LIFE CYCLE MODELS

Life Cycle Model (CONT.)

- Many life cycle models have been proposed.
- We confine our attention to only a few commonly used models.

Traditional models:

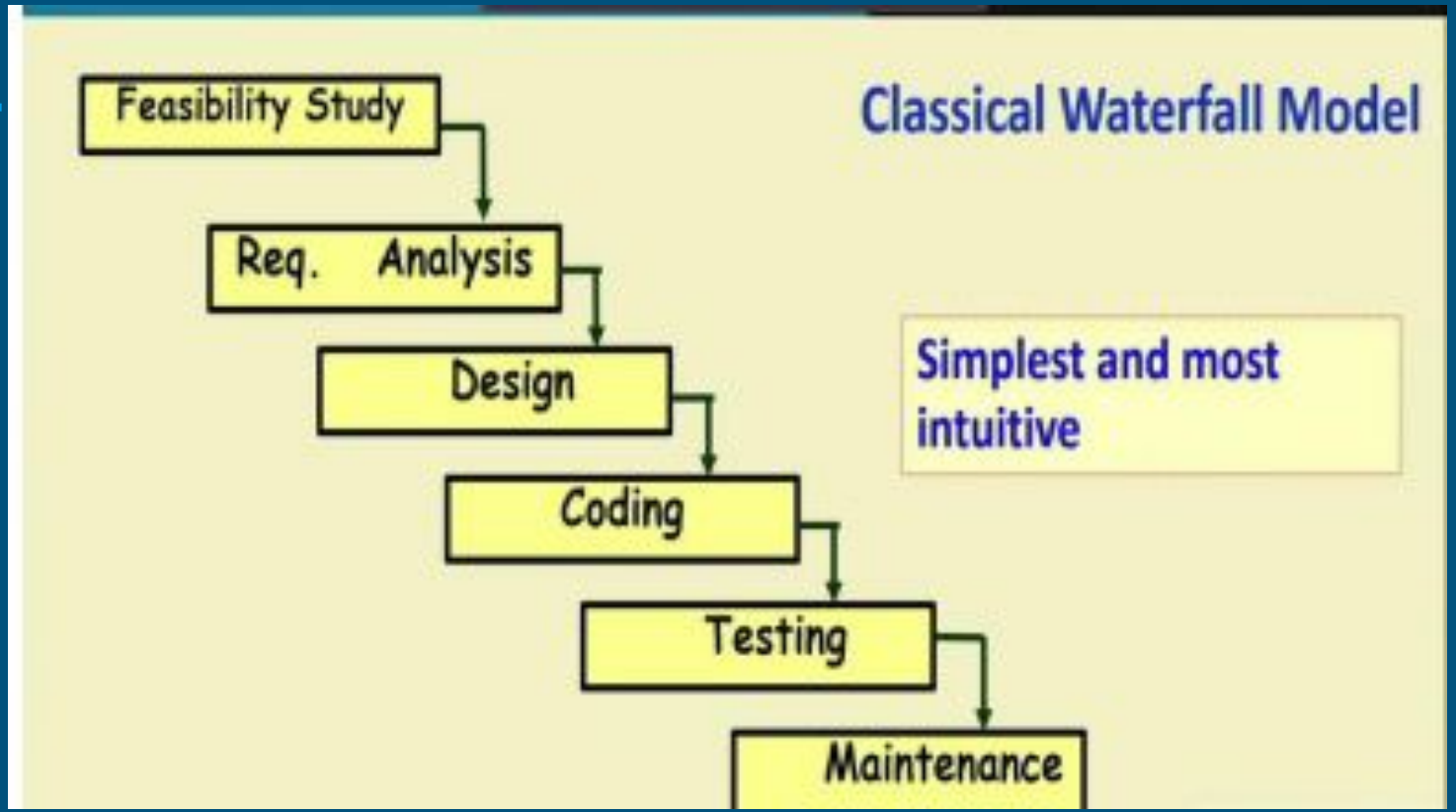
- Waterfall
- V model,
- Evolutionary,
- Prototyping
- Spiral model,
- Agile models

- Software life cycle (or software process):
  - Series of identifiable stages that a software product undergoes during its life time:
    - Feasibility study
    - Requirements analysis and specification,
    - Design,
    - Coding,
    - Testing
    - Maintenance.

Software Life Cycle

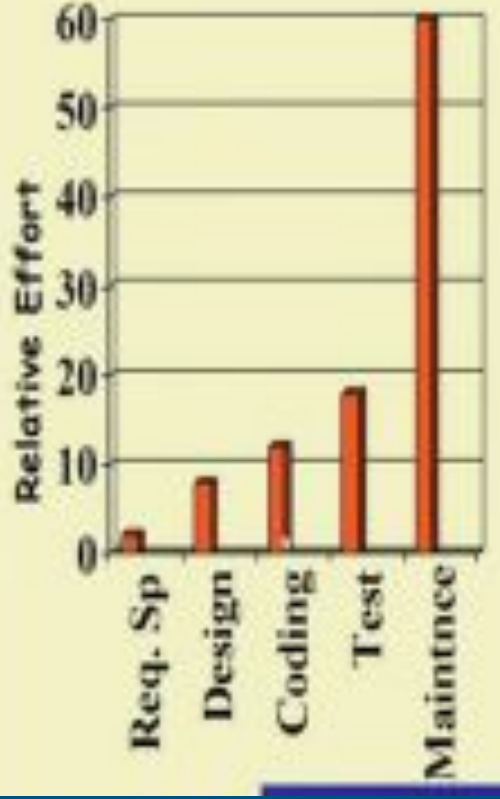
Classical Waterfall Model

- Classical waterfall model divides life cycle into following phases:
  - Feasibility study,
  - Requirements analysis and specification,
  - Design,
  - Coding and unit testing,
  - Integration and system testing,
  - Maintenance.


Feasibility Study
Req. Analysis
Design
Coding
Testing
Maintenance
Classical Waterfall Model
Simplest and most intuitive

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.

