Chapter 1
Introduction

Two Orthogonal view of software

- ❑**Traditional technique**
  - ❑The traditional techniques view software, as a collection of programs (or functions) and isolated data.
  - ❑focuses on the functions of the system-What is it doing?
- ❑**Object oriented methodologies**
  - ❑ Object-oriented systems development centers on the object, which combines data and functionality.

SUMMARY OBJECT ORIENTED SYSTEM DEVELOPMENT Vs TRADITIONAL APPROACH

TRADITIONAL APPROACH	OBJECT ORIENTED SYSTEM DEVELOPMENT
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step of process	Focuses on object, classes, modules that can be easily replaced, modified and reused.
Moving from one phase to another phase is complex.	Moving from one phase to another phase is easier.
Increases complexity	Reduces complexity and redundancy
Increases duration of project	decreases duration of project

Software Process

- Process: A particular method, generally involving a number of steps
- Software Process: A set of steps, along with ordering constraints on execution, to produce software with desired outcome (high P&Q)

- Many types of activities performed by different people in a software project
- Better to view software process as comprising of many component processes

- A process is a means to reach the goals of high quality, low cost, and low cycle time, and a process model provides generic guidelines for developing a suitable process for a project.

- A process model specifies a general process, usually as a set of stages in which a project should be divided, the order in which the stages should be executed, and any other constraints and conditions on the execution of stages.

Component Software Processes

- Two major processes
  - **Development** – focuses on development and quality steps needed to engineer the software
  - **Project management** – focuses on planning and controlling the development process

Development process

- Development process is the heart of software process; other processes revolve around it
- These are executed by different people
  - developers execute engineering process
  - project manager executes the management process

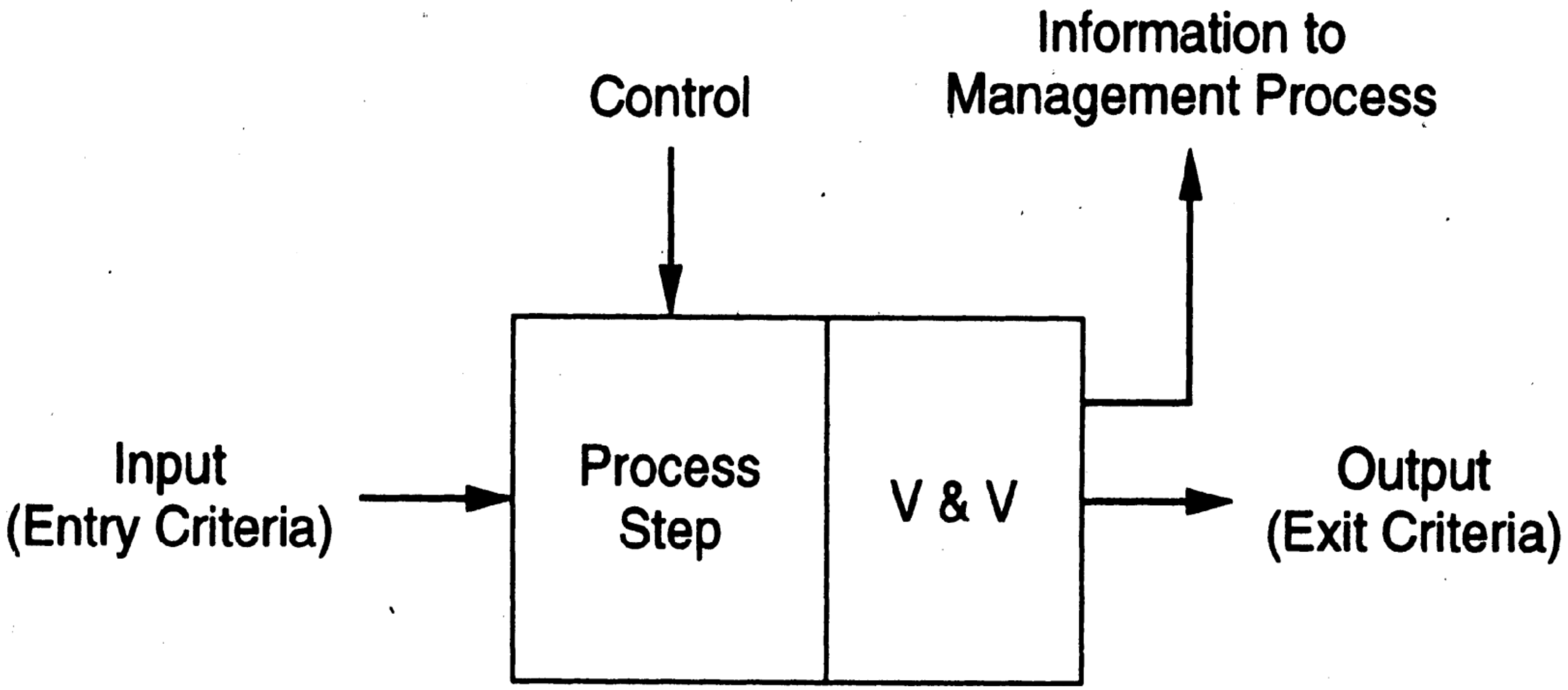
- Other processes
  - **Configuration management process:** manages the evolution of artifacts
  - **Change management process:** how changes are incorporated
  - **Process management process:** management of processes themselves
  - **Inspection process:** How inspections are conducted on artifacts

ETVX Approach for Process Specification

- Process is generally a set of phases
- Each phase performs a well defined task and generally produces an output
    - Output of development process, which are not the final output, are frequently called the *work products.*
        - a work product can be the requirements document, design document, code, prototype, and the like.

- Output must be a formal and tangible entity.
- Output of each step must be some work product that can be verified
  - process should have a small number of steps.
    - Due to this, at the top level, a process typically consists of a few steps
- How to perform the activity of the particular step or phase is generally addressed by *methodologies for that activity.*

- (Entry-Task-Verification-Exit)ETVX approach to specify a step
    - Entry criteria: what conditions must be satisfied for initiating this phase
    - Task: what is to be done in this phase
    - Verification: the checks done on the outputs of this phase
    - eXit criteria: when can this phase be considered done successfully
- A phase also produces info for management


Control
Information to Management Process
Input (Entry Criteria)
Process Step
V & V
Output (Exit Criteria)

Characteristics of Software Process

- **Predictability and repeatability**
  - Predictability of a process determines how accurately the outcome of following that process in a project can be predicted before the project is completed.
  - Process should repeat its performance when used on different projects
    - outcome of using a process should be predictable

- Without predictability, cannot estimate, or say anything about quality , cost or productivity
- With predictability, past performance can be used to predict future performance

- Predictable process is said to be under statistical control
  - Repeatedly using the process produces similar results
- To consistently develop sw with high Q&P, process must be in control

- **Support Change**
  - Software changes for various reasons
  - Requirements change is a key reason
  - Requirement changes cannot be wished away or treated as “bad”
  - They must be accommodated in the process for sw development

