
  - can be derived from the requirements/use cases
  - responsibilities that are "obvious" from the name of the class (be cautious, avoid extraneous responsibilities)
- Filter and test responsibilities and collaborators
- Attributes (only the primary ones)

Finding Responsibilities

• Things that the class has knowledge about, or things that the class can do with the knowledge it has
• Tips/Indicators

- Verb phrases in the problem or use case

- Ask what the class knows? What/how the class does ?
- Ask what information must be stored about the class to make it unique?

Finding Collaborators

- A class asks another class when it
  - needs information that it does not have or
  - needs to modify information that it does not have
- Client - Server relationship
- Tips/Indicators
  - Ask what the class does not know and needs to know? And who can provide that

Scenario Execution

- Identify Scenarios (By domain experts)
- Main scenarios: **check-out, return and search**
- Start with the simple ones
- The first one always takes the longest
- Domain experts have high level of contribution during the early scenarios

Checkout Scenario

- Who should have the overall responsibilities for the task/check out? *Librarian*.
- What does the task entail?
- Shouldn't there be collaborations in the opposite direction?
  - Collaborations in CRC cards are one-way relationships from the client to the server (OO)
- Who should do the checking out of the Book?
  Librarian or Book itself?

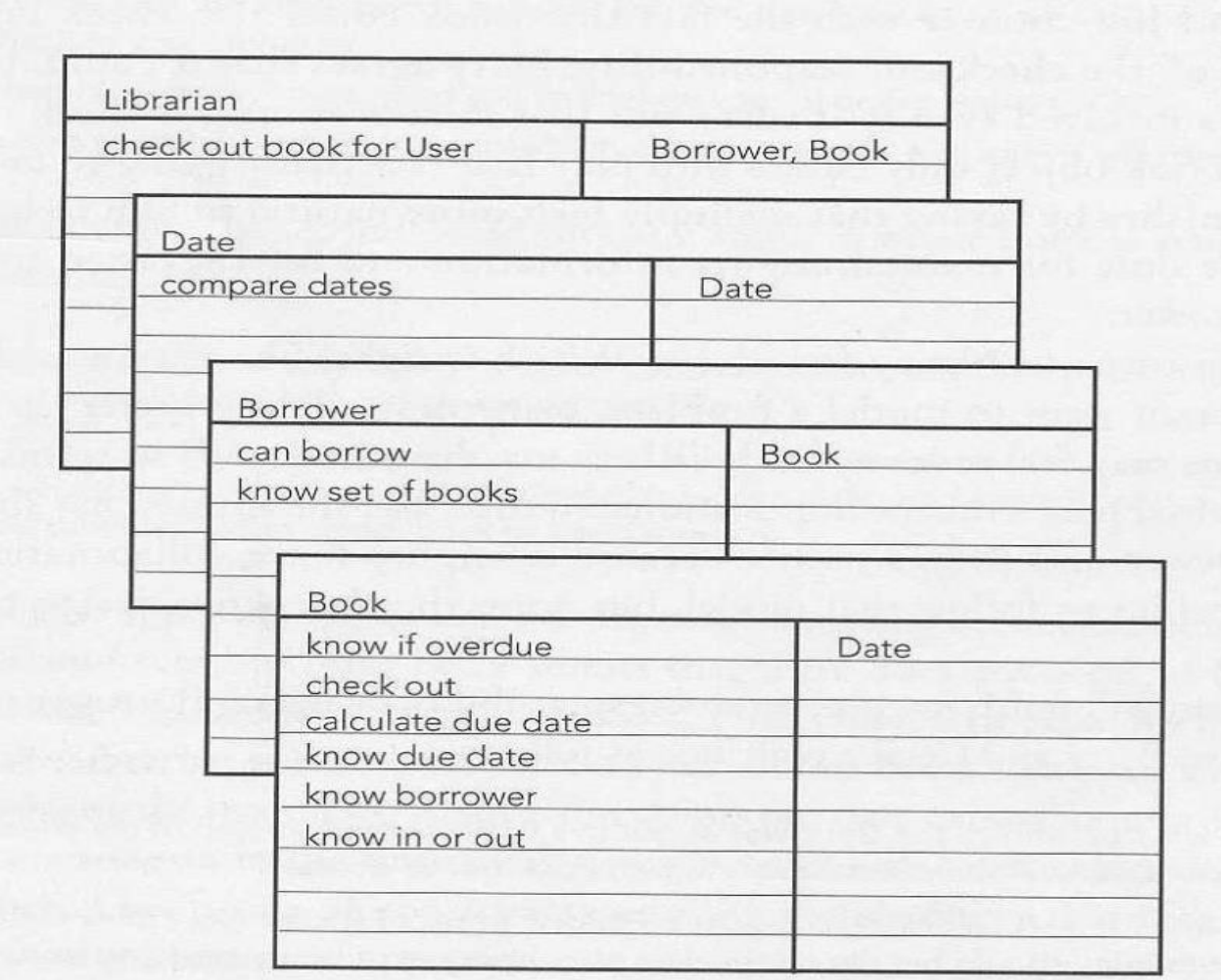
•Who should tell Borrower to update its knowledge about outstanding Book? Librarian or Book?