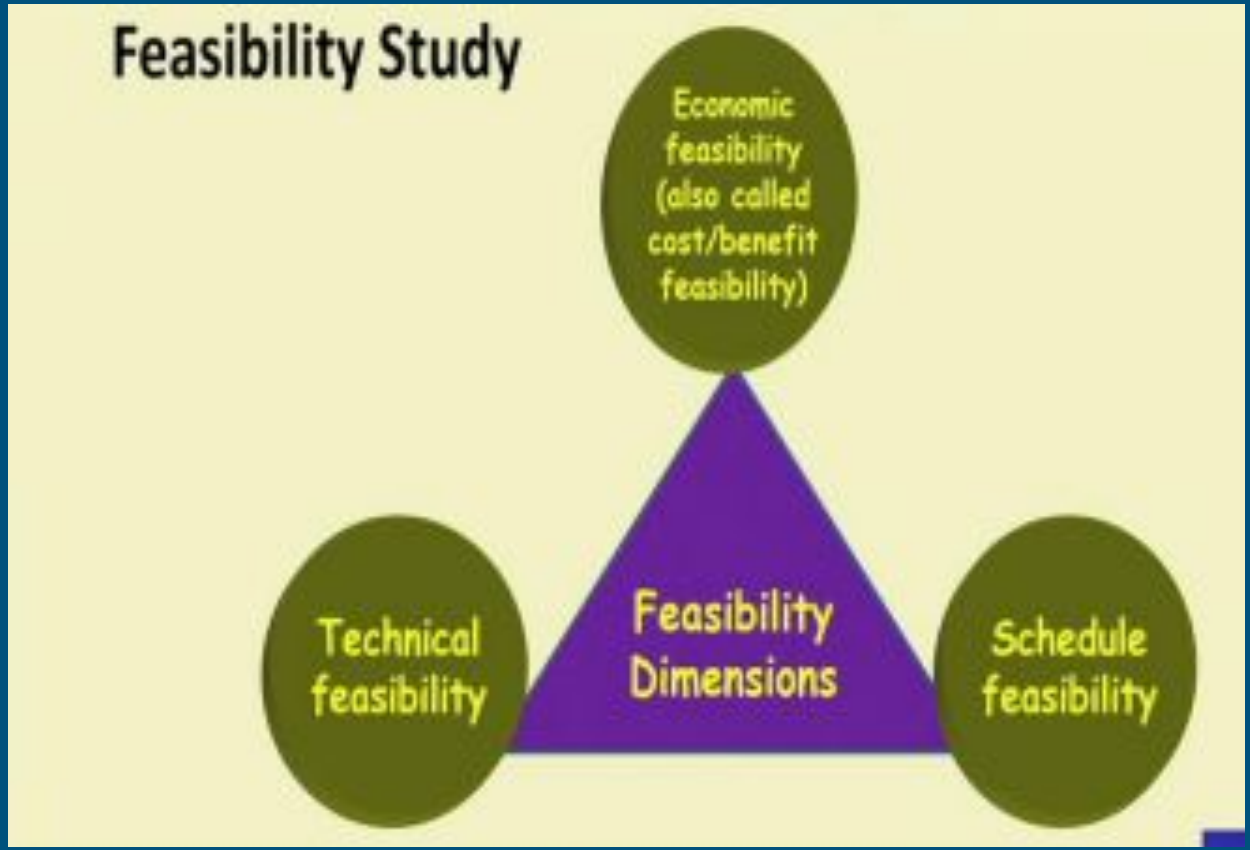


- Most organizations usually define:
  - Standards on the outputs (deliverables) produced at the end of every phase
  - Entry and exit criteria for every phase.
- They also prescribe methodologies for:
  - Specification,
  - Design,
  - Testing,
  - Project management, etc.

Process Model

Classical Waterfall Model (CONT.)

- The guidelines and methodologies of an organization:
  - –Called the organization's software development methodology.
- Software development organizations:
  - – Expect fresh engineers to master the organization's software development methodology.


Feasibility Study
Economic feasibility (also called cost/benefit feasibility)
Technical feasibility
Feasibility Dimensions
Schedule feasibility

- Main aim of feasibility study: determine whether developing the software is:
  - Financially worthwhile
  - Technically feasible.

Feasibility Study

- Roughly understand what customer wants:
  - Data which would be input to the system,
  - Processing needed on these data,
  - Output data to be produced by the system,
  - Various constraints on the behavior of the system.

First Step

Requirements Analysis and Specification

- Aim of this phase:
  - –Understand the exact requirements of the customer,
  - –Document them properly.
- Consists of two distinct activities:
  - –Requirements gathering and analysis
  - –Requirements specification.

Requirements Analysis and Specification

- Gather requirements data from the customer:
  - Analyze the collected data to understand what customer wants
- Remove requirements problems:
  - Inconsistencies
  - Anomalies
  - Incompleteness
- Organize into a Software Requirements Specification (SRS) document.

Requirements Gathering

- Gathering relevant data:
  - –Usually collected from the end-users through interviews and discussions.
  - –Example: for a business accounting software:
    - Interview all the accountants of the organization to find out their requirements.

Requirements Analysis (Cont..)

- The data you initially collect from the users:
  - –Usually contain several contradictions and ambiguities.
  - –Why?
  - **–Each user typically has only a partial and incomplete view of the system.**

Requirements Analysis (Cont...)

- Ambiguities and contradictions:
  - –must be identified
  - –resolved by discussions with the customers.
- Next, requirements are organized:
  - –into a Software Requirements Specification (SRS) document.

Design

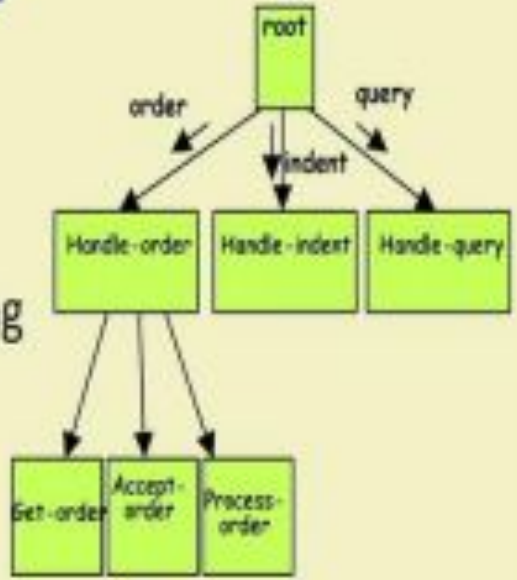
- During design phase requirements specification is transformed into :
  - A form suitable for implementation in some programming language.
- Two commonly used design approaches:
  - Traditional approach,
  - Object oriented approach

Traditional Design Approach

- Consists of two activities:
  - –Structured analysis (typically carried out by using DFD)
  - –Structured design

Structured Design

- High-level design:
  - –decompose the system into modules,
  - –represent invocation relationships among the modules.
- Detailed design:
  - –different modules designed in greater detail:
    - data structures and algorithms for each module are 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.

