SDLC Models

SDLC

- SDLC stands for " Software Development Life Cycle".
- It is the series of identifiable stages that a software product undergoes during its lifetime.
- SDLC model is a descriptive and diagrammatic representation of the software life cycle.
- A life cycle model represents all the activities required to make a software product transit through its life cycle phases.

Software Development Life Cycle

Software development projects are often very large projects. A number of people work on such a project for a very long time and therefore the whole process needs to be controlled; progress needs to be monitored, people and resources need to be allocated at the right point in time. Hence some software process models are used for development of quality software product. Such software development models are called Generic software process models, or software development life cycle (SDLC) models.

A framework that describes the activities performed at each stage of a software development project.

Software Development Life Cycle

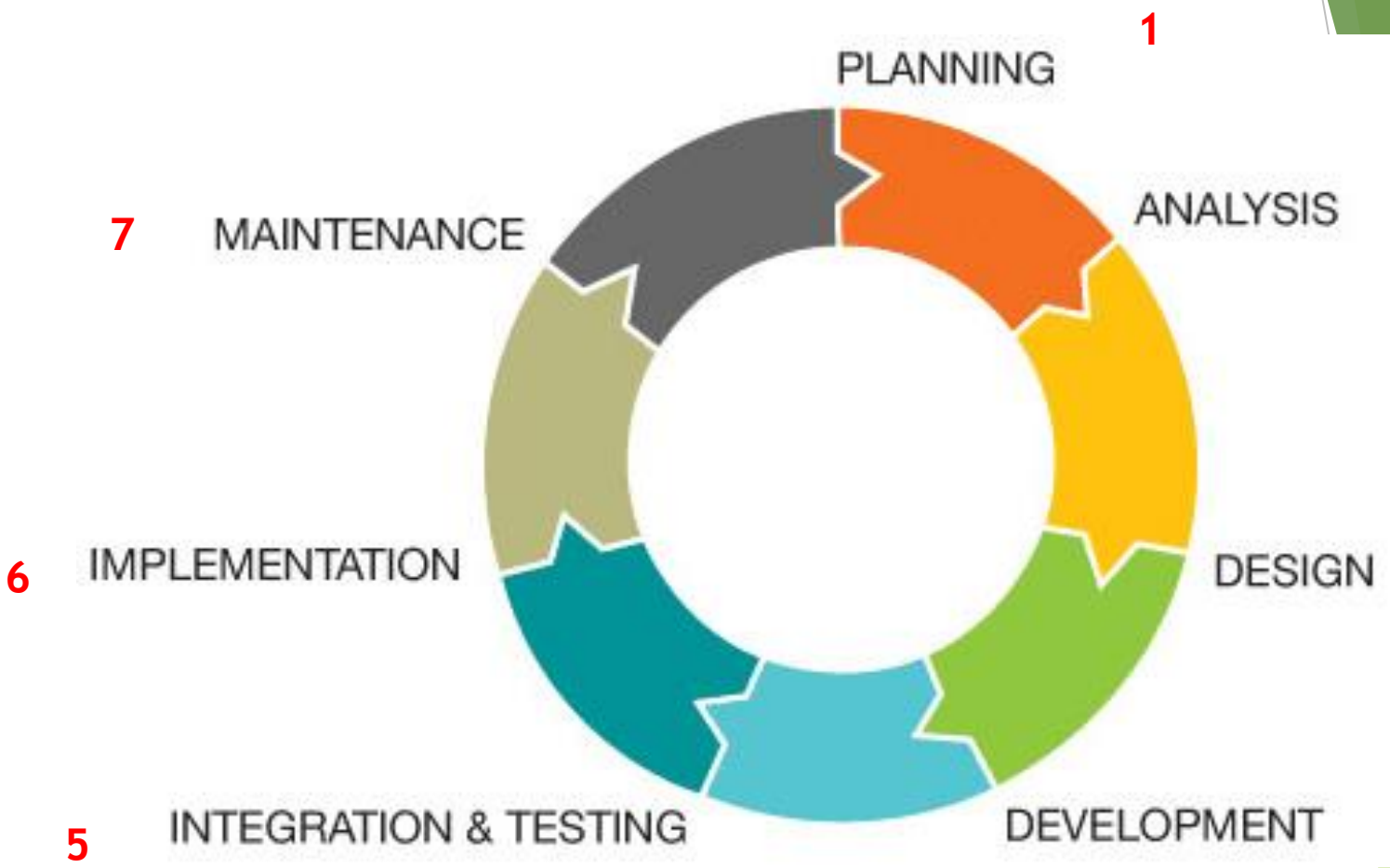
- An information system goes through a series of phases from conception to implementation.
- This process is called the Software-Development Life-Cycle. Various reasons for using a life-cycle model include:-
  - Helps to understand the entire process
  - Enforces a structured approach to development
  - Enables planning of resources in advance
  - Enables subsequent controls of them
  - Aids management to track progress of the system

Software Development Life Cycle

- Software development life-cycle can be divided into 5-9 phases:
  - Project initiation and planning/Recognition of need/Preliminary investigation
  - Project identification and selection/Feasibility study
  - Project analysis
  - System design
  - Coding
  - Testing
  - Implementation
  - Maintenance

Outputs of SDLC Phases

SDLC Phase	Input	Output
Requirement gathering /elicitation/Initial phase	Customer or user requirement	URS (user requirement specification)
Analysis phase	URS document	SRS (Software Requirement Specification)
Designing Phase	SRS document	TDD(Technical design document) [UML diagrams etc.]
Coding phase	TDD	Developed software , Abstract code
Testing Phase	Developed software(Build),Test Case,Test Environment.	Quality Product,Test Deliverables,Test Reports, modification/change request.
Implementation	Quality product,Tested and build module / application / solution.	Real time Issues, Application live for User Acceptance Testing and ensure its running with out bug and errors, Operational System
Maintenance	Operational system	working business solution



Software Development Life Cycle (SDLC)

Software Development Life Cycle

Background on software process models

- 1950 ：Code-and-fix model
- 1956 ： Stage-wise model (Bengington)
- 1970 ： Waterfall model (Royce)
- 1971 ： Incremental model(Mills)
- 1977 ： Prototyping model(Bally)
- 1988 ： Spiral model(Boehm)

Generic S/W Process Models-

Generic software process models are

- The Waterfall Model – Separate and distinct phases of specification and development.
- Prototyping Model – A quick design approach is adopted.
- Incremental Models – It emphasizes on short development cycle.
  - Rapid Application and Development (RAD) Model
- Evolutionary Process Models – specification, development and validation are interleaved.
  - Incremental Model
  - Spiral Model
  - WINWIN spiral model
  - Concurrent Development

SDLC Models

- Waterfall Model
- Prototype Model
- Spiral Model
- Evolutionary Development Model
- Iterative Enhancement Model
- RAD Model
- Agile Model