• Do we need a collaboration between Book a Borrower for the “known set of books” responsibility?

- Collaborations are not usually needed for responsibilities that simply hold information, only for situations where an object actually **sends a message** to a Collaborator.

- Borrower does not need Book's help to put a Book in a set.

The following is an example of CRC Cards after executing first scenario i.e checkout



Library application cards after the first scenario.

Search Scenario

- "What happens when Mr. John comes to Library in search of a book entitled The *Database Systems*?"

- Discovery of new class: Collection class (Why?)
    - Book can't look for itself
    - Collection looks over a set of Books to find the correct one

- When to end scenario execution?
    - When you have a stable model (does not cause new C or R to be added)

CASE TOOLS

CASE stands for Computer Aided Software Engineering. It means development and maintenance of software projects or products with the help of various automated software tools.

CASE tools can be defined as a set of software application programs, which are used to automate (Software Development Life Cycle) SDLC activities.

CASE tools are used by software project managers, analysts and engineers to develop software system smoothly and reliably.

There are a number of CASE tools available to simplify various stages of SDLC such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools etc.

Some of these CASE tools assist in phase related tasks such as specification, structured analysis, design, coding, testing, etc, and others to non-phase activities such as project management and configuration management.

Need of CASE Tools

A CASE environment facilitates the automation of the step-by-step methodologies for software development. In contrast to a CASE

environment, a programming environment is an integrated collection of tools to support only the coding phase of software development.
The ultimate goal of CASE is to provide a language for describing the overall system that is sufficient to generate all the necessary programs needed.

Use of CASE tools accelerates the development of project to produce the desired result and helps to uncover flaws before moving ahead with next stage in software development.

The primary utility of using a CASE tool:

- To increase productivity
- To help produce better quality software at a lower cost
- To develop more reliable and fault torant software product

Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

➢ Central Repository

CASE tools require a central repository, which can serve as a source of of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and

diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.

➢ Upper Case Tools

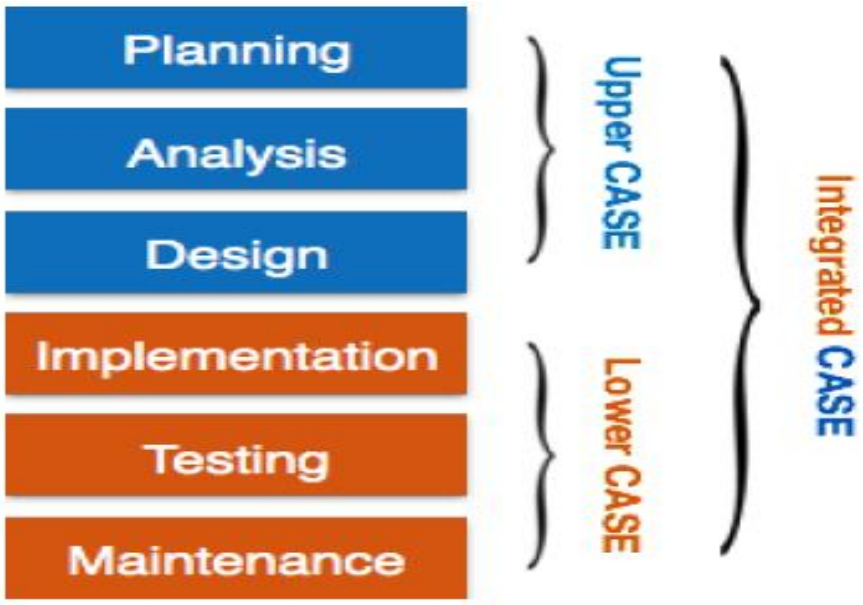
Upper CASE tools are used in planning, analysis and design stagesof of SDLC.

