Software Engineering

- An engineering approach to develop software.
- It focuses systematic and cost-effective techniques to software development.
- Software engineering is defined as the systematic approach to the development, operation, maintenance, and retirement of software.

Why Study Software Engineering?

- To acquire skills to develop large software products.
- Can effectively handle complexity in a software development problem.
- Learn techniques of:
  - specification, design, interface development, testing, project management, etc.
- To acquire skills to be a better programmer.

Emergence of Software Engineering

1) Early Computer Programming (1950s):
   - Programs were being written in assembly language.
   - Programs were limited to about a few hundreds of lines of assembly code.
   - Every programmer developed his own style of writing programs:
     - according to his intuition (exploratory programming – build and fix).

2) High-Level Language Programming (Early 60s)

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:
  - This reduced software development efforts greatly.
- Software development style was still exploratory.
- Typical program sizes were limited to a few thousands of lines of source code.

3) Control Flow-Based Design (late 60s)

- Size and complexity of programs increased further:
  - exploratory programming style proved to be insufficient.
- Programmers found:
  - very difficult to write cost-effective and correct programs.
  - programs written by others are very difficult to understand and maintain.

Control Flow-Based Design (late 60s)

- To cope up with this problem, experienced programmers advised:

  ``Pay particular attention to the design of the program's control structure.‘’

- A program's control structure indicates:
  - the sequence in which the program's instructions are executed.

Control Flow-Based Design (late 60s)

- To help design programs having good control structure:
  - flow charting technique was developed.

- Using flow charting technique:
  - one can represent and design a program's control structure.
- Usually one understands a program:
  - by mentally simulating the program's execution sequence.

4) Data Structure Oriented Design (Early 70s)

It gives more attention to the design of data structures of a program than to the design of its control structure.

- Techniques which emphasize
  - designing the data structure
  - derive program structure from it

Data Structure Oriented Design (Early 70s)

- Example of data structure-oriented design technique:
  - Jackson's Structured Programming(JSP) methodology
- In JSP methodology:
  - a program's data structures are first designed using notations for
    - sequence, selection, and iteration.
  - Then data structure design is used :
    - to derive the program structure.

5) Data Flow-Oriented Design (Late 70s)

- In Data flow-oriented technique
  - the data items input to a system must first be identified
  - processing required on the data items to produce the required outputs should be determined.
- Data flow technique identifies:
  - different processing stations (functions) in a system
  - the items (data) that flow between processing stations.

Data Flow-Oriented Design (Late 70s)

- Data flow technique is a generic technique:
  - can be used to model the working of any system.
- A major advantage of the data flow technique is its simplicity.

6) Object-Oriented Design (80s)

- In Object-oriented technique:
  - natural objects (such as employees, pay-roll-register, etc.) occurring in a problem are first identified.
- Relationships among objects:
  - such as composition, reference, and inheritance are determined.
- Each object essentially acts as
  - a data hiding (or data abstraction) entity.

Object-Oriented Design (80s)

- Object-Oriented Techniques have gained wide acceptance:
  - Simplicity
  - Reuse possibilities
  - Lower development time and cost
  - More robust code
  - Easy maintenance

Software Process

- A process is the **sequence of steps** executed to achieve a goal.

- To satisfy different goals while developing software, multiple processes are needed.

- Many of these do not concern software engineering, but they have impact on software development.

- These are nonsoftware processes - Business processes, social processes, and training processes.

