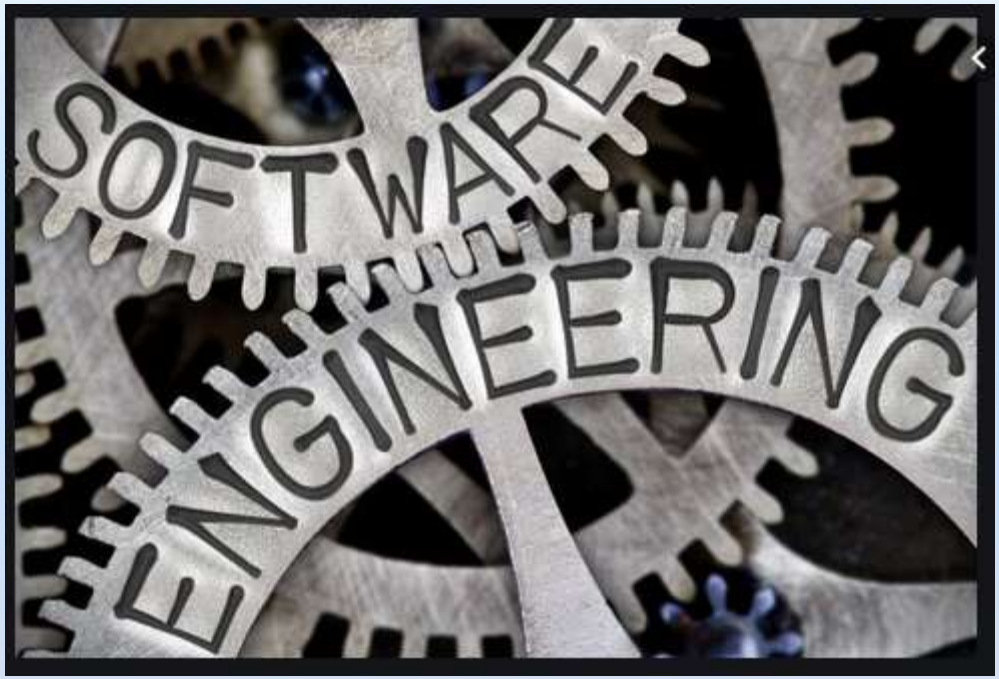
Software Engineering
Module-1



Faculty :

Dr. Pulak Sahoo

Associate Professor

Silicon Institute Of Technology

Text Books

(1) **Roger S. Pressman**, ***Software Engineering, A Practitioner's Approach***, Mc Graw Hill, 7th Edition, 2010

(2) **I Summerville**, ***Software Engineering,*** Pearson Education, 9th Edition, 2013

Reference Books

(1) **RAJIB MALL**, ***Fundamentals of Software Engineering***, PHI Learning, 4th Edition, 2014

Module-1 Contents

- **Software Engineering:**
  - Introduction & Evolving role of software, Process framework, **CMM**
- **Software Life Cycle Models:**
  - **Waterfall model, Iterative Waterfall model**
  - **V-Process model**
  - **Incremental Process model**
  - **Evolutionary Process models**
    - **Prototyping & Spiral model**
  - **6. Agile & RAD models**
    - **Extreme Programming, Scrum, Crystal models,**
  - **Unified Process**

Introduction to S/W Engineering

- What is **Software Engineering**?
- Problem complexity reduction using:
  - **Abstraction** & **Decomposition**
- **S/W Crisis**
- Diff between **S/W Programs** vs. **S/W Products**
- Diff between **Systems Engg** vs **S/W Engg**
- **Evolution** of **Software Engineering**
- Introduction to **Life Cycle Models**

What is S/W Engg.

- What is **Engineering**?

- What is **Software Engineering**?

- Is writing program straightaway in computer is S/W Engg ?

Introduction to S/W Engineering

- What is **Software Engineering**?
- Problem complexity reduction using:
  - **Abstraction** & **Decomposition**
- **S/W Crisis**
- Diff between **S/W Programs** vs. **S/W Products**
- Diff between **Systems Engg** vs **S/W Engg**
- **Evolution** of **Software Engineering**
- Introduction to **Life Cycle Models**

SDLC Models

- 1. Waterfall model
- 2. Iterative Waterfall model
- 3. V-Process model
- 4. Incremental Process model
- 5. Evolutionary Process models
  - 5.1. Prototyping model
  - 5.2. Spiral model
- 6. Agile & RAD models
  - 6.1. Extreme Programming
  - 6.2. Scrum
  - 6.3. Crystal
- Unified Process

S/W Engineering

- **S/W Engineering**
  - Is **Methodical approach** to **software development**
  - Makes use of **past experience**
  - Systematic use of **techniques**, **methodologies** & **guidelines**




- **Purpose**
  - To achieve **quality s/w** which is **cost effective**
- Two important techniques used to **reduce problem complexity**
  - **Abstraction** and **Decomposition**

S/W Development Myths

Myth-1:

- We already have a book that's full of standards & procedures for building software. Won't that provide my people with everything they need to know?

Myth:-2:

- If we get behind schedule, we can add more programmers & catch up (“Mongolian horde” concept).

Myth-3:

If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Legacy software - Maintainance

• **Legacy software systems** were developed decades ago & have been continually modified to meet changes in business requirements & computing platforms

• The **maintenance** of such systems is causing Difficulties for large organizations who find them costly to maintain & risky to evolve.

Abstraction

- **Abstraction** is a powerful way of reducing complexity of problem
- Simplify a problem by :
  - **Considering** relevant aspects & **suppressing** irrelevant aspects
    - Top managers are not interested in tech. details of each program
    - **Ex:** Inputs (loan amount, duration) & outputs (EMI) of "Loan EMI Calc" system
- The omitted details are considered in next level abstraction

Decomposition

- Another approach to tackle problem complexity
- A complex problem is divided into **smaller problems**
- The smaller problems are then **solved one by one**
- But, random decompositions does not reduce complexity
- Problem is decomposed such that each component can be **solved independently**
- Then component solutions are **combined** to get the full solution

Software Crisis

- Occurs when **S/w Products:**
  - **Fail** to meet **user requirements**
  - Frequently **crash** due to bad design
  - Difficult to **debug** & **alter**
  - **Delivered late**
  - **Over-budget**
  - Skill Shortage
  - Low productivity
  - **Lack of adequate training in s/w engg**

Programs Vs S/W Products

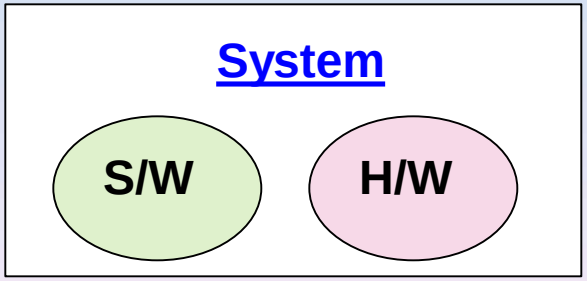
- Usually small in size
- Author may be the sole user
- Single developer
- Lacks proper user interface
- Lacks proper documentation
- Ad hoc development

- Large size
- Large number of users
- Team of developers
- Well-designed interface
- Well documented & user-manual prepared
- Systematic development

Computer Systems Engg.

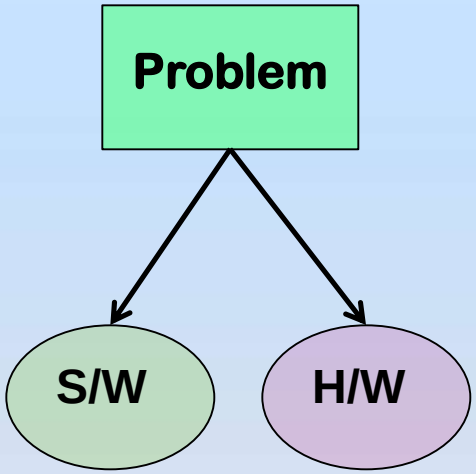
- Many **products** require development of **software** as well as **specific hardware** to run it
  - Robots
  - Coffee vending machine
  - Mobile equipments

- **Systems engineering** Contains
  - **Software engineering**

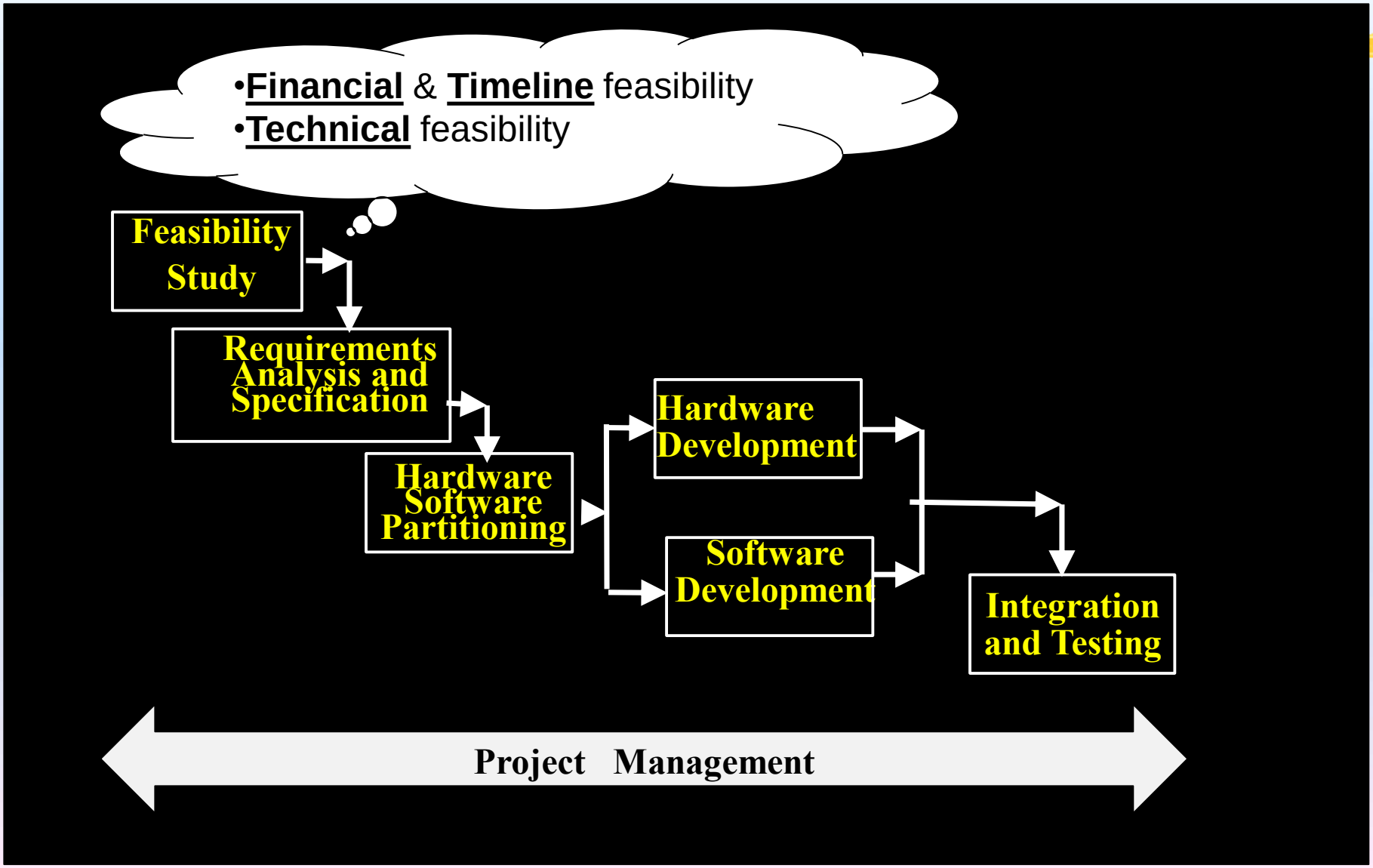


Computer Systems Engg

- Divide the high-level problem in terms of
  - Which tasks to be done by Software
  - Which tasks to be done by Hardware



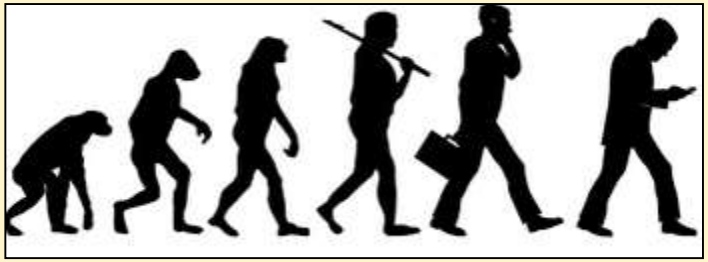
Computer Systems Engg (CONT.)



Computer Systems Engg.

- Sometimes **H/W** and **S/W** are developed together
  - Hardware is used during s/w development
    - **Ex**: *Electronic devices*
- Integration of **H/W** & **S/W** parts
- Final system testing

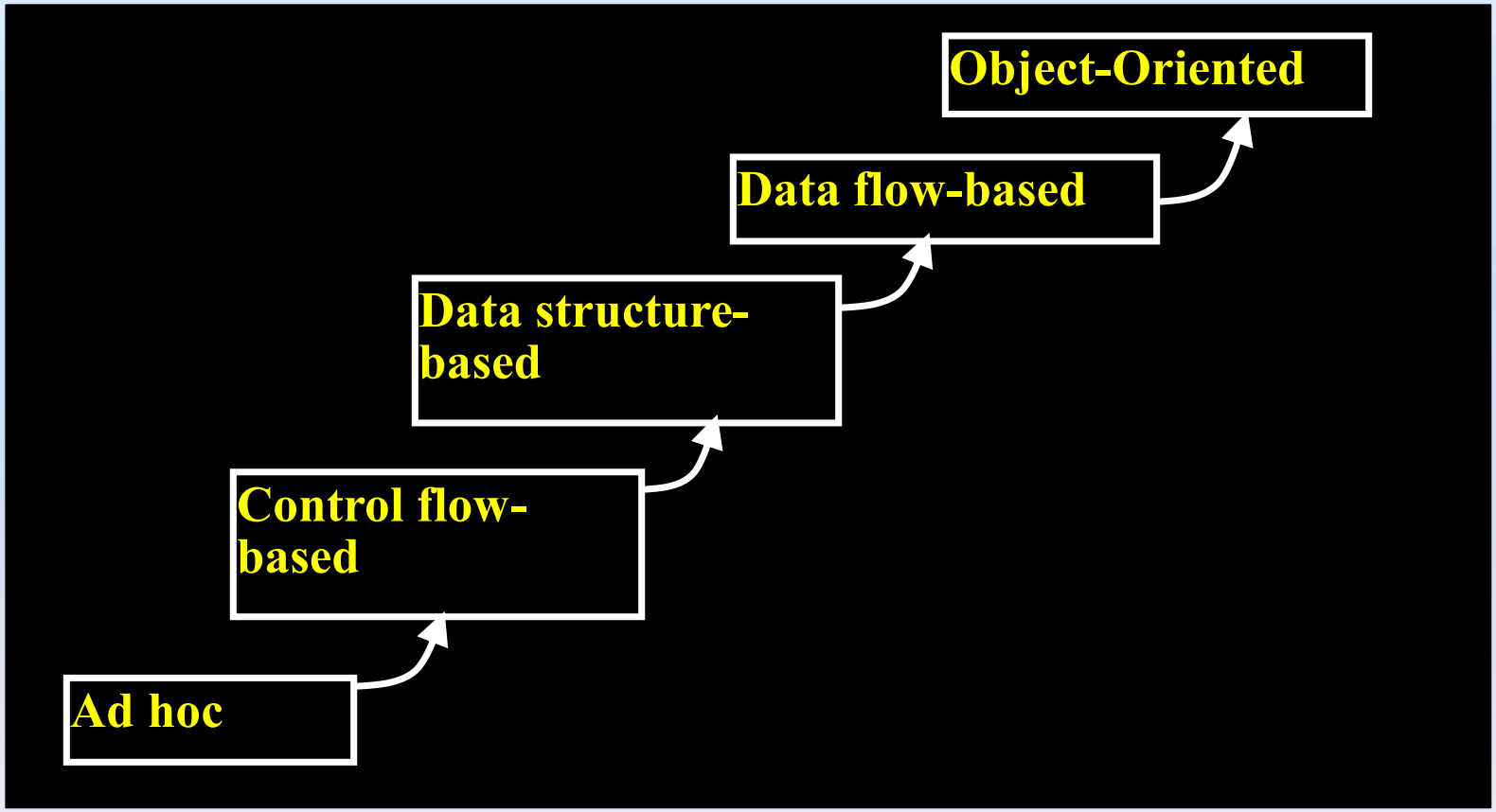
Evolution & Emergence of Software Engineering



Evolution of Software Engineering

- Early Programming Style *(Exploratory/ Adhoc, Assembly Language prog.)*
- High Level Programming *(Exploratory/ Adhoc, **HLL-** Fortran, Cobol…)*
- Control flow based Programming *(Design using **Flow-charts**)*
- Structured Programming *(Decompose into set of modules)*
- Data Structure Oriented Programming *(Design data structure – Then design program structure)*
- Data Flow Oriented Design *(Input-Process-Output : **DFD**)*
- Object Oriented Design *(Design objects & relationships between them)*
- Modern S/W Engg Techniques *(SDLC, CASE tools)*

Evolution of Design Techniques



1. Early Computer Programming (1950s)

- Programs were written in *assembly language*

- Every programmer uses his *own style*
  (Called exploratory programming)
- Difficult to learn & understand

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

There are many high level languages

Some Examples:

COBOL	Business applications
FORTRAN	Engineering & Scientific Applications
PASCAL	General use and as a teaching tool
C & C++	General Purpose - currently most popular.
PROLOG	Artificial Intelligence
JAVA	General all purpose programming
.NET	General or web applications.