- **Support Testability and Maintainability**
    - In the life of software the maintenance costs generally exceed the development costs.
    - Objective of development
        - to produce software that is easy to maintain. And the process used should ensure this maintainability
    - Support testability as testing is the most expensive task

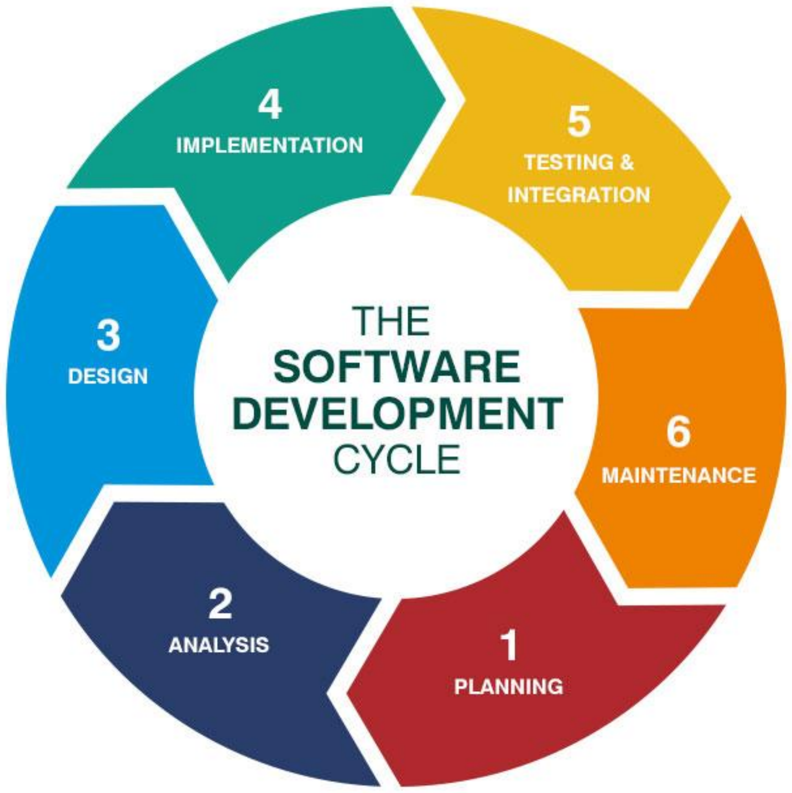
- **Early Defect Removal**
  - Errors can occur at any stage during development.
  - Cost of correcting errors of different phases is not the same and depends on when the error is detected and corrected.

- The greater the delay in detecting an error after it occurs, the more expensive it is to correct it.
- Attempt to detect errors that occur in a phase during that phase itself and should not wait until testing to detect errors.

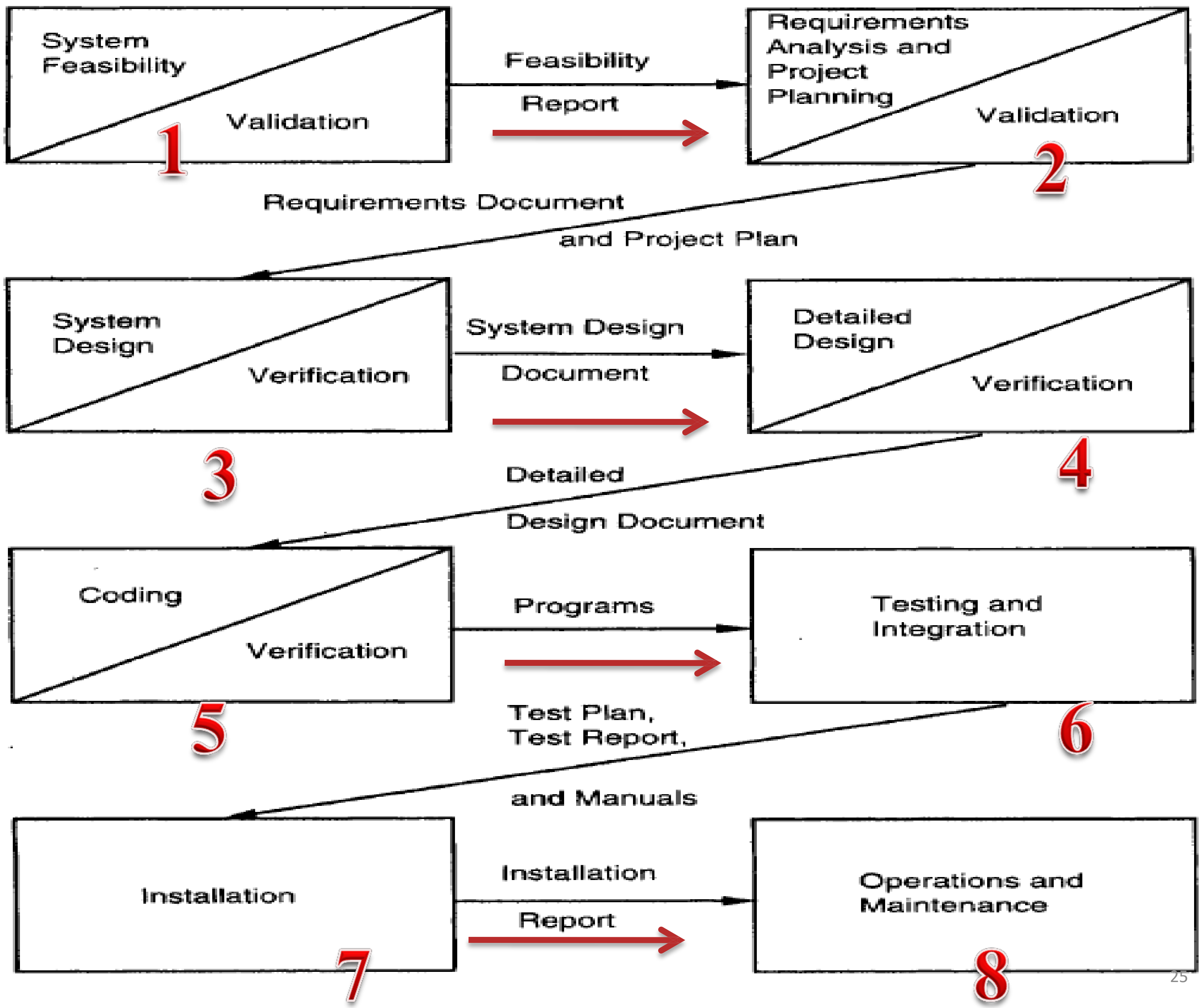
- A process should have quality control activities spread through the process and in each phase.

SOFTWARE LIFE CYCLE AND PROCESS MODELS

- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares.
- The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.