1. Classical Waterfall Model

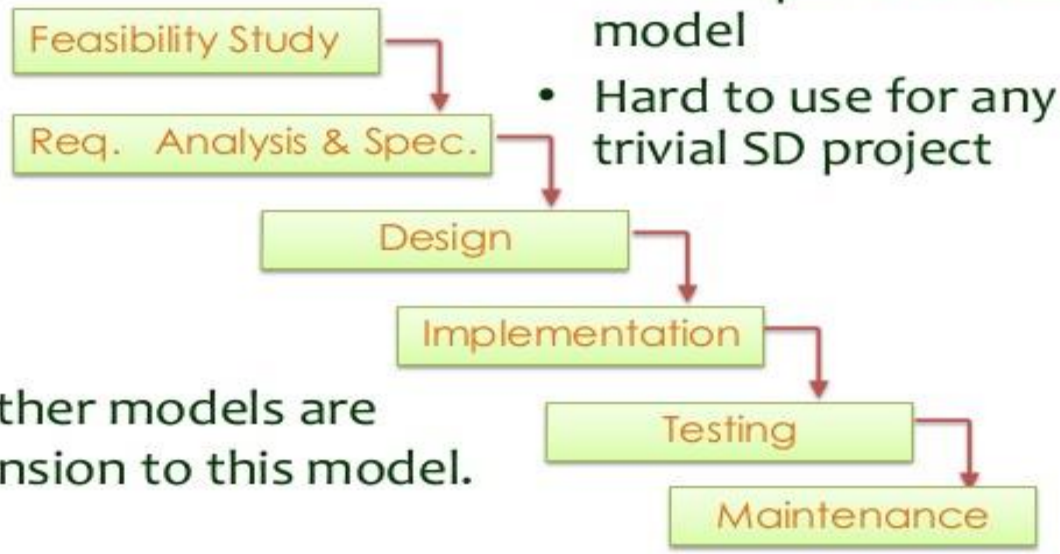
This model is easy to understand and reinforces the notion of "define before design" and "design before code".

The model expects complete & accurate requirements early in the process, which is unrealistic

Classical Waterfall Model(Cont..)

Classical Waterfall Model

- Classical waterfall model divides life cycle into phases:
  - It is simple and idealistic model
  - Hard to use for any non-trivial SD project



- All other models are extension to this model.

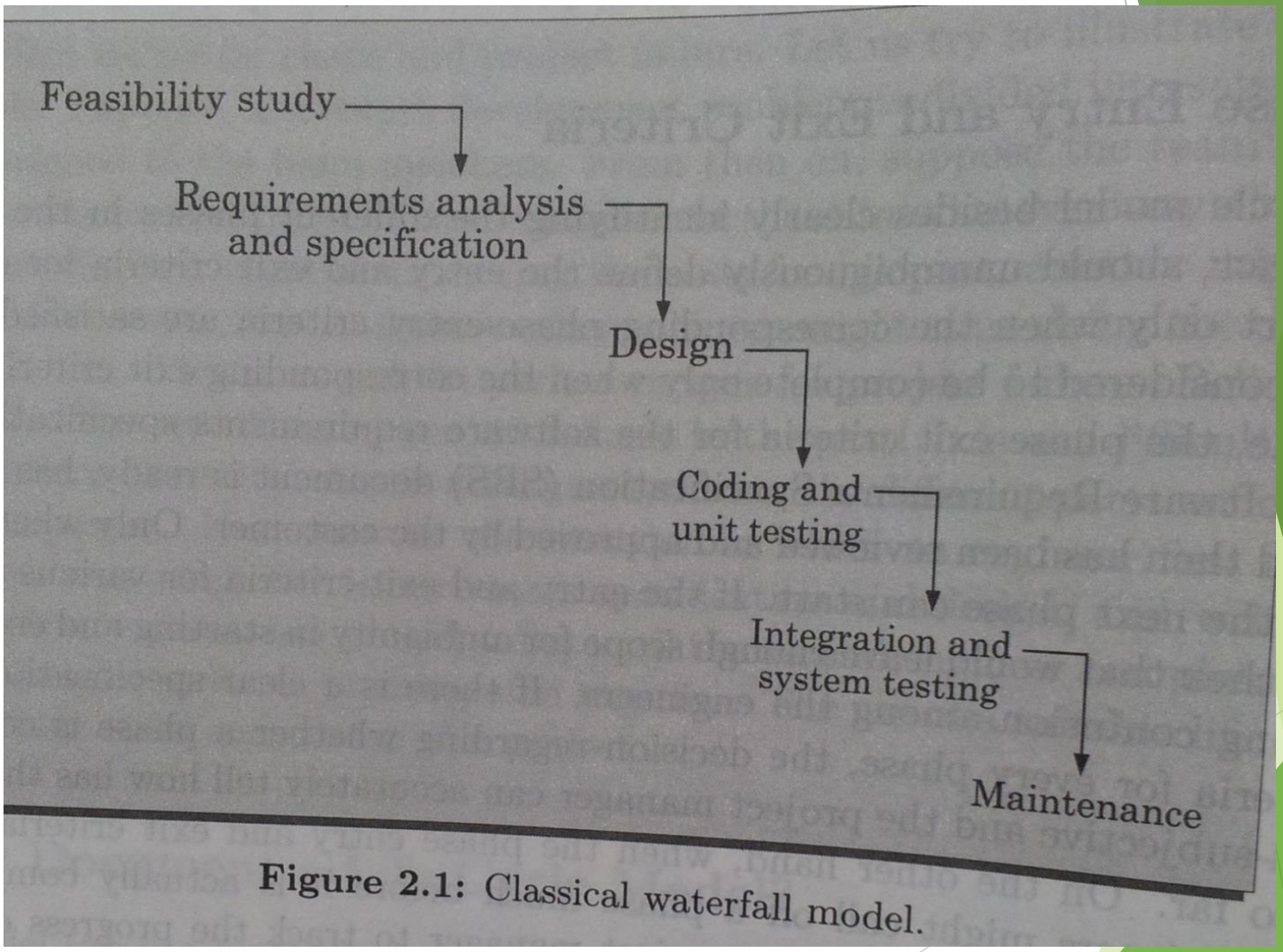


Figure 2.1: Classical waterfall model.

Phases of Waterfall Model(Cont..)

- Feasibility study: checking of financially and technical possibility to develop the product.
- Requirement Analysis and Specification: The aim of this phase is to understand the exact requirement of the customer and to document them properly.
- Requirement Gathering and Analysis: This phase is done to get a clear understanding of the requirements and to remove the inconsistencies and incompleteness in the requirements.
- Requirement Specification: The requirement which are gathered are now converted into a specification document known as **SOFTWARE REQUIREMENT SPECIFICATION(SRS).**

Phases of Waterfall Model(Cont..)

