

# Software Process Models

## Lecture#01-04



Dr. Sanjeev Patel

Asst. Professor,

Department of Computer Science and Engineering

National Institute of Technology Rourkela, Odisha

# Outline

- Software
- Software development process model
- Attributes of Good Software [1-3]
- Software Development Activities [1]
- Software Life Cycle [1]
- Waterfall Model [1]
- Object-oriented SDLC
- Building high quality software

# Evaluation Scheme

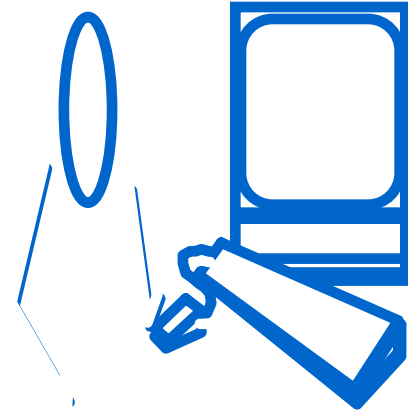
- Mid Sem Exam- 30 Marks
- End Sem Exam-50 Marks
- TA -20 Marks
  - Assignments
  - Viva
  - Quiz
  - Project / Presentation

# Books

1. Rajib Mall, “Fundamentals of Software Engineering”, 3<sup>rd</sup> edition, PHI, 2009
2. R.S. Pressman, “Software Engineering: A Practitioner's Approach”, 7th Edition, McGraw
3. James Rumbaugh, Ivar Jacobson, Grady Booch, “The Unified Modeling Language Reference Manual”, Second Edition, Addison- Wesley, 2004.
4. Grady Booch et al, Object-Oriented Analysis and Design with Applications, Addison Wesley, 2007.
5. Rumbaugh and Blaha, Object Oriented Modeling and Design with UML, Pearson.
6. Bernd Bruegge and Allen H Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, Pearson.
7. Bernd Oestereich, Developing Software with UML: Object-Oriented Analysis and Design in Practice, Addison Wesley.
8. Erich Gamma, Richard help, Ralph Johnson and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1<sup>st</sup> Edition, Addison Wesley.

# What is Software?

- Software is a set of items or objects that form a “configuration” that includes
  - **Programs:** instructions (computer programs) that when executed provides desired function and performance
  - **Documents:** that describe the operation and use of the programs. (such as requirements, design models and user manuals)
  - **Data:** Data and data structures that enable the programs to adequately manipulate information



Software is a **digital form of knowledge**. (“Software Engineering,” 6ed. Sommerville, Addison-Wesley, 2000)

Ref: (“Software Engineering- a practitioner’s approach,” Pressman, 5ed. McGraw-Hill)

# Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

# What is a software process model? [3]

- A simplified representation of a software process, presented from a specific perspective.
- Examples of process perspectives are
  - Workflow perspective - sequence of activities;
  - Data-flow perspective - information flow;
  - Role/action perspective - who does what.
- Generic process models are (Paradigms of Software Development):
  - Waterfall;
  - Iterative development:
    - During software development, more than one iteration of the software development cycle may be in progress at the same time
  - Component-based software engineering:
    - It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems.

# Software Process Model

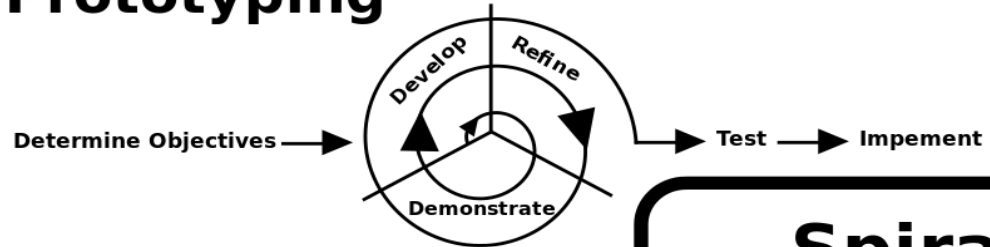
- A set of activities whose goal is the development or evolution of software.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective
- Generic activities in all software processes are:
  - **Specification** - what the system should do and its development constraints (Stakeholder analysis, Feasibility study, requirement analysis)
  - **System Design**- Designing of the system at high level and detailed level
  - **Development** - production of the software system
  - **Validation** - checking that the software is what the customer wants
  - **Evolution** - changing the software in response to changing demands.



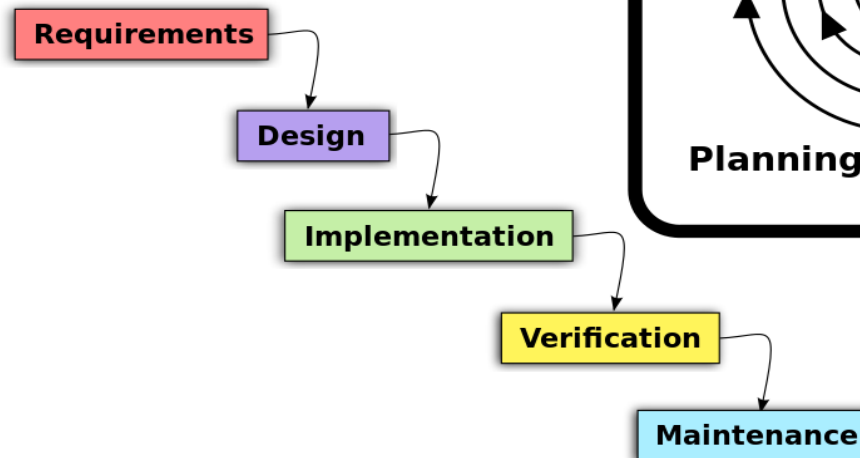
# Software Process Models

- Waterfall
- V model,
- Evolutionary,
- Prototyping
- Spiral model,
- Agile models

## Prototyping



## Waterfall



## Spiral



# What are the attributes of good software?

- Quality attributes expected of software
  - **Reliability**
  - **Extensibility/Maintainability**
  - **Performance**
  - **Usability**
  - **Security**
  - **Availability**
  - **Portability**
- Quality attributes expected of software engineering
  - Productivity
  - Cost

# Attributes of good software

- **Reliability:**

- Probability of failure free operation for a specified time in a specified environment for a given purpose
- Informally, reliability is a measure of how well system users think it provides the services they require
- One measure of Reliability is the **MTF (mean-time-between-failures)** OS/400 has a MTF of 1.3 years and MS Windows 5.0 had a MTF of 1.3 hours; incidentally, OS/400 is the **ONLY** software to have won the **Malcolm-Baldrige Quality Award** so far!!