- Due to the importance of the development process, various models have been proposed.
  - **Waterfall Model**
  - **Prototyping**
  - **Iterative Development**
  - **Timeboxing Model**
  - **Spiral model**
  - **Agile model**


System Feasibility
Validation
1
Feasibility Report
Requirements Analysis and Project Planning
Validation
2
Requirements Document and Project Plan
System Design
Verification
3
System Design Document
Detailed Design
Verification
4
Detailed Design Document
Coding
Verification
5
Programs
Testing and Integration
6
Test Plan, Test Report, and Manuals
Installation
7
Installation Report
Operations and Maintenance
8

- advantages of this model is:
  - its simplicity.
  - divides the large task of building a software system into a series of cleanly divided phases, each phase dealing with a separate logical concern.
  - It is also easy to administer

- Limitations
  - It assumes that the requirements of a system can be frozen (i.e., base lined) before the design begins.
    - This is possible for systems designed to automate an existing manual system.
  - But user requirements are usually unknown and can be changed

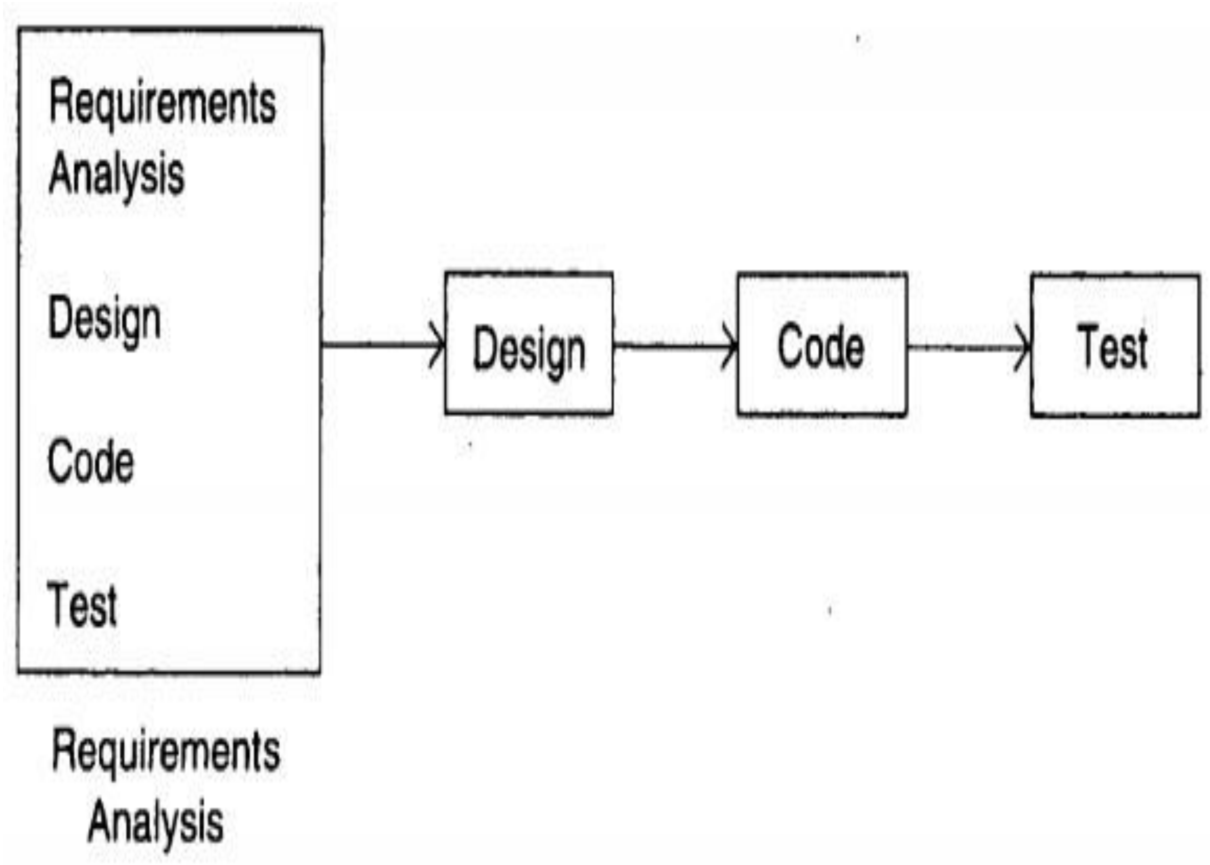
- Freezing the requirements usually requires choosing the hardware (because it forms a part of the requirements specification).
  - Large projects can take few years to complete by then the hardware specified might be obsolete.

- The entire software is delivered in one shot at the end. This entails heavy risks, as the user does not know until the very end what they are getting.
- It is a document-driven process that requires formal documents at the end of each phase.

Prototyping

- Instead of freezing the requirements before any design or coding can proceed
  - a throw-away prototype is built to help understand the requirements.

- Based on known requirement design, coding, and testing of the prototype is done.
- The user interact with prototype and give feedback
- Final results in more stable requirements that change less frequently.


Requirements Analysis
Design
Code
Test
Design
Code
Test
Requirements Analysis

- Suitable for
  - complicated and large systems for which there is no manual process or existing system to help determine the requirements.
  - It is also an effective method of demonstrating the feasibility of a certain approach.

- Exception handling, recovery, and conformance to some standards and formats are typically not included in prototypes.
- minimal documentation need (design documents, a test plan, and a test case specification are not needed)