Object-Oriented Design

Object Oriented Design (CONT.)

- Object structure:
  - Refined to obtain the detailed design.
- OOD has several advantages:
  - Lower development effort,
  - Lower development time,
  - Better maintainability.

Coding and Unit Testing

- During this phase:
  - –Each module of the design is coded,
  - –Each module is unit tested
    - That is, tested independently as a stand alone unit, and debugged.
  - –Each module is documented.

Integration and System Testing

- Different modules are integrated in a planned manner:
  - –Modules are usually integrated through a number of steps.
- During each integration step,
  - –the partially integrated system is tested.

System Testing

- After all the modules have been successfully integrated and tested:
  - System testing is carried out.
- Goal of system testing:
  - Ensure that the developed system functions according to its requirements as specified in the SRS document.

Maintenance

- Maintenance of any software:
  - –Requires much more effort than the effort to develop the product itself.
  - –Development effort to maintenance effort is typically 40:60.

Types of Maintenance?

- **Corrective maintenance:**
  - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance:**
  - Improve implementation of the system
  - enhance functionalities of the system.
- **Adaptive maintenance:**
  - Port software to a new environment,
    - e.g. to a new computer or to a new operating system.

Iterative Waterfall Model

- Classical waterfall model is idealistic:
  - –Assumes that no defect is introduced during any development activity.
  - –In practice:
    - Defects do get introduced in almost every phase of the life cycle.

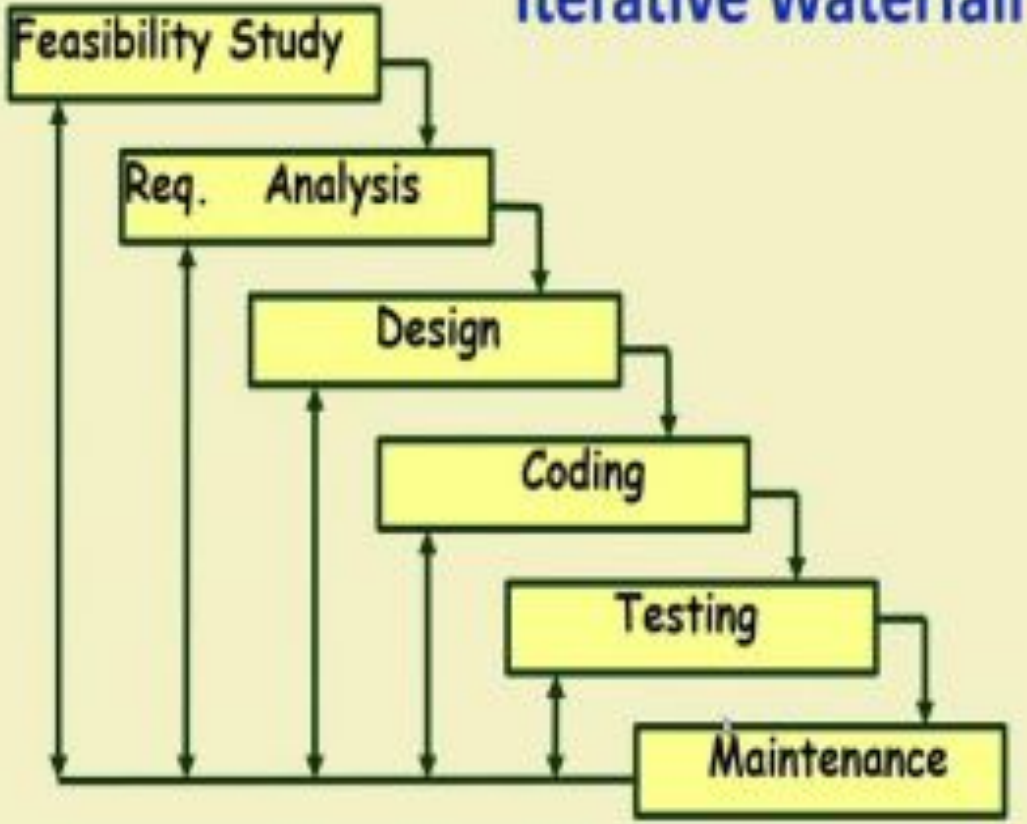
Iterative Waterfall Model (CONT.)

- Defects usually get detected much later in the life cycle:
  - –For example, a design defect might go unnoticed till the coding or testing phase.
  - **–The later the phase in which the defect gets detected, the more expensive is its removal --- why?**

Iterative Waterfall Model (CONT.)

- Once a defect is detected:
  - –The phase in which it occurred needs to be reworked.
  - – Redo some of the work done during that and all subsequent phases.
- Therefore need feedback paths in the classical waterfall model.

Iterative Waterfall Model (CONT.)



Phase Containment of Errors (Cont...)

- **Errors should be detected:**
  - **In the same phase in which they are introduced.**
- For example:
  - If a design problem is detected in the design phase itself,
    - The problem can be taken care of much more easily
    - Than say if it is identified at the end of the integration and system testing phase.

Phase Containment of Errors

- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible:
  - is known as **phase containment of errors.**
- Iterative waterfall model is by far the most widely used model.
  - Almost every other model is derived from the waterfall model.