Software Process

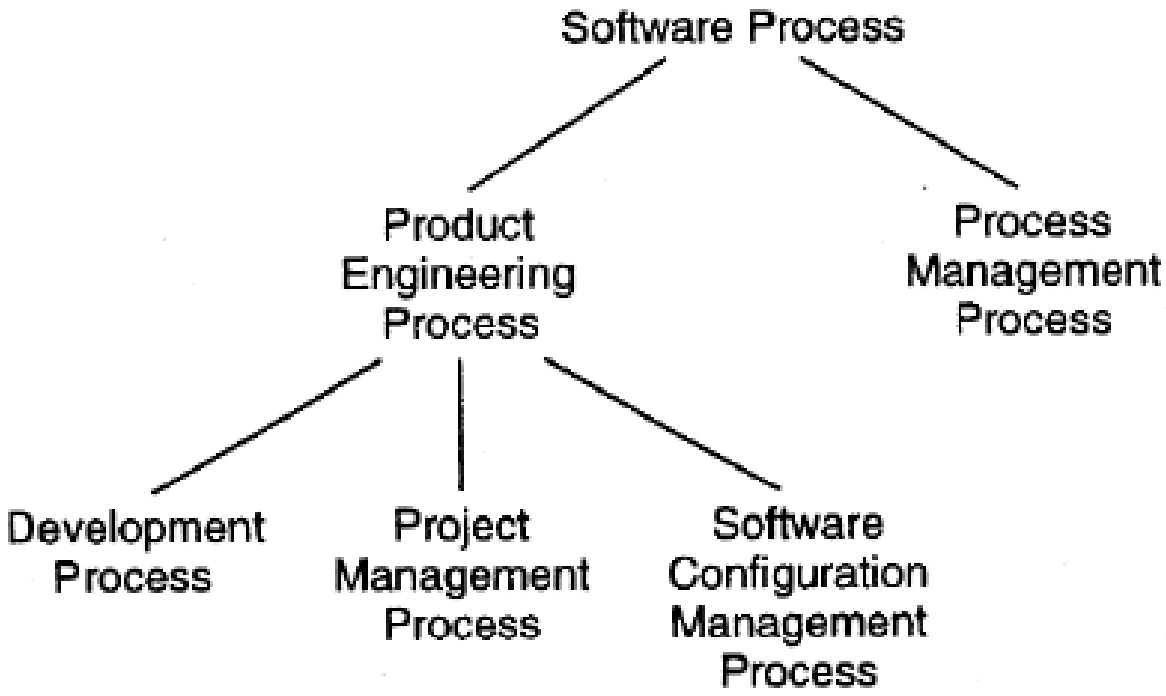
- The processes that deal with the **technical** and **management** issues of software development are collectively called the **software process**.

- There are two major components in a software process—a **development process** and a **project management process.**
- The **development process** specifies all the engineering activities that need to be performed.

- The **management process** specifies how to plan and control these activities so that cost, schedule, quality, and other objectives are met.

Software Process

- Effective **development** and **project management** processes are the key to achieving the objectives of software satisfying the user needs, while ensuring high productivity and quality.



Phases of Software Development

- A **software life cycle** is a series of identifiable stages that a software product undergoes during its lifetime.

- Software life cycle is also called **Software Development Life Cycle (SDLC)**

- The first stage of life cycle is **feasibility study**.

- Subsequent stages are – **requirement analysis and specification, design, coding, testing,** and **maintenance**.

- Each of these stages is called – **life cycle phase**.

Phases of Software Development

1. Feasibility study

- Aim is to determine whether it would be **financially** and **technically feasible** to develop the product.

- It involves **analysis of the problem** and **collection of all relevant information** related to the product – input data items, processing required, the output data, and various constraints.

After data collection, this phase arrives at:

- An abstract problem definition
- Formulation of different strategies for solving problem
- Evaluation of different solution strategies
  – examine benefits and shortcomings.

Phases of Software Development

2. Requirement analysis and specification

- This phase determines exact **requirements** of customers and to **document** them properly.

1. Requirement gathering and analysis – First gathering the requirements and then analysing the gathered requirements.

The aim of requirement analysis is to remove the **incompleteness** and **inconsistencies** in requirement.

2. Requirement specification – The requirements identified are organised **into Software Requirement Specification (SRS)** document.

Phases of Software Development

3. Design

- Design transforms the requirements specified in SRS document into structure suitable for implementation in programming language.
- During this phase the **software architecture** is derived from the SRS document.

4. Coding

- It translates the software design into source code.
- This phase is also called **Implementation** phase.
- Each component of the design is implemented as a program module.

Phases of Software Development

5. Testing

The aim of testing is to **identify all defects** in a program.

- Testing a program involves providing a program with a set of test inputs and observing whether the program behaves as expected.

