UNIT I

Object Oriented Methodologies: Software System Life Cycle – Traditional cycle models – Object Oriented approach – Rambaugh et al Object Modeling Technique – Booch Methodology – Jacobsonet al methodology –Rational Unified Process (RUP) – Unified Modeling Language (UML) – UML Models.

1.1 THE SYSTEM LIFE CYCLE

Framework is very important for the development of a software system. An agreed framework for development brings many advantages:

1. A framework provides an overall picture of the development process; this picture is not cluttered by detail of what goes on at any stage in the process, but is useful as a high-level view of the major areas of activity, milestones and project deliverables.
2. A framework provides a basis for development and ensures a certain level of consistency in how the work is approached.
3. Consistency approach is very important when large of developers are involved in the project after it has started
4. A framework plays a significant role in ensuring quality, both of the development process and of the final system, by providing a structure for project management-planning, monitoring and controlling the development project.

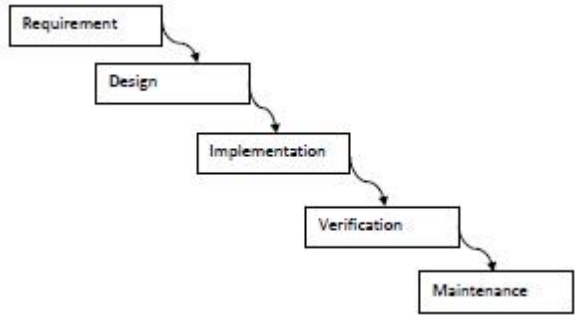
In software system development, a framework has traditionally been known as a **system life cycle model** The stages that have been referred for life cycle as requirements, analysis, design, implementation and installation. Each stage is concerned with particular issues and produces a set of outputs or deliverables shown in the below table

Stage of life cycle	*Issues addressed*	*Deliverables*
Requirements	What are the problems, needs and wishes of clients and users? What are the objectives and scope of the proposed system? What are the major risks involved?	List of requirements that can be used as a starting point for development. List of problem areas that fall within the scope of the proposed system. Assessment of risk factors.
Analysis	What does the system look like from the perspective of the clients and users?	A set of models, each taking a different view of the system, which together give a complete picture. The models may be text, diagrams or early prototypes.
Design	How can the system be constructed, so as to satisfy the requirements?	Models from the analysis stage, refined to illustrate the underlying architecture of the system. These models take account of technological considerations and constraints arising from the implementation environment.
Implementation	How can the models produced be translated into code?	A fully tested suite of programs.
Installation	What is needed to support clients and users so that they can use the new system effectively?	User manual, technical documentation, user training. Conversion from current system to new system.

1.2 TRADITIONAL LIFE CYCLE: The most important traditional life cycle models are:

1.2.1 Waterfall Model :

1. This is the early life cycle model; stages of development are straightforward sequence.
2. It describes a development method that is linear and sequence
3. It has distinct goals for each phase of development.
4. Once a phase of development is completed, the development proceeds to the next phase and is turning back.



Requirements: List of requirements for development.
Design: Process of problem solving and planning for a software solution.
Implementation: Coding
Testing: Make sure that the complete system meets software requirements.
Maintenance: modification of the product after deliver to correct faults.

1.2.2 V-model:

1. Stages are visualized in the form of the letter 'V'.
2. It emphasizes how later stages of development are related to earlier stages; for example, how testing should be derived from the activities that are carried out during requirements and analysis.

1.2.3 Spiral.

1. It incorporates iteration of life cycle stages and focuses on identifying and addressing the risks involved in development.
2. At each iteration around the cycle, the products are extensions of an earlier stage.

1.2.4 Prototyping.

1. In the prototyping life cycle, implementation takes place early in the development process.
2. The working model produced is subsequently refined and enhanced during a series of iterations until it is acceptable to the client.

1.2.5 Iterative Development:

1. This approach is closely related to the spiral model and to prototyping.
2. It covering the complete functionality of the system is produced and then refined as development progresses.

1.2.6 Incremental development.

1. In this life cycle model the system is partitioned according to areas of functionality.
2. Each major functional area is developed and delivered independently to the client.

1.3 THE OBJECT-ORIENTED APPROACH

One of the differences that is immediately obvious between traditional life cycle models and the object-oriented approach is the way that the various stages are named.