- Design: Transform the SRS Document into a structure that can be implemented into some programming language.
- Two different approaches

1. Procedural Design Approach- based on Data Flow oriented design -SA and SD
2. Object –oriented design- Objects are identified in the problem and solution domain and also identified the relationship among the objects.

Phases of Waterfall Model(Cont..)

- Coding and Unit Testing: Translate software design into a source code

This phase is also known as implementation phase.

- Integration and System Testing: Integration of different modules are undertaken and system testing perform.
- System testing consist of:

Alpha testing

Beta testing

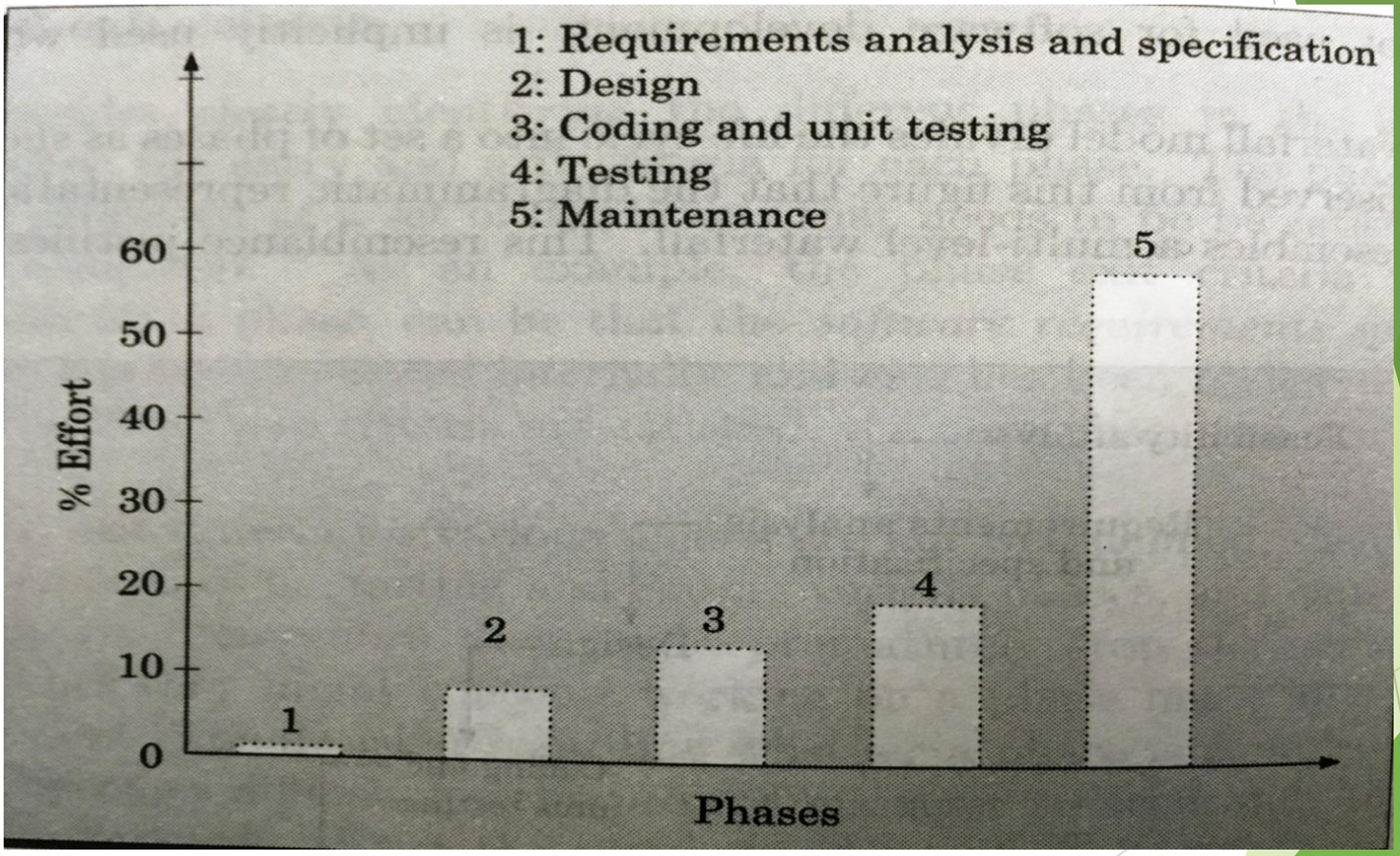
Acceptance testing

Phases of Waterfall Model(Cont..)

- Maintenance: last phase of SDLC
- It starts when the software is delivered to the user or launched in the market.
- Types

1. Corrective maintenance
2. Perfective Maintenance
3. Adaptive Maintenance

The Effort estimation of Waterfall Model



Advantages

- Simple and easy to understand
- Easy to manage due to rigidity of requirements
- Phases are complete one at a time
- Estimation activity is less complex
- Suitable for small projects

Waterfall Model (cont...)

Problems of waterfall model

i. It is difficult to define all requirements at the beginning of a project

ii. This model is not suitable for accommodating any change

iii. A working version of the system is not seen until late in the project's life

iv. It does not scale up well to large projects.

v. Real projects are rarely sequential.