# Attributes of good software

- **Extensibility / Maintainability**
  - Process of modifying a software system or component after delivery
    - to correct faults,
    - to enhance functionality,
    - to improve performances or other quality attributes
    - to adapt to a changed environment, etc.
- One of the measures of Extensibility / Maintainability is the average person-days of effort to extend / maintain a software.
  - modularity quotient of a software has a direct impact on the extensibility / maintainability.

# Attributes of good software

- **Usability**

- The ease with which a user can utilize software package.
- Usability is a culturally sensitive subject
- A measure of usability is the average satisfaction rating of a suitably sampled user group. (Note: Attempts by industry to mine texts of blogs, news-groups to get a measure for usability of their software)

- **Security:**

- Security of a software is the degree of its capability to protect its services and data from unauthorized users
- Authentication (to log-in) and authorization (to use only certain services) are used to protect the services and encryption to protect the data

# Attributes of good software

- **Availability**

- Software packages contain many modules, which are integrated and work together; e.g Web servers, App servers, DBMS servers, application software modules, OS, N/W layers put together offer Infosys Finacle based banking services!
- Availability refers to the percentage time that such as a complex software package is continuously available measured over a 100 units of time!
- Many banks of reputation demand 99.9% availability for every 1000 days from the IT service provider!

- **Portability**

- Portability of a software is the degree of ease with which one can install and use that software in **different environments**;

# Attributes of good software engineering

- **Productivity**

- Average number of LoC (Line of code) per person taken over the complete life-cycle of a software project
- well engineered projects eliminate or minimize re-design or re-coding or re-testing and thereby achieve high productivity;
- use of engineering models, methodologies, standards, tools and training of personnel etc. contribute towards high productivity;

- **Cost**

- Appropriate balance between high productivity and high cost is very essential
- High productivity and low cost is the dream of every software project manager!

# Software Development Activities

1. Stake Holder Analysis
2. System development requirements (Requirement analysis)
3. Feasibility study
4. System Design
5. Coding and Integration
6. Debugging and Testing
7. Implementation and post implementation



# 1. Stakeholder Analysis

Stakeholder Management is the process by which you identify your key stakeholders and win their support.

A stakeholder-based approach gives you four key benefits:

## **1. Getting Your Projects Into Shape:**

Opinions of most powerful stakeholders help to define projects at an early stage.

## **2. Winning Resources**

Gaining support from powerful stakeholders can help you to win more resources, such as people, time or money.

## **3. Building Understanding**

ensure that stakeholders fully grasp what you're doing and understand the benefits of your project.

## **4. Getting Ahead of the Game**

Understanding your stakeholders means that you can anticipate and predict their reactions to your project as it develops.

# How to Conduct a Stakeholder Analysis

- There are three steps to follow in Stakeholder Analysis.
- **Step 1: Identify Your Stakeholders:**
  - Anyone who operates the system
  - Anyone who benefits from the system
  - Anyone involved in purchasing or procuring the system.
  - Organizations which regulate aspects of the system
  - People or organizations opposed to the system

Types of stakeholders include:

- **Key stakeholders:** those with **significant influence** upon or importance within an organization
- **Primary stakeholders:** those ultimately **most affected** by an organization's actions
- **Secondary stakeholders:** persons or organizations who are **indirectly affected** by an organization's actions
- **Tertiary stakeholders:** those who will be **impacted the least**

# How to Conduct a Stakeholder Analysis

- **Step 2: Prioritize Your Stakeholders**
  - Some of these may have the power either to block that work or to advance it.
  - Some may be interested in what you are doing,
  - while others may not care
- Classify according to their power and interest on a Power/Interest Grid.