Waterfall Strengths

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

Waterfall Deficiencies

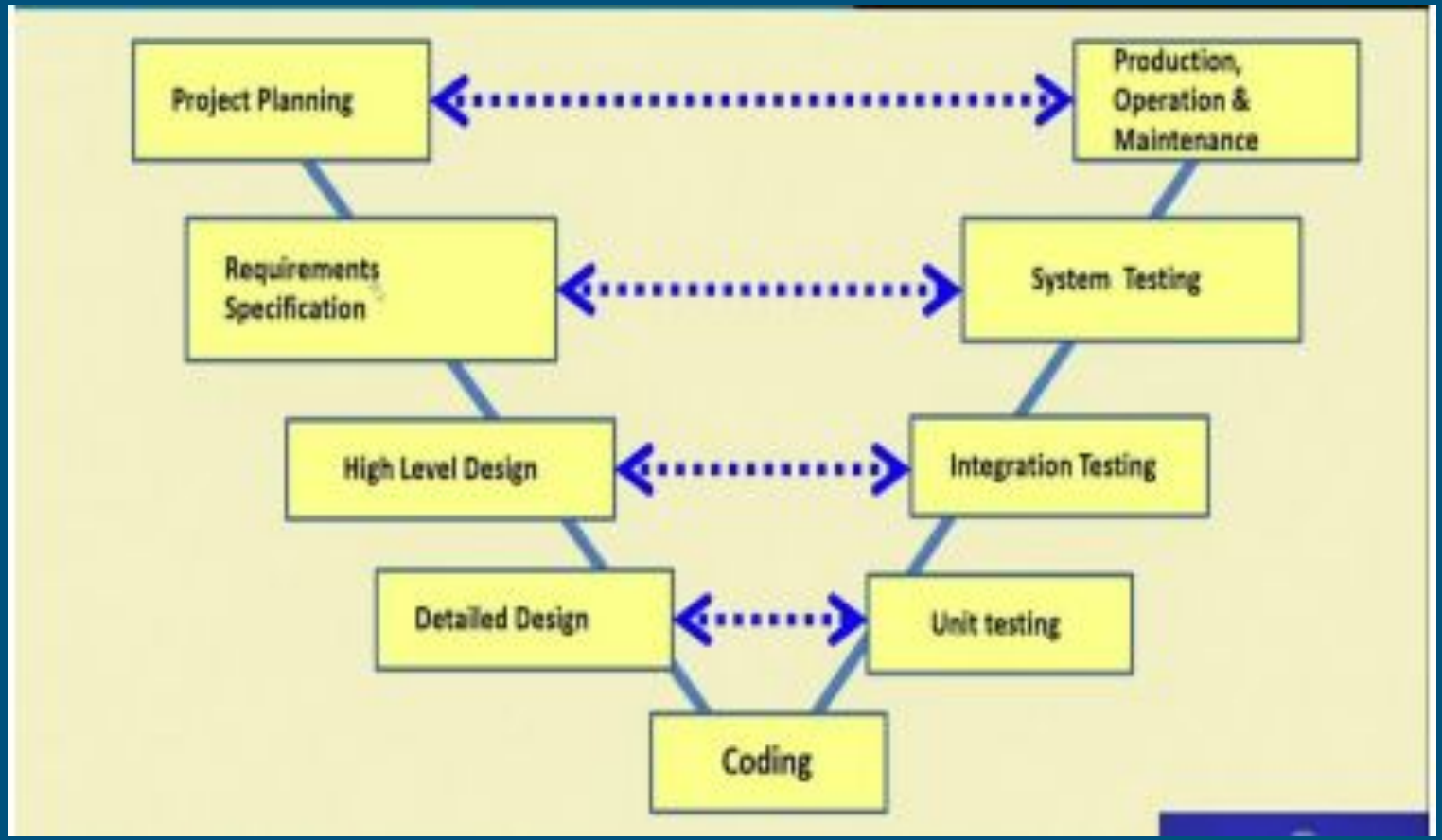
- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – **inhibits flexibility**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Experienced Development team

V Model

- It is a variant of the Waterfall
  - emphasizes verification and validation
  - V&V activities are spread over the entire life cycle.
- In every phase of development:
  - Testing activities are planned in parallel with development.


Project Planning
Production, Operation & Maintenance
Requirements Specification
System Testing
High Level Design
Integration Testing
Detailed Design
Unit testing
Coding

V Model Steps

- Planning
- Requirements Analysis and Specification
- High-level Design
- Detailed Design
- System test design
- Integration Test dsign
- Unit test design

V Model: Strengths

- Starting from early stages of software development:
  - **Emphasizes planning for verification and validation of the software**
- Each deliverable is made testable
- Easy to use

V Model Weaknesses

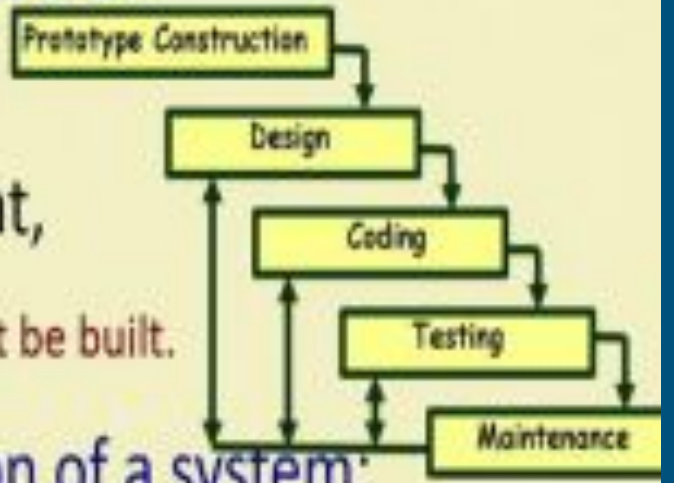
- Does not support overlapping of phases
- Does not handle iterations or phases
- Does not easily accommodate later changes to requirements
- Does not provide support for effective risk handling

When to use V Model

- Natural choice for systems requiring high reliability:
  - Embedded control applications, safety-critical software
- All requirements are known up-front
- Solution and technology are known

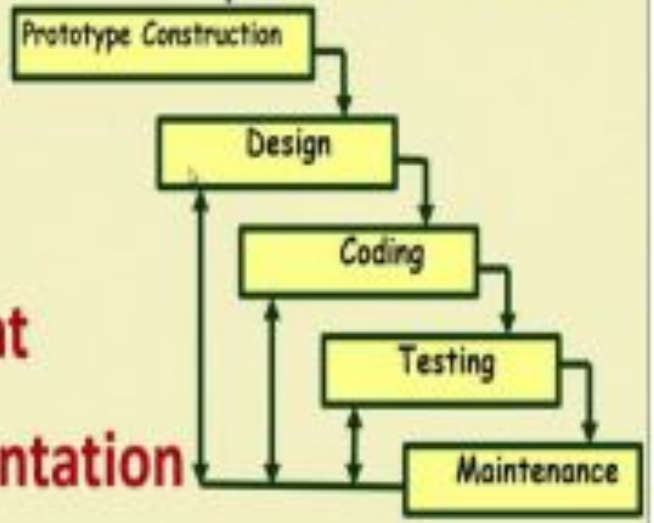
Prototyping Model

- A derivative of waterfall model.
- Before starting actual development,
  - A working prototype of the system should first be built.
- A prototype is a toy implementation of a system:
  - Limited functional capabilities,
  - Low reliability,
  - Inefficient performance.