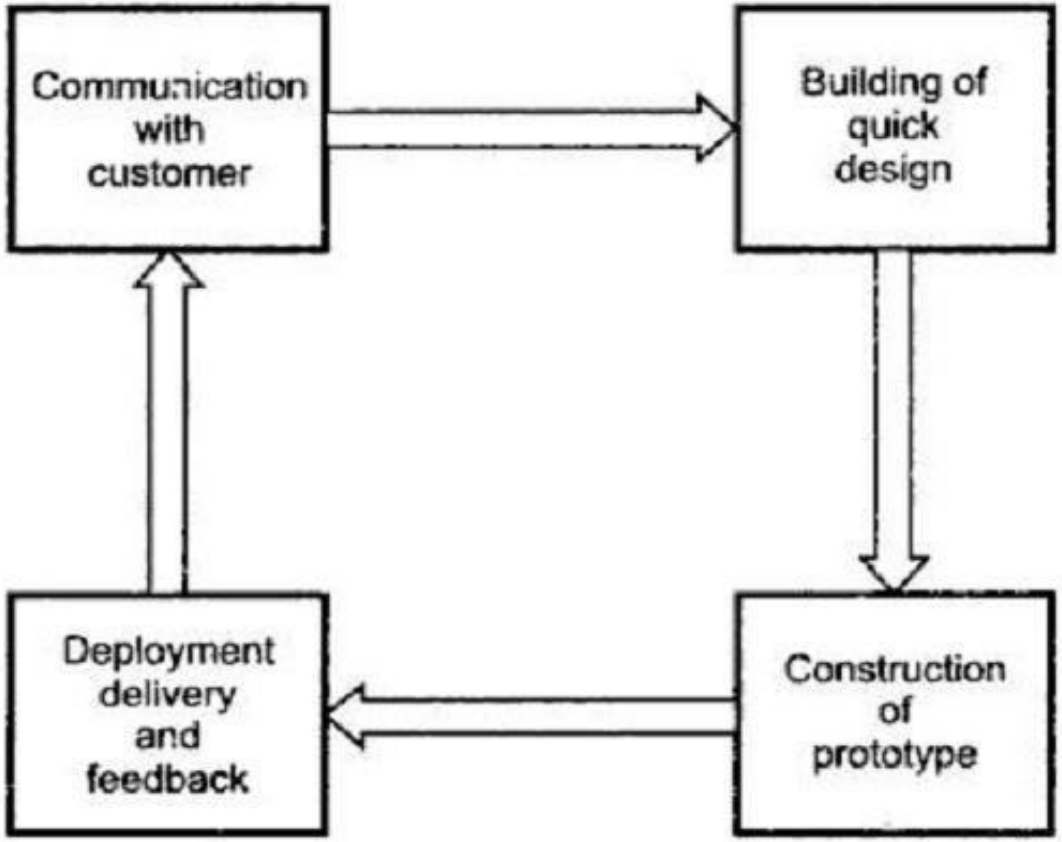
When to use the Waterfall Model

- **Requirements are very well known.**
- **When it is possible to produce a stable design.**
- **a new version of an existing product.**
- **porting an existing product to a new platform.**

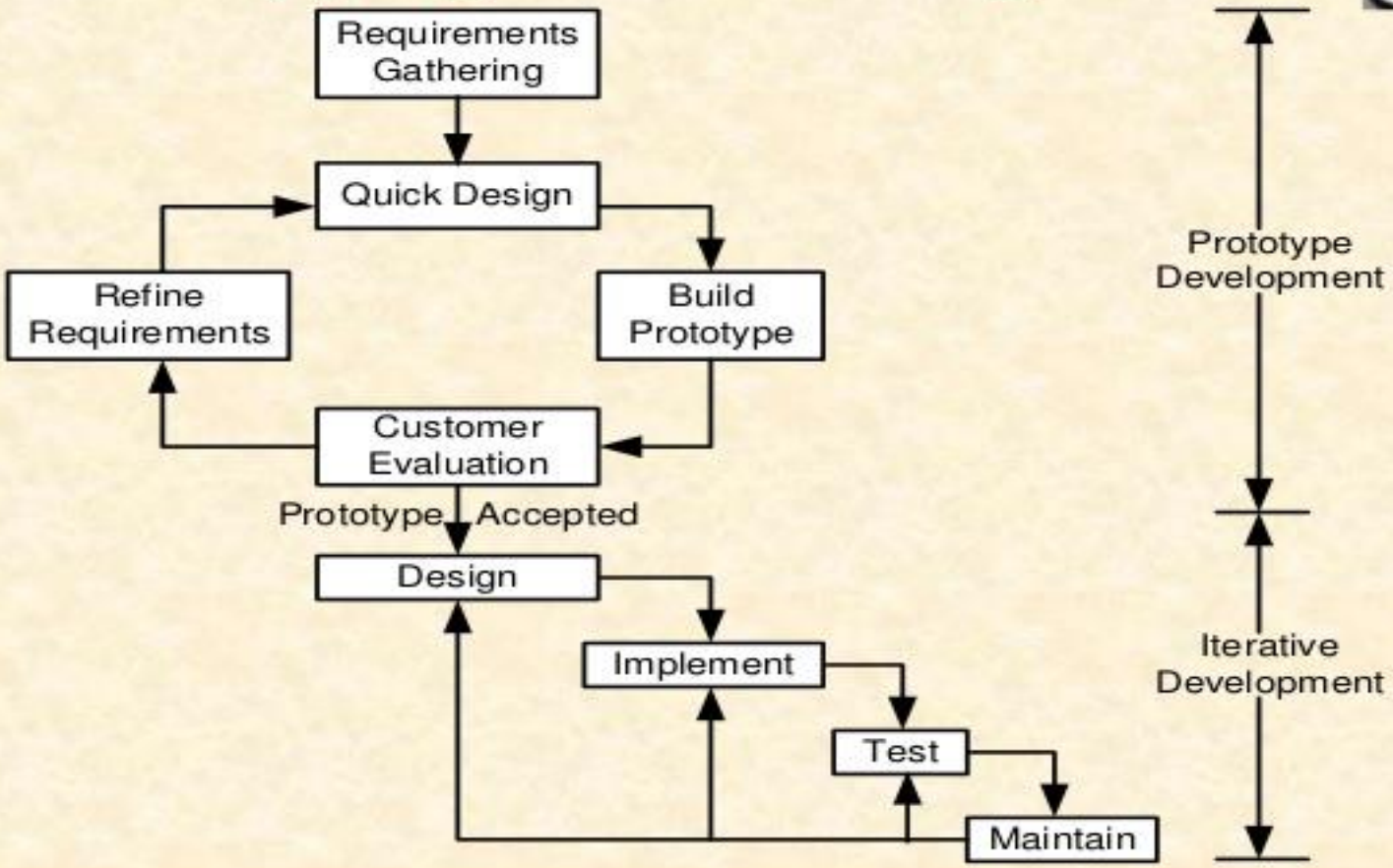
2. Prototype Model

- In prototyping model initially the requirement gathering is done.
- Developer & customer define overall objectives; identify areas needing more requirement gathering.
- Then a quick design is prepared. This design represents what will be visible to user- in input and output format.
- From the quick design a prototype is prepared. Customer or user evaluates the prototype in order to refine the requirements. Iteratively prototype is tuned for satisfying customer requirements. Thus prototype is important to identify the software requirements.
- When working prototype is built, developer use existing program fragments or program generators to throw away the prototype and rebuild the system to high quality.