- **Unit testing** - testing each module, debugging, and documenting.
- **Integration testing** - different modules are integrated. After each step, the resultant module is tested.
- **System testing** - System testing is designed to validate a fully developed system to assure that it meets its requirements.

Phases of Software Development

6. Maintenance

- Software needs to be maintained because of the defects remaining in the system. Developing software with zero defect is not possible.

1. Corrective maintenance – correcting errors that were not discovered during development.

2. Perfective maintenance – adding functionalities according to customer's requirement.

3. Adaptive maintenance – porting the software to work in a new environment – new computer platform/operating system

Life Cycle Models

A life cycle model explains the **different activities** that needed to be carried out to develop a software product and **sequencing** of these activities.

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.

Life Cycle Models

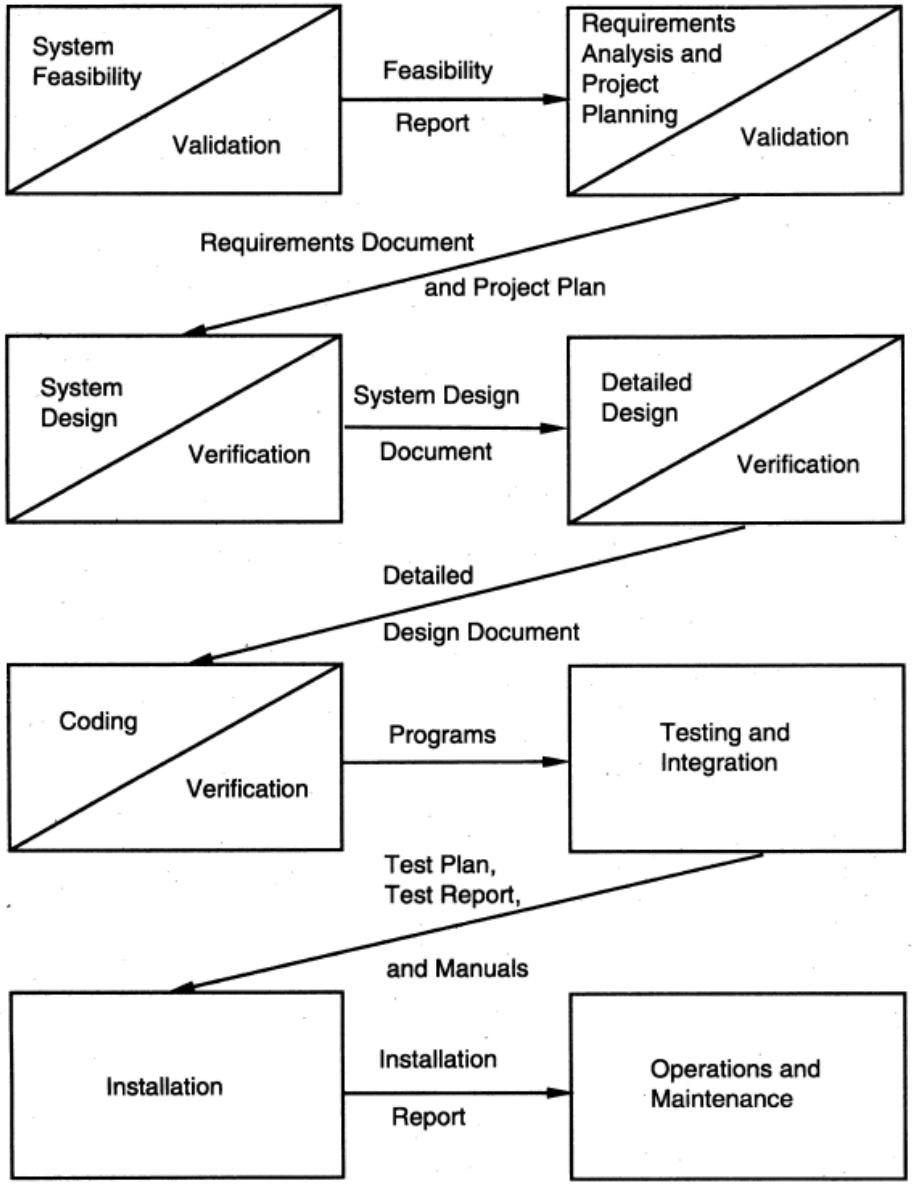
1. Waterfall Model
2. Prototyping Model
3. Iterative Model
4. Spiral Model
5. Agile Process

1. Waterfall Model

- The simplest process model is the waterfall model, in this **the phases are organized in a linear order**.