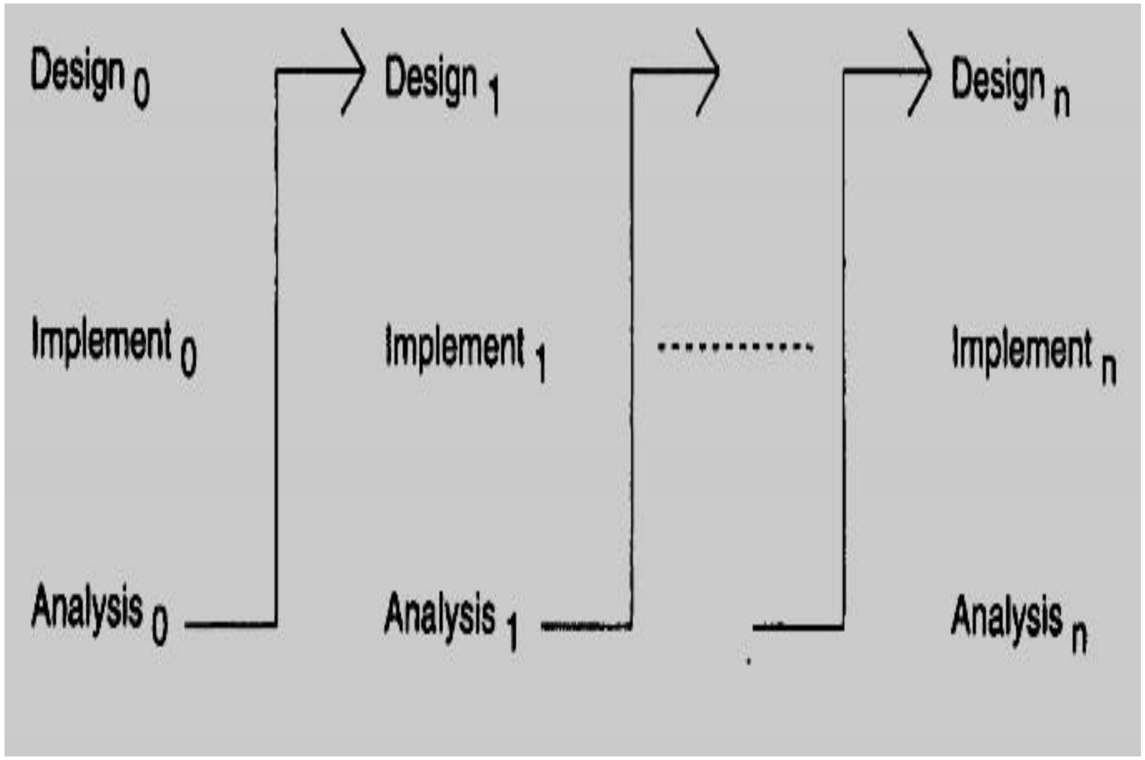
Iterative Development

- Basic idea
  - Software developed in increments, each increment adding some functional capability to the system until the full system is implemented
- Combine the benefits of both prototyping and the waterfall model.

- Advantage
  - better testing because testing each increment is likely to be easier than testing the entire system as in the waterfall model.
  - as in prototyping, the increments provide feedback to the client that is useful for determining the final requirements of the system

The iterative enhancement model

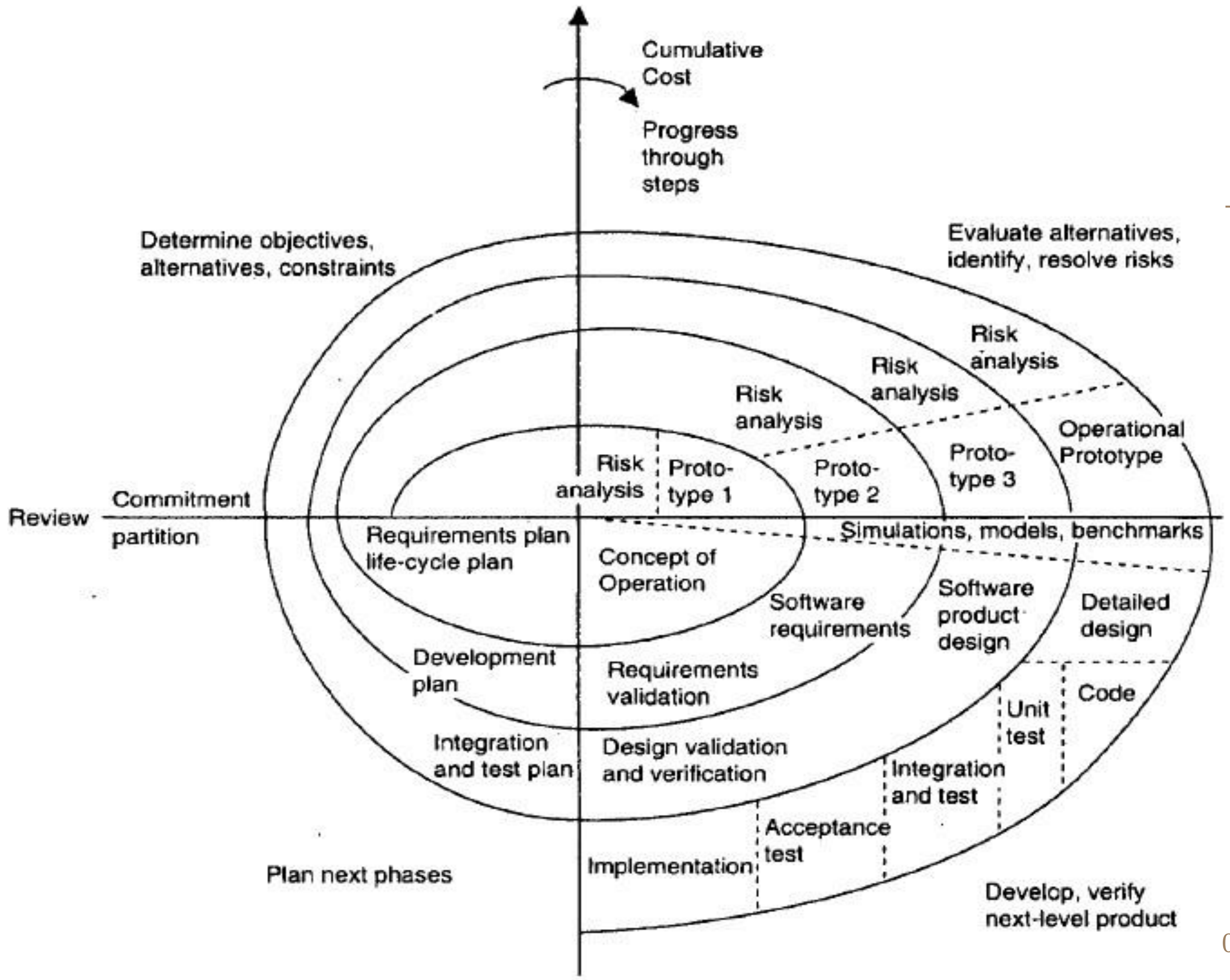
- The iterative enhancement model is an example of this approach.
- steps
  - First create initial implementation & project control list (tasks that must be performed to obtain the final implementation.)
  - Remove one task at a time and
    - design → implement → analysis
  - The process is iterated until the project control list is empty


Design $_0$
Implement $_0$
Analysis $_0$
Design $_1$
Implement $_1$
Analysis $_1$
Design $_n$
Implement $_n$
Analysis $_n$

The spiral model

- It is another iterative model that has been proposed
- Models do not deal with uncertainly which is inherent to software projects.
- Important software projects have failed because project risks were neglected & nobody was prepared when something unforeseen happened.