Traditional Life Cycle model	Object Oriented approach

Traditional model the name, such as 'analysis' or 'implementation', reflects the activities that are intended to be carried out in that stage.	A clear distinction is made between the activities and the stages (generally referred to as phases) of development.
	Phases are inception, elaboration, construction and transition. indicating the state of the system

1.3.1 Phases

o **Inception:**

It covers the initial work required to set up and agree terms for the project. It includes establishing the business case for the project, incorporating basic risk assessment and the scope of the system that is to be developed.

o **Elaboration**:

It deals with putting the basic architecture of the system in place and agreeing a plan for construction. During this phase a design is produced that shows that the system can be developed within the agreed constraints of time and cost.

o **Construction**:

It involves a series of iterations covering the bulk of the work on building the system; it ends with the beta release of the system, which means that it still has to undergo rigorous testing.

o **Transition**

It covers the processes involved in transferring the system to the clients and users. This includes sorting out errors and problems that have arisen during the development process.

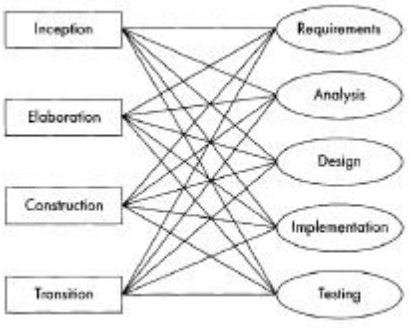
In object-orientation, activities such as analysis or design are referred to as workflows. The below figure shows the different workflows that typically take place during a system development project.

Workflows that take place during development of a system

It is recognized that a workflow may be carried out at more than one development phase and that developers may well engage in the whole range of workflows during every phase of building a system.

During the construction phase the main activities will be implementation and testing, but if bugs are found there will have to be some requirements and analysis as well.

The OO approach to development views the relationships between workflows and phases of development rather like the spider's web in the below figure, where any phase may involve all workflows, and a workflow may be carried out during any phase.



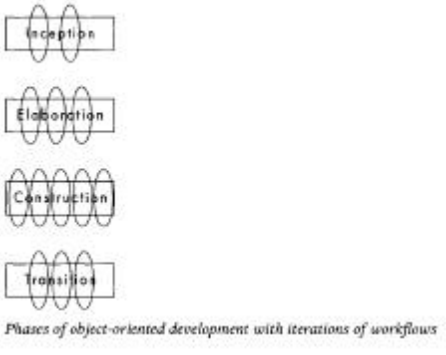
An object-oriented view of development phases and workflows

The object-oriented approach also recognizes fully the reality of iterative development. Activities at any phase do not take place in a neatly ordered fashion.

A developer may have to revisit a range of workflows several times during one phase of development, before it is possible to move on to the next phase.

The below figure illustrates the phases of the object-oriented life cycle with iteration of workflows at each phase.

In the diagram that iterations are most likely during construction, but can occur during any phase of development. Each ellipse represents a range of workflows.



Phases of object-oriented development with iterations of workflows

In addition to the emphasis on iterative development, the object-oriented approach also differs from traditional life cycle models in that it stresses the importance of a seamless development process.

This means that the separate phases are less distinct from each other than in a traditional system life cycle; it is not considered essential, nor is it often easy, to be able to say precisely when one phase is completed and another begins.

Although the traditional system life cycle was concerned about issues such as quality, ease of modification and potential reuse, it tended to regard them as add-ons to the core development process.

In the object-oriented approach such issues are regarded as central, and developers are encouraged to bear them in mind throughout the time they are working on the system.

1.3.1 RAMBAUGH ET AL OBJECT MODELING TECHNIQUE Object modelling techniques (OMT) presented by Jim Rambaugh describes a method for the analysis, design and implementation of a system using OOT. It is a fast, intuitive approach for identifying and modelling all the object making up a system. This model lets you specify detailed state transitions their descriptions within in a system. It consists of 4 phases:

1. ***Analysis:*** The results are objects and dynamic and functional models.
2. ***System Design:*** The results are structure of a basic architecture of the system along with the high –level strategy decisions.
3. ***Object Design:*** This phase produce a design document, consisting of a detailed objects static, dynamic and functional models.
4. ***Implementation:*** This activity produces reusable, extendible and robust code.

OMT separates modeling into three different parts:

1. Object model: presented by the object model and data dictionary.
2. Dynamic model: presented by the state diagrams and event flow diagrams.
3. Functional model: presented by data flow and constraints.