- In this model, a project begins with **feasibility analysis**.
- Upon successfully demonstrating the feasibility of a project, **the requirements analysis** and **project planning** begins.
- The **design** starts after the requirements analysis is complete.
- **Coding** begins after completion of design.
- After coding completed, the code is **integrated** and **testing** is done.
- After successful completion of testing, the system is **installed**.
- After this, the **operation** and **maintenance** of the system takes place.

1. Waterfall Model



1. Waterfall Model

- The idea behind the phases is **separation of tasks**- each phase deals with a distinct and separate set of tasks.

- The large and complex task of building the software is broken into smaller tasks of specifying requirements.

- Separating the tasks and focusing on a few in a phase gives a better handle to the engineers and managers in dealing with the complexity of the problem.

1. Waterfall Model

- The **requirements analysis** phase is mentioned as **"analysis and planning."**

- Planning is a critical activity in software development.

- A good plan is based on the requirements of the system and should be done before later phases begin.

- To clearly identify the end of a phase and the beginning of the next, some certification mechanism has to be employed at the end of each phase.

- This is done by some verification and validation means, that will ensure that the output of a phase is consistent with its input.

1. Waterfall Model

- The outputs of the earlier phases are often called **work products** and are usually in the form of **documents** like the **requirements document** or **design document**.

- For the coding phase, the output is the code.

- The following documents generally produced in each project:

  - **Requirements document**
  - **Project plan**
  - **Design documents (architecture, system, detailed)**
  - **Test plan and test reports**
  - **Final code**
  - **Software manuals (e.g., user, installation, etc.)**

1. Waterfall Model

Advantages

1. **Simplicity**. It is conceptually straightforward and divides the large task of building a software system into a series of cleanly divided phases, each phase dealing with a separate logical concern.

2. It is also **easy to administer** in a contractual setup - as each phase is completed and its work product produced, some amount of money is given by the customer to the developing organization.

1. Waterfall Model

Disadvantages

1. **The requirements of a system can be frozen before the design begins**.
    - Determining the requirements is difficult as the user does not even know the requirements.

2. Freezing the requirements usually **requires choosing the hardware** (a part of the requirements specification).
    - If the hardware is selected early, the final software will use a hardware technology on the limit of becoming outdated.

1. Waterfall Model

Disadvantages

3. It follows the **“big bang”** approach - the entire software is delivered in one shot at the end.
    - This imposes heavy risks, as the user does not know until the very end what they are getting.