Prototype Model



Prototyping Model: Figure



Advantages

The advantages of the Prototyping Model are as follows –

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

Disadvantages

The Disadvantages of the Prototyping Model are as follows –

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

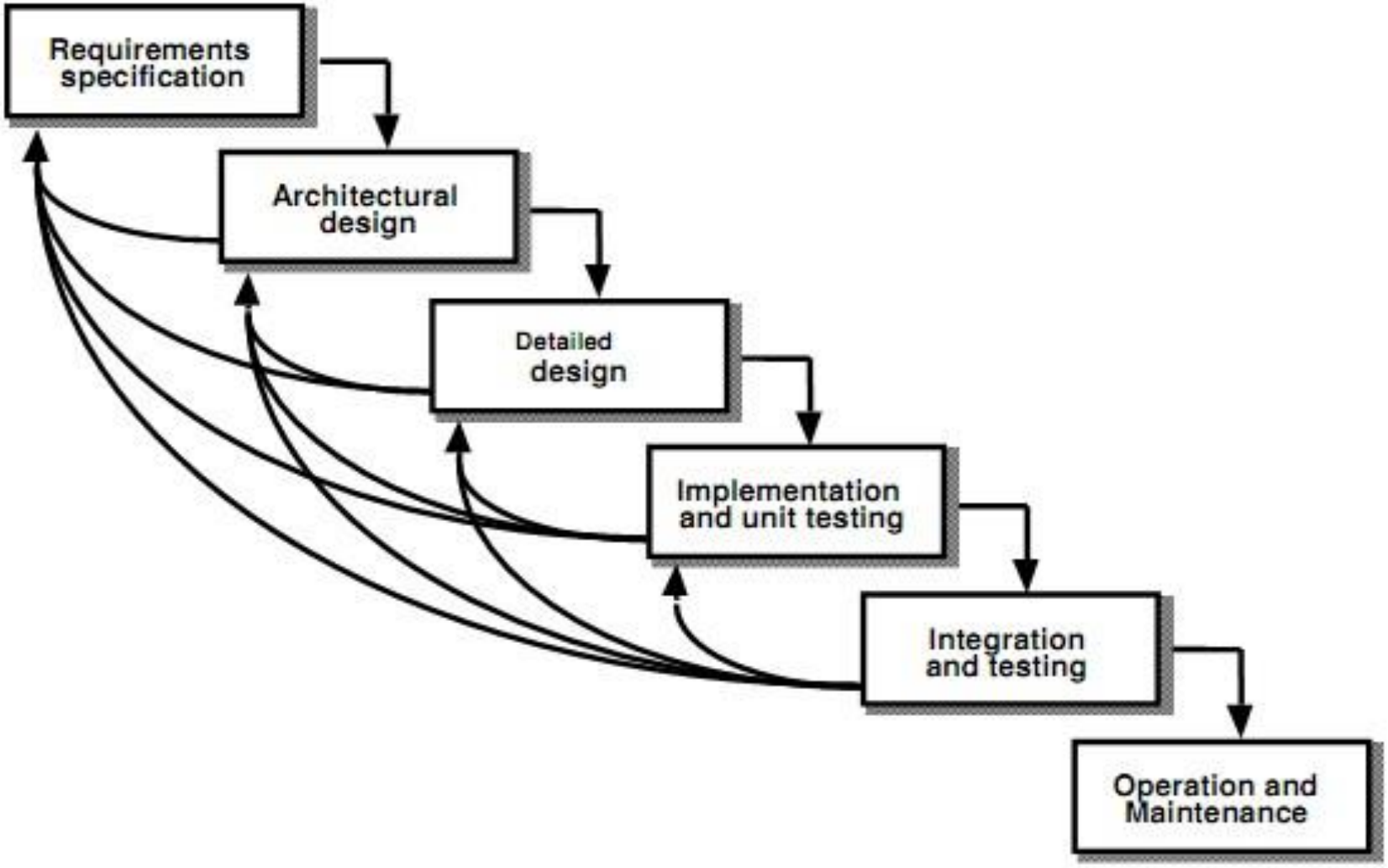
When to Use…..

- Requirements are unstable
- Short lived demonstration required
- when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

3. Iterative Enhancement Model

- This model has the **same phases as the waterfall model**, but with **fewer restrictions**.
- Generally the phases occur in the same order as in the waterfall model, but they may be **conducted in several cycles.**
- **Useable product** is **released at the end of the each cycle**, with **each release providing additional functionality.**

Iterative Enhancement Model

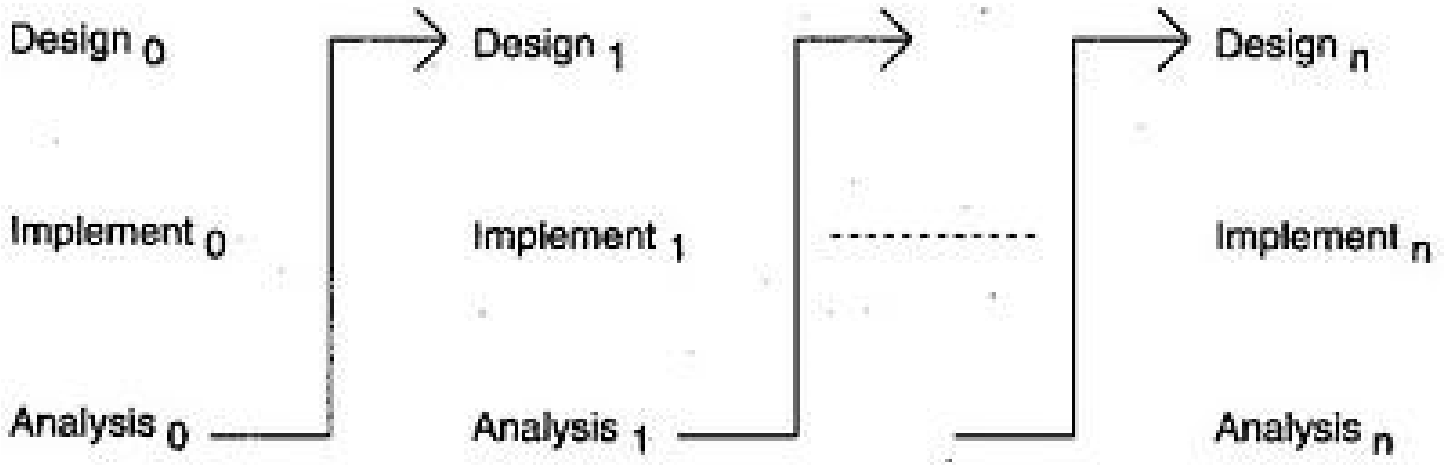


Iterative Enhancement Model(Incremental Model)