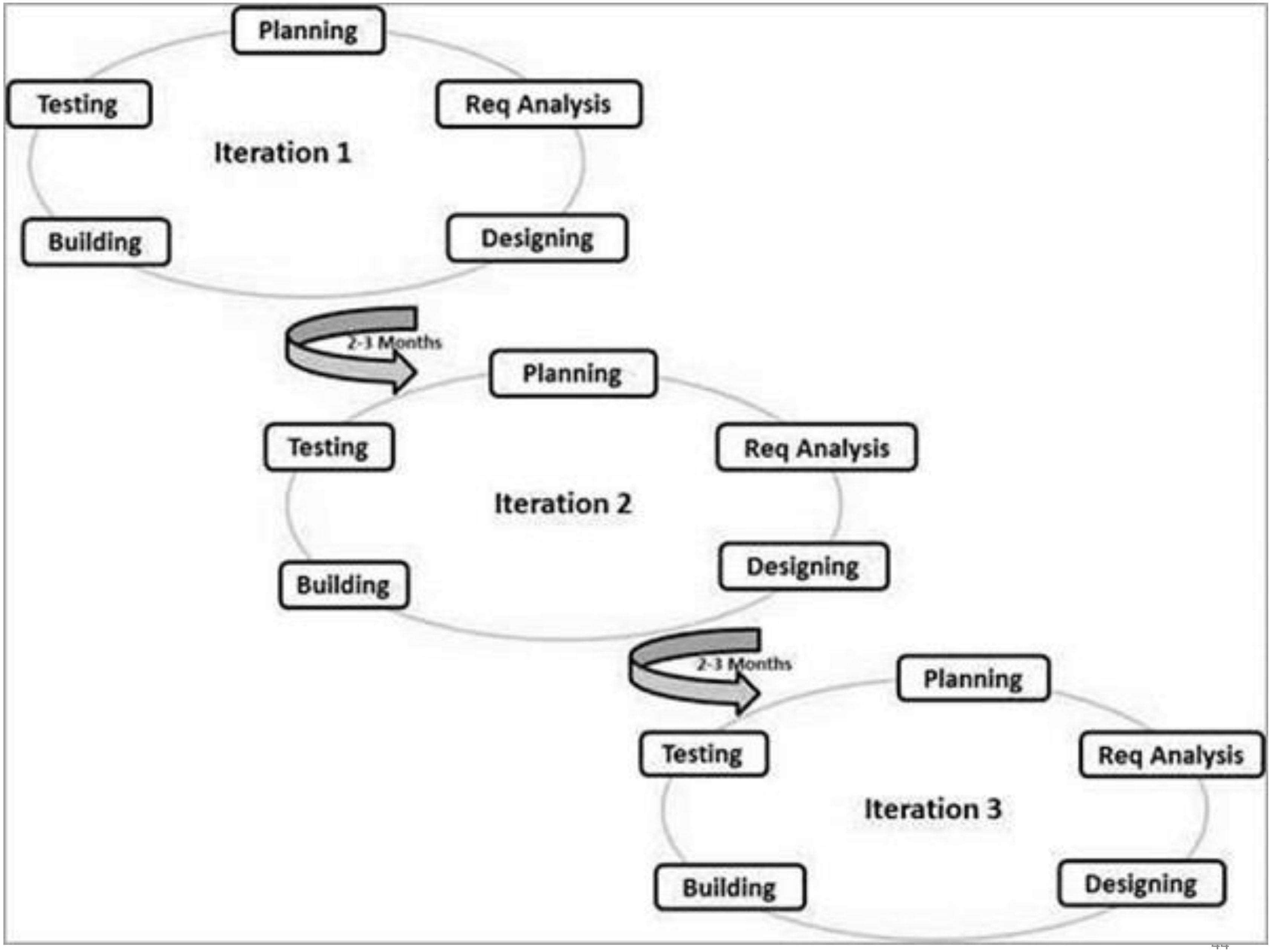
- Barry Boehm recognized this and tired to incorporate the “project risk” factor into a life cycle model.
- The result is the spiral model, which was presented in 1986.


Cumulative Cost
Progress through steps
Determine objectives, alternatives, constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Proto-type 1
Proto-type 2
Proto-type 3
Operational Prototype
Review
Commitment partition
Requirements plan life-cycle plan
Concept of Operation
Simulations, models, benchmarks
Software requirements
Software product design
Detailed design
Development plan
Requirements validation
Code
Unit test
Integration and test plan
Design validation and verification
Integration and test
Acceptance test
Implementation
Plan next phases
Develop, verify next-level product

Agile model

- Agile SDLC model is a combination of iterative incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements.
- In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

- Iterative approach is taken and working software build is delivered after each iteration.
- Each build is incremental in terms of features; the final build holds all the features required by the customer.
- Every iteration involves cross functional teams working simultaneously on various areas like –
- Planning
    - Requirements Analysis
    - Design
    - Coding
    - Unit Testing and
    - Acceptance Testing.
- At the end of the iteration, a working product is displayed to the customer and important stakeholders.


Planning
Testing
Req Analysis
Iteration 1
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 2
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 3
Building
Designing

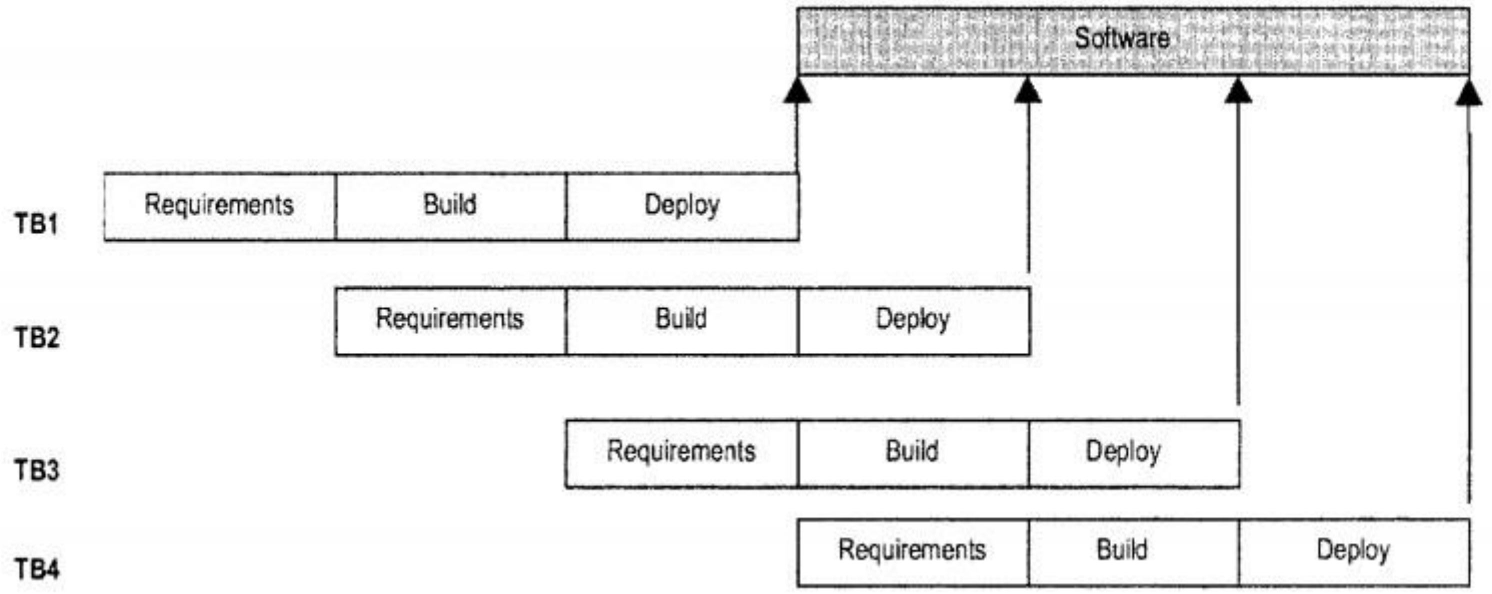
- The advantages of the Agile Model are as follows –
  - Is a very realistic approach to software development.
  - Promotes teamwork and cross training.
  - Functionality can be developed rapidly and demonstrated.
  - Resource requirements are minimum.
  - Suitable for fixed or changing requirements
  - Delivers early partial working solutions.
  - Good model for environments that change steadily.
  - Minimal rules, documentation easily employed.
  - Enables concurrent development and delivery within an overall planned context.
  - Little or no planning required.
  - Easy to manage.
  - Gives flexibility to developers.