Reasons for prototyping

- **Learning by doing:** useful where requirements are only partially known
- **Improved communication**
- **Improved user involvement**
- **Reduced need for documentation**
- **Reduced maintenance costs**



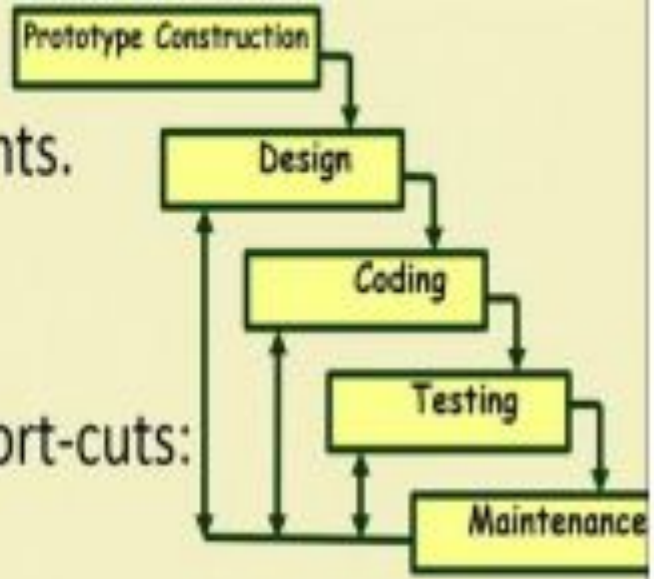
Reasons for Developing a Prototype

- **Illustrate to the customer:**
  - –input data formats, messages, reports, or interactive dialogs.
- **Examine technical issues associated with product development:**
  - –Often major design decisions depend on issues like:
    - Response time of a hardware controller,

Prototyping Model (CONT.)

- Another reason for developing a prototype:
  - **It is impossible to "get it right" the first time,**
  - We must plan to throw away the first version:
    - If we want to develop a good software.

Prototyping Model



- Start with approximate requirements.
- Carry out a quick design.
- Prototype is built using several short-cuts:
  - –Short-cuts might involve:
    - Using inefficient, inaccurate, or dummy functions.
    - A table look-up rather than performing the actual computations.

Prototyping Model (CONT.)

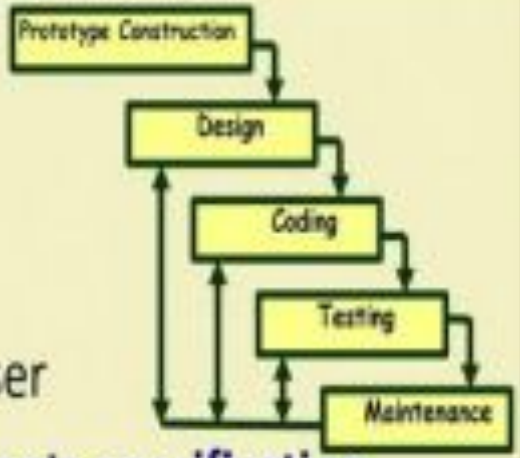
Prototyping Model (CONT.)

- The developed prototype is submitted to the customer for his evaluation:

  - –Based on the user feedback, the prototype is refined.
  - –This cycle continues until the user approves the prototype.

- The actual system is developed using the waterfall model

Prototyping Model



- Requirements analysis and specification phase becomes redundant:
  - Final working prototype (incorporating all user feedbacks) serves as an **animated requirements specification.**
- **Design and code for the prototype is usually thrown away**:
  - However, experience gathered from developing the prototype helps a great deal while developing the actual software.

Prototyping Model (CONT.)

- Even though construction of a working prototype model involves additional cost --- **overall development cost usually lower for:**
  - Systems with unclear user requirements,
  - Systems with unresolved technical issues.
- Many user requirements get properly defined and technical issues get resolved:
  - These would have appeared later as change requests and resulted in incurring massive redesign costs.

Prototyping: advantages

- The resulting software is usually more usable
- User needs are better accommodated
- The design is of higher quality
- The resulting software is easier to maintain
- Overall, the development incurs less cost

Prototyping: disadvantages

- For some projects, it is expensive
- Susceptible to over-engineering:
  - Designers start to incorporate sophistications that they could not incorporate in the prototype.