**Figure: Power/Interest Grid for Stakeholder Prioritization**



# How to Conduct a Stakeholder Analysis

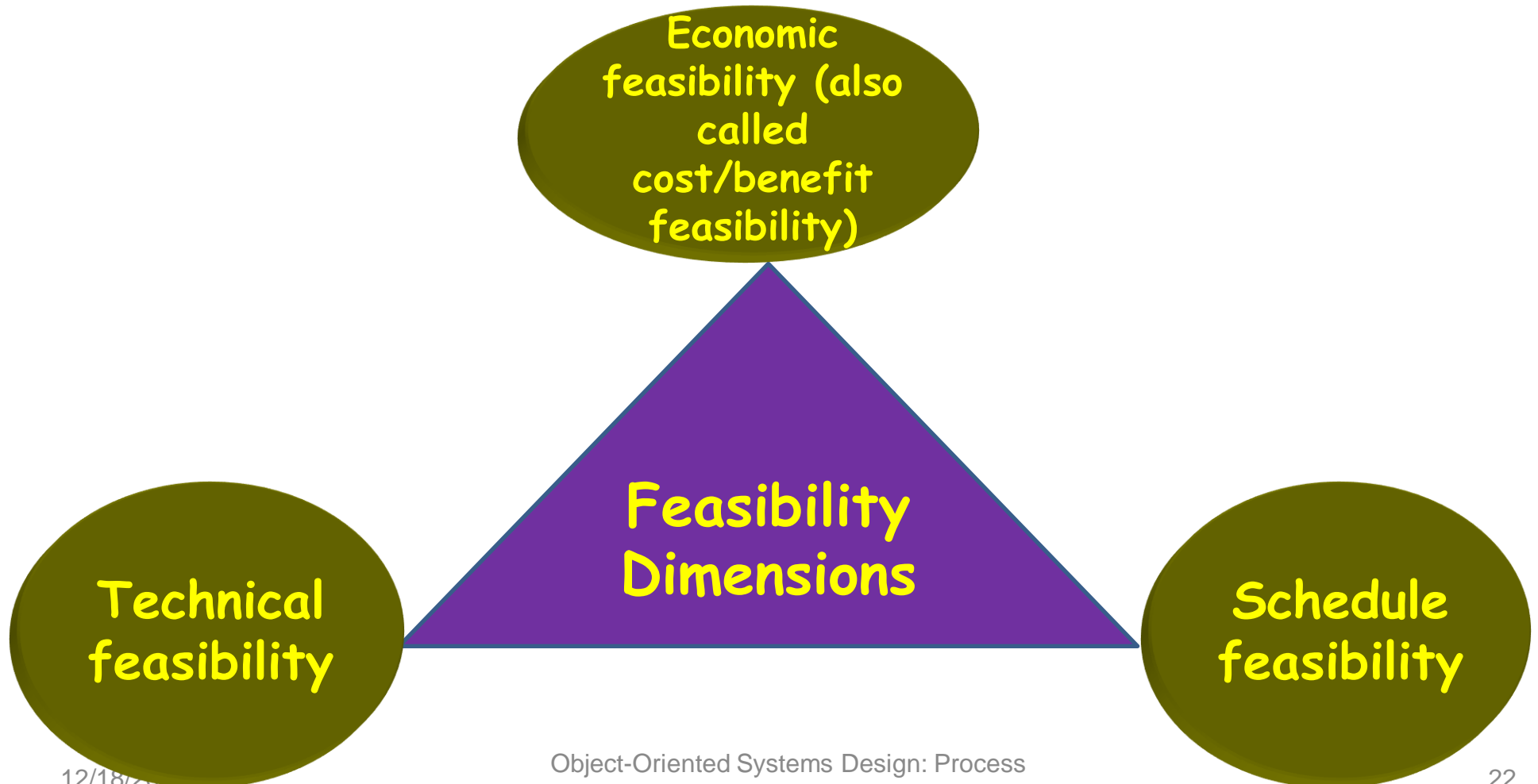
- **Step 3: Understand Your Key Stakeholders**
  - How key stakeholders feel about the project?
  - Key questions that can help you understand your stakeholders include:
    - What financial or emotional interest?
    - What motivates them most of all?
    - What information do they want?
    - What is their current opinion of your work?
  - If they aren't likely to be positive, what will win them around to support your project?
  - If you think that you'll not be able to win them around, how will you manage their opposition?

## 2. System development requirements

- **Requirements analysis**
  - Determining the needs or conditions to meet for a new or altered product/project.
- **Conceptually, requirements analysis includes three types of activities:**
  1. **Eliciting requirements:** This is sometimes also called requirements **gathering** or requirements discovery.
  2. **Analyzing requirements:** determining whether the stated requirements are **clear, complete, consistent** and **unambiguous**, and resolving any apparent conflicts.
  3. **Recording requirements (Software Requirements Specification):** Requirements may be documented in various forms, usually including a summary list and may include natural-language documents, use cases, user stories, process specifications and a variety of models including data models.

# 3. Feasibility Study

- **Feasibility Study** is an assessment of the **practicality** of a proposed project or system.



# Areas of Feasibility study

**A. Technical feasibility:** determine whether the company has the technical expertise to handle completion of the project.

- **Method of production**

- Availability of inputs or raw materials and their quality and prices.
- Availability of markets for outputs
- Various efficiency factors such as the expected increase in one of the additional production unit.

- **Production technique**

- Tools and equipment needed for the project
- Construction requirement such as buildings, storage, and roads ...etc.
- Requirements of skilled and unskilled labor and managerial and financial labor.

- **Project location**

- Availability of land (proper acreage and reasonable costs).
- The costs of transporting inputs and outputs to the project's location.
- Availability of various related resources: water or electricity or good roads ...etc.

# Areas of Feasibility study

## B. Financial feasibility

- In case of a new project, financial viability can be judged on the following parameters:
  - Total **estimated cost** of the project
  - **Financing** of the project in terms of its capital structure
  - **Existing investment** by the promoter in any other business
  - Projected **cash flow** and
  - **Profitability**

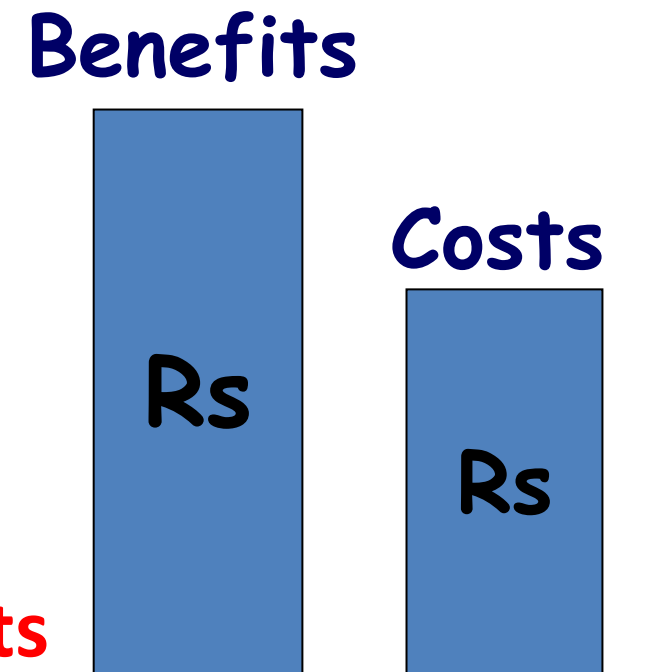




# Cost benefit analysis (CBA)

- Need to identify all costs --- these could be:
  - Development costs
  - Set-up
  - Operational costs
- Identify the value of benefits
  - Quantifiable
  - Non-quantifiable
  - Needs to take account of business risks
  - Identify risks.
  - Explain how these will be managed.