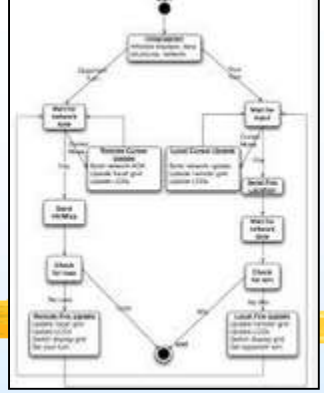
- High-level languages such as FORTRAN, ALGOL & COBOL were introduced

  Easy to learn & understand

- This reduced s/w development efforts

- S/w development style was still exploratory

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

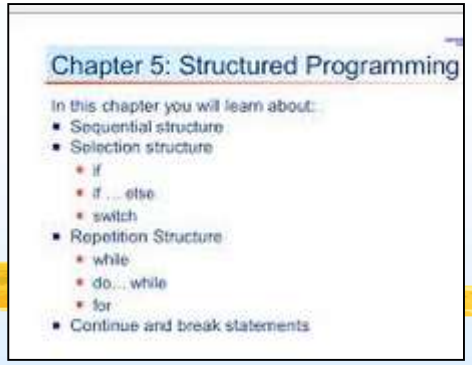


- Size & complexity of programs increased
- Exploratory programming style was insufficient
- Difficulty faced to write complex programs
- Difficulty faced to understand & maintain **other's code**
- Focus on **Control flow (Flow Chart)** based design
- **Flow charting technique** was developed

Disadv.s of Control Flow-Based Design

- Messy flow charts are difficult to understand & debug
  - GO TO statements makes a program messy
  - Alter the flow of control arbitrarily
  - Need to restrict use of GO TO statements

4. Structured Programming



- It was proved that a program needs only below 3 types of statements (GO-TO not needed)

- A program is called **structured**, if it uses below types of constructs :
  - **Sequence** (eg: a=0;b=5;)
  - **Selection** (eg: if(c=true) k=5 else m=5;)
  - **Iteration** (eg: while(k>0) k=j-k;)

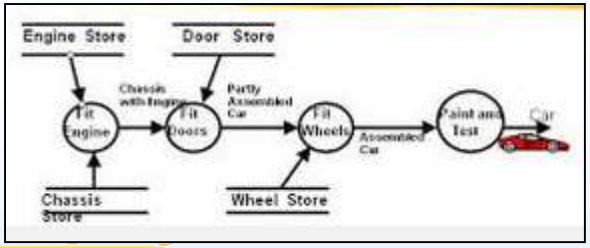
Structured programming

- Unstructured control flows are avoided
- Consist of a clean set of **modules**
- Use single-entry, single-exit program constructs
- **Structured programs** are :
  - Easier to read & understand & maintain
  - Require less effort & time to develop

5. Data Structure Oriented Design (Early 70s)

- In this methodology:
  - Program's data structures are first designed
  - From that program structure is derived

6. Data Flow-Oriented Design (Late 70s)

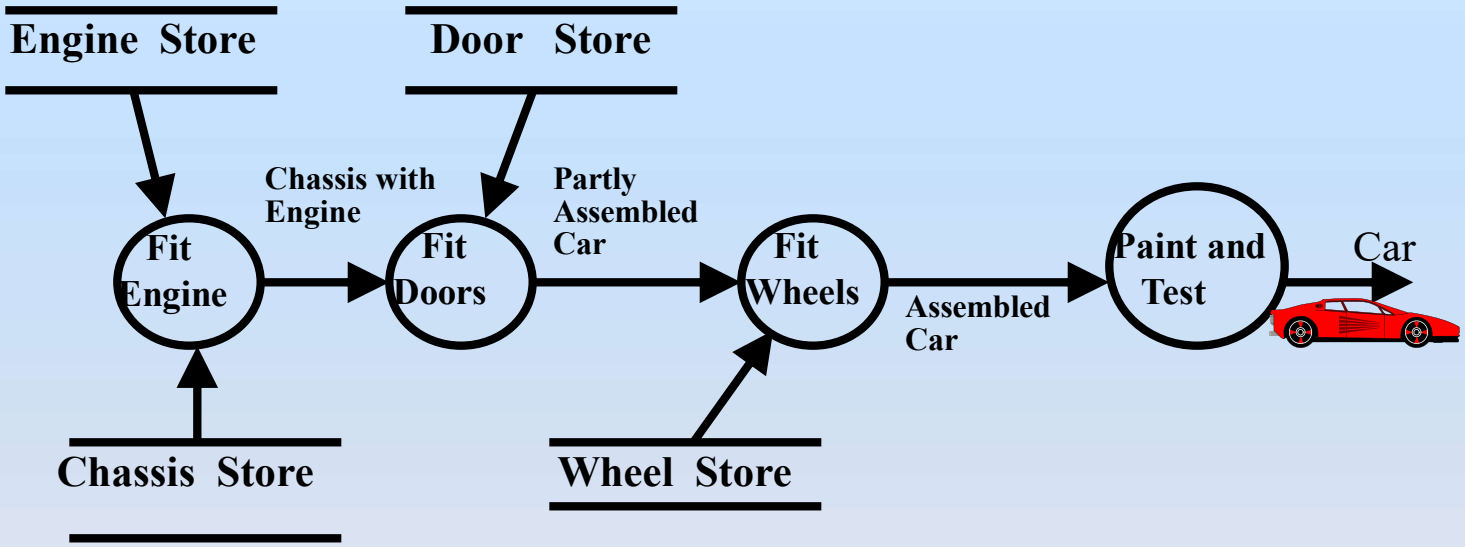


- In Data flow-oriented technique:
  - First Identify input data items of the system
  - Then Processing required to produce the outputs is determined
  - **Ex**: Payroll System, Credit card approval system

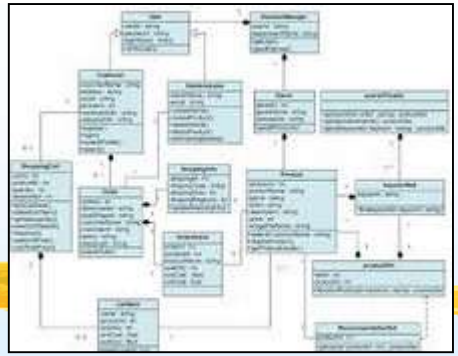
Data Flow-Oriented Design (Late 70s)

- Data flow technique is a generic technique:
  - Can be used to model the working of any system not just s/w systems
- A major advantage of the data flow technique is its **simplicity**

Data Flow Model of a Car Assembly Unit



7. Object-Oriented Design (80s)



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

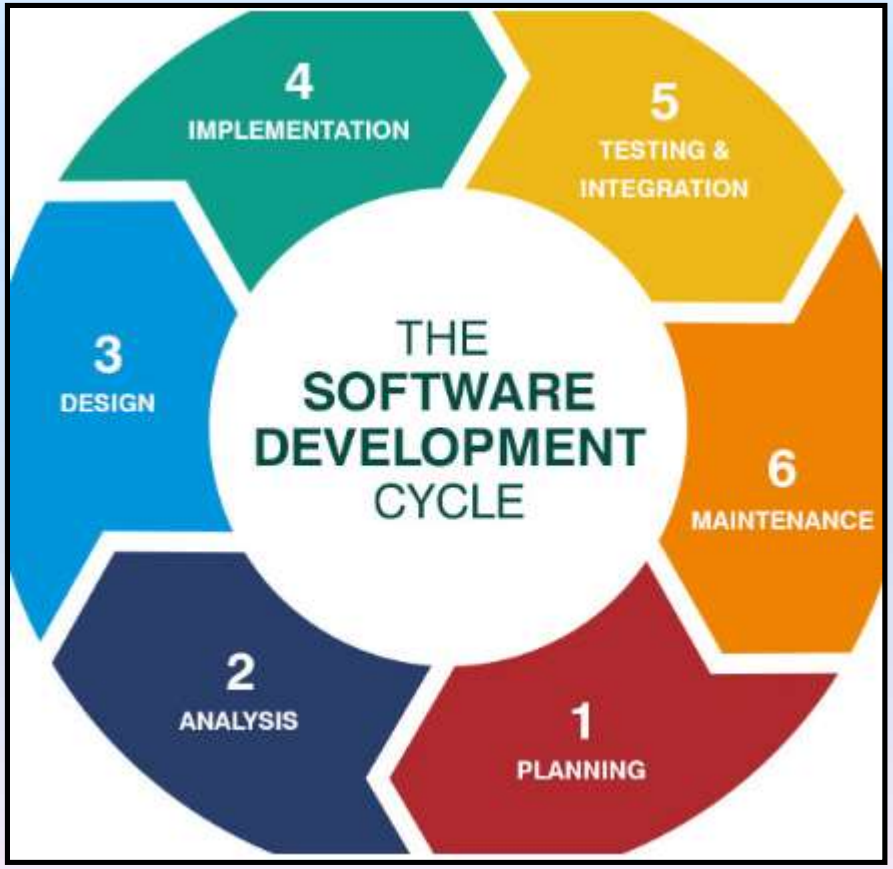
Advantages of Object-Oriented Design

- OO Techniques are very popular & well-accepted due to:
  - Simplicity
  - Reusability
  - Lower development time & cost
  - More robust code
  - Easy maintenance

Changes in S/W Engg Techniques

- Development of below Software Engg techniques :
  - Software Development Life cycle models **(SDLCs)**
  - Specification techniques (formal techs like State Transition diagrams)
  - Project management techniques (Chief-prog, Democratic)
  - Testing techniques (Automated, Manual)
  - Debugging techniques (Brute-force, Backtracking, Prog slicing)
  - Quality assurance techniques (ISO 9001, Six Sigma)
  - Software measurement techniques (LOC, FP)
  - CASE tools (Rational Rose/Architect) etc

Software Life Cycle Models



Topics:

- **Exploratory** Vs **Modern** S/W Engg techniques
- **Software Life Cycle**
- **Software Life Cycle Models**

Exploratory style VS modern s/w development practices

- Use of Life Cycle Models
- S/W is developed through well-defined stages:
  - Feasibility study
  - Requirements analysis & specification
  - Design
  - Coding
  - Testing
  - Maintenance

- Emphasis has shifted
  - From error correction to error prevention

Exploratory style Vs modern s/w development practices

- In exploratory style, errors are detected **only** during **testing**
  - Now, focus is on detecting errors in **each phase of s/w development**
- In exploratory style, main focus was on **coding**
  - Now, coding is considered only a small part of s/w dev.
- A lot of effort and attention is on requirements specification

Exploratory style Vs modern s/w development practices

- Creation of good quality documents
  - In the past, very little attention was being given to producing good quality documents

✓ **Several metrics are being used:**

✓ To help in s/w project mgmt, quality assurance etc.

CASE tools are being used

Process Framework

- **Software Process** - *A process is a collection of activities, actions & tasks that are performed when some work product is to be created*
- **The Process Framework** - *A process framework provides the foundation for a complete s/w engg. process by identifying a no. of framework activities*
- Process framework specifies a set of *umbrella activities that are* applicable across the entire s/w process
- A generic process framework for s/w engg contains five activities:

1. Communication	2. Planning	3. Modeling	4. Construction	5. Deployment

Process Framework

1. Communication

- Communication with customer & stakeholders is very important in order to understand the project objectives & to gather requirements that help define s/w features & functions.

2. Planning

- Creation of a map, called a s/w project plan to define the s/w engg work by describing the tech. tasks to be conducted, the likely risks, the resources required, the work products & the work schedule.

3. Modeling

- A s/w engineer needs to create models of the system to understand the requirements & the design that will achieve the requirements.

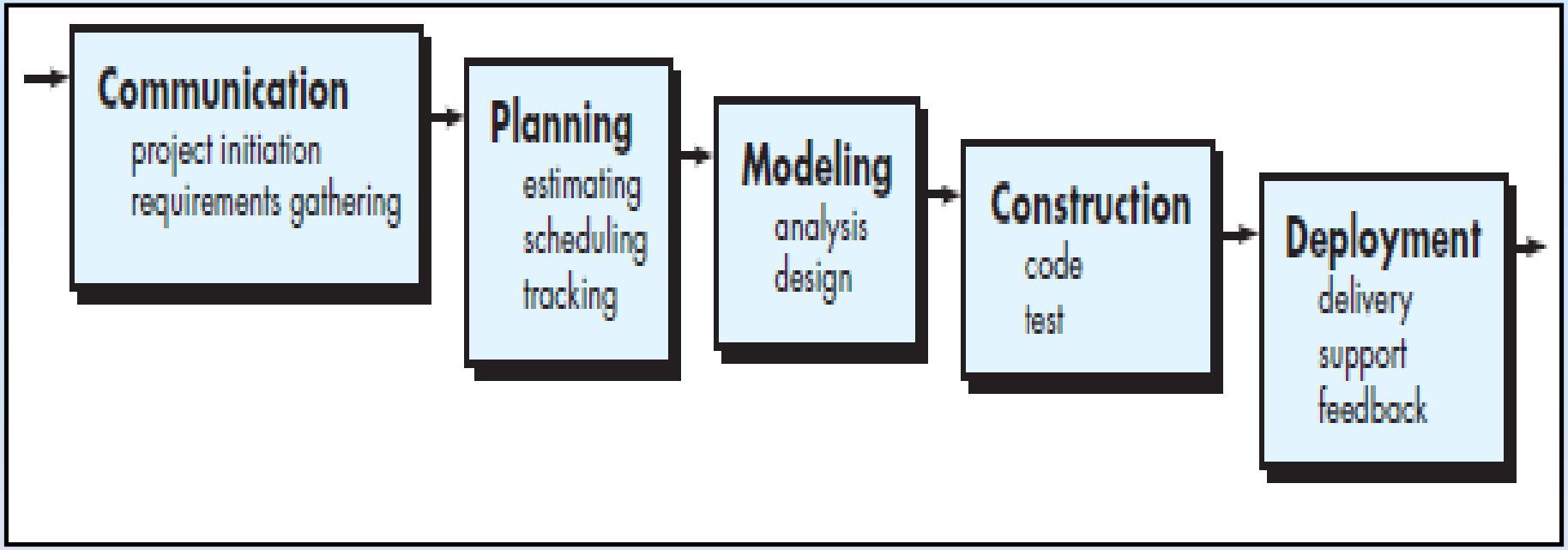
4. Construction

- Building what was designed. It is a combination of code generation & testing to uncover errors in the code.

5. Deployment

- The process of delivering the completed product to the customer.

Process Framework



Capability Maturity Model (CMM)

by SEI

Process Capability Models : SEI Capability Maturity Model (CMM)

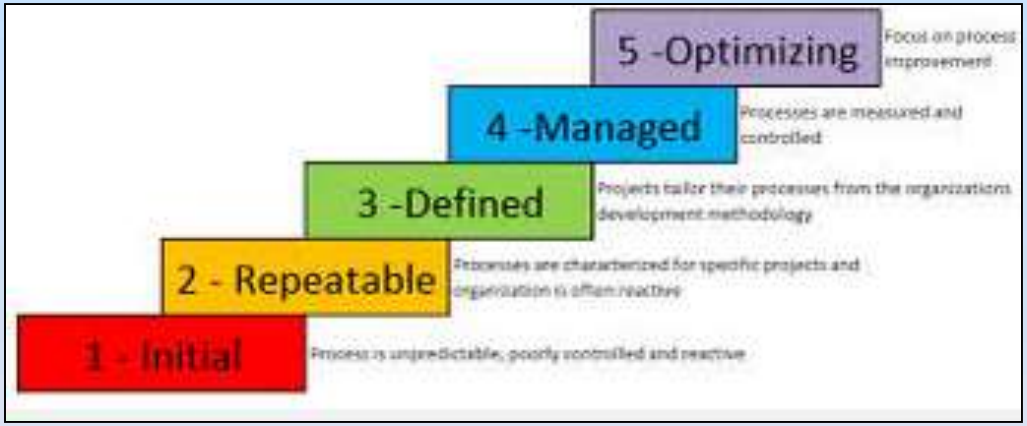
- **SEI-CMM** is a widely-accepted **quality certification** offered by **SEI** mainly to s/w organizations
- **SEI** - *Software Engineering Institute (SEI),* Mellon University, USA
- **Aim:** To improve **quality** of *s/w products* through **5 stages**

Process Capability Models : SEI Capability Maturity Model (CMM)

- *SEI CMM* helps organizations:
  - **To improve quality of s/w developed**
- *Very popular* & *adopted* by many organizations