Major difficulties of Waterfall-Based Models

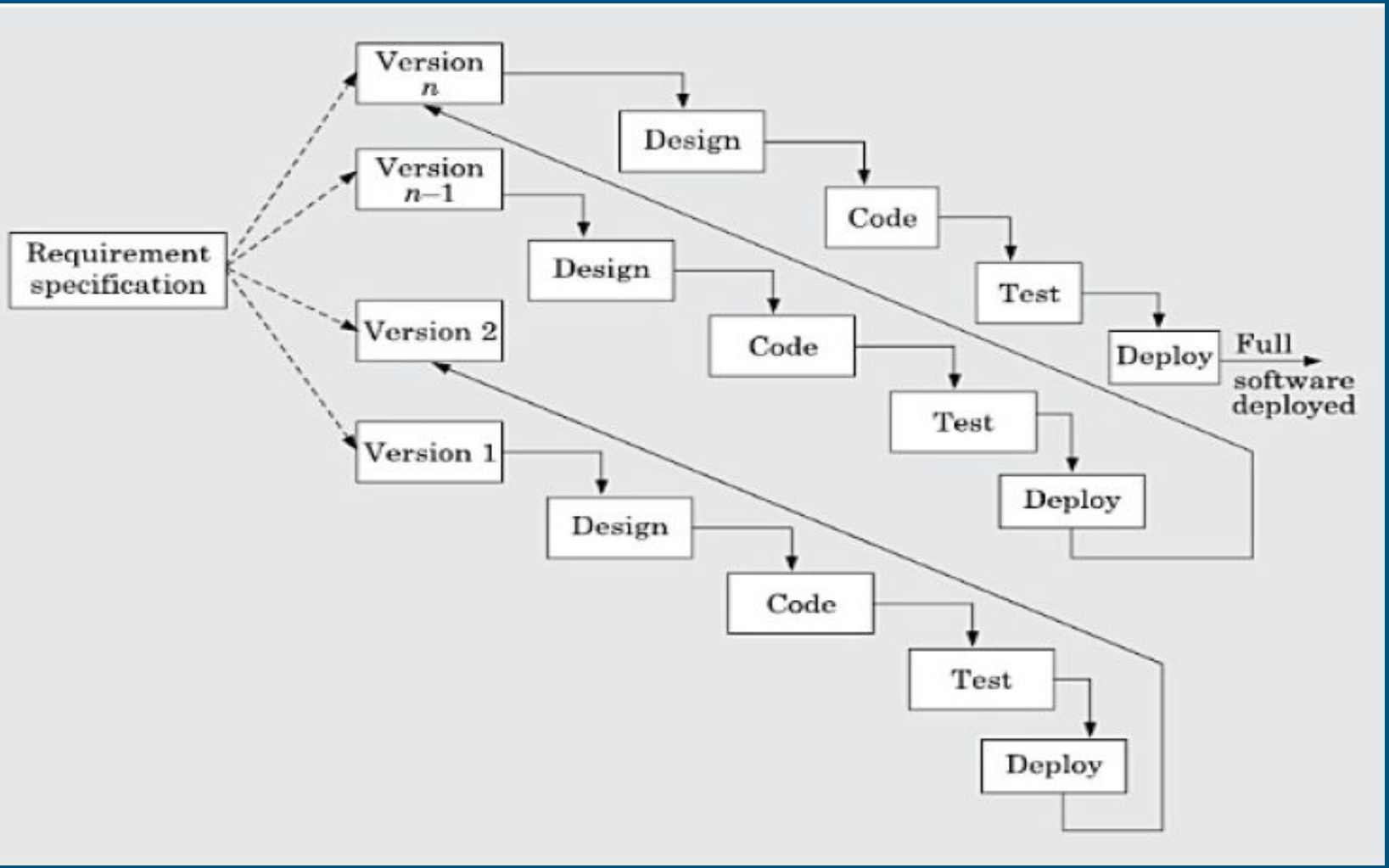
1. Difficulty in accommodating change requests during development.
   - 40% of the requirements change during development
2. High cost incurred in developing custom applications.
3. "Heavy weight processes."

Major difficulties of Waterfall-Based Life Cycle Models

- Requirements for the system are determined at the start:
  - Are assumed to be fixed from that point on.
  - Long term planning is made based on this.

Incremental Model

- Waterfall: single release
- Iterative: many releases (increments)
  - –First increment: core functionality
  - –Successive increments: add/fix functionality
  - –Final increment: the complete product
- Each iteration: a short mini-project with a separate lifecycle
  - –e.g., waterfall


Requirement specification
Version n
Version n–1
Version 2
Version 1
Design
Code
Test
Deploy
Full software deployed

Advantages

The incremental development model offers several advantages. Two important ones are the following:

- **Error reduction:** The core modules are used by the customer from the beginning and therefore these get tested thoroughly. This reduces chances of errors in the core modules of the final product, leading to greater reliability of the software.
- **Incremental resource deployment:** This model obviates the need for the customer to commit large resources at one go for development of the system. It also saves the developing organisation from deploying large resources and manpower for a project in one go.

Evolutionary Model

- First develop the core modules of the software.
- The initial skeletal software is refined into increasing levels of capability: (Iterations)

  –By adding new functionalities in successive versions.

Evolutionary Model

- First develop the core modules of the software.
- The initial skeletal software is refined into increasing levels of capability: (Iterations)

  –By adding new functionalities in successive versions.

Activities in an Iteration

- Software developed over several "mini waterfalls".
- The result of a single iteration:
  - –Ends with delivery of some tangible code
  - –An incremental improvement to the software --- leads to evolutionary development

Evolutionary Model with Iteration

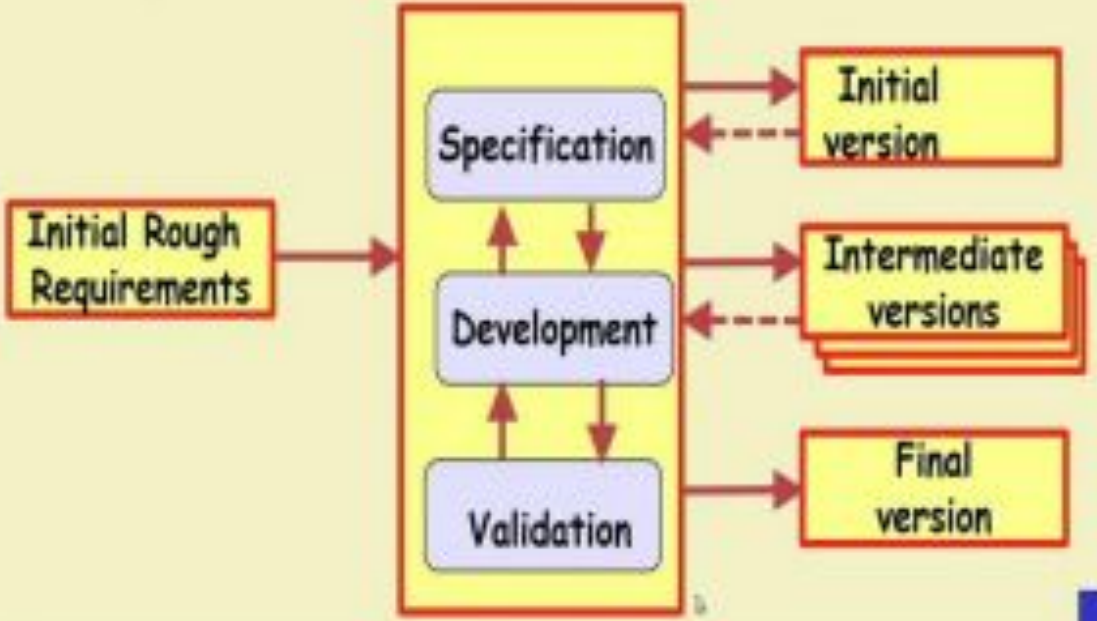
- Outcome of each iteration: tested, integrated, executable system
- Iteration length is short and fixed
  - Usually between 2 and 6 weeks
  - Development takes many iterations (for example: 10-15)
- Does not "freeze" requirements and then conservatively design :
  - Opportunity exists to modify requirements as well as the design...