➢ Lower Case Tools

Lower CASE tools are used in implementation, testing and maintenance.

➢ Integrated Case Tools

Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.



Scope of Case Tools

The scope of CASE tools goes throughout the SDLC.

Case Tools Types

Various case tools are as follows.

➢ Diagram tools

These tools are used to represent

- system components,
- data and control flow among various software components
- and system structure in a graphical form.

For example, Flow Chart Maker tool tool for creating state-of-the-art flowcharts.

➢ Process Modeling Tools

Process modeling is method to create software process model, which is used to develop the software.

Process modelling tools help the managers to choose a process model or modify it as per the requirement of software product.

For example, EPF Composer (The **Eclipse Process Framework** (EPF) is an open source project that is managed by the Eclipse Foundation.

➢ Project Management Tools

These tools are used for

project planning,

cost and effort estimation,

project scheduling and resource planning.

Managers have to strictly comply project execution with every mentioned step in software project management.

Project management tools help in storing and sharing project information in real-time throughout the organization.

For example, Creative Pro Office, Trac Project, Basecamp etc.

➢ Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project.

Documentation tools generate documents **for technical users** and **end users**.

Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc.

The end user documents describe the functioning and how-to use the system such as user manual.

For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

➢ Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions.

For example, Accept 360, Accompa, CaseComplete for requirement analysis, Visible Analyst for total analysis.

➢ Design Tools

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules.

For example, Animated Software Design

➢ Configuration Management Tools

An instance of software is released under one version.

Configuration Management tools deal with –

- Version and revision management
- Baseline configuration management
- Change control management

CASE tools help in this by automatic tracking, version management and release management. For example, Fossil, Git, Accu REV.

➢ Programming Tools

- These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools.
- These tools provide comprehensive aid in building software product and include features for simulation and testing.
- For example, Cscope to search code in C, Eclipse.

➢ Prototyping Tools

- Software prototype is simulated version of the intended software product.

- Prototype provides initial look and feel of the product and simulates few aspect of actual product.
- Prototyping CASE tools essentially come with graphical libraries.
- They can create hardware independent user interfaces and design.
- These tools help us to build rapid prototypes based on existing information.
- In addition, they provide simulation of software prototype.
- For example, Serena prototype composer, Mockup Builder.

➢ Web Development Tools

- These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on.
- Web tools also provide live preview of what is being developed and how will it look after completion.
- For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

➢ Quality Assurance Tools

- Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure the quality as per organization standards.

- QA tools consist of configuration and change control tools and software testing tools.
- For example, SoapTest, AppsWatch, JMeter.

➢ Maintenance Tools

- Software maintenance includes modifications in the software product after it is delivered.
- Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC.
- For example, Bugzilla for defect tracking, HP Quality Centre.

Advantages of the CASE approach

- The overall quality of the product is improved as an organized approach is undertaken during the process of development.
- Chances to meet real-world requirements are more likely and easier with a computer-aided software engineering approach.
- As special emphasis is placed on redesign as well as testing, the servicing cost of a product over its expected lifetime is considerably reduced.
- CASE indirectly provides an organization with a competitive advantage by helping ensure the development of high-quality products.

Disadvantages of the CASE approach:

- **Cost:** Using case tool is a very costly. Mostly firms engaged in software development on a small scale do not invest in CASE tools because they think that the benefit of CASE are justifiable only in the development of large systems.
- **Learning Curve:** In most cases, programmers productivity may fall in the initial phase of implementation, because user need time to learn the technology. Many consultants offer training and on-site services that can be important to accelerate the learning curve and to the development and use of the CASE tools.
- **Tool Mix:** It is important to build an appropriate selection tool mix to urge cost advantage CASE integration and data integration across all platforms is extremely important.