**Check benefits are greater than costs**



# Areas of Feasibility study

## **C. Schedule feasibility**

- Feasibility is a measure of how reasonable the project timetable is?
- Given our technical expertise, are the project deadlines reasonable?
- When it can be built?
- A project will fail if it takes too long to be completed before it is useful.
- It is necessary to determine whether the deadlines are mandatory or desirable.

# Areas of Feasibility study

## **D. Operational feasibility**

- Operational feasibility is the measure of
  - how well a proposed system solves the problems?
  - how it satisfies the requirements?
- Desired operational outcomes: reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others.

## **E. Legal feasibility**

- Determines whether the proposed system conflicts with legal requirements, e.g.,
  - if the proposed venture is acceptable in accordance to the laws of the land.
- The impact of project on the environment and the approval of the concerned authority.

# 4. System Design

- Software design is a technical description about **how the system will implement the requirements**
- During design phase requirements specification is transformed into :
  - A form suitable for implementation in some programming language.
- The system architecture describes:
  - How the system will be decomposed into subsystems (modules)
  - Responsibilities of each module
  - Interaction between modules
  - Platforms and technologies

# System Design

## A. Logical design

- Abstract representation of the data flows, inputs and outputs of the system.
- This is often conducted via **modelling**
- Logical design includes entity-relationship diagrams(ER diagrams).

## B. Physical design

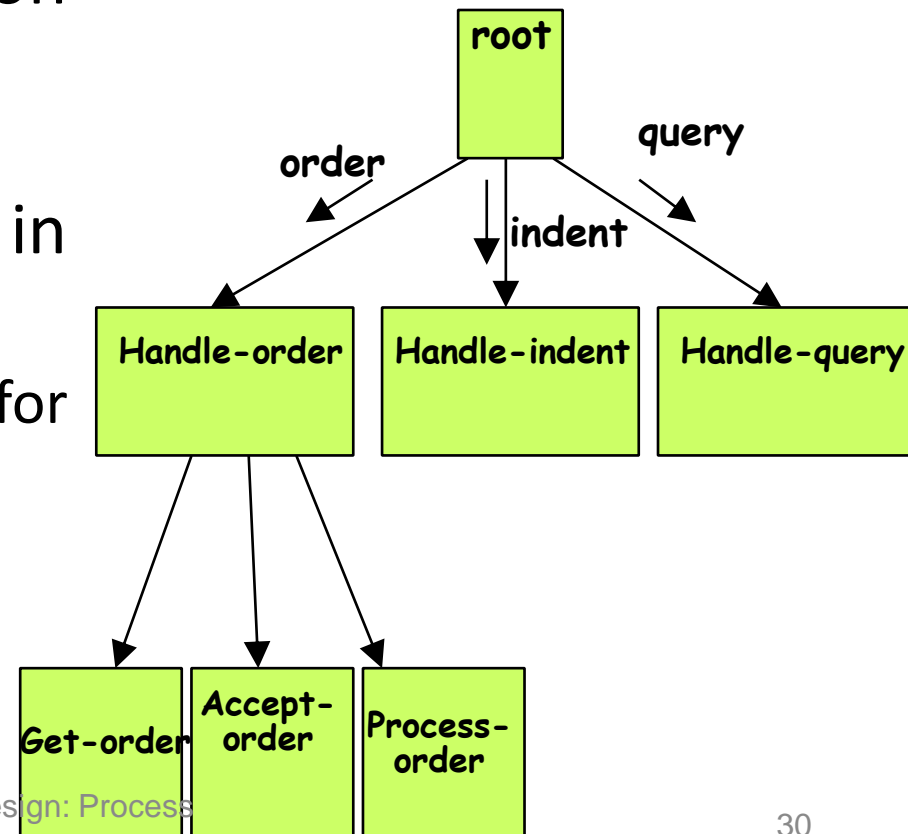
- In physical design, the input, output, storage, processing, and recovery requirements about the system are decided.
  - User Interface Design
  - Data Design: concerned with how the data is represented and stored within the system.
  - Process Design: concerned with how data moves through the system (DFD)

## C. Architectural design

- Design of the system architecture that describes the structure, behavior and more views of that system and analysis.

# Structured Design

- **High-level design:**
  - Decompose the system into **modules**,
  - Represent invocation relationships among modules.
- **Detailed design:**
  - Different modules designed in greater detail:
    - Data structures and algorithms for each module are designed.



# Structured Design

- Two commonly used design approaches:
  - Traditional approach,
  - Object oriented approach
- Consists of two activities:
  - **Structured analysis:** typically carried out by using DFD, Data Dictionary, State Transition diagram and ER diagram.
  - **Structured design:** It uses Structure Chart

# Object-Oriented Design

- First identify various objects (real world entities) occurring in the problem:
  - Identify the relationships among the objects.
  - For example, the objects in a pay-roll software may be:
    - employees,
    - managers,
    - pay-roll register,
    - Departments, etc.



# 5. Coding and Integration

- Coding is the process of writing the programming code (the source code)
- Inexperienced developers consider coding the core of development
  - In most projects, coding is only 20% of the project activities!
  - The important decisions are taken during the requirements analysis and design
  - Documentation, testing, integration, maintenance, etc. are often disparaged
- Software engineering is not just coding!

**Programmer != software engineer**

# Quantifying program quality

- RUC (Readability, Understandability, and Comprehensibility)
- Logical structure : logical designing of modules
- Physical Layout: listing of the source code
- Robustness: how well the program can handle incorrect data
- CPU efficiency (execution time)
- Memory efficiency (Space consumption)
- Complexity (Algorithmic and structural complexity)
- Human Factors (Human-to-computer interface)
- System interface (system-to-environment interface)
- Reusable code

# Integration

- **System integration** process of bringing together the component sub-systems into one system and ensuring that the subsystems function together as a system

# Methods of integration

**A. Vertical integration:** is the process of integrating subsystems according to their functionality by creating functional entities.