Evolutionary Model (CONT.)

- Successive versions:
  - –Functioning systems capable of performing some useful work.
  - –A new release may include new functionality:
    - Also existing functionality in the current release might have been enhanced.

Evolutionary Model

- Evolves an initial implementation with user feedback:
  - **Multiple versions until the final version.**



Advantages of Evolutionary Model

- Users get a chance to experiment with a partially developed system:
  - Much before the full working version is released,
- **Helps finding exact user requirements:**
  - Software more likely to meet exact user requirements.
- **Core modules get tested thoroughly:**
  - Reduces chances of errors in final delivered software.

Advantages of evolutionary model

- Better management of complexity by developing one increment at a time.
- Better management of changing requirements.
- Can get customer feedback and incorporate them much more efficiently:
  - As compared when customer feedbacks come only after the development work is complete.

Evolutionary Model: Problems

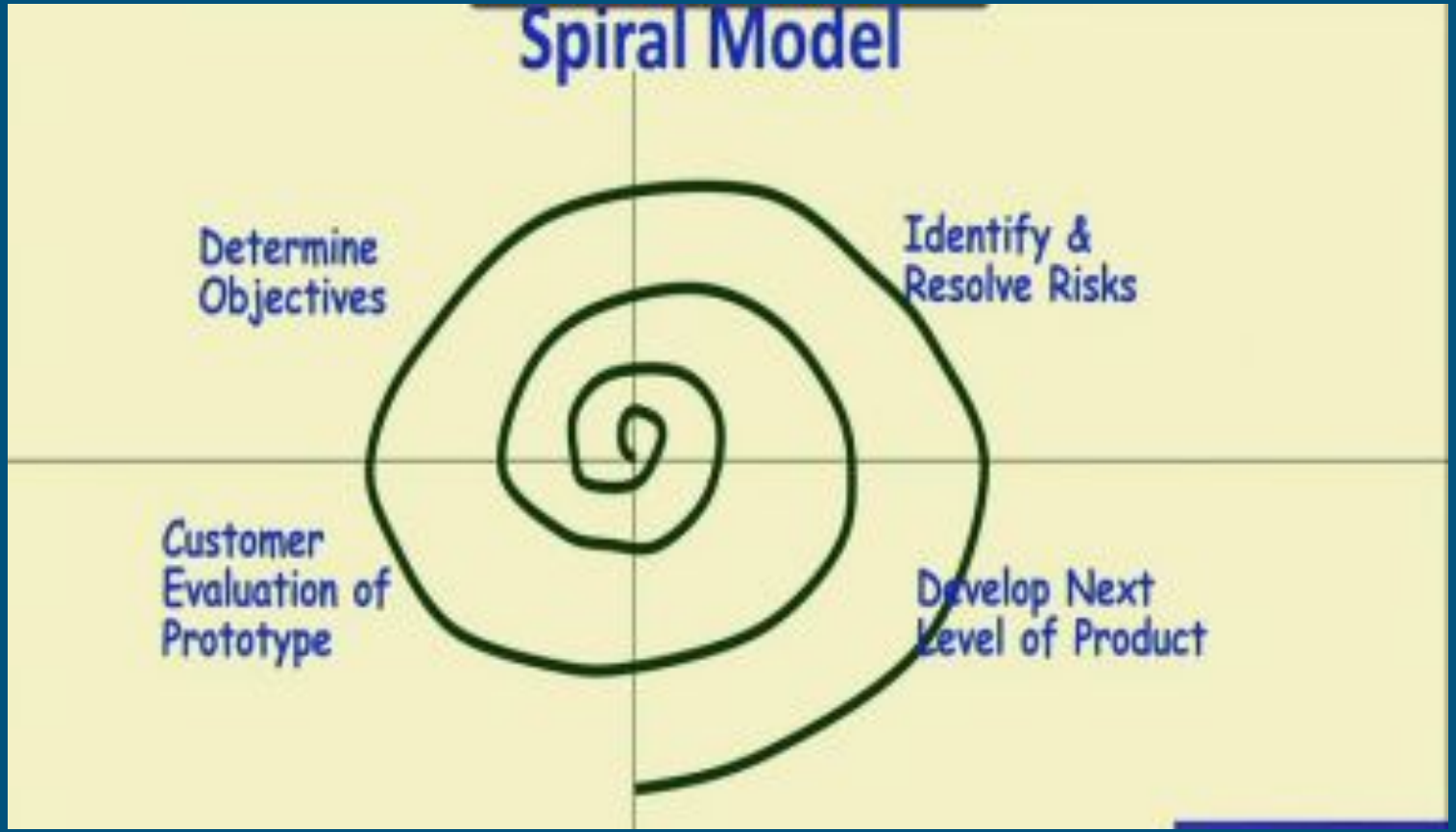
- **The process is intangible:**
  - No regular, well-defined deliverables.
- **The process is unpredictable:**
  - Hard to manage, e.g., scheduling, workforce allocation, etc.
- **Systems are rather poorly structured:**
  - Continual, unpredictable changes tend to degrade the software structure.
- **Systems may not even converge to a final version.**

Spiral Model

- Proposed by Boehm in 1988.
- Each loop of the spiral represents a phase of the software process:
  - the innermost loop might be concerned with system feasibility,
  - the next loop with system requirements definition,
  - the next one with system design, and so on.
- There are no fixed phases in this model, the phases shown in the figure are just examples.

Spiral Model (CONT.)