1.3.1 THE OBJECT MODEL It describes structure of object in the system; their identity and relationship to other objects, attributes and operations. The figure below shows object model with graphical representation

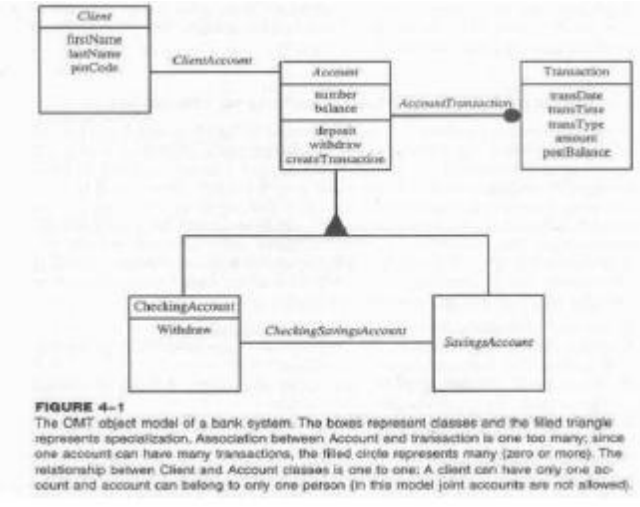


FIGURE 4–1
The OMT object model of a bank system. The boxes represent classes and the filled triangle represents specialization. Association between Account and transaction is one too many; since one account can have many transactions, the filled circle represents many (zero or more). The relationship betwen Client and Account classes is one to one: A client can have only one account and account can belong to only one person (in this model joint accounts are not allowed).

1.3.2 THE DYNAMIC MODEL It provides detailed and comprehensive dynamic model, in addition to letting you depict states, transitions, events and actions. The below figure shows state transition is a network of states and events.

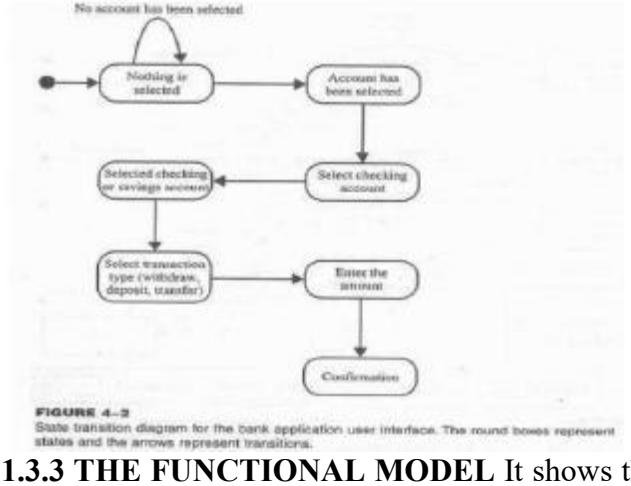


FIGURE 4–2
State transition diagram for the bank application user interface. The round boxes represent states and the arrows represent transitions.

1.3.3 THE FUNCTIONAL MODEL It shows the flow of data between different processes in a business. The OMT DFD provides a simple and intuitive method for describing business processes without focusing on the details of computer systems. DFD use 4 primary symbols:

1. The *process* is any function being performed; ***example*** verifying Password/PIN in ATM.

2. The *data flow* shows the direction of data element movement; **example** PIN code.
3. The *data store* is a location where the data are stored; **example** account data store in ATM
4. The *external entity* is a source/ destination of a data element; **example** ATM card reader

1.4 THE RATIONAL UNIFIED PROCESS (RUP)

A life cycle provides a high-level representation of the stages that a development project must go through to produce a successful system.

A development method, on the other hand, is much more prescriptive, often setting down in detail the tasks, responsibilities, processes, prerequisites, deliverables and milestones for each stage of the project.

Nowadays, almost all object-oriented projects use the Unified Modeling Language as the principal tool in their development process.

Use of the UML has been approved by the Object Management Group (OMG), which controls issues of standardization in this area. This has resulted in conformity between projects in terms of notation and techniques.

The creators of the UML have proposed a generic object-oriented development The Unified Software Development Process (Jacobson et al., 1999) and this generic method has been adopted and marketed by the Rational Corporation under the name of the Rational Unified Process (RUP).

RUP is based on the following six 'Best Practices'

1 Develop software iteratively
2 Manage requirements
3 Use component-based architectures
4 Visually model software
5 Verify software quality
6 Control changes to software.

1. Develop software iteratively

- RUP follows the phases of the generic object-oriented life cycle (inception, elaboration, construction and transition). It is built on the central concept of iterative development and each of its phases defines a series of activities that may be performed once or a number of times.
- Each iteration is defined as a complete development loop resulting in the release of an executable product that is a subset of the final system.
- In this way RUP supports incremental development- the frequent release of small packages of software that gradually build up to become the final system.
- Iteration and incremental development encourage involvement and feedback from clients and users; they make it easier to cope with changes, and reduce the risk factors associated with any development project.