## Benefits:

- The integration is performed quickly
- Involves only the necessary vendors, therefore, this method is cheaper in the short term.

## Shortcomings

- Cost-of-ownership can be substantially higher than seen in other methods, since in case of new or enhanced functionality, the only possible way to implement (scale the system) would be by implementing another entity.



# Methods of integration

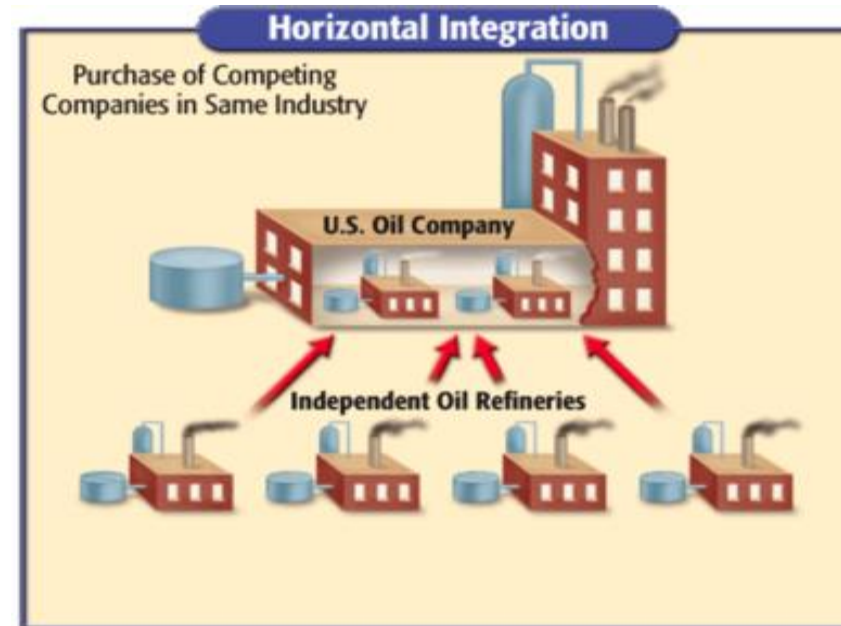
**B. Horizontal integration:** is an integration method in which a specialized subsystem is dedicated to communication between other subsystems.

## Benefits:

- Only one connection per subsystem which will connect directly to the master subsystem.
- Allows cutting the costs of integration and provides extreme flexibility.
- it is possible to completely replace one subsystem with another subsystem which provides similar functionality but exports different interfaces, all this completely transparent for the rest of the subsystems.

## Shortcoming:

- The only action required is to implement the new interface between the master and the new subsystem.



# Methods of integration

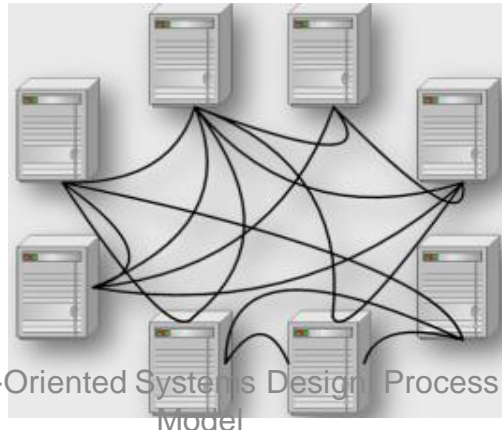
**C. Matrix integration:** is a process of systems integration where each system is interconnected to each of the remaining subsystems.

## Benefits:

- From the feature perspective, this method often seems preferable, due to the extreme flexibility of the reuse of functionality.
- The cost varies because of the interfaces that subsystems are exporting.

## Shortcomings:

- In a case where the subsystems are exporting heterogeneous or proprietary interfaces, the integration cost can substantially rise.
- Time and costs needed to integrate the systems increase exponentially when adding additional subsystems.



# 6. Testing and debugging

- Testing:
  - A systematic attempt to find faults in planned way in the implemented software
  - A fault detection technique to create failures or erroneous states in a planned way
- Terminologies:
  - **Component:** part of system that can be isolated for testing
  - **Fault:** bug or defect, is a design or coding mistake that may cause abnormal component behavior
  - **Erroneous state:** manifestation of a fault during the execution of the system. Caused by one or more fault and lead to failure
  - **Failure:** deviation between the specification and the actual behavior
  - **Test case:** set of inputs and expected results

# Software Testing Activities

- **Component Inspection:** The goal of the inspection is to identify defects. In an inspection, a source code is selected for review and a team is gathered for an inspection meeting to review the work product. (static verification)
- **Usability testing:** a small set of target end-users, of a software system, "use" it to expose usability defects. a **non-functional testing** technique that is a measure of how easily the system can be used by end users.
- **Unit testing:** Unit testing is the process of testing individual components in isolation. It is a defect testing process.
- **Integration testing:** individual software modules are integrated logically and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
- **System testing:** system testing is a level of software testing where a complete and integrated software is tested.



# Testing methods

- **Static vs. dynamic testing**
  - **Static:** Reviews, walkthroughs, or inspections are referred to as static testing, when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow.
  - **Dynamic:** executing programmed code with a given set of test cases is referred to as dynamic testing. Dynamic testing takes place when the program itself is run.
- **The "box" approach:** describe the point of view that the tester takes when designing test cases
  - **White box testing:** verifies the internal structures or workings of a program.
  - **Black box testing:** examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it
  - **Grey box testing:** design test cases based on knowledge of internal data structures and algorithms while executing those tests at the user, or black-box level.