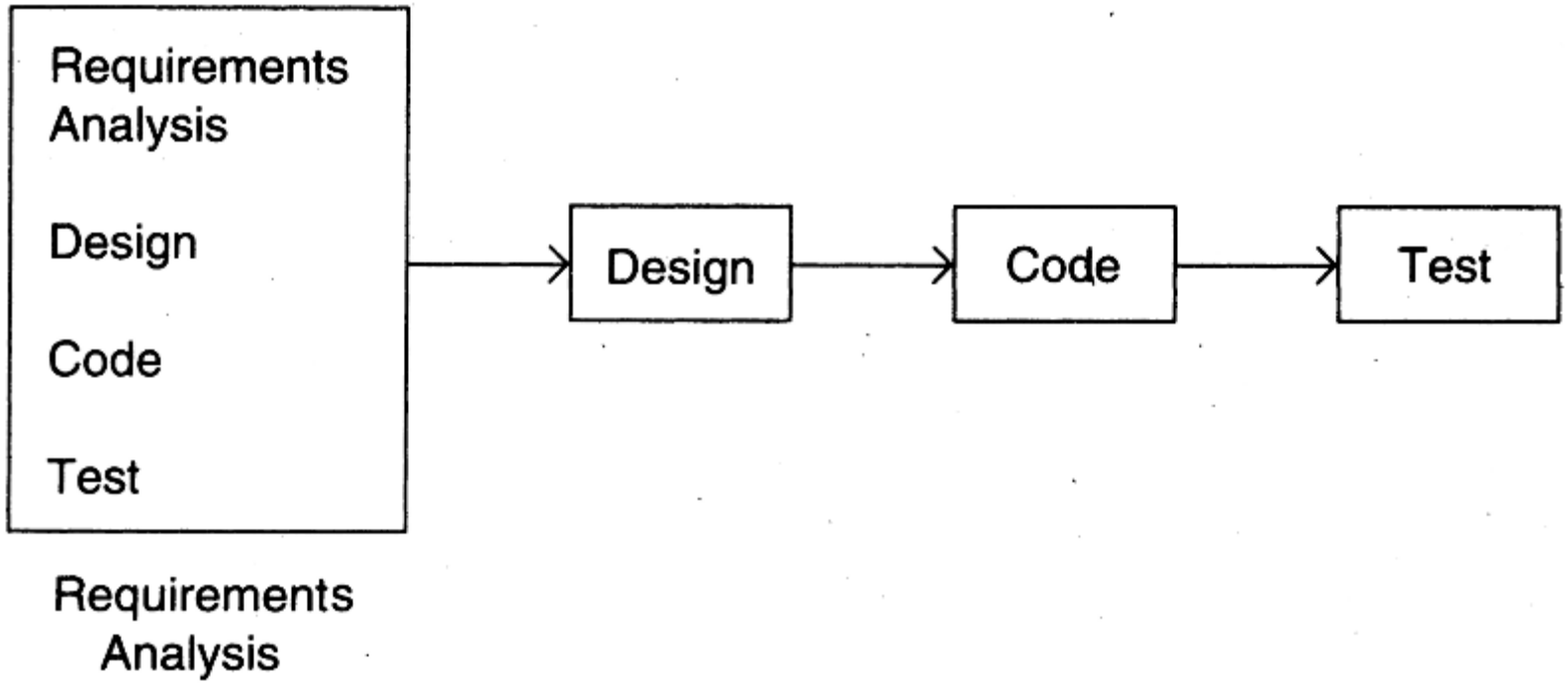
4. It encourages “**requirements bloating**”. - all requirements must be specified at the start and only what is specified will be delivered.

5. It is a **document-driven** process that requires formal documents at the end of each phase.

2. Prototyping Model

- The goal of a prototyping-based development process is to counter the first limitation of the waterfall model.
- In this, instead of freezing the requirements before design or coding, a **throwaway prototype** is built to help understand the requirements.
- This **prototype** is developed based on the currently known requirements.
- Development of the **prototype** undergoes **design**, **coding**, and **testing** phases.
- By using this prototype, the client can better understand the requirements of the desired system.
- This results in more stable requirements that change less frequently.

2. Prototyping Model



2. Prototyping Model

- The development of the prototype starts when the preliminary version of the requirements specification document has been developed.
- After the prototype has been developed, the end users and clients are given an opportunity to use the prototype.
- They provide feedback to the developers regarding the prototype: what is correct, what needs to be modified, what is missing, what is not needed, etc.
- Based on the feedback, the prototype is modified, and the users and the clients are again allowed to use the system.
- This cycle repeats.