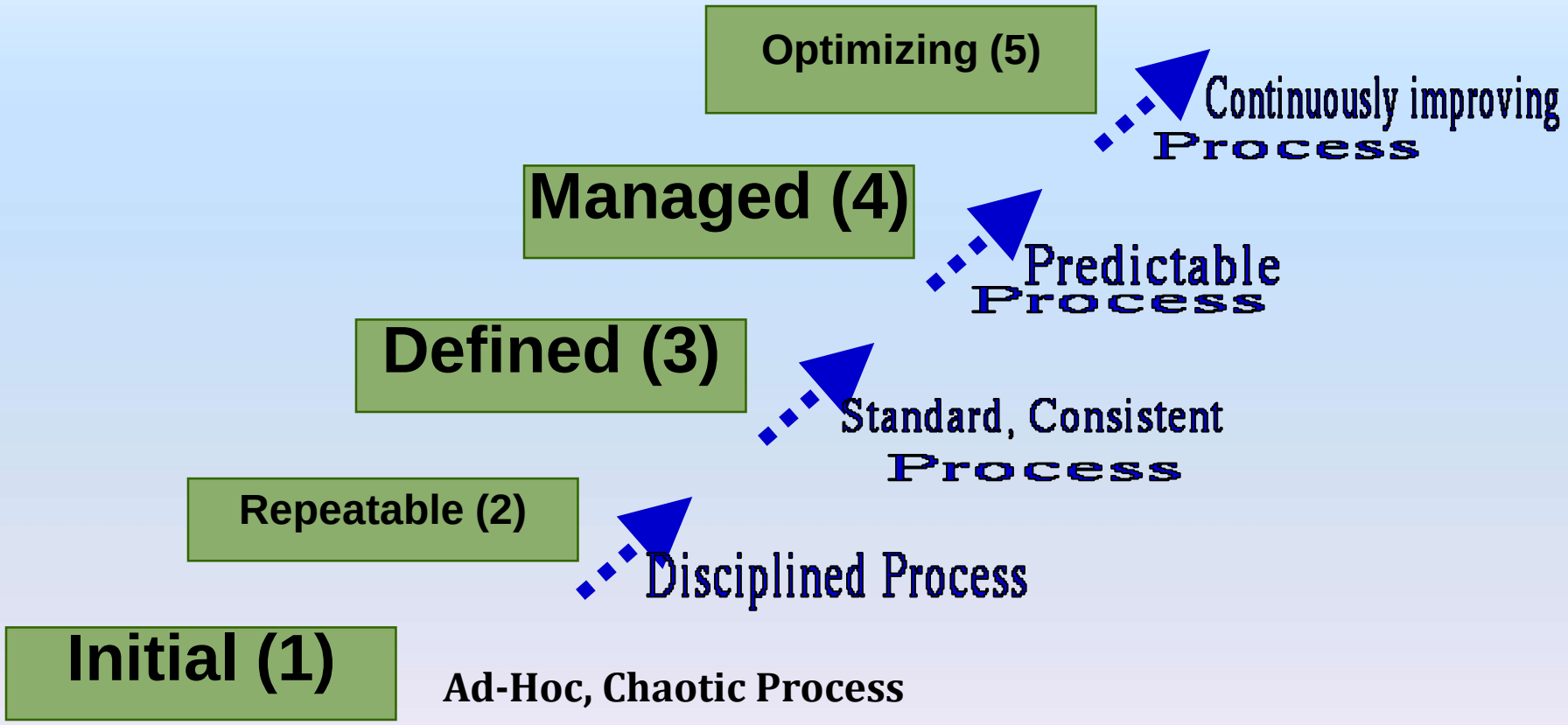
Process Capability Models : SEI Capability Maturity Model (CMM)

- **CMM** is a model for **evaluating** the **s/w process maturity** of an Org. into one of **five different levels**



- Can be used in two ways:
  - For Capability evaluation of an Org.
  - Software process assessment

SEI -CMM


Optimizing (5)
Continuously improving Process
Managed (4)
Predictable Process
Defined (3)
Standard, Consistent Process
Repeatable (2)
Disciplined Process
Initial (1)
Ad-Hoc, Chaotic Process

Level 1: (Initial)

- Org. operates **without** any **formalized process** or **project plan**
- Characterized by **ad hoc** and often **chaotic activities**
- **Different engineers** follow their **own process**
- The **success** of projects depend on **individual efforts** & **heroics**

Level 2: (Repeatable)

- Basic project management practices like
  - Tracking of *cost*, *schedule* & *functionality* are followed
- Size and cost estimation techniques like
  - *Function point analysis*, *COCOMO*.. are used
- Development process is still ad hoc (*neither formally defined & nor documented)*
- Process may vary between different projects
- **Earlier success** on projects can be **repeated**

Level 3: (Defined)

- Management & development activities are :
  Defined & Documented
- Common org-wide standards of activities, roles & responsibilities exist
- The processes are **defined**
- But, process & product qualities are **not measured**

Level 4: (Managed)

- **Quantitative quality goals** for products are ***set***
  - **Ex:** Defects per Kloc, MTBF
- *Software processes* & *product qualities* are **measured**
- The *measured values* are used to improve the product quality
- But, these are **not** used to ***improve the processes***

Level 5: (Optimizing)

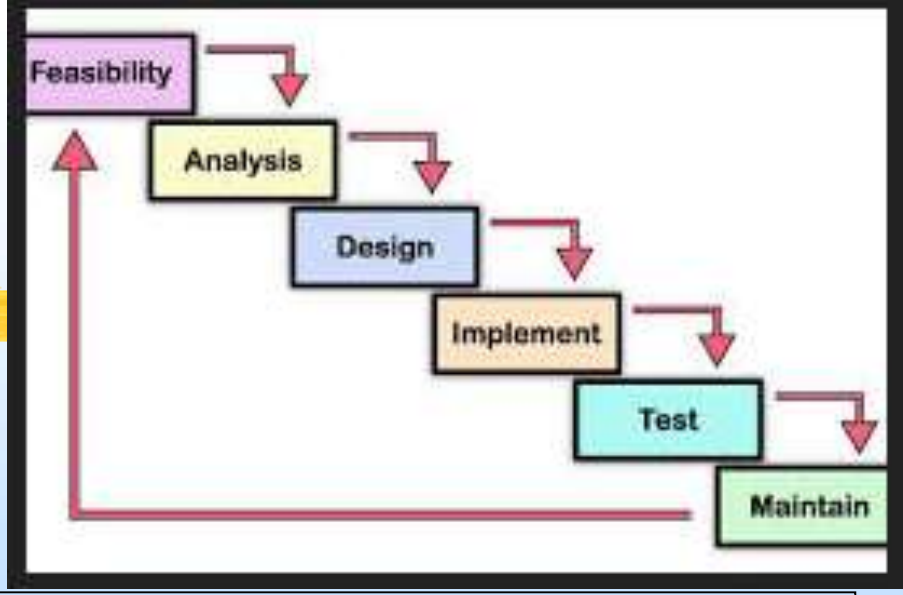
- *Statistics collected* from process/product measurements are analyzed:
- **Continuous process improvement** is done *based on the measurements*
- *Known types of defects* are **prevented** from recurring
- Lessons learned from projects incorporated into the process
- **Best software engineering** practices, methods & **innovations** are identified & promoted throughout the org.

Software Life Cycle

Objectives :

- What is a **life cycle model**
- What **problems** would occur if life cycle model is not used
- Different **software life cycle models**
- Different **phases** of the **software lifecycle**
- **Activities** undertaken in each **phase**
- **Shortcomings** of the each life cycle model

Life Cycle Model



- A **software life cycle model** (or process model):
  - Is a descriptive & **diagrammatic representation** of software life cycle
  - It shows all the activities required for s/w product development
  - Establishes a **precedence** ordering among the activities

Software Life Cycle Model

- **Software life cycle** (or software process):
  - **Series of stages/phases** that a software product undergoes during its life time are:
    - Feasibility study
    - Requirements analysis and specification
    - Design
    - Coding
    - Testing
    - Maintenance
  - Helps dev. of s/w in a systematic & disciplined manner

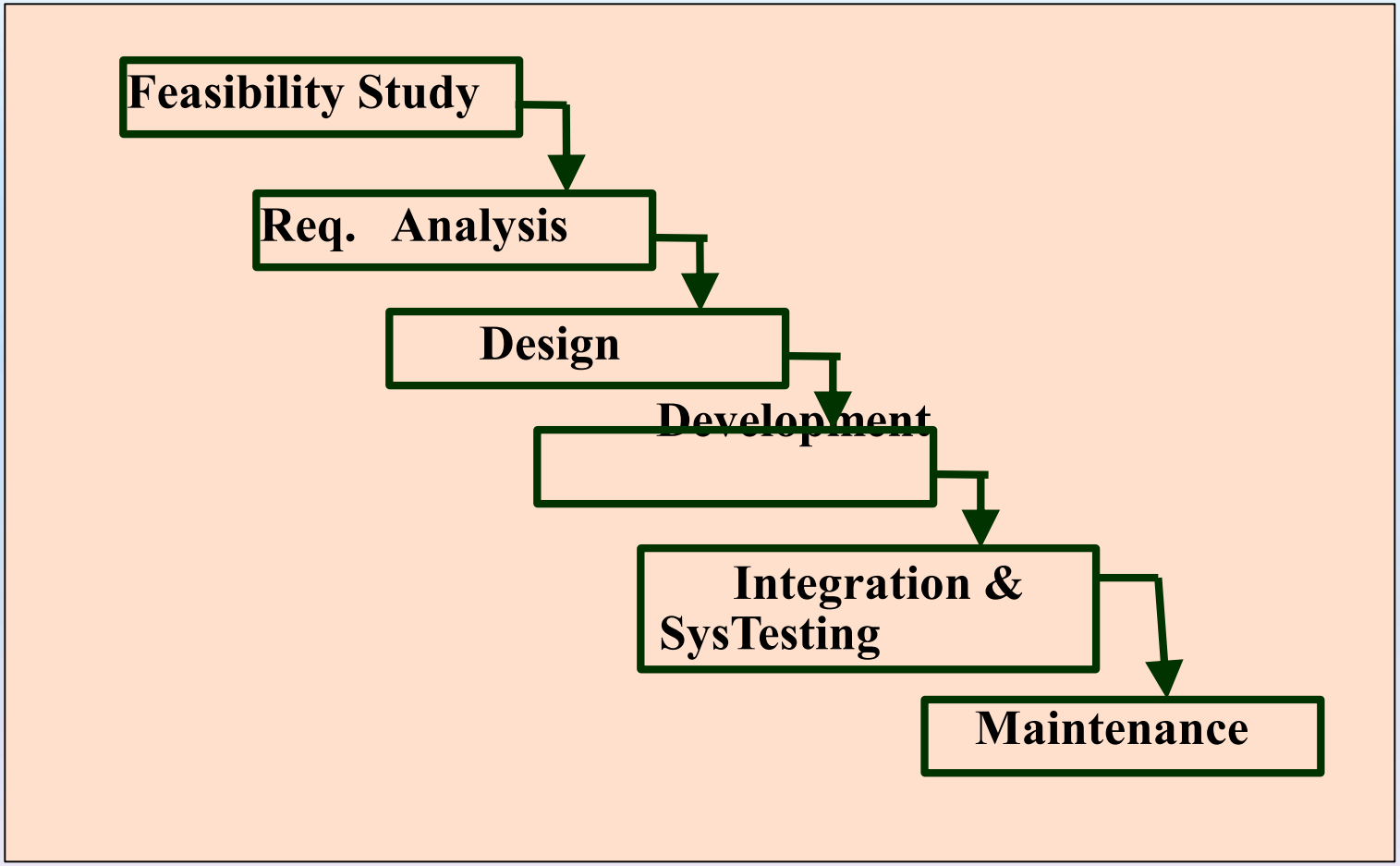
SDLC Models

- 1. Waterfall model
- 2. Iterative Waterfall model
- 3. V-Process model
- 4. Incremental Process model
- 5. Evolutionary Process models
  - 5.1. Prototyping model
  - 5.2. Spiral model
- 6. Agile & RAD Models
  - 6.1. Extreme Programming
  - 6.2. Scrum
  - 6.3. Crystal
- Unified Process

1. Classical Waterfall Model

- Classical waterfall model divides a project's life cycle into below phases:
  - *Feasibility study*
  - *Requirements analysis & specification*
  - *Design*
  - *Coding & Unit testing*
  - *Integration & system testing*
  - *Maintenance*

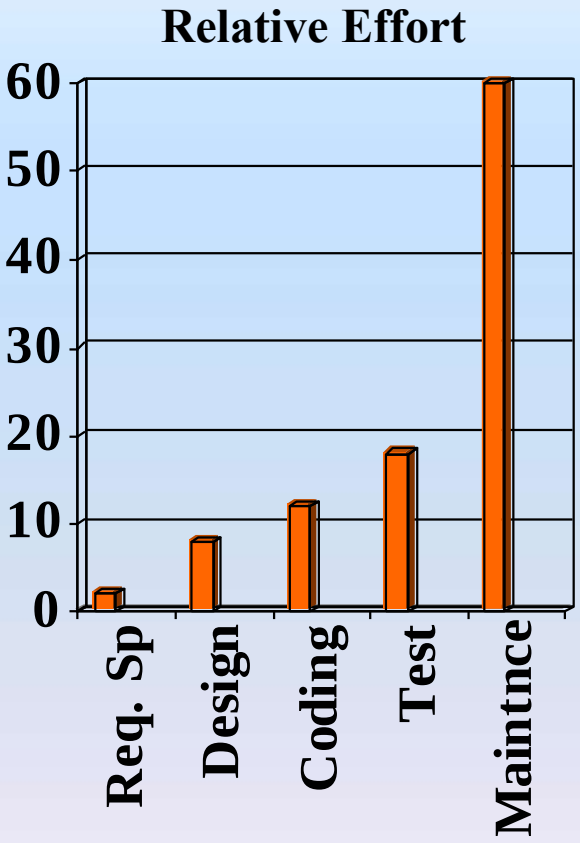
Waterfall model stages



Quiz: **Which stage takes most effort ??**

Relative Effort for Phases

- The phases between feasibility study & testing are
  - Known as **development phases**
- Among all life cycle phases
  - **Maintenance phase consumes maximum effort**
- Among development phases,
  - **Testing phase consumes the maximum effort**



Phase-1: Feasibility Study

- Main aim of feasibility study: is to determine whether developing the software product is :
  - Financially worthwhile
  - Technically feasible
- Work out an overall understanding of the problem :
  - Data inputs
  - Processing needed
  - Output data
  - Various constraints (timeline, resource, performance..)

Activities during Feasibility Study

- Formulate different solution strategies (Alternatives)
- Examine each strategy
- Based on that Decide whether the project is feasible

Phase-2: Requirements Analysis & Specification

- The aim of this phase is:
  - To understand the **requirements** of the customer accurately
  - Then **Document** them properly
- Consists of two distinct activities:
  - Requirements gathering and analysis
  - Writing Requirements specification

2.1. Requirements Gathering

- **Gathering Requirements**:
  - Requirements are usually collected from the **end-users** through

  - Interviews
  - Discussions
  - Survey/ Questionnaire
  - Workshops
  - Group or one-to-one meetings etc..

2.2. Requirements Analysis

- The data you initially collected from the users:
  - May contain several **contradictions** & **ambiguities**
- These **ambiguities & contradictions**:
  - Must be *identified*
  - *Resolved* by discussions with the customers
- Then requirements are *organized*:
  - Into **Software Requirements Specification (SRS)** document

Stage-3: Design

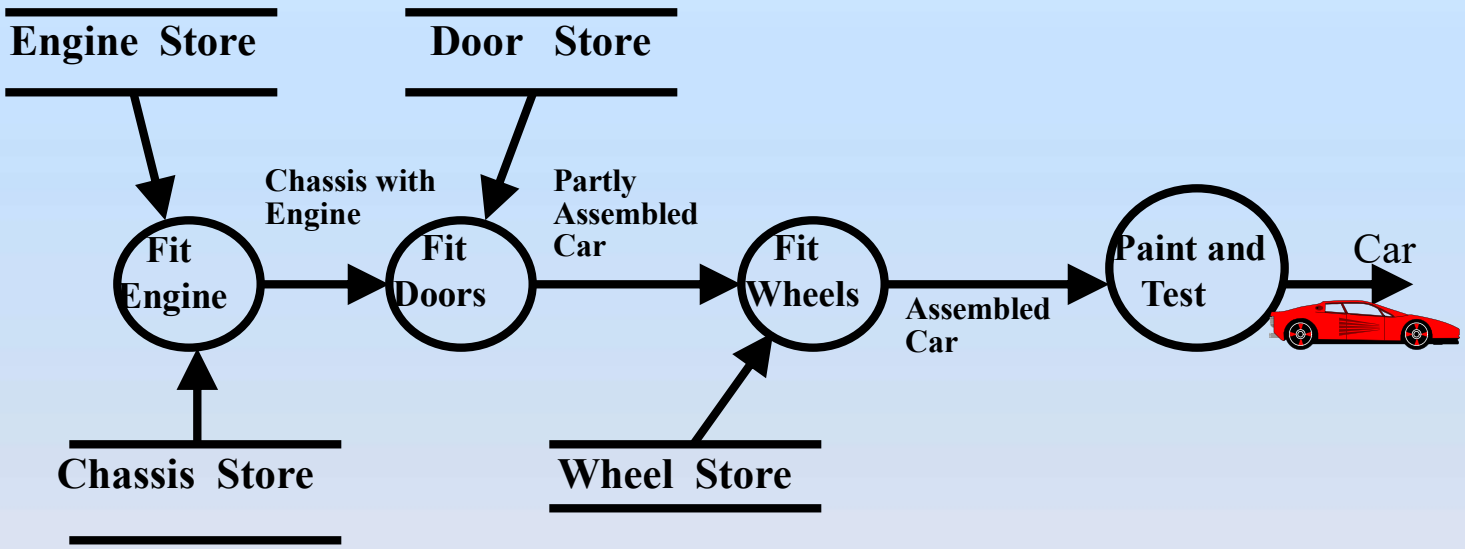
- Design phase **transforms** the "*Requirements specification*"
  - into **a form** suitable for **implementation** in some programming language

- Two design approaches are there:
  - **Function oriented approach**
  - **Object oriented approach**