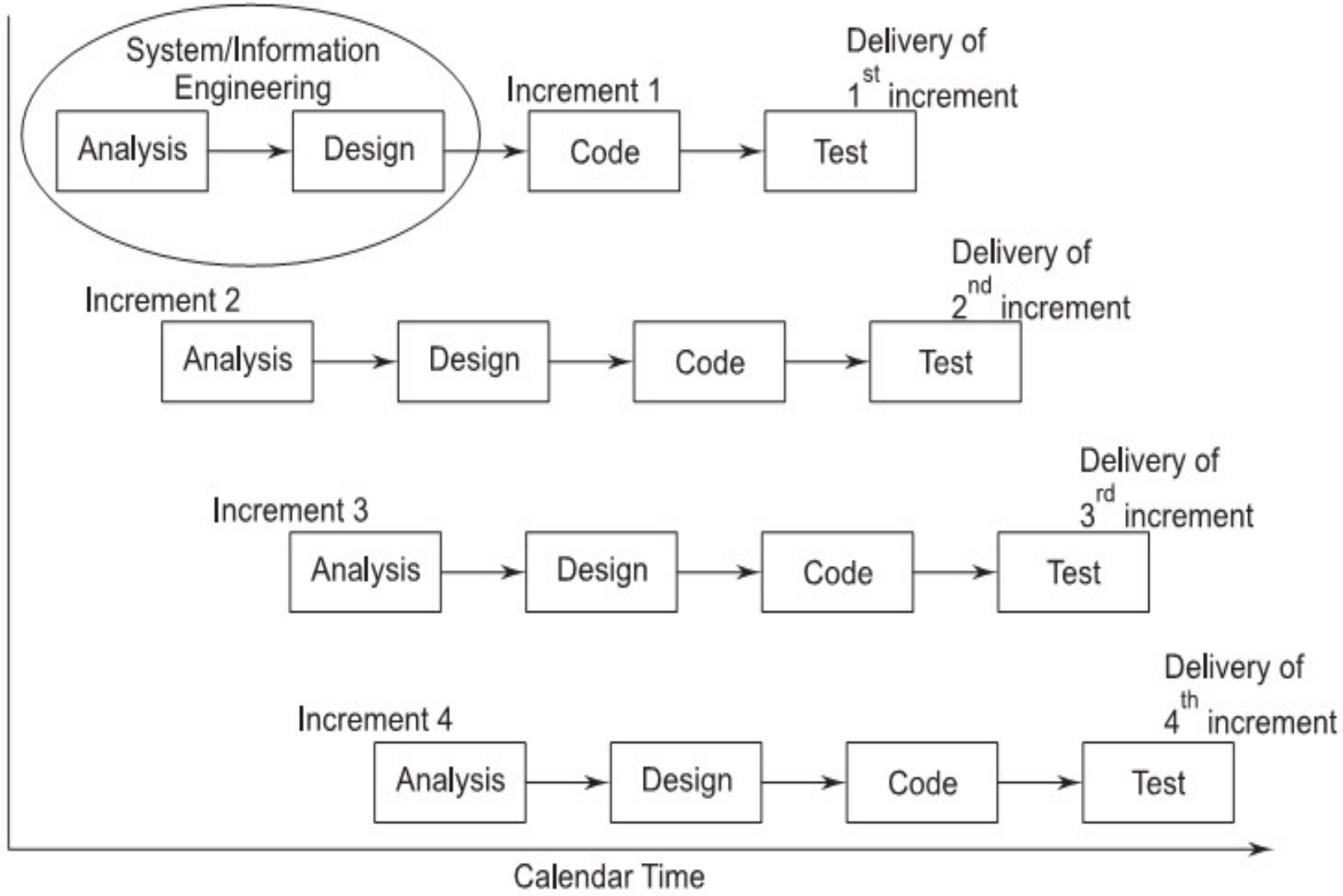
- The incremental model has same phases that are in waterfall model. But it is iterative in nature. The incremental model has following phases.

  1. Analysis
  2. Design
  3. Code
  4. Test

- The incremental model delivers series of releases to the customer. These releases are called increments. More and more functionality is associated with each increment.
- The first increment is called core product. In this release the basic requirements are implemented and then in subsequent increments new requirements are added.
- The word processing software package can be considered as an example of incremental model. In the first increment only the document processing facilities are available. In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given. In the next increment spelling and grammar checking facilities can be given. Thus in incremental model progressive functionalities are obtained with each release.


Design 0
Implement 0
Analysis 0
Design 1
Implement 1
Analysis 1
Design n
Implement n
Analysis n

Iterative Enhancement Model(Incremental Model)



Iterative Enhancement Model(Incremental Model)

When to choose it?

1. When requirements are reasonably well-defined.
2. When overall scope of the development effort suggests a purely linear effort.
3. When limited set of software functionality needed quickly.

Merits of incremental model

1. The incremental model can be adopted when there are less number of people involved in the project.
2. Technical risks can be managed with each increment.
3. For a very small time span, at least core product can be delivered to the customer.

The advantages of the Iterative and Incremental SDLC Model are as follows –

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.

Demerits of Iterative model:

The disadvantages of the Iterative and Incremental SDLC Model are as follows –

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.

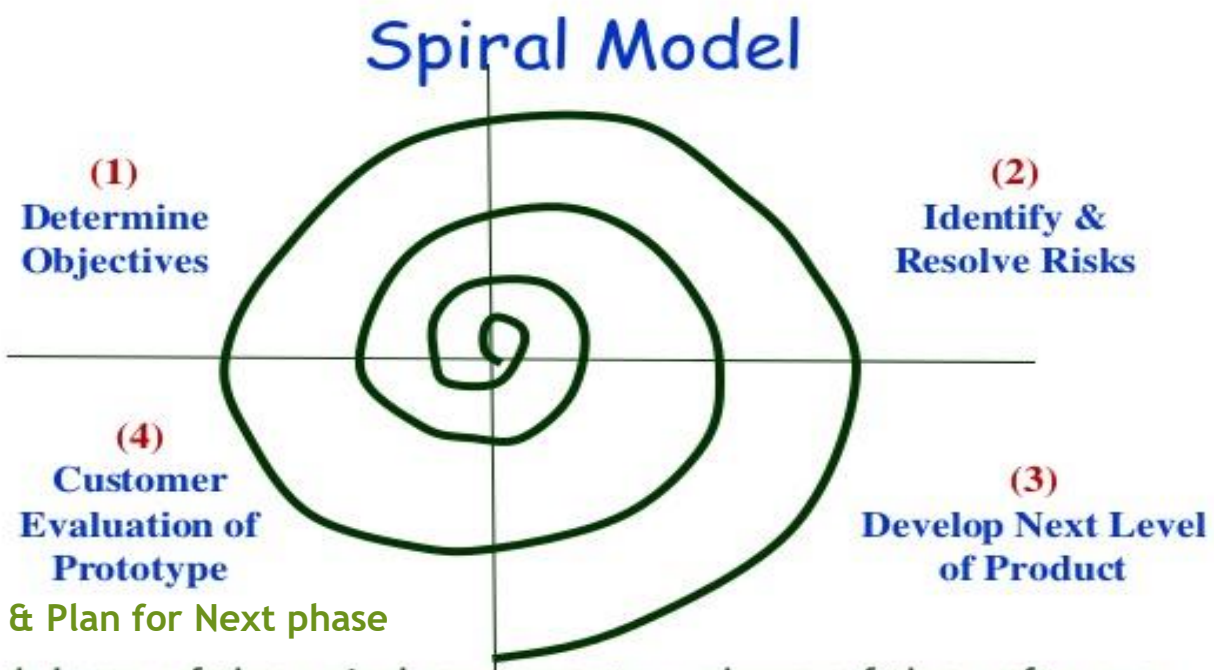
- Each phase of an iteration is rigid with no overlaps

- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

4. Spiral Model

- Thespiral model is intended for large, expensive and complicated projects.
- The spiral model focuses on identifying and eliminating high-risk problems by careful process design.
- Boehm defines risk management as a discipline whose objectives are to-
  - identify, address, and eliminate software risk items before they become either threats to successful software operation or a major source of expensive software rework