- The disadvantages of the Agile Model are as follows –
  - Not suitable for handling complex dependencies.
  - More risk of sustainability, maintainability and extensibility.
  - An overall plan, an agile leader and agile PM practice is a must without which it will not work.
  - Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
  - Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
  - There is a very high individual dependency, since there is minimum documentation generated.
  - Transfer of technology to new team members may be quite challenging due to lack of documentation.

Timeboxing Model

- Uses parallelism between the different iterations
- a new iteration commences before the system produced by the current iteration is released
- Each time box is divided into a sequence of stages,

- Each stage performs some clearly defined task for the iteration and produces a clearly defined output.
- duration of each stage is the same
- team for a stage performs only tasks of that stage


Software
TB1
Requirements
Build
Deploy
TB2
Requirements
Build
Deploy
TB3
Requirements
Build
Deploy
TB4
Requirements
Build
Deploy

- Example
  - If the time box is of size T days, then the first software delivery will occur after T days. The subsequent deliveries, however, will take place after every T/3 days.
    - T is 9 weeks
      - first delivery is made on 9th week, then 12th , 15th week

- Linear execution of iterations,
  - The first delivery will be made after 9 weeks,
  - the second will be made after 18 weeks,
  - the third after 27 weeks,

- **Man-power**
  - Suppose ;
    - it takes 2 people 2 weeks to do the requirements for iteration,
    - 4 people 2 weeks to do the build for the iteration, and
    - it takes 3 people 2 weeks to test and deploy.
      - If the iterations are serially executed, then the team for the project will be 4 people (the maximum size needed for a stage)
      - If this project is executed using the timeboxing process model separate team is needed so (2+4+3=9)peoples are needed
  - Hence, the timeboxing provides an approach for utilizing additional man-power to reduce the delivery time.

- Timeboxing is well suited for projects that require a large number of features to be developed in a short time

- Timeboxing is not suitable for
  - Projects where it is difficult to partition the overall development into multiple iterations of approximately equal duration.
  - It is also not suitable for projects where different iterations may require different stages, and
  - for projects whose features are such that there is no flexibility to combine them into meaningful deliveries.

Process assessment models

- A software process assessment is a disciplined examination of the software processes used by an organization, based on a process model.
- The assessment includes the identification and characterization of current practices, identifying areas of strengths and weaknesses, and the ability of current practices to control or avoid significant causes of poor (software) quality, cost, and schedule.
- A software assessment (or audit) can be of three types.
  - A **self-assessment (first-party assessment)** is performed internally by an organization's own personnel.
  - A **second-party assessment** is performed by an external assessment team or the organization is assessed by a customer.
  - A **third-party assessment** is performed by an external party or (e.g., a supplier being assessed by a third party to verify its ability to enter contracts with a customer).

- Software process assessments are performed in an open and collaborative environment.
- They are for the use of the organization to improve its software processes, and the results are confidential to the organization.
- The organization being assessed must have members on the assessment team.

Software Process Assessment Cycle

- According to Paulk and colleagues (1995), the CMM-based assessment approach uses a six-step cycle. They are:
  - Select a team - The members of the team should be professionals knowledgeable in software engineering and management.
  - The representatives of the site to be appraised complete the standard process maturity questionnaire.
  - The assessment team performs an analysis of the questionnaire responses and identifies the areas that warrant further exploration according to the CMM key process areas.
  - The assessment team conducts a site visit to gain an understanding of the software process followed by the site.
  - The assessment team produces a list of findings that identifies the strengths and weakness of the organization's software process.
  - The assessment team prepares a Key Process Area (KPA) profile analysis and presents the results to the appropriate audience.

Software process metrics

- Process metrics are the measures of the development process that creates a body of software. A common example of a process metric is the length of time that the process of software creation tasks.
- Based on the assumption that the quality of the product is a direct function of the process, process metrics can be used to estimate, monitor, and improve the reliability and quality of software
- Process metrics are often collected as part of a model of software development. Models such as Boehm’s COCOMO (**Constructive Cost Model**) make cost estimations about software projects. Thebaut’s COPMO makes predictions about the need for additional effort on large projects.
- Although valuable management tools, process metrics are not directly relevant to program understanding. They are more useful in measuring and predicting such things as resource usage and schedule.

Object oriented system development methodology.

- object-oriented systems development is a way to develop software by building self-contained modules or objects that can be easily replaced, modified, and reused.
- It encourages a view of the world as a system of cooperative and collaborating objects.
- In an object-oriented environment, software is a collection of discrete objects that encapsulate their data as well as the functionality to model real-world "objects."
- An object orientation yields important benefits to the practice of software construction.

- Each object has attributes (data) and methods (functions).
- Objects are grouped into classes; in object-oriented terms, we discover and describe the classes involved in the problem domain.

WHY AN OBJECT ORIENTATION?

- **Higher level of abstraction:**
  - The object-oriented approach supports abstraction at the object level.
  - Since objects encapsulate both data (attributes) and functions (methods), they work at a higher level of abstraction.
  - The development can proceed at the object level and ignore the rest of the system for as long as necessary.
  - This makes designing, coding, testing, and maintaining the system much simpler.

- ***Seamless transition among different phases of software development:***
    - The traditional approach to software development requires different styles and methodologies for each step of the process.
    - Moving from one phase to another requires a complex transition of perspective between models that almost can be in different worlds.
    - This transition not only can slow the development process but also increases the size of the project and the chance for errors introduced in moving from one language to another.
    - The object-oriented approach, on the other hand, essentially uses the same language to talk about analysis, design, programming, and database design.
    - This seamless approach reduces the level of complexity and redundancy and makes for clearer, more robust system development

- ***Encouragement of good programming technique:***
    - A class in an object-oriented system carefully delineates between its interface (specifications of what the class can do) and the implementation of that interface (how the class does what it does).
    - The routines and attributes within a class are held together tightly.
    - In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact on other classes, and so, the impact is minimized.