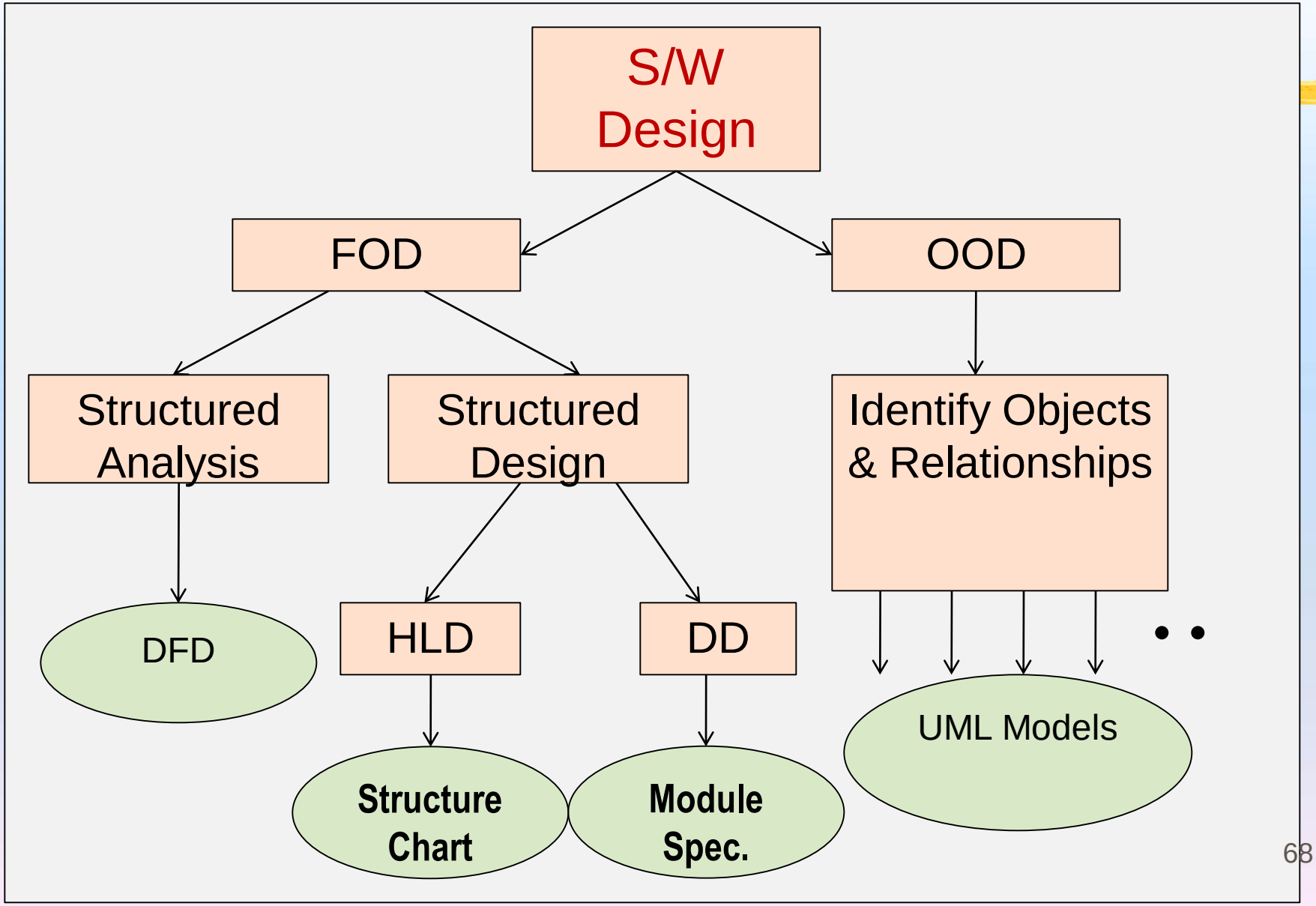
Function Oriented Design Approach

- Consists of two activities:
  - **Structured analysis**
    - Output is "**Data flow Diagram**"
  - **Structured design**
    - Outputs :
      1. High Level Design - "**Structure Chart**"
      2. Detail Design – "**Module Specification**"

Data Flow Model of a Car Assembly Unit



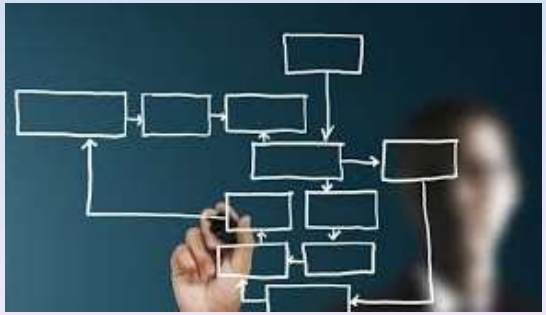
S/W Design Process



Object Oriented Design Approach

- First **identify various objects** (real world or conceptual entities) in the problem & **relationships** among them

- **Ex:** The objects in a pay-roll software may be:
  - Employees
  - Managers
  - Payroll register
  - Departments etc.



Stage-4: Implementation

- Purpose of ***implementation*** **(coding + unit testing)** phase is to :
  - **Translate software design into source code**

- During the **implementation** phase**:**
  - Each designed module is **coded** & **unit tested** *independently* for *correctness*

- The end product of **implementation phase:**
  - Program **modules** that have been **tested individually**

Stage-5: Integration & System Testing

Integration Testing :

- The **modules** are **integrated** in a **planned manner:**
  - Integrated in a **number of steps**
- During **each integration step**
  - **The partially integrated system is tested**

Integration Testing

System Testing

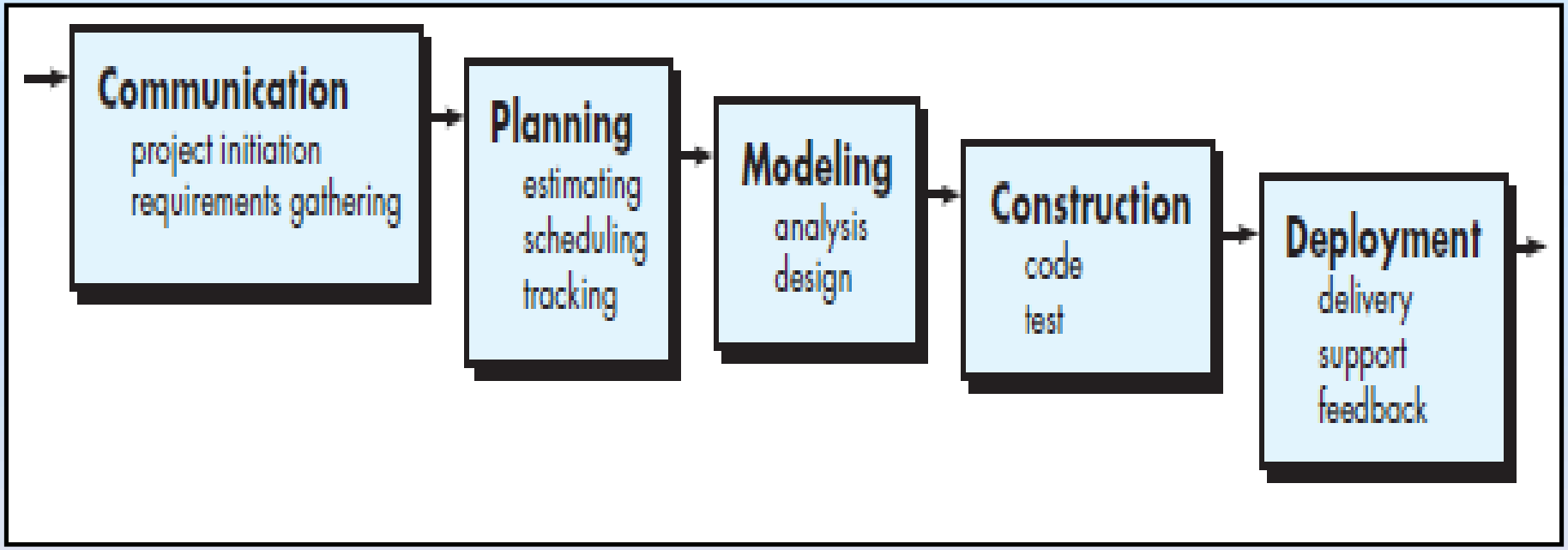
- **System testing** is carried out after Integration testing

- **Goal of system testing:**
  - Ensure that the "**Developed system**" **works** according to "***Requirements***" specified in the **SRS document**

Stage-6: Maintenance

- **Maintenance** of any software product:
  - Involves any **change** done to the product after it is delivered
  - Requires **more effort** than **development**
    - **40:60**

Waterfall model – an alternate diagram



WF model - Description of each stage

1. Communication

- Communication with customer & stakeholders is very important in order to understand the project objectives & to gather requirements that help define s/w features & functions.

2. Planning

- Creation of a map, called a s/w project plan to define the s/w engg work by describing the tech. tasks to be conducted, the likely risks, the resources required, the work products & the work schedule.

3. Modeling

- A s/w engineer needs to create models of the system to understand the requirements & the design that will achieve the requirements.

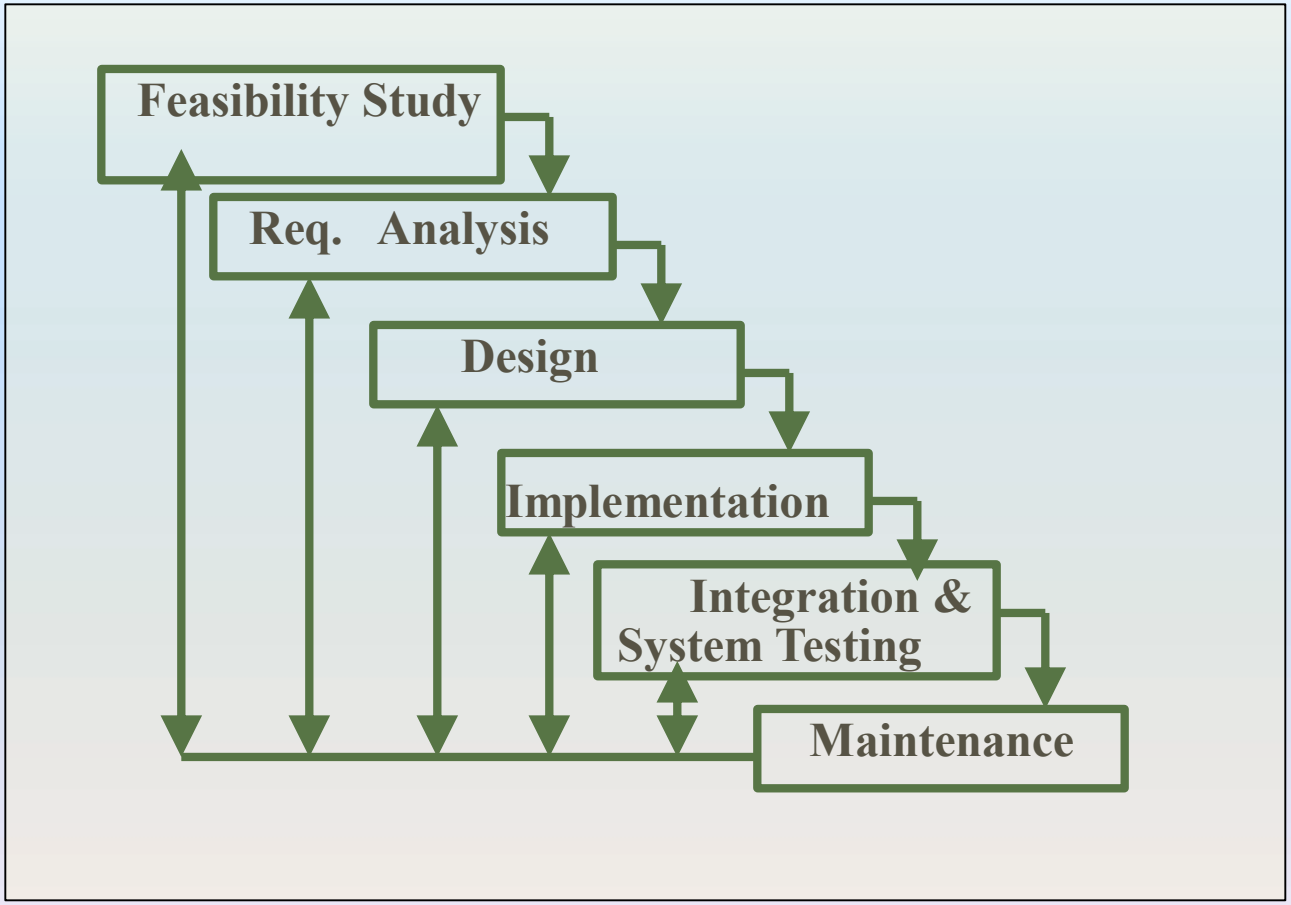
4. Construction

- Building what was designed. It is a combination of code generation & testing to uncover errors in the code.

5. Deployment

- The process of delivering the completed product to the customer.

(2) Iterative Waterfall Model



Iterative Waterfall Model

- **Classical waterfall model** is **idealistic** & **rigid**:
  - It assumes, **no defect is found** in earlier phases, after we have moved to the next phase
  - **In practice:** ***Defects*** are discovered in earlier phases
    - **Ex:** *A design defect might go unnoticed till the coding or testing phase*

Iterative Waterfall Model

- Once a **defect is detected**:
  - We need to **go back to the phase** where it was **introduced**
  - **Redo** some work in **that phase** & **subsequent phases**
- Therefore we need **feedback paths** in ***waterfall model***
- *Iterative waterfall model* was the **most widely used model**

Phase containment of errors

- **Principle of S/W Engg.** recommends:

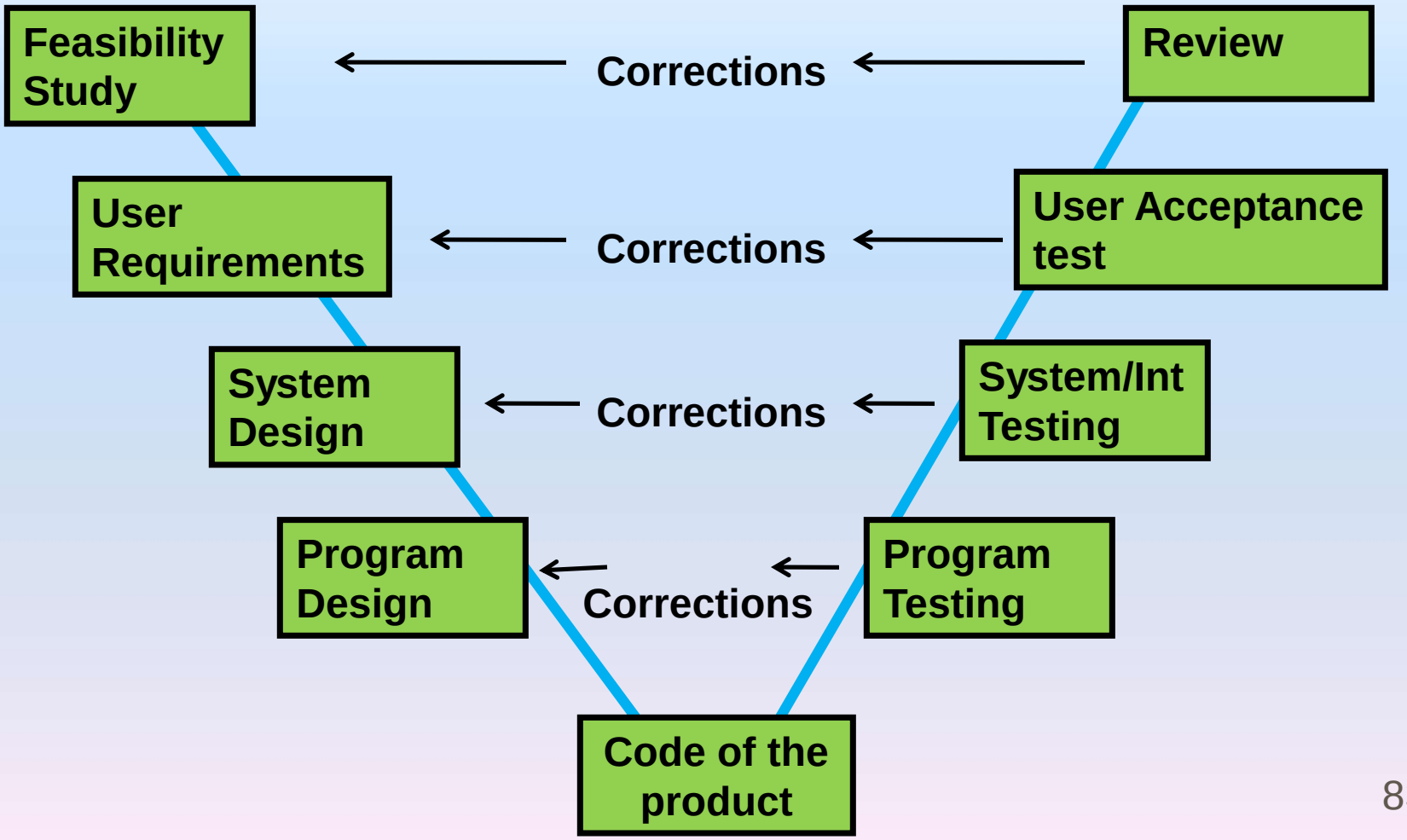
Detection of errors as close to its point of introduction as possible