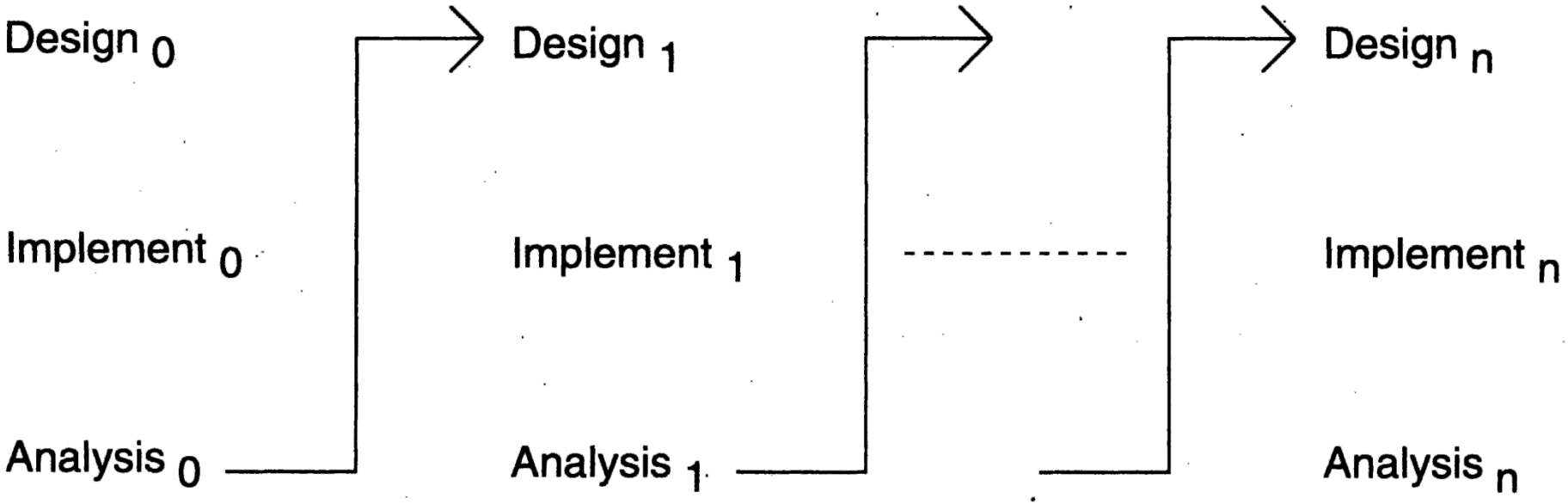
3. Iterative Model

- The iterative development process model tries to **combine the benefits of** both **prototyping** and the **waterfall** models.
- In this, the software is developed in **increments**, each increment adding some **functional capability** to the system until the full system is implemented.
- In the first step, a simple initial implementation is done for a subset of the overall problem.
- This subset contains some of the key aspects of the problem that are easy to understand and implement.

3. Iterative Model

- A **project control list** is created, that contains all the **tasks** that must be performed to obtain the final implementation.
- Each step consists of removing the next task from the list, **designing** the implementation for the selected task, **coding** and **testing** the implementation, performing an **analysis** of the partial system obtained after this step, and **updating** the list as a result of the analysis.
- These three phases are called the **design phase**, **implementation phase, and analysis phase.**
- The process is iterated until the project control list is empty, at which time the final implementation of the system will be available.