- The team must decide:
  - how to structure the project into phases.
- Start work using some generic model:
  - add extra phases
    - for specific projects or when problems are identified during a project.
- Each loop in the spiral is split into four sectors (quadrants).


Spiral Model
Determine Objectives
Identify & Resolve Risks
Customer Evaluation of Prototype
Develop Next Level of Product

Objective Setting (First Quadrant)

- Identify objectives of the phase,
- Examine the risks associated with these objectives.
  - Risk:
    - Any adverse circumstance that might hamper successful completion of a software project.
- Find alternate solutions possible.

Risk Assessment and Reduction (Second Quadrant)

- For each identified project risk,
  - –a detailed analysis is carried out.
- Steps are taken to reduce the risk.
- For example, if there is a risk that requirements are inappropriate:
  - –A prototype system may be developed.

Spiral Model (CONT.)

- Development and Validation (Third quadrant):
  - develop and validate the next level of the product.
- Review and Planning (Fourth quadrant):
  - review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral:
  - progressively more complete version of the software gets built.

Spiral Model as a Meta Model

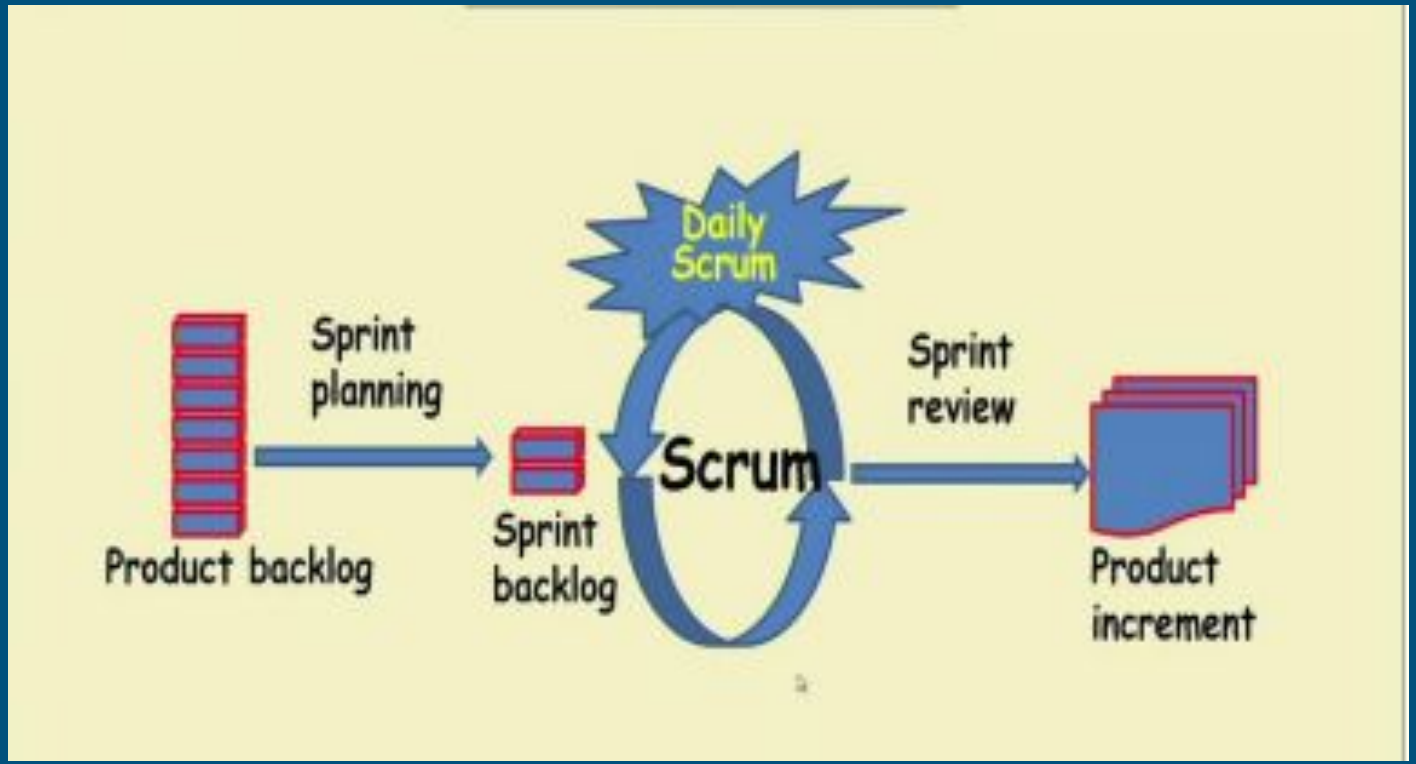
- Subsumes all discussed models:
  - a single loop spiral represents waterfall model.
  - uses an evolutionary approach --
    - iterations over the spiral are evolutionary levels.
  - enables understanding and reacting to risks during each iteration along the spiral.
  - Uses:
    - prototyping as a risk reduction mechanism
    - retains the step-wise approach of the waterfall model.

Agile Methodologies

- XP
- Scrum
- Unified process
- Crystal
- DSDM
- Lean

Scrum: Characteristics

- Self-organizing teams
- Product progresses in a series of month-long **sprints**
- Requirements are captured as items in a list of **product backlog**
- One of the agile processes


Daily
Scrum
Sprint
planning
Sprint
review
Scrum
Product backlog
Sprint
backlog
Product
increment

Sprint

- Scrum projects progress in a series of "sprints"
  - Analogous to XP iterations or time boxes
  - Target duration is one month
- **Software increment is designed, coded, and tested during the sprint**
- **No changes entertained during a sprint**

Key Roles and Responsibilities in Scrum Process

- Product Owner
  - Acts on behalf of customers to represent their interests.
- Development Team
  - Team of five-nine people with cross-functional skill sets.
- Scrum Master (aka Project Manager)
  - Facilitates scrum process and resolves impediments at the team and organization level by acting as a buffer between the team and outside interference.

Product Owner

- Defines the features of the product
- Decide on release date and content
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results.

The Scrum Master

- Represents management to the project
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

Scrum Team

- Typically 5-10 people
- Cross-functional
  - QA, Programmers, UI Designers, etc.
- Teams are self-organizing
- Membership can change only between sprints