- This is known as ***phase containment of errors***

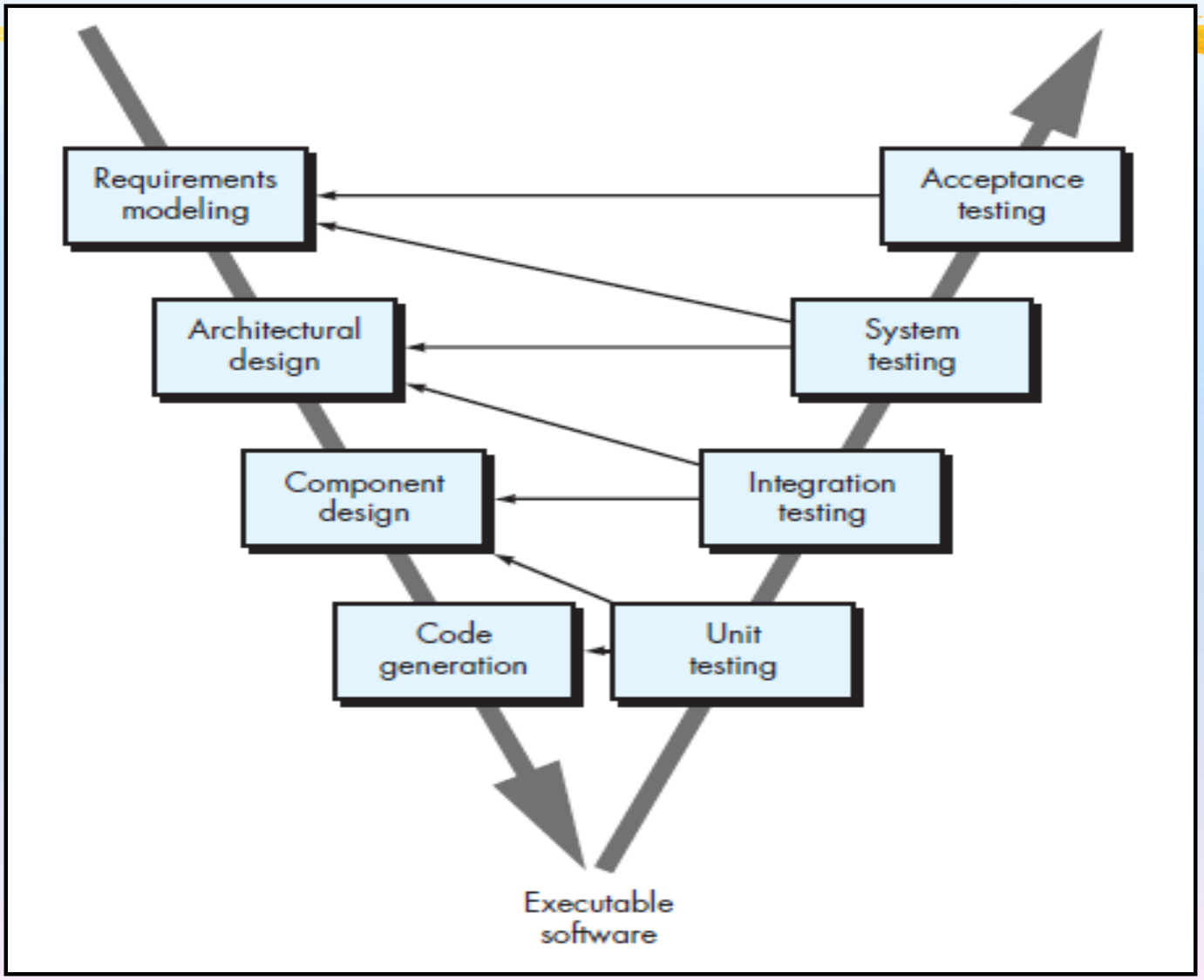
3. V-Process Model

- **Elaboration** of "Waterfall model" with stress on testing/validation of each phase
- **Expands** the Test/validation activities of Waterfall model
- Each Phase has a "matching validation" process
- If defects found, then "loop back" to corresponding development stage

V-Process Model



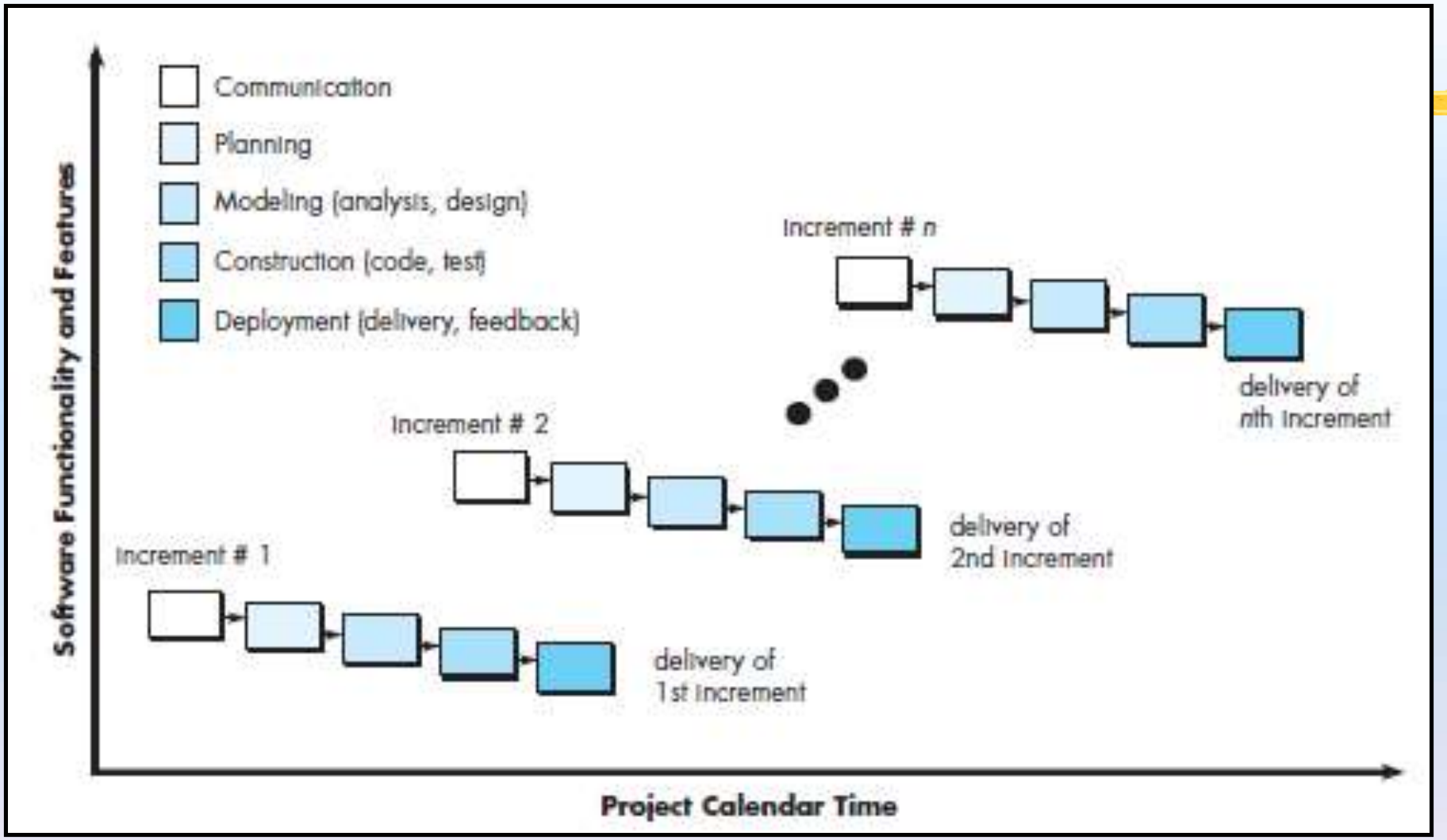
V-Process Model – alternate diagram



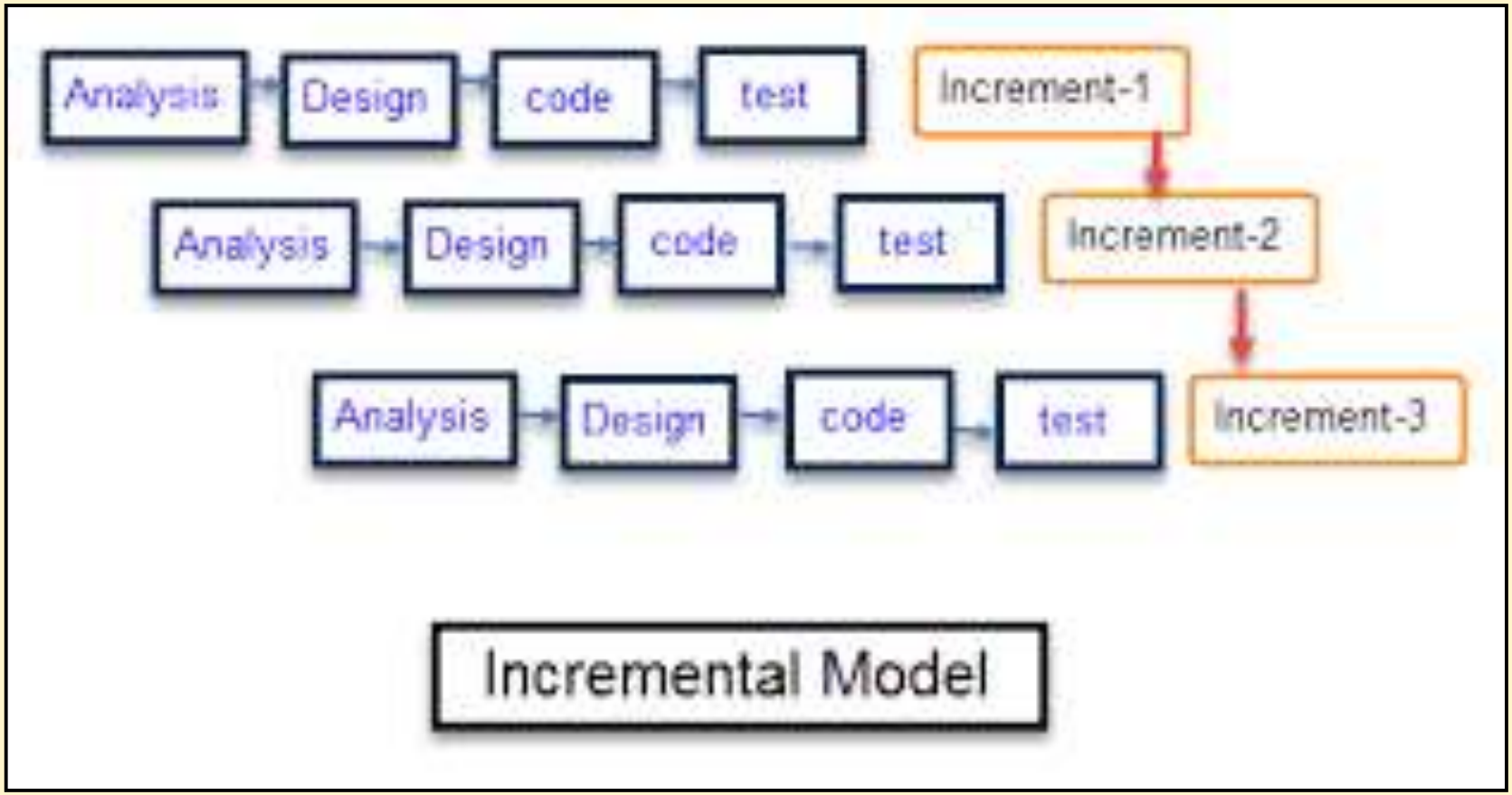
REVISION TO CONTINUE

(4) Incremental Process Model

- This model is suitable when:
  - There is a compelling need to provide a limited set of s/w functionality to users quickly & then expand that functionality in later s/w releases
- This model that is designed to produce the s/w in increments
- It combines both linear & parallel process flow
- It applies linear sequence in a staggered fashion as time progresses
- Each linear sequence produces deliverable "increment" of the s/w
- The 1st increment normally contains the "**Core**" part of the product
- Additional features are delivered in subsequent increments until complete product is produced


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
Increment # n
delivery of
nth increment
Increment # 2
delivery of
2nd increment
Increment # 1
delivery of
1st increment
Software Functionality and Features
Project Calendar Time

Incremental Model



(5) Evolutionary Process Models

- Software usually evolves over a period of time
- Business & product reqts often change as development progresses
- If the situation requires to release a limited version of the product to meet competitive or business pressure

  or
- The core system requirements are well understood, but additional features are yet to be defined
- Evolutionary SDLC model is suitable

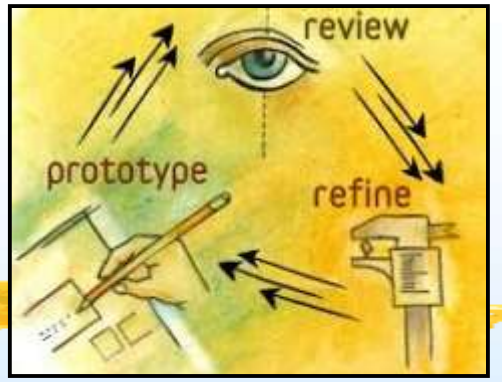
1. Evolutionary models are iterative in nature
2. They facilitate development of increasingly more complete versions of the s/w

- Two common evolutionary models are:

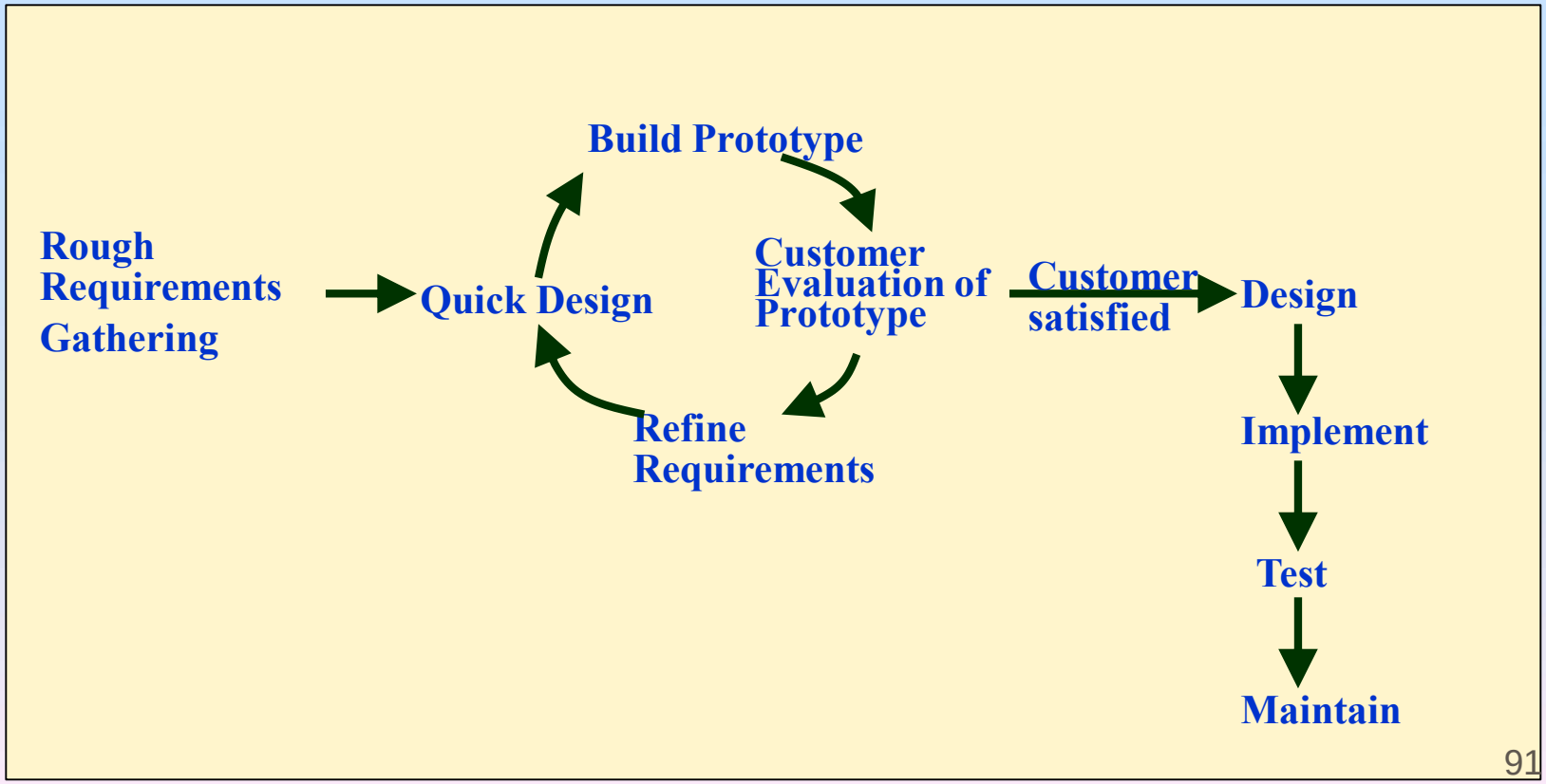
  5.1. Prototyping model

  5.2. Spiral model

(5.1) Prototyping Model



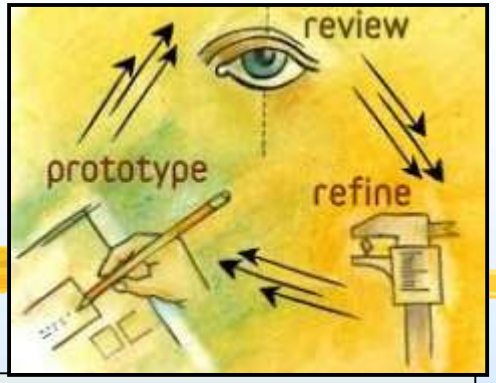
- Used for systems with :
  - ***Unclear*** *user requirements*
  - ***Unresolved*** *technical issues*