Spiral Model



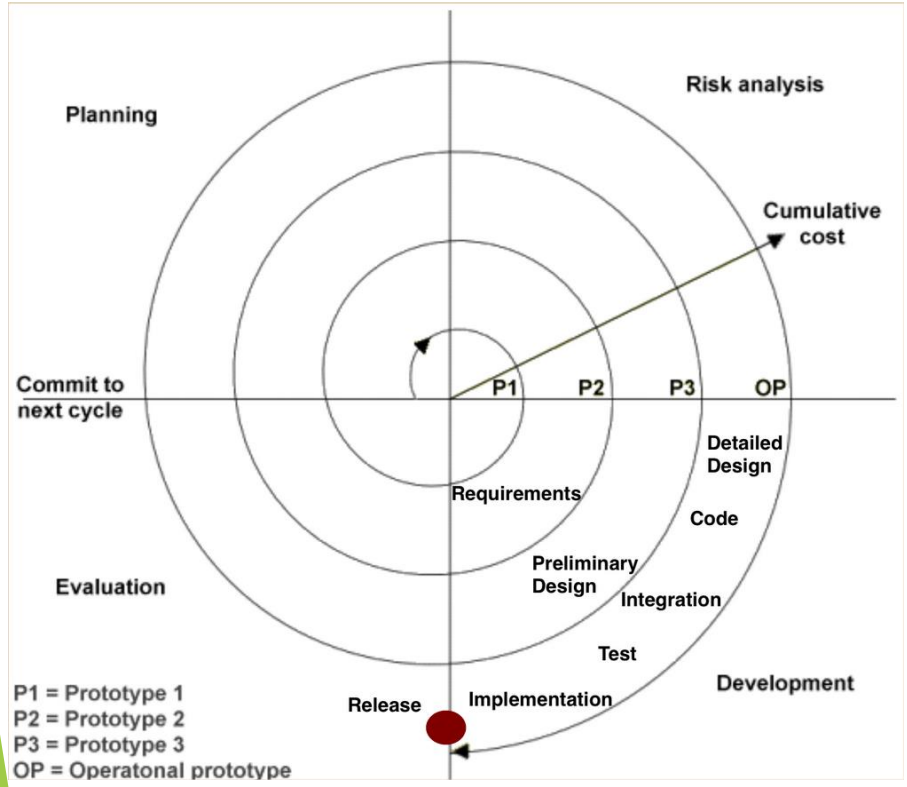
- Each loop of the spiral represents a phase of the software process:
  - the innermost loop might be concerned with system feasibility,
  - the next loop with system requirements definition,
  - the next one with system design, and so on.

Each phase in this model is split into four sectors (or quadrants)

- The **first quadrant** identifies the objectives of the phase and the alternative solutions possible for the phase under consideration.
- In the **second quadrant** we evaluate different alternatives based on the objectives and constraints. The evaluation is based on the risk perceptions for the project.

Each phase in this model is split into four sectors (or quadrants)

- The **third quadrant** is the next step that emphasizes development of strategies that resolve the uncertainties and risks. This may involve activities, such as benchmarking, simulation, and prototyping.
- In the last or **fourth quadrant** we determine the objective that should be fulfilled in the next cycle of our software development in order to build the complete system.



Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Development Phase: In this phase software is **developed**, along with **testing** at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project

Advantages

The advantages of the Spiral SDLC Model are as follows –

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

Disadvantages

The disadvantages of the Spiral SDLC Model are as follows –

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

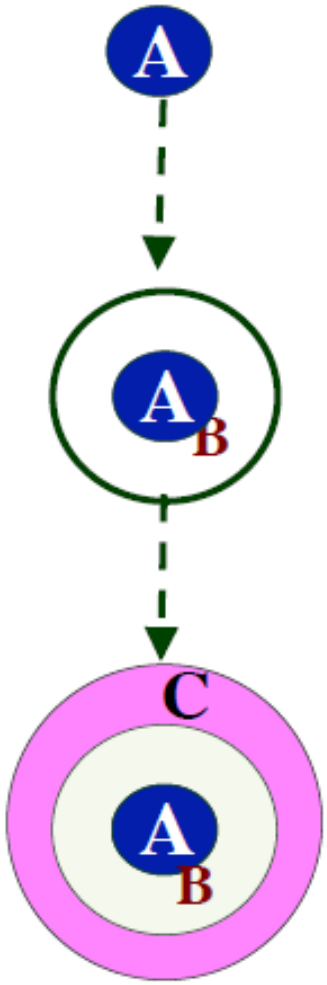
When to Use Spiral Model….

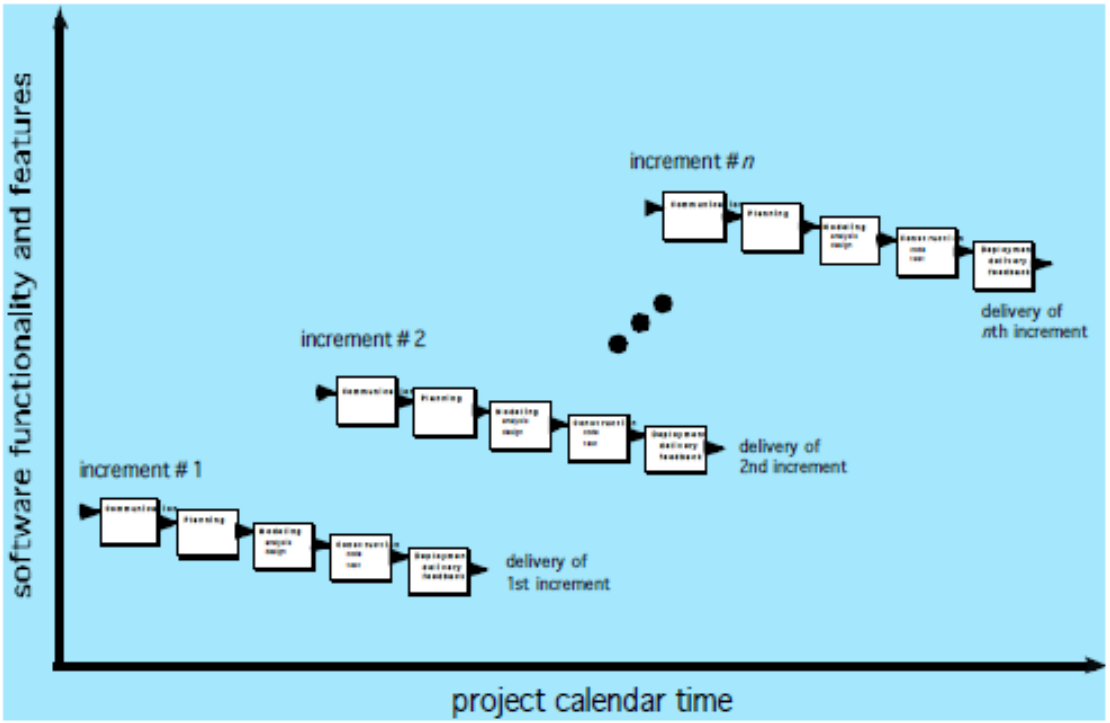
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