2. Manage requirements

- RUP offers sound support for eliciting, organizing and recording requirements. Precise documentation of requirements facilitates traceability through the development process, which enhances the quality of the final system.
- The emphasis on the activities that take place early on in the life cycle provides a sound foundation for the later stages and results in systems that are robust, reliable and meet the needs of their users.

3. Use component-based architectures

- RUP prescribes the early identification and development of a system structure that is at the same time robust enough to ensure system reliability, and flexible enough to accommodate changes. This is achieved through the use of components subsystems that each have a single, well-defined function.
- RUP describes how to construct an architecture combining both new and previously existing components, thus encouraging the reuse of software as part of the development process.

4. Visually model software

- RUP is based around the Unified Modelling Language (UML) as a vehicle for development. UML has become an industry standard, and incorporates a wide range of techniques and tools to support developers. The techniques offered by UML bring with them all the advantages of visual modelling.
- ***For example,*** UML diagrams facilitate communication between developers and users and between members of the development team, they offer a number of different views of the system which combine to give a complete picture, they help developers to decompose the problem into smaller, more manageable chunks, and they provide a means of abstraction, concentrating on important information while hiding details that are currently irrelevant.

5. Verify software quality

- RUP provides the techniques to support quality assessment of functionality, reliability and performance throughout the development process.
- The RUP approach to quality is based on objective measures and criteria for success; it involves all members of the development team and applies to all the activities that are carried out as part of the system development.

6. Control changes to software

- Changes are the norm in a software development project, so an effective development process must be able to monitor and control them.
- RUP provides tools to do this, and also supports the work of developers by offering protection in one area of development from changes that occur in another.

1.5 UNIFIED MODELLING LANGUAGE (UML)

The Unified Modelling Language, or UML, is a set of diagrammatic techniques, which are specifically tailored for OOD, and which have become an industry standard for modelling object-oriented systems.

1.5.1 Modelling:

Software developers use specialized diagrams to model the system that they are working on throughout the development process. Each model produced represents part of the system or some aspect of it, such as the structure of the stored data, or the way that operations are carried out. Each model provides a view of the system, but not the whole picture.

1.5.2. Abstraction:

The characteristic of a model to provide some but not all the information about the person or thing being modelled is known as ***abstraction.*** Each of the modelling techniques in the Unified Modelling Language provides a particular view of the system as it develops; each UML model is an abstraction of the complete system. Abstraction, concentrates on only those aspects of the system that are currently of interest, and putting other details to the side for the time being.

1.5.3. Decomposition:

This is the breaking down of a large, complex problem or system into successively smaller parts, until each part is a 'brain-size' chunk and can be worked on as an independent unit. Traditionally software systems used to be decomposed according to their functions - the tasks that the system had to carry out. In OO, systems are decomposed according to the data that they have to store, access and manipulate.

1.6 UML MODELS

The UML is not a development method since it does not prescribe what developers should do, it is a diagrammatic language or notation, providing a set of diagramming techniques that model the system from different points of view.

The below table shows the principal UML models with a brief description of what each can tell us about the developing system.

Table 1.3: The principal UML diagrams with brief descriptions

Model	*View of the system*
Use case	How the system interacts with its users.
Class	The data elements in the system and the relationships between them.
Interaction (sequence and collaboration)	How the objects interact to achieve the functionality of a use case.
State	How the different objects of a single class behave through all the use cases in which the class is involved.
Activity	The sequence of activities that make up a process.
Component	The different software components of the system and the dependencies between them.
Deployment	The software and hardware elements of the system and the physical relationships between them.

The 4 + 1 view. The authors of UML, Booch et al., (1999), suggest the architecture of a system from five different perspectives or views:

- The use case view
- The design view
- The process view
- The implementation view
- The deployment view.

This is known as the 4 + 1 view (rather than the 5 views) because of the special role played by the use case view.

The Use Case view:

it specifies what the user wants the system to do; the other 4 views describe how to achieve this.

The use case view describes the external behavior of the system and is captured in the use case model

The Design view:

It sometimes called as ***logical view.*** Describes the logical structures required to provide the functionality specified in the use case view.

The design view describes the classes (including attributes and operations) of the system and their interactions.

The Process view:

It is concerned with describing concurrency in the system.

Sequence diagram can be used to achieve it.

The Implementation view:

It describes the physical software components of the system, such as executable files, class libraries and databases.

The view of the system can be modeled using component diagram

The Deployment view:

This view describes the hardware components of the system such as PCs, mainframes, printers and the way they are connected.

This view can also be used to show where software components are physically installed on the hardware elements.