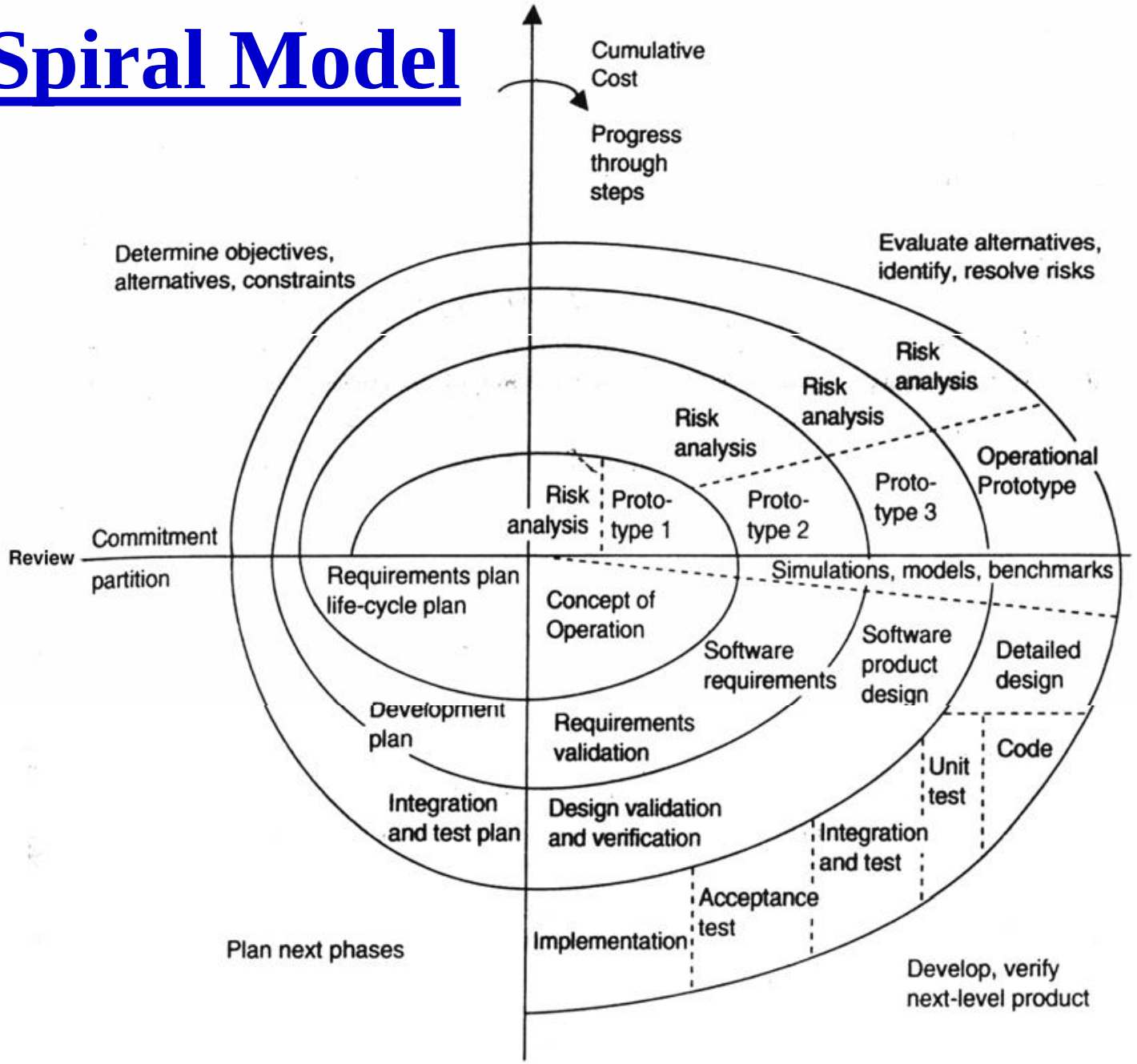
3. Iterative Model



4. Spiral Model

- The activities in this model can be organized like a **spiral** that has **many cycles**.
- The **radial dimension** represents the **cumulative cost** incurred in accomplishing the steps done so far.
- The **angular dimension** represents the **progress** made in completing each cycle of the spiral.
- Each cycle in the spiral is split into four sectors:
  - **Objective setting**
  - **Risk Assessment and reduction**
  - **Development and validation**
  - **Planning**

4. Spiral Model



4. Spiral Model

Objective setting: Each cycle in the spiral begins with

- the identification of objectives for that cycle,
- the different alternatives that are possible for achieving the objectives, and
- the constraints that exist.

Risk Assessment and Reduction: The next step is to evaluate these different alternatives based on the objectives and constraints.

The focus of evaluation is based on the risk awareness for the project.

Risks reflect the chances that some of the objectives of the project may not be met.

4. Spiral Model

Development and Validation:

The next step is to develop strategies that resolve the uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

Planning:

Next, the software is developed, keeping in mind the risks.

Finally, the next stage is planned.

The project is reviewed and a decision made whether to continue with a further cycle of the spiral.

If it is decided to continue, plans are drawn up for the next phase of the project

5. Agile Process

Agile development approaches evolved as a reaction to documentation based processes, particularly the waterfall approach.

Agile approaches are based on some common principles,

- Working software is the key measure of progress in a project.
- For progress in a project, software should be developed and delivered rapidly in small increments.
- Even late changes in the requirements should be entertained.

5. Agile Process

- Face-to-face communication is preferred over documentation.
- Continuous feedback and involvement of customer is necessary for developing good-quality software.
- Simple design which evolves and improves with time is a better approach than doing an elaborate design for handling all possible scenarios.
- The release dates are decided by teams of talented individuals.