# Debugging

- Debugging aims to find the source of already identified defect and to fix it
  - Performed by developers
- Steps in debugging:
  - Attempt to reproduce the problem.
  - After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug.
  - After the test case is sufficiently simplified, a programmer can use a debugger tool to examine program states (values of variables, plus the call stack) and track down the origin of the problem(s).
  - Fix the defect
  - Test to check if the fix is correct

# 7. Implementation and Maintenance

- **Software implement** is all of the activities that make a software system available for use.
- Deployment activities
  - Release
  - Installation and activation
  - Deactivation
  - Uninstallation
  - Update
  - Version tracking

# Software Implementation Challenges

- **Code-reuse** - There are huge issues faced by programmers for compatibility checks and deciding how much code to re-use.
- **Version Management** - Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation. This documentation needs to be highly accurate and available on time.
- **Target-Host** - The software program, which is being developed in the organization, needs to be designed for host machines at the customers end. But at times, it is impossible to design a software that works on the target machines.


# Project maintenance

- Reasons for which modifications are required:
  - **Market Conditions** - Change in policies, such as taxation and new constraints
  - **Client Requirements** - Customer may ask for new features or functions.
  - **Host Modifications** - If any of the hardware or platform of the target host change.
  - **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business.

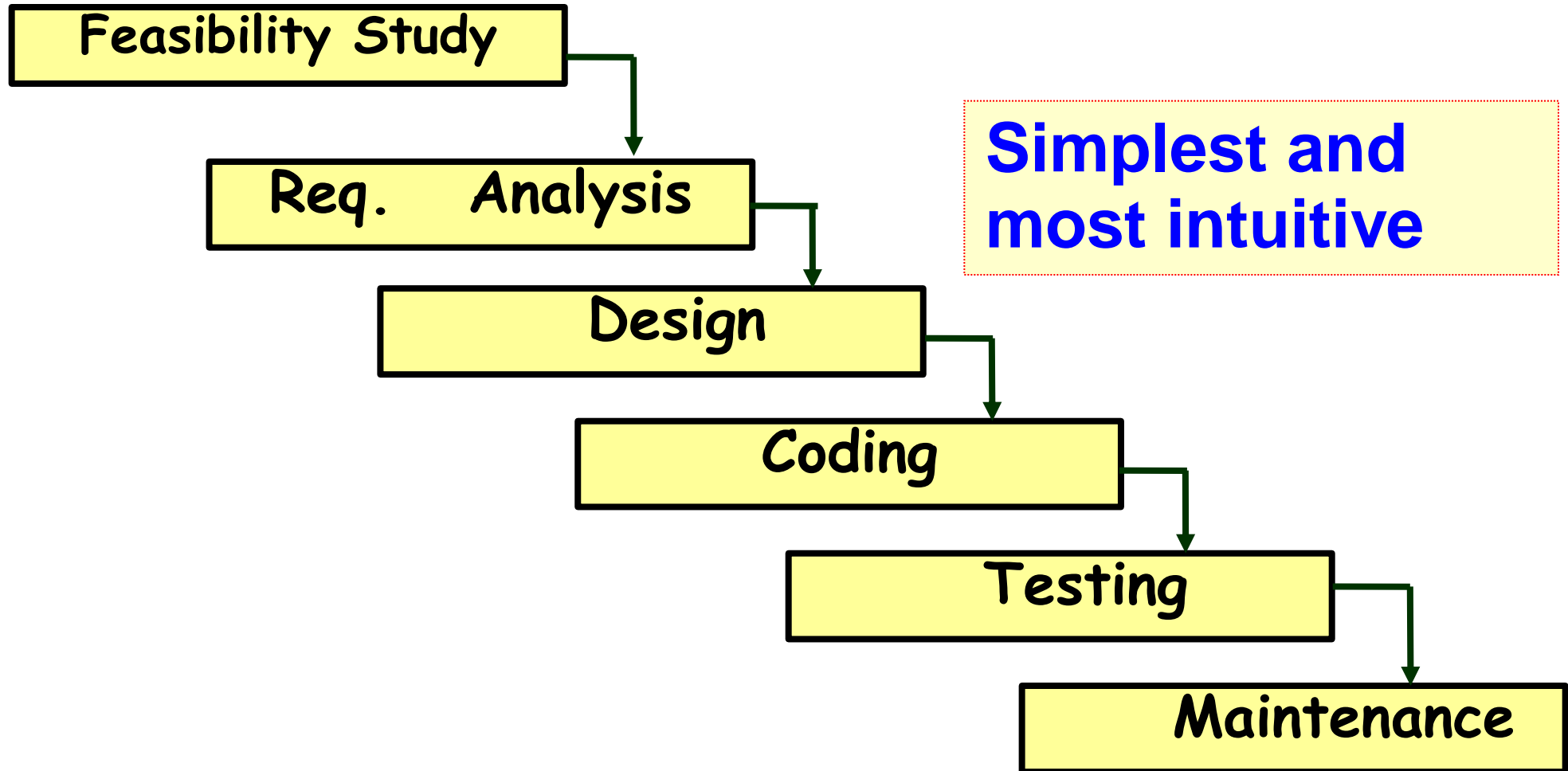
## Types of maintenance

- **Corrective Maintenance** - to correct or fix problems.
- **Adaptive Maintenance** - to keep the software product up-to date.
- **Perfective Maintenance** - to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