- The object-oriented approach will promote perfect design or perfect code by raising the level of abstraction from the function level to the object level and by focusing on the real-world aspects of the system, the object-oriented method tends to promote clearer designs, which are easier to implement, and provides for better overall communication.
- Using object-oriented language is not strictly necessary to achieve the benefits of an object orientation.
- However, an object-oriented language such as C+ +, Smalltalk, or Java adds support for object-oriented design and makes it easier to produce more modular and reusable code via the concept of class and inheritance.

- ***Promotion of reusability:***
  - Objects are reusable because they are modeled directly out of a real—world problem domain.
  - Each object stands by itself or within a small circle of peers (other objects).
  - Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.
  - This means that classes are designed generically, with reuse as a constant background goal.
  - Furthermore, the object orientation adds inheritance, which is a powerful technique that allows classes to be built from each other, and therefore, only differences and enhancements between the classes need to be designed and coded.
  - All the previous functionality remains and can be reused without change.

Overview of the unified approach (UA)

- ❑The Unified Approach (UA) is a methodology for software development.
- ❑It tries to combine the best practices, processes and guidelines along with the Object Management Group's unified modeling language.
- ❑The unified modeling language (UML) is a set of notations and conventions used to describe and model an application.
- ❑However, the UML does not specify a methodology or what steps to follow to develop an application; that would be the task of the UA.
- ❑The heart of the UA is Jacobson's use case.
- ❑The use case represents a typical interaction between a user and a computer system to capture the users' goals and needs.

- ❑In its simplest usage, you capture a use case by talking to typical users and discussing the various ways they might want to use the system.
- ❑The use cases are entered into all other activities of the UA.
- ❑The UA establishes a unifying and unitary framework around their works by utilizing the UML to describe the model and document the software development process.
- ❑The idea behind the UA is not to introduce yet another methodology.
- ❑The main motivation here is to combine the best practices, processes, methodologies, and guidelines along with UML notations and diagrams for better understanding of object-oriented concepts and system development.

Unified approach road map

- **Analysis**
  - Identify the user
  - Develop a simple business model
  - Develop use case
  - Interaction diagram
    - Develop sequence diagram
    - Develop collaboration diagram
  - Classification
    - Identify classes
    - Identify relationship
    - Identify attributes
    - Identity methods

- **Design**
  - Apply design axioms to design classes
  - Design the access layer
  - Design view layer classes
    - Macro level UI design
    - Micro level UI design
    - Usability and user satisfaction test

- The unified approach to software development revolves around (but is not limited to) to the following processes and concepts. The processes are:
- 1. Use-case driven development
- 2. Object-oriented analysis
- 3. Object-oriented design
- 4. Incremental development and prototyping
- 5. Continuous testing

- The UA allows iterative development by allowing you to go back and forth between the design and the modeling or analysis phases.
- It makes backtracking very easy and departs from the linear waterfall process, which allows no form of back tracking.

Object-Oriented Analysis

- ❑Analysis is the process of extracting the needs of a system and what the system must do to satisfy the users' requirements.
- ❑The goal of object-oriented analysis is to first understand the domain of the problem and the system's responsibilities by understanding how the users use or will use the system.
- ❑It concentrates on describing what the system does rather than how it does it.
- ❑Separating the behavior of a system from the way it is implemented require viewing the system from the user's perspective rather than that of the machine.
- ❑OOA process consists of the following steps:
  1. Identify the Actors.
  2. Develop a simple business process model using UML Activity diagram.
  3. Develop the Use Case.
  4. Develop interaction diagrams.
  5. Identify classes.

Object-Oriented Design

- OOD Process consists of:
  1. Designing classes, their attributes, methods, associations, structures and protocols, apply design axioms.
  2. Design the Access Layer
  3. Design and prototype User interface
  4. User Satisfaction and Usability Tests based on the Usage/Use Cases
  5. Iterated and refine the design

Iterative Development and Continuous Testing

- You must iterate and reiterate until, eventually, you are satisfied with the system.
- Sine testing often uncovers design weaknesses or at least provides additional information you will want to use, repeat the entire process, taking what you have learned and reworking your design or moving on the prototyping and retesting.
- Continue this refining cycle through the development process until you are satisfied with the results.

- During this iterative process, your prototypes will be incrementally transformed into the actual application.
- The UA encourages the integration of testing plans from day 1 of the project.
- Usage scenarios can become test scenarios; therefore, use case will drive the usability testing.
- Usability testing is the process in which the functionality of software is measured.

An object oriented philosophy Object-Oriented Analysis (OOA)

- ❑The object model has influenced even earlier phases of the software development lifecycle.
- ❑Traditional structured analysis techniques focus on the flow of data within a system.
- ❑Object-oriented analysis (OOA) emphasizes the building of real-world models, using an object-oriented view of the world:
  - ❑Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.

Object-Oriented Design (OOD)

- Design methods emphasize the proper and effective structuring of a complex system.
- ***Object-oriented design:*** is a method of design encompassing the process of object oriented decomposition and a notation for depicting both logical (class and object structure) and physical ((module and process architecture) as well as static and dynamic models of the system under design.

- The support for object-oriented decomposition is what makes object-oriented design quite different from structured design:
- The former uses class and object abstractions to logically structure systems, and the latter uses algorithmic abstractions.

Object oriented programming (OOP)

- ❑Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.
- ❑There are three important parts to this definition:
  - ✓Object-oriented programming uses objects, not algorithms, as its fundamental logical building blocks
  - ✓each object is an instance of some class; and
  - ✓classes may be related to one another via inheritance relationships