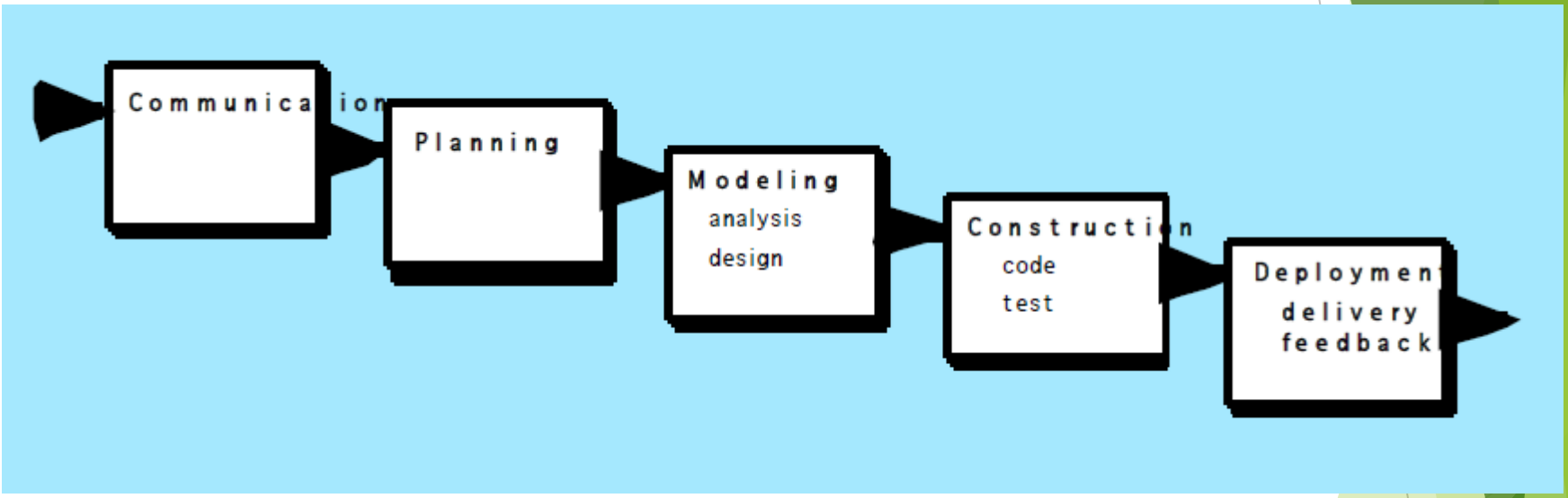
5. Evolutionary/Incremental Development Model

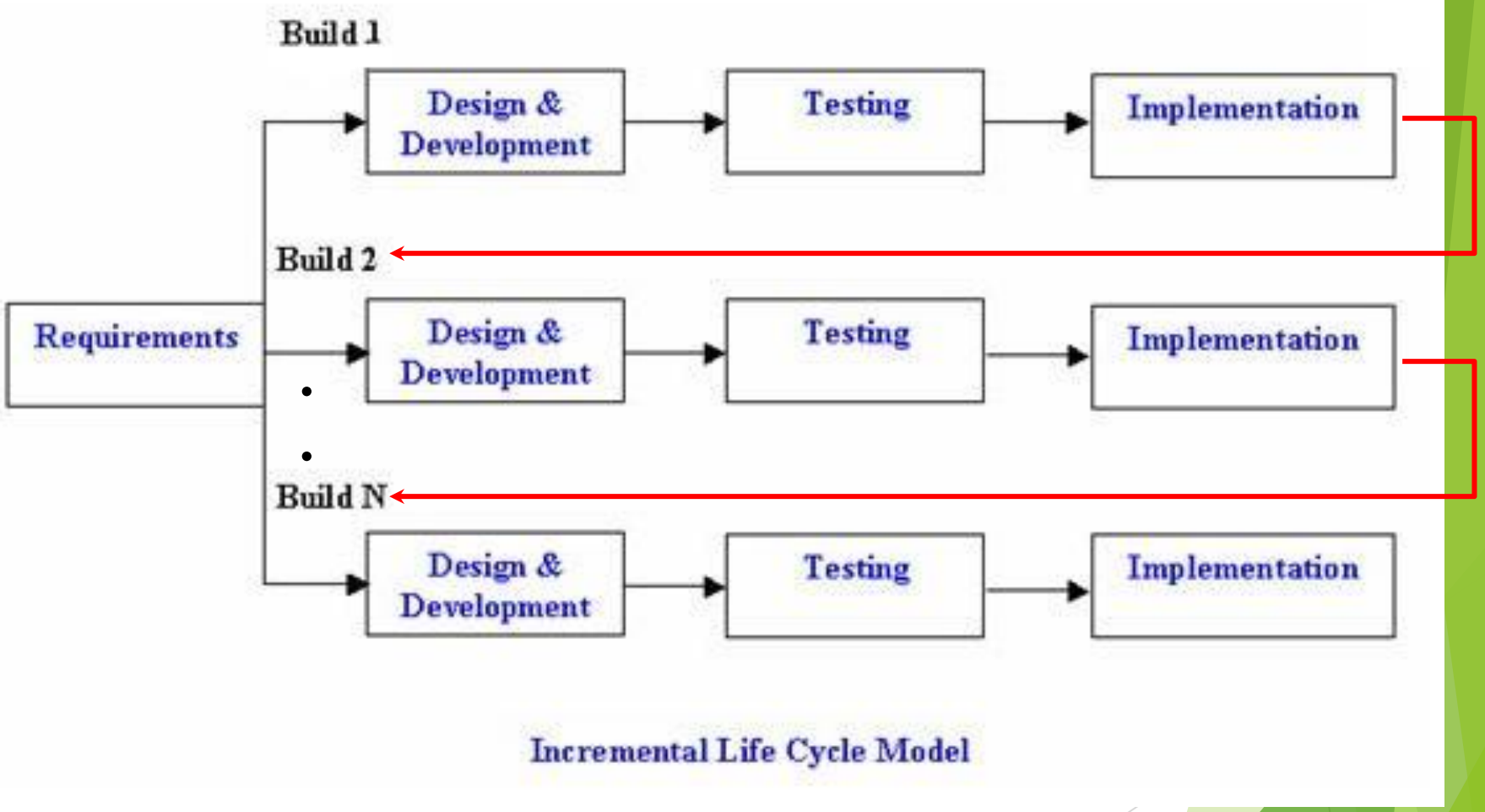
- Evolutionary model (aka successive versions or incremental model):
  - > The system is broken down into several modules which can be incrementally implemented and delivered.
- First develop the core modules of the system.
- The initial product skeleton is refined into increasing levels of capability