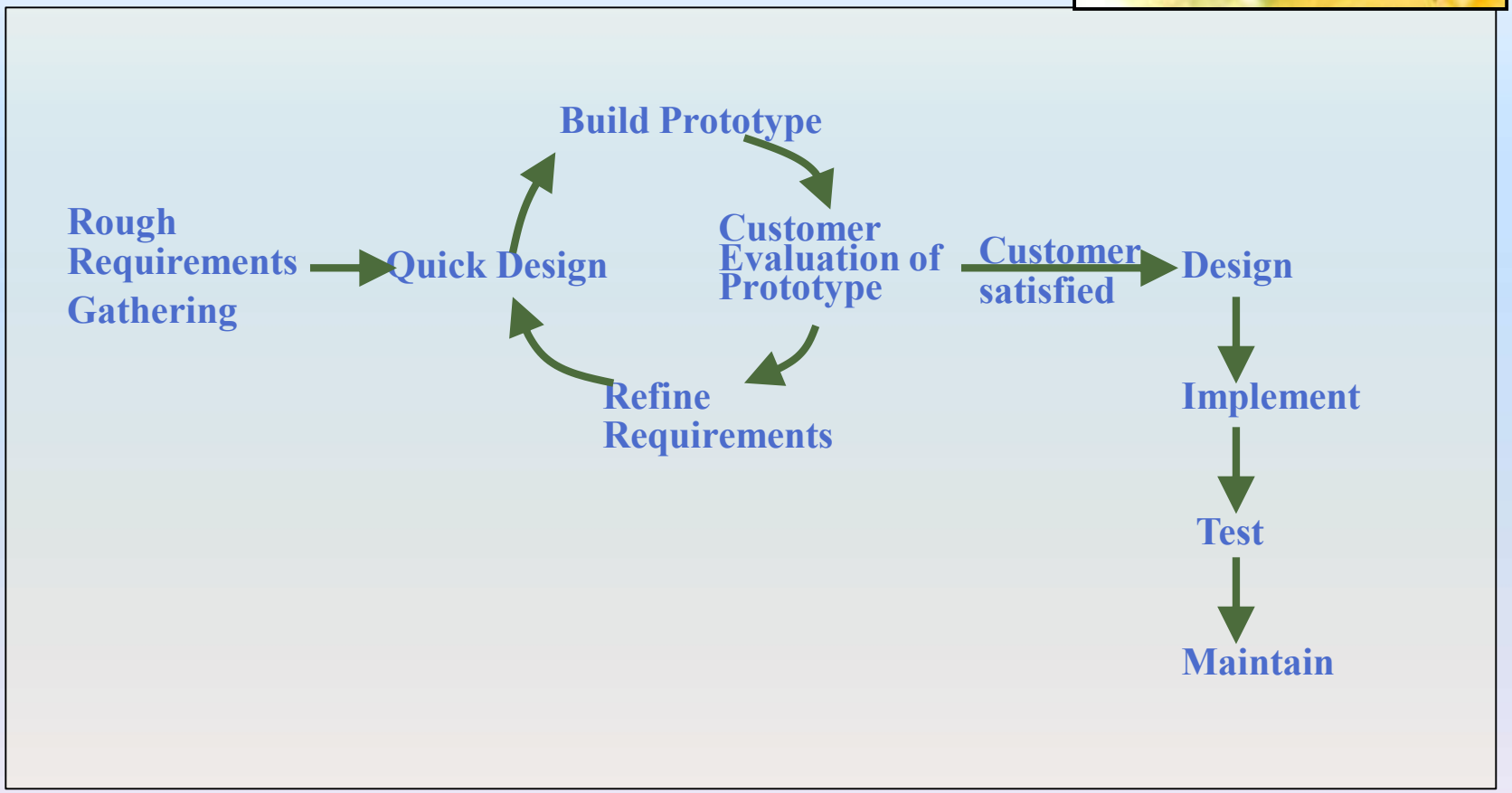


Prototyping Model

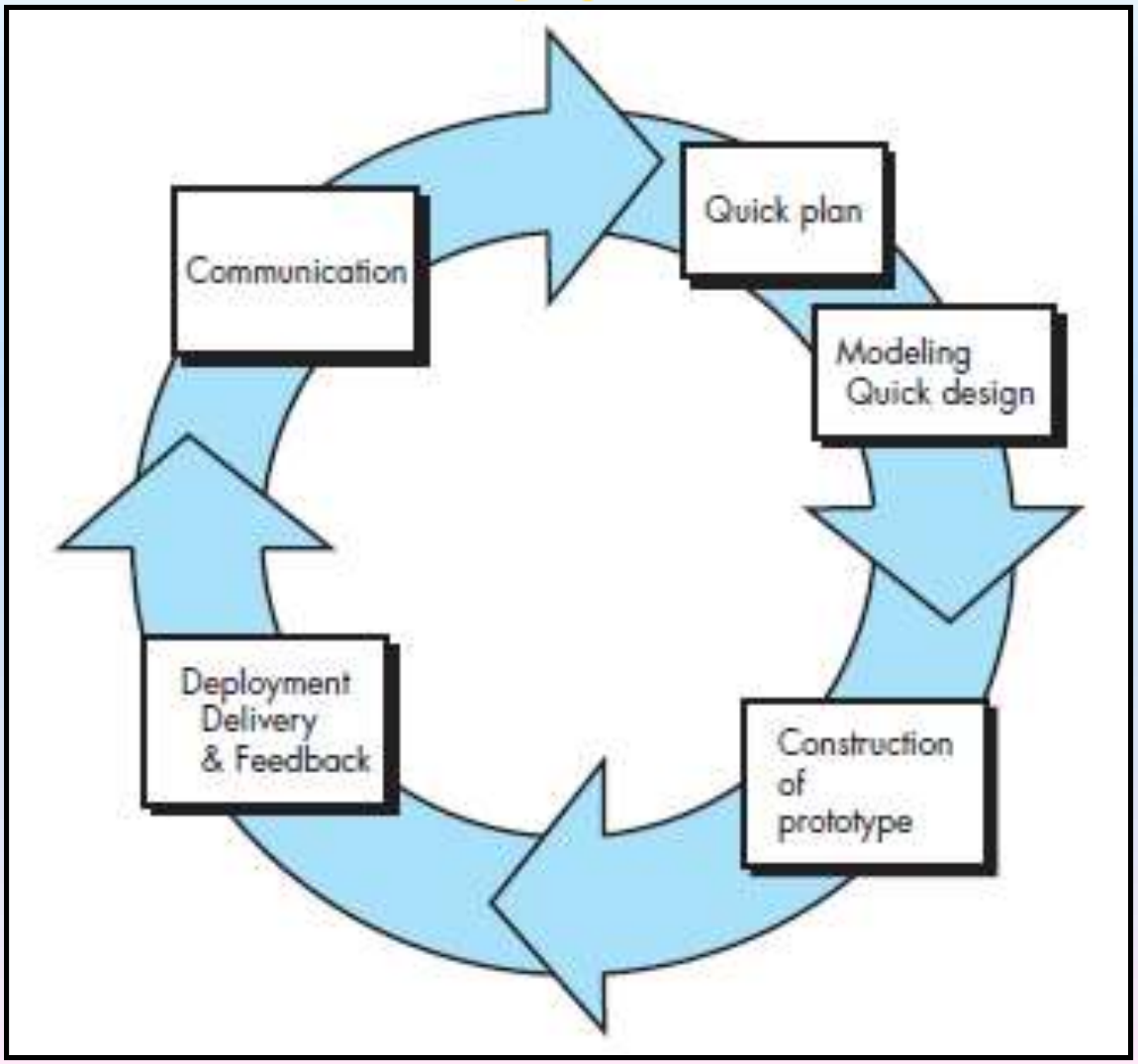
- **Water fall model** assumes that **requirements are completely known before** *development starts*
  - **It's not the case always**
- In such cases, before starting actual development,
  - A **"working prototype"** of the system should first be built
- **A prototype is a toy implementation of a system with:**
  - limited functional capabilities (dummy functions) , low reliability & inefficient performance
- It **Illustrates** the **system** to the **customer**
  - For providing **complete requirements**

Prototyping Model phases





Prototyping Model – alternate diagram



Prototyping Model Phases

- **Start with approximate or rough requirements**
- **Carry out a quick design**
- The ***prototype*** is submitted to ***customer*** for **evaluation**
  - Based on the *user feedback*, requirements are refined
  - This cycle continues until the user **approves** the prototype
- The actual system is developed using the **classical waterfall** approach

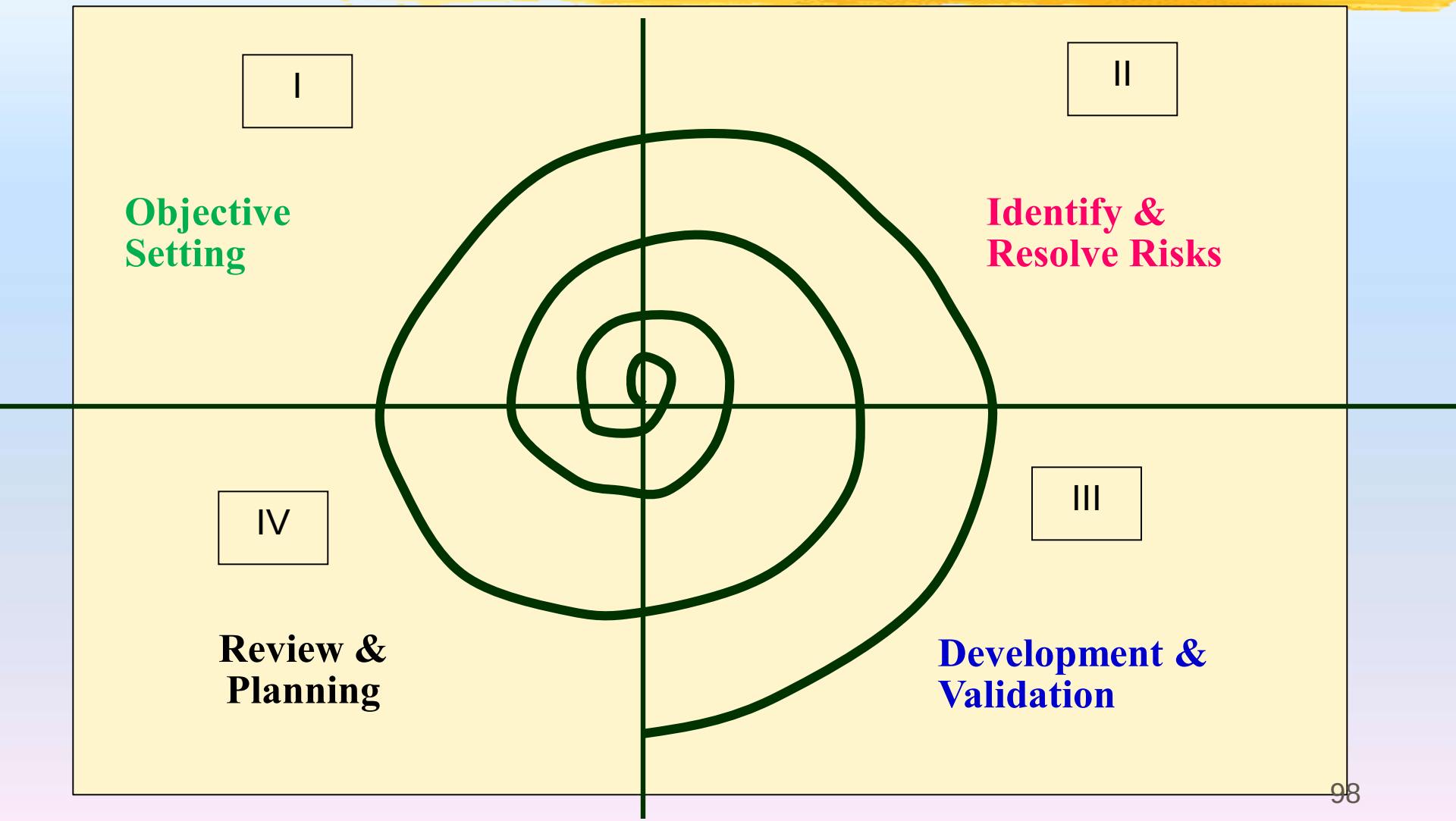
Prototyping Model Summary

- **Final working prototype** (with all user feedbacks) serves as an **animated requirements specification**
- The **prototype** is usually **thrown away:**
  - But, the experience gained **helps** with developing the actual product

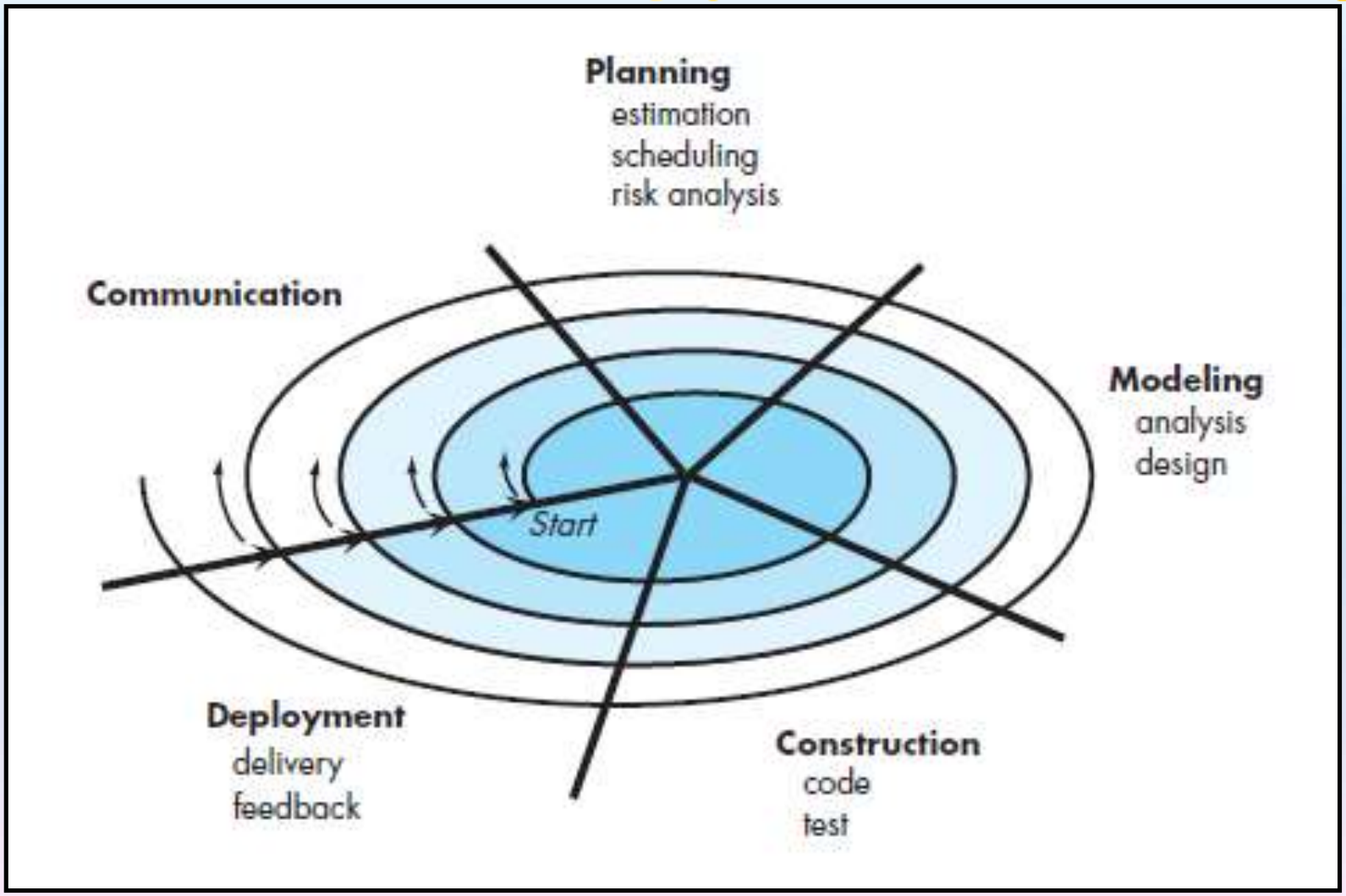
5.2. Spiral Model

- Proposed by **Barry Boehm** in **1988**
- This model appears like a **spiral with many loops**
- Each **loop** represents a **phase of the s/w dev. process**
  - The *innermost loop* may be *feasibility study* phase
  - *Next loop: requirements Analysis* phase
  - *Next one: system design*, and so on
- There are ***no fixed number of loops*** *or* phases
- The team *decides*: *How to structure the project into phases*

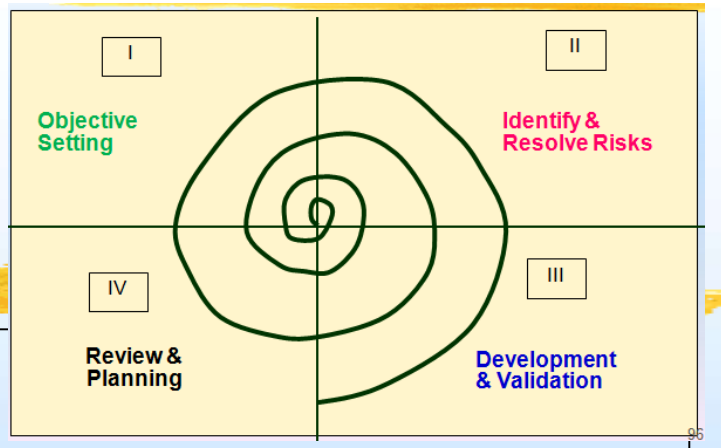
Spiral Model Graphical Representation



Spiral Model – alternate diagram

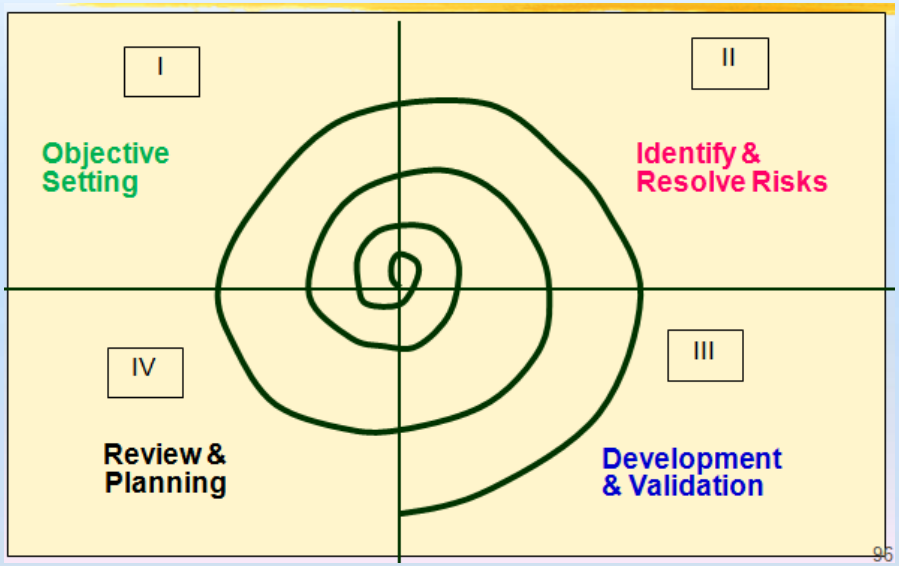


Activities Of Each Phase



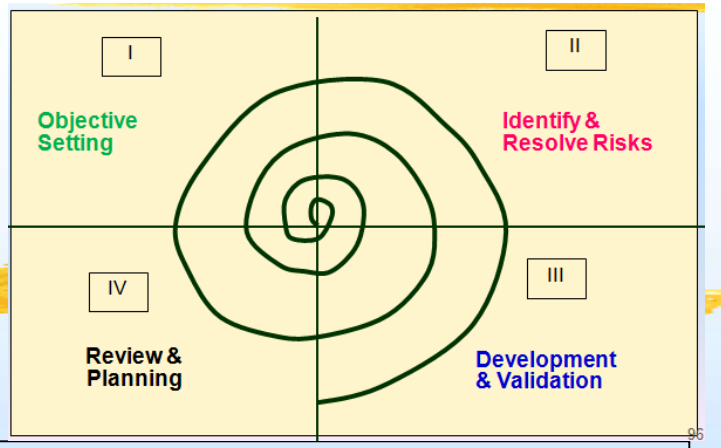
- *Each loop* in the spiral is ***split into 4 quadrants***
- The following *activities* are carried out in **each phase** :
  - **Objective Setting** (1st Quadrant)
  - **Identify & Resolve Risks** (2nd Quadrant)
  - **Development & Validation** (3rd Quadrant)
  - **Review & Planning** (4th Quadrant)