# Life Cycle Model

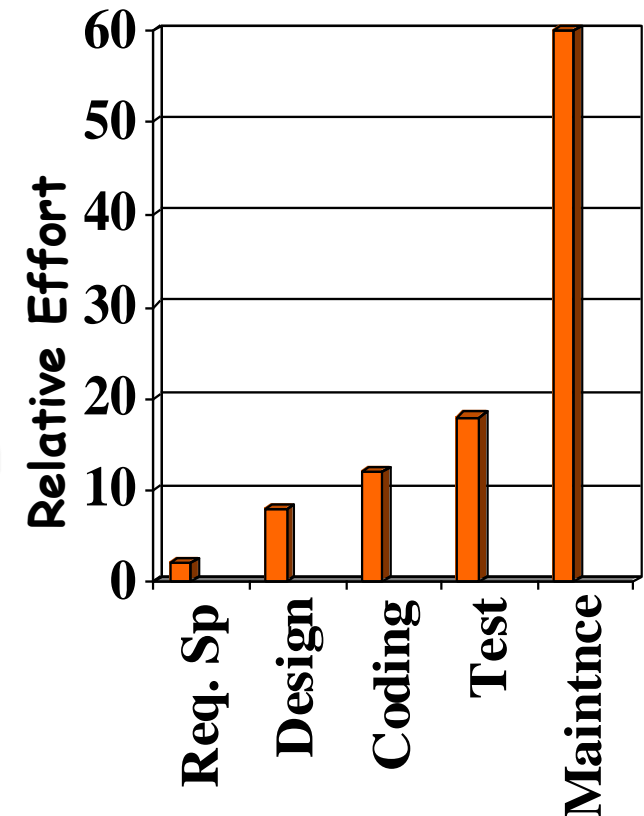
- A descriptive and diagrammatic model of software life cycle:
  - We confine our attention to only a few commonly used models.
    - **Waterfall**
    - **V model,**
    - **Evolutionary,**
    - **Prototyping**
    - **Spiral model,**
    - **Agile models**
- 
- Traditional models**

# Classical Waterfall Model



# Relative Effort for Phases

- Phases between feasibility study and testing Called **development phases**.
- Among all life cycle phases
  - Maintenance phase consumes maximum effort.
- Among development phases,
  - Testing phase consumes the maximum effort.





# Waterfall Strengths

- Easy to understand, easy to use, especially by inexperienced staff
- Milestones are well understood by the team
- Provides requirements stability during development
- Facilitates strong management control (plan, staff, track)

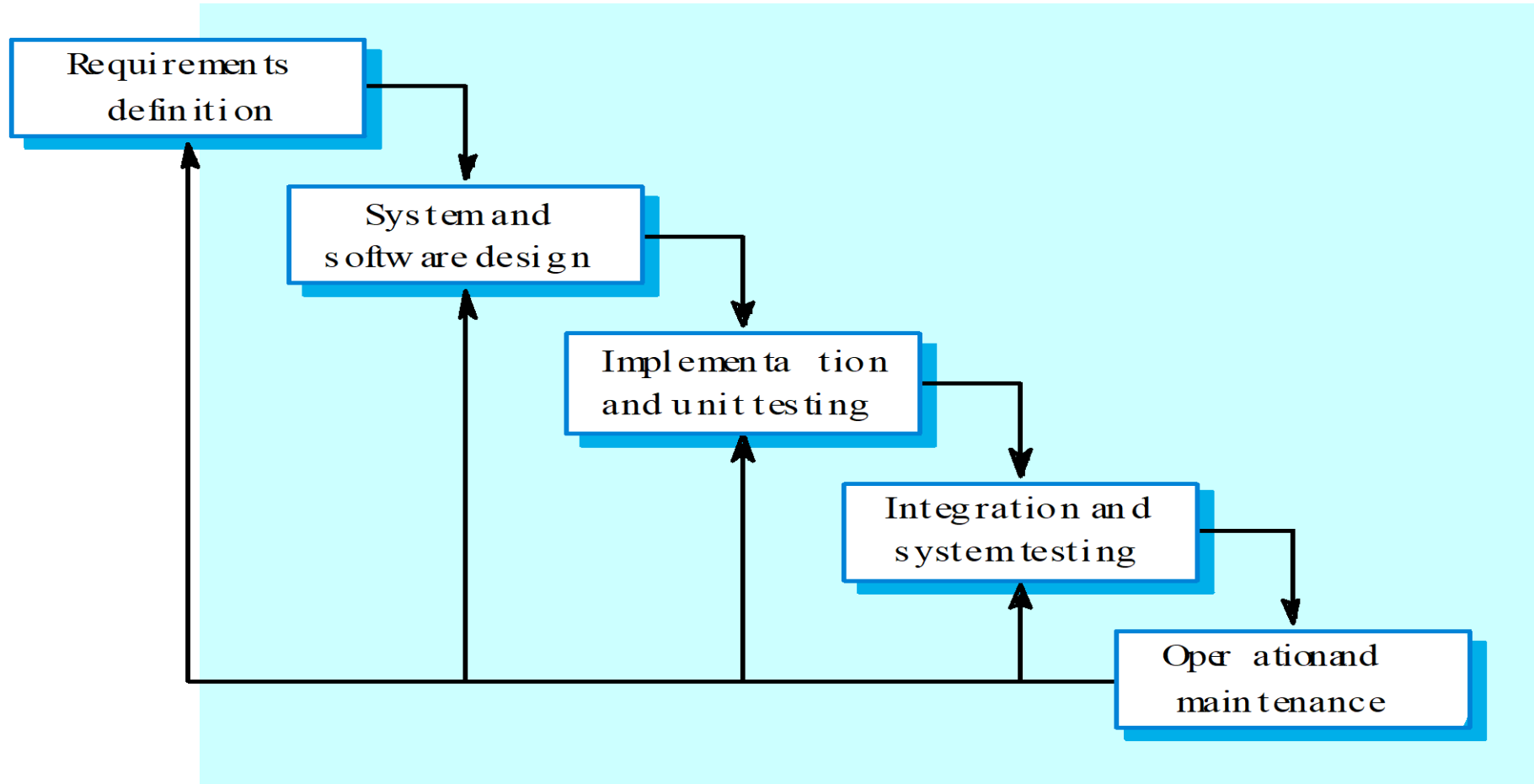
# Waterfall Deficiencies

- All requirements must be known upfront – **in most projects requirement change occurs after project start**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

# When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Development team have experience with similar projects

# Waterfall Model [3]



# Waterfall Model Phases

- Phases of waterfall model are;
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.
- One phase has to be complete before moving onto the next phase.

# Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a software project is part of a large systems engineering project.

# Specialized Process Models

- Component based development—the process to apply when reuse is a development objective
- Formal methods—emphasizes the mathematical specification of requirements
- AOSD—provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- Unified Process—a “use-case driven, architecture-centric, iterative and incremental” software process closely aligned with the Unified Modeling Language (UML)

# Object-oriented Life Cycle

- The Object-Oriented approach of Building Systems takes the objects as the basis.
- First the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development.
- The objects in the required system are identified i.e. for a Banking System, a customer, chequebook, and an account are object.
- The essence of the software development process that consists of analysis, design, implementation, testing, and refinement is to transform user's needs into a software solution that satisfies those needs.



# Object-oriented Life Cycle[8]

- Object-oriented model employs an object-oriented strategy.
- The primary objectives are
  - Object-oriented analysis
    - Conceptual Model
    - System Requirements
  - Object-oriented design
    - System Design
    - Detailed Design
  - Object-oriented programming
    - Coding
    - Testing

# Object-oriented Life Cycle[8]

- Object-oriented analysis develops an object-oriented model of the application domain and software system.
- Object oriented programming realizes the software design with an object-oriented programming language
- It supports direct implementation of objects, classes, and inheritance.
- The traditional conception of the software life-cycle is known as the *waterfall model*, which prescribes a strictly sequential transition between the successive phases
- Strict regulations with respect to validation of the products resulting from each phase may be imposed to avoid the risk of backtracking that leads to a severe problem.

# Analysis Phase[8]

- Software development can be seen as a series of transformations
- The output of one transformation becomes the input of the subsequent transformation.
- Translates the user's needs into system requirements and responsibilities.
- The way they use the system can provide insight into the user's requirements.
- During analysis the focus is on aspects of the problem domain and the goal is to arrive at a description of that domain to which the user and system requirements can be related.
- It can be done using
  - Objects, Inheritance, and Message passing

# Design Phase [8]

- Begins with a problem statement and ends with a detailed design that can be transformed into a operational system.
- The design phase must result in an architectural model of the system, for which we can demonstrate that it fulfills the user needs and the additional requirements expressed as the result of analysis.
- This transformation includes the bulk of the software development activity
- It includes the definition of how to build the software, its development, and its testing and it also includes design descriptions, the programs and the testing material.

# Design Phase [8]

- The distinction between *analysis* and *design* is primarily one of emphasis.
- Emphasis on modeling the reality of the problem domain versus emphasis on providing an architectural model of a system that lends itself to implementation.
- One of the attractive features of such an approach is the opportunity of a seamless transition between the respective phases of the software product in development.
- The classical waterfall model can no longer be considered as appropriate for such an approach

# Implementation [8]

- Refines the detailed design into the system deployment that will satisfy the user's needs
- This takes into account equipment, procedures, people, and the like.
- It represents embedding the software product within its operational environment.
- Design is meant to clarify the conceptual structure of a system, whereas the implementation must include all the details needed for the system to run.
- At the end, the design must serve both as *justification* and *clarification* of the actual implementation.

# Building High Quality Software [6]

- The software process transforms the users' needs via the application domain to a software solution that satisfies those needs.
- High-quality products must meet users' needs and expectations.
- Furthermore, the products should attain this with minimal or no defects, the focus being on improving products (or services) prior to delivery rather than correcting them after delivery.

# Building High Quality Software [6]

- The ultimate goal of building high-quality software is user satisfaction.
- To achieve high quality in software we need to be to answer the following questions:
  - How do we determine when the system is ready for delivery?
  - Is it now an operational system that satisfies users' needs?
  - Is it correct and operating as we thought it should?
  - Does it pass an evaluation process?



# Building High Quality Software [6]

- There are two basic approaches to systems testing.
- We can test a system according to how it has been built or, alternatively, what it should do.
- Blum describes a means of system evaluation in terms of four quality measures:
  - Correspondence
  - Correctness
  - Verification
  - validation.

# Building High Quality Software [6]

- **Correspondence**
  - It measures how well the delivered system matches the needs of the operational environment
  - True correspondence cannot be determined until the system is in place.
- **Validation**
  - Validation is the task of predicting correspondence.
  - Validation, however, is always subjective, and it addresses a different issue
  - the appropriateness of the specification.
  - It answers : Am I building the right product?

# Building High Quality Software [6]

- **Correctness**
  - It measures the consistency of the product requirements with respect to the design specification.
- **Verification**
  - It is the exercise of determining correctness.
  - However, correctness always is objective.
  - It answers: Am I building the product right?

# Building High Quality Software [6]

- Validation vs Verification
  - Validation begins as soon as the project starts, but verification can begin only after a specification has been accepted.
  - Verification and validation are independent of each other.
  - It is possible to have a product that corresponds to the specification
  - If the specification proves to be incorrect
  - Then, we do not have the right product;

# References

1. Rajib Mall, “Fundamentals of Software Engineering”, 3<sup>rd</sup> edition, PHI, 2009
2. R.S. Pressman, “Software Engineering: A Practitioner's Approach”, 7th Edition, McGraw
3. Sommerville, “ Introduction to Software Engineering”, 8th Edition, Addison-Wesley, 2007
4. [GRADY BOOCH, JAMES RUMBAUGH, IVAR JACOBSON, The Unified Modeling Language User Guide](#), Addison Wesley, Reading, Massachusetts, May 2005
5. PPT available for the respective books
6. [https://www.slideshare.net/kunalkishornirala/object-orientedsystemsdevelopmentlifecycle-ppt?from\\_action=save](https://www.slideshare.net/kunalkishornirala/object-orientedsystemsdevelopmentlifecycle-ppt?from_action=save)
7. <https://www.geeksforgeeks.org/software-engineering-object-oriented-life-cycle-model/>
8. <https://www.cs.vu.nl/~eliens/oop/1-3.html>

# THANK YOU