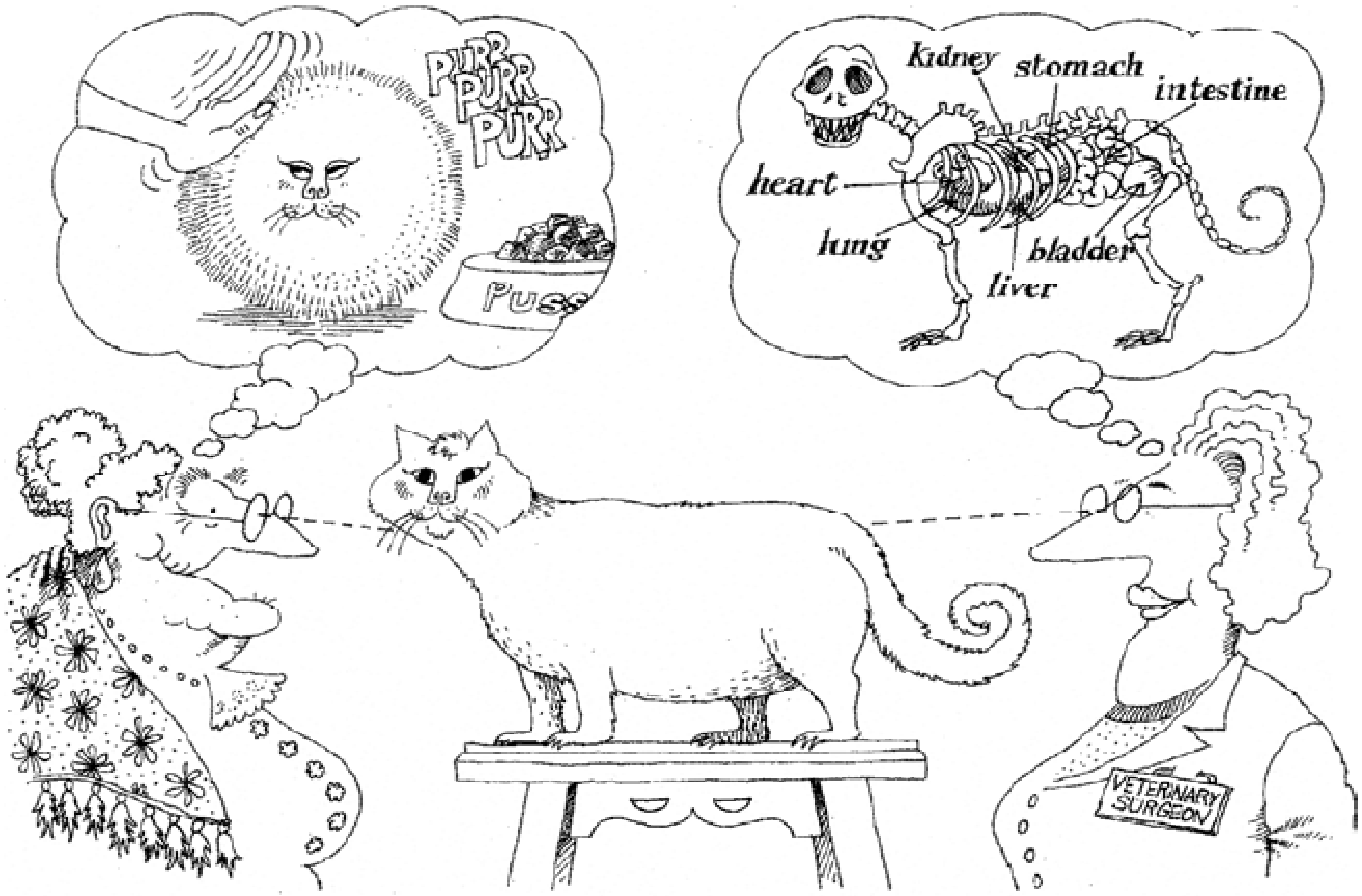
- How are OOA, OOD, and OOP related?
  - Basically, the products of object-oriented analysis serve as the models from which we may start an object-oriented design the products of object-oriented design can then be used as blueprints for completely implementing a system using object-oriented programming methods.

Elements of Object model

- There are four major elements of this model:
  1. Abstraction
  2. Encapsulation
  3. Modularity
  4. Hierarchy
- By *major, we mean that a model without any one of these elements is not object oriented.*
- There are three minor elements of the object model:
  1. Typing
  2. Concurrency
  3. Persistence
- By *minor, we mean that each of these elements is a useful, but not essential, part* of the object model.

The Meaning of Abstraction

- ❑An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.
- ❑An abstraction focuses on the outside view of an object and so serves to separate an object’s essential behavior from its implementation.

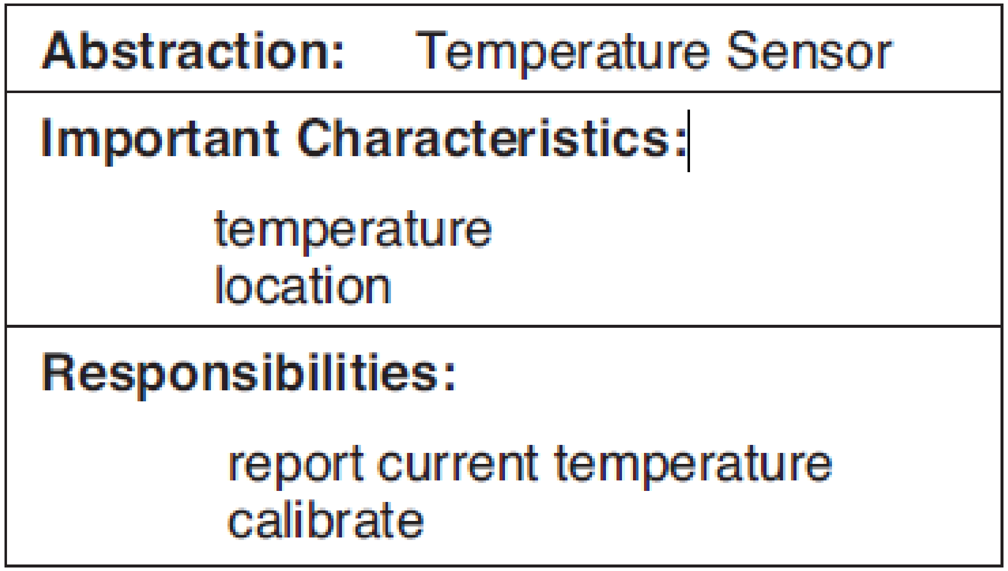


Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Examples of Abstraction

- ❑On a hydroponics farm, plants are grown in a nutrient solution, without sand, gravel, or other soils.
- ❑Maintaining the proper greenhouse environment is a delicate job and depends on the kind of plant being grown and its age.
- ❑One must control diverse factors such as temperature, humidity, light, pH, and nutrient concentrations.
- ❑On a large farm, it is not unusual to have an automated system that constantly monitors and adjusts these elements.
- ❑Simply stated, the purpose of an automated gardener is to efficiently carry out, with minimal human intervention, growing plans for the healthy production of multiple crops.

- One of the key abstractions in this problem is that of a sensor.
- Actually, there are several different kinds of sensors.
- Anything that affects production must be measured, so we must have sensors for air and water temperature, humidity, light, pH, and nutrient concentrations, among other things.



Abstraction: Temperature Sensor

Important Characteristics:

temperature
location

Responsibilities:

report current temperature
calibrate

Abstraction of a `Temperature Sensor`

The Meaning of Encapsulation

- ❑Abstraction and encapsulation are complementary concepts:
- ❑Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

The Meaning of Modularity

- ❑The act of partitioning a program into individual components can reduce its complexity to some degree.
- ❑Although partitioning a program is helpful for this reason, a more powerful justification for partitioning a program is that it creates a number of well-defined, documented boundaries within the program.
- ❑For larger applications, in which we may have many hundreds of classes, the use of modules is essential to help manage complexity
- ❑Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.

The Meaning of Hierarchy

- ❑Hierarchy is a ranking or ordering of abstractions.
- ❑The two most important hierarchies in a complex system are its class structure
  - ✓ the “is a” hierarchy; generalization/specialization and
  - ✓its object structure (the “part of” hierarchy; aggregation).