1. Objective Setting (First Quadrant)



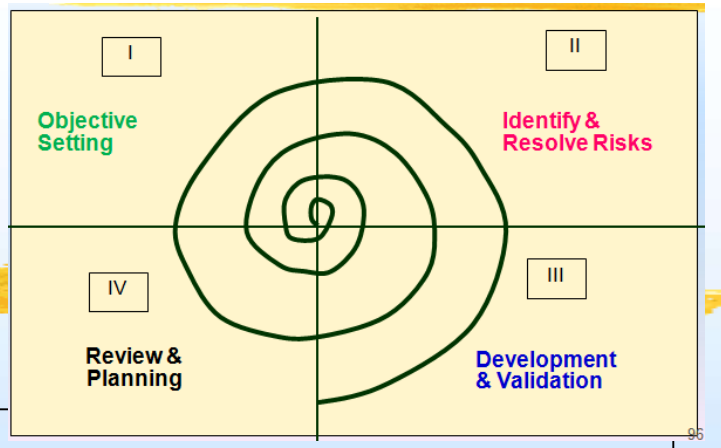
- Identify ***objectives of the phase***
- Identify **deliverables** for the phase *(SRS, Design docs ..)*

2. Identify & Resolve Risks (2nd Quadrant)



- Identify the ***risks*** associated with the objectives
  - **Risk**: Any *adverse circumstance* that might impact successful completion of a project
- **Analyze** each identified project risk
- Take steps to **resolve** or **reduce** the risk

Spiral Model



3. Development & Validation (Third quadrant):

- ***Develop*** &***validate*** the product

4. Review & Planning (Fourth quadrant):

- ***Review the results*** with the ***customer*** & ***plan the next iteration*** in the spiral

- **With *each iteration in the spiral*:**
  - ***More complete version*** of the s/w gets built

Adv.s & Disadv.s of Spiral Model

- **ADV.s:**
  - Suitable for projects with many unknown risks
  - More powerful than Prototyping model
- **DISADV.s:**
  - Complex model to follow unless have experienced staff
  - Not suitable for outsourced projects (as continuous risk assessment is needed)

Spiral Model as a meta model

- **Encompasses all discussed models:**
  - A single loop of spiral represents a **waterfall model** phase
  - Uses an **evolutionary** approach
    - Iterations through the spiral are evolutionary levels
  - Uses **Prototyping** as a risk reduction mechanism

Write more details

6. Agile Development Models
(Mid 90's)

- **Unsuitability** of Water Fall & Iterative models due to:
  - Late changes to requirements discouraged
  - No customer interaction allowed after Req. Stage
  - For customized s/w needing component reuse
- **Agile Models** help :
  - Avoid above shortcomings
  - to adapt to change requests quickly
  - Quick project completion by removing unnecessary activities

12 Agile Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.

12 Agile Principles

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Development Models

Features of Agile Models

- Reqts are divided into small parts for incremental development
- An iteration : approx. 2 weeks (time box)
- Delivery date is fixed
- Customer encouraged to give change requests
- Competent team
- Continuous customer interaction
- Pair programming

Adv.s & Disadv.s of Agile Methods

- **ADV**:
  - Agility to requirement changes
  - Highly effective if team members are competent
  - Reduced development time & cost
  - Elimination of overheads like formal docs & reviews
- **DISADV**:
  - Lack of formal docs lead to confusion & maintenance issues
  - Lack of review by external experts

Agile Development Models

- **Popular Agile Models :**
  - **Crystal**
  - Atern
  - **Scrum**
  - **Extreme Programming (XP)**
  - Lean Development
  - **Unified process**

6.1. Extreme Programming (XP)

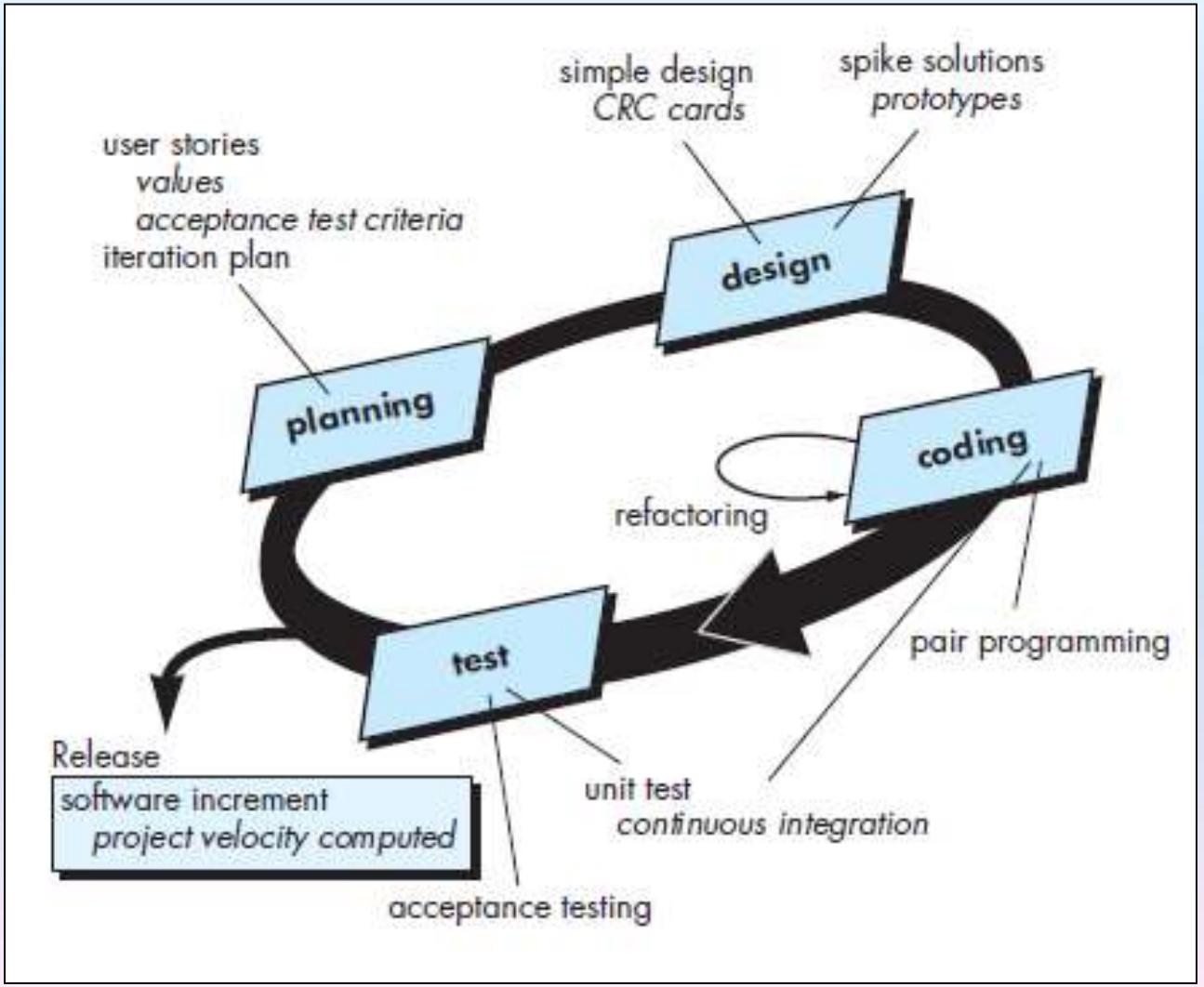
(Kent Beck – 1999)

- **Takes below best practices to extreme levels**
  - **Code Review:** Pair programming for continuous review
  - **Testing**: **Test driven dev. (TDD)** – write code & execute TCs
  - **Incremental dev.** to Implement customer feedbacks
  - **Simplicity:** Start with the simplest approach, enhance later
  - **Design:** Continuously improve design
  - **Int. Testing:** Continuous integration (no. of times a day)

Extreme Programming (XP)

- **Features of XP**
  - **Frequent releases** (Iterations): implement "user stories"
    - **User stories** *: Informal description of a feature by user*
      - **Ex**: *A Library member can issue a book if available*
  - **Metaphors:** How the system should work
    - Spike : A simple solution
  - **Coding** – Is the most important activity
  - **Testing**– High importance given
  - **Designing** – Effective simple design to be followed
  - **Feedback** – Frequent feedback obtained from customer
  - **Simplicity** – Build something simple that will work today

Extreme Programming (XP)



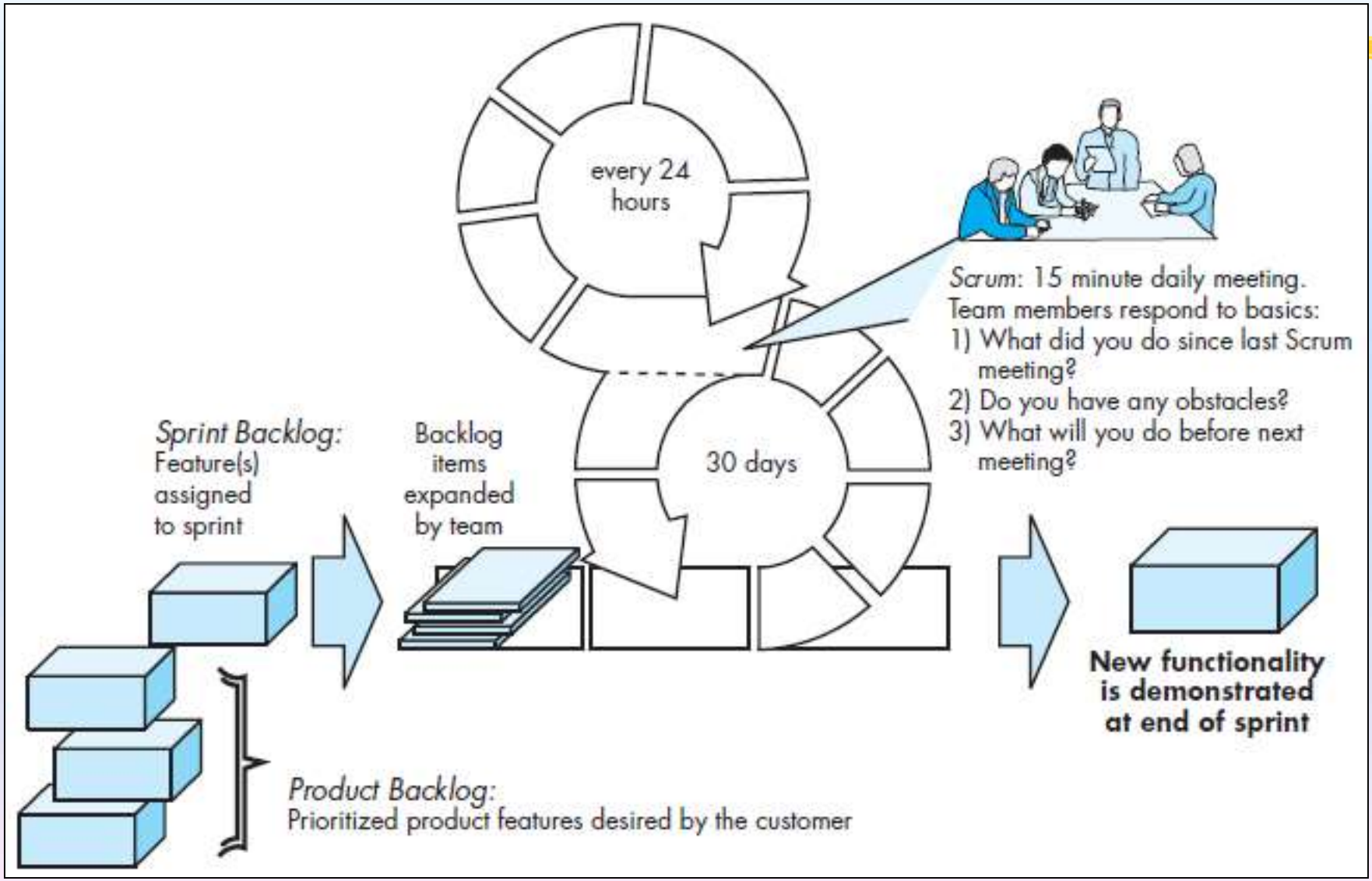
Extreme Programming (XP)

- **Applicability of XP**
  - Projects with new tech.(research projects)
  - Small projects
- **Unsuitability of XP**
  - Projects with stable requirements
  - Mission critical projects (need high reliability)

6.2. SCRUM Models

- **Features of Scrum model**
  - Project divided into small parts – developed incrementally
  - Time period of an increment – **Sprint** *(couple of weeks)*
    - Each Sprint will contain **multiple tasks** as decided by the **stakeholders**
  - Team & stake holders meet after sprint to discuss progress
  - Team members have **3 roles** :
    - **S/w owner** – communicates customer vision to team
    - **Scrum master** – liaison between s/w owner & team members
    - **Team member**

SCRUM Models



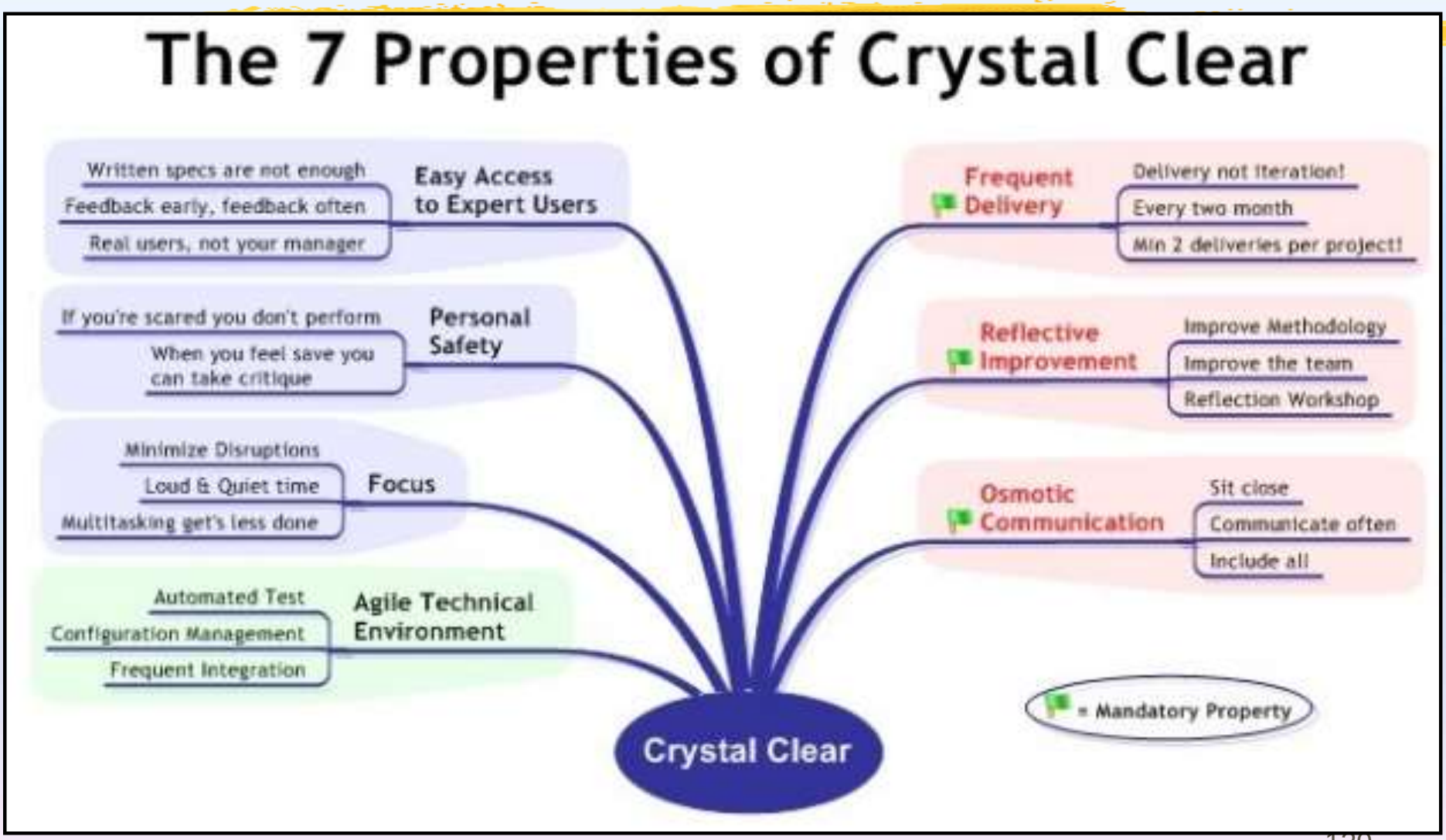
6.3. Crystal Methodology

- Developed by **Alistair Cockburn** in the **mid-1990s**
- Described as **"lightweight methodologies"**
- **Crystal methods are focused on:**
  - **People**
  - **Interaction**
  - **Community**
  - **Skills**
  - **Talents**
  - **Communications**

7 Properties of Crystal

- **The seven properties are:**
  - Frequent delivery
  - Reflective improvement
  - Close or osmotic communication
  - Personal safety
  - Focus
  - Easy access to expert users
  - Technical env with automated tests, config mgmt & frequent integration

Crystal Model

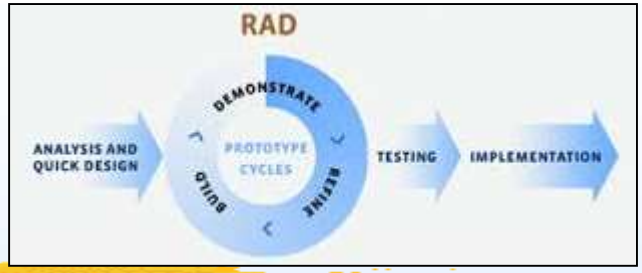


6.4. RAD Model (Agile, XP & Scrum)
(early 90's)

- **Goals of RAD model**
  1. Decrease time/cost of building s/w systems
  2. Reduce communication gap between customer & developers

- **RAD** (Rapid Application Development) **Model :**
  - Combines features from Prototyping & Evolutionary Models
  - Prototypes are first constructed
    - But not thrown away, used in system construction later
  - Features are developed incrementally & delivered to customer

RAD & Agile concepts

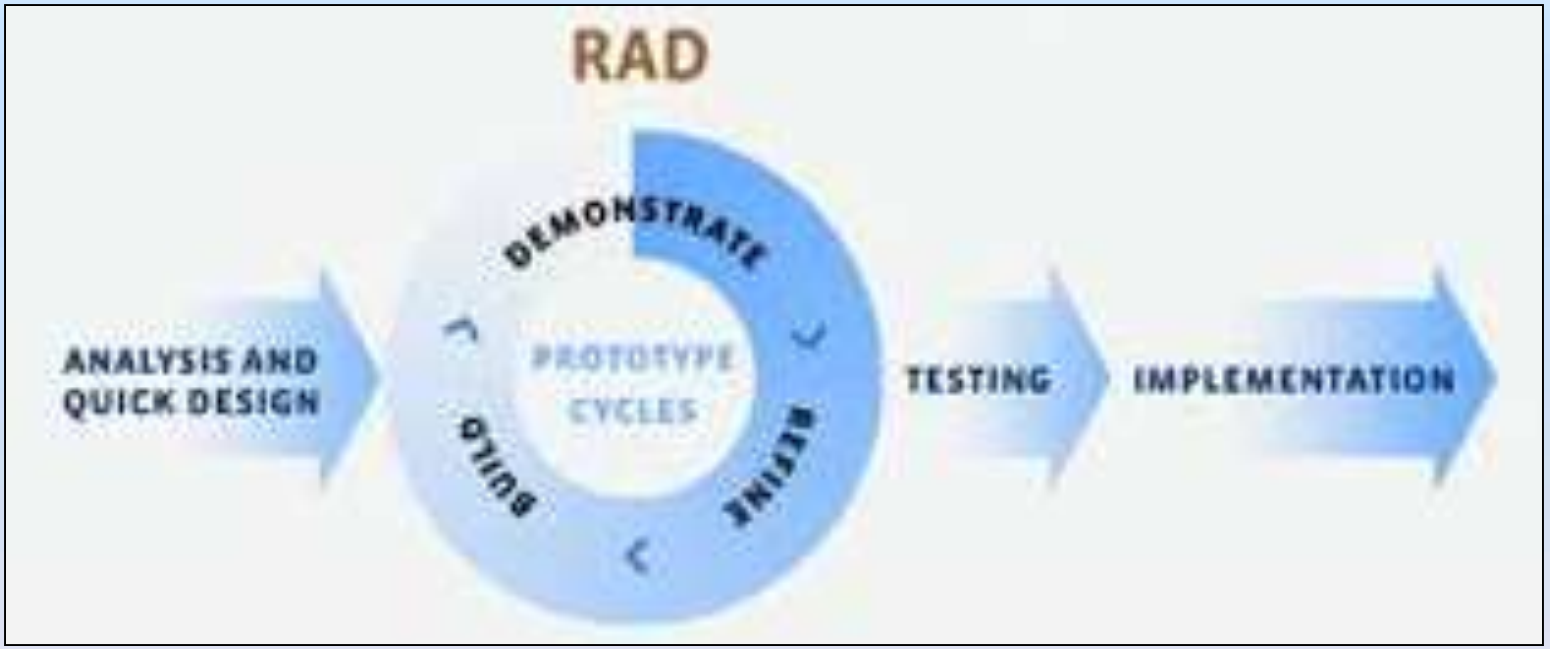


Agile models and **RAD** has many features in common.

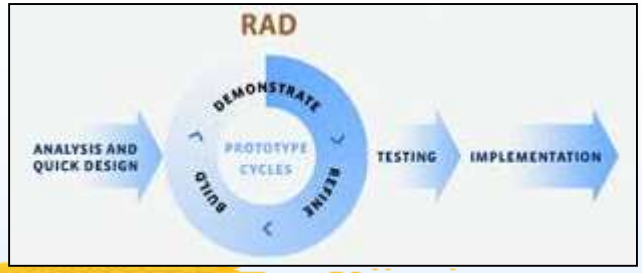
Both models:

- Break the product into small increments (time box) & deliver iteratively
- Focus on adaptability to customer requirement changes
- Aim at reducing time & cost by rapid delivery of working s/w by doing :
  - Minimal planning
  - Heavy reuse of code by rapid prototyping

RAD Model


RAD
ANALYSIS AND QUICK DESIGN
DEMONSTRATE
REFINE
BUILD
PROTOTYPE CYCLES
TESTING
IMPLEMENTATION

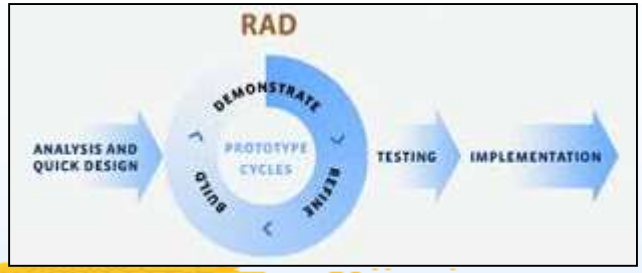
RAD Model



Working of RAD Model :

- Development takes place in short iterations (**Time box)**
- **During each time box :**
  - Quick prototype is made & refined based on customer feedbacks
  - Dev. Team contains customer rep.s to bridge communication gaps
- Customer is encouraged to give change requests

RAD Model



- **RAD Model is applicable to:**
  - Customized products
  - Non-critical & large s/w
  - Product with tight schedule
- **RAD Model is unsuitable for :**
  - Generic products (wide distribution)
  - Products needing reliability
  - Monolithic s/w (small – difficult to divide)

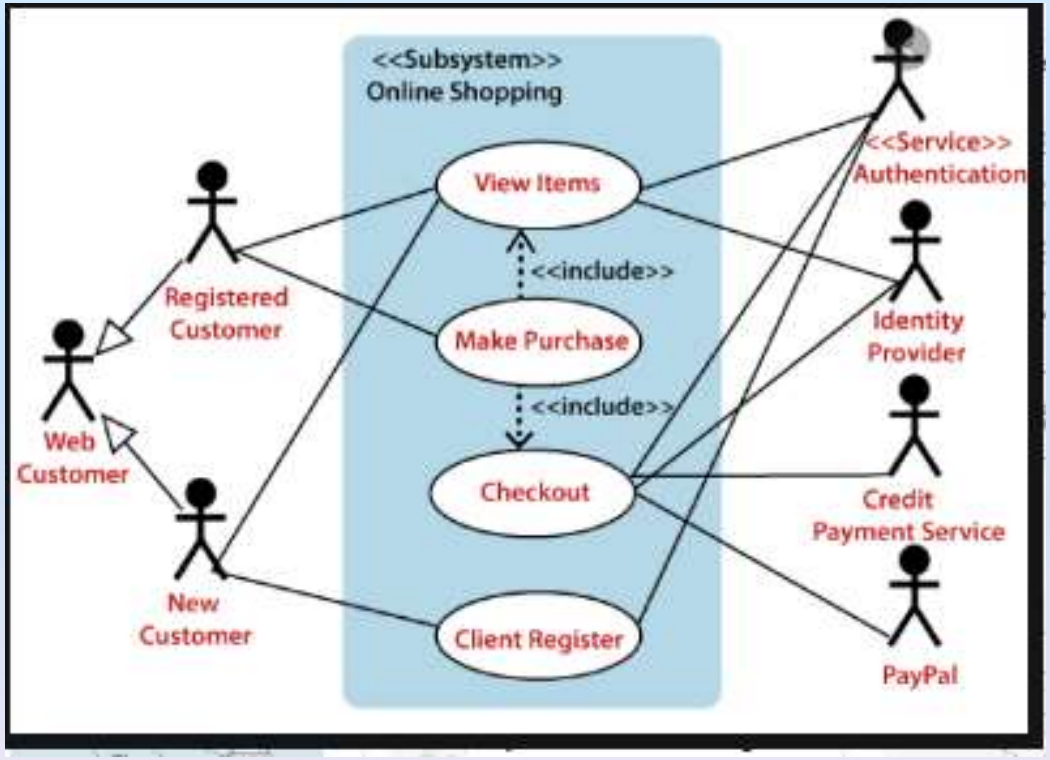
Example systems & model suitability (justify)

- Nuclear Plant Cooling System (V-Process model)
- Medical Diagnostic System (V-Process model)
- Student Info Mgmt System (Prototyping model)
- Animated computer game system (Prototyping model)
- ERP System (Agile model)
- Payroll System (Iterative waterfall model)

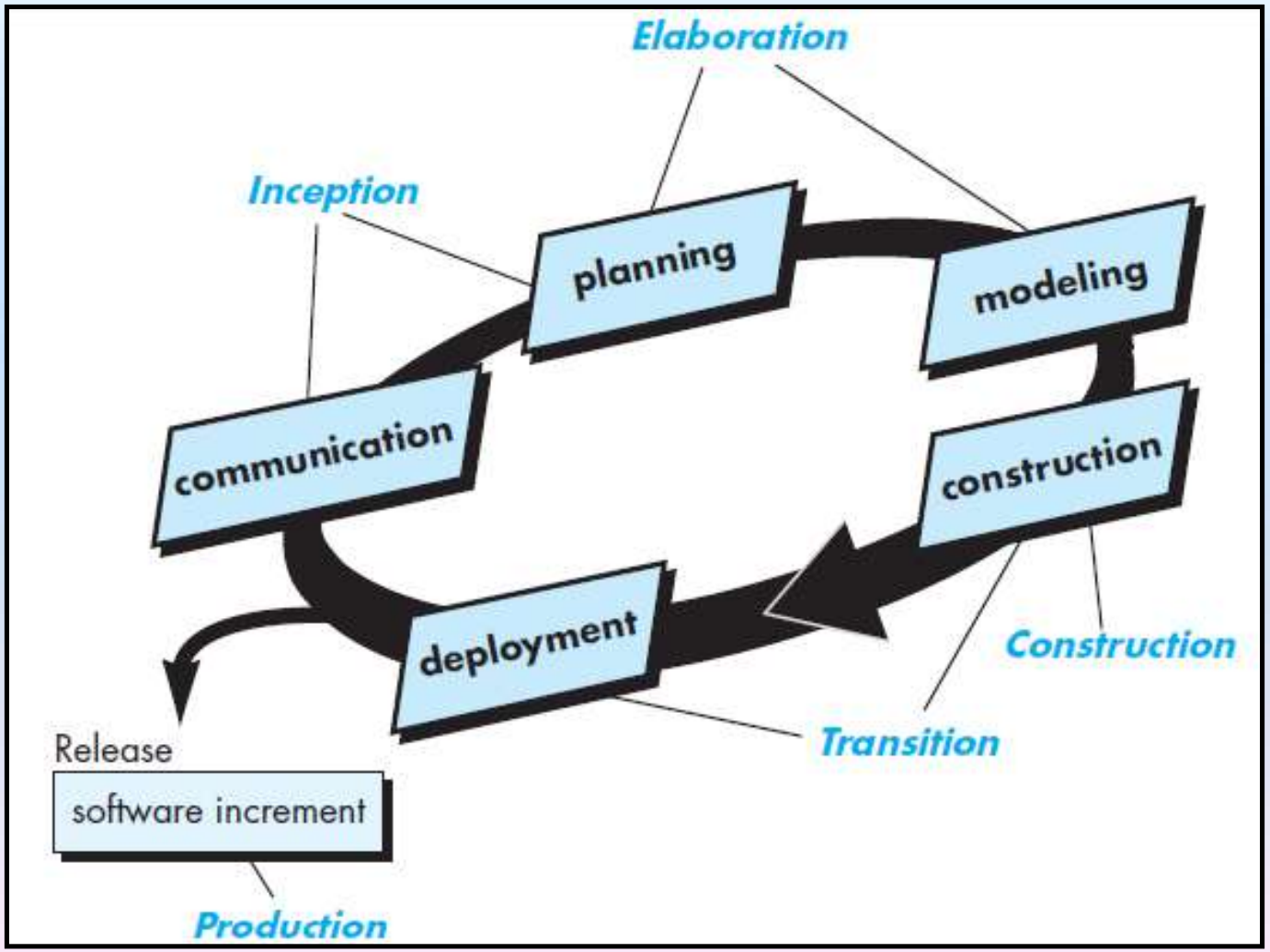
7. Unified Process

- Proposed by Rumbaugh, Booch & Jacobson in 1990's
- The **Unified Process** derives the best features of traditional s/w process models & implements many principles of agile s/w development
- It gives importance to customer communication & describing the customer's view of a system (the **use case**)
- Emphasizes the importance of **s/w architecture**

Use case diagram



Unified Process - diagram



Phases of Unified Process

1. Inception phase -

- Consists of customer communication & planning activities
- Business requirements are identified
- A rough architecture for the system proposed
- Business requirements are described using a set of use cases

2. Elaboration phase -

- Consists of the communication & modelling activities
- Refines & expands the preliminary use cases developed earlier
- Architectural representation to includes 5 different views of the s/w

  Use case, Analysis, Design, Implementation & Deployment models

Phases of Unified Process

3. Construction phase –

- The s/w components are developed using architectural model as input
- Use cases are made operational
- Features for the s/w increments are then implemented in source code
- Integration & Acceptance testing done

4. Transition phase -

- Contains latter stages of construction & 1st part of the deployment
- Software is given to end users for beta testing & defects are fixed
- Support information is created

5. Production phase -

- Ongoing use of the software is monitored
- Support for the operating env is provided
- Defect reports & requests for changes are submitted & evaluated

Comparison of Different Life Cycle Models

- **Iterative waterfall model**
  - most widely used model
  - But, suitable only for well-understood problems
- **Prototype model is suitable for projects not well understood from below aspects :**
  - user requirements
  - technical aspects

Comparison of Different Life Cycle Models

- **Evolutionary model is suitable for large problems:**
  - That can be decomposed into a set of modules that can be incrementally implemented
  - incremental delivery is acceptable to the customer

- **The spiral model:**
  - Suitable for development of **technically challenging s/w products** that are subject to several kinds of risks

★Comparison of Different Life Cycle Models

- **RAD model is suitable for** customized, non-critical & large s/w products with tight schedule & **unsuitable** for small, generic, high reliability products
  - Unlike **Prototyping** model, the prototypes are not thrown away
  - Unlike **Iterative WF** model, functionalities are developed incrementally not together with heavy code/design reuse
  - Unlike **Evolutionary** model, each increment is shorter & builds a prototype not systematic development

★Comparison of Different Life Cycle Models

- **Agile models** are **suitable for** customized, large s/w products that can be released in increments & **unsuitable** for stable requirement, mission-critical products
  - Unlike **RAD** model, developing prototypes are not recommended
  - In Agile model, only completed work is demonstrated to customer
  - Unlike **Iterative WF** model, if the project is cancelled mid-way, still some functionality is implemented
  - Progress is measured in terms of functionalities delivered

Entry and Exit criteria for a particular phase of software life-cycle

- For the **testing phase** of "Library Information System – Renew Book" module:
- **Entry criteria:**
  - All the sub-modules/programs for the "Renew Book" module are coded & unit tested successfully
  - Test suite containing all test cases for Black-Box & White-Box testing are written
- **Exit criteria:**
  - All the sub-modules/programs for the "Renew Book" module are tested by executing all test cases of the test suite
  - All defects discovered are fixed


The End

Possible questions

Q: *Suggest a suitable life-cycle model for development of a* ***ECG*** *(Electrocardiography) Monitoring System. Justify your choice.*

A: *V-process model will be suitable. Because, this system needs to be highly reliable and fault-proof. It is important to validate and test all the components thoroughly in all stages of software development.*

Justify this in detail

Q: *Suggest a suitable life-cycle model for development of a gaming application. Justify your choice.*

A: *Prototyping model will be suitable as it is important to get all requirements clearly from customer before building the actual system.*

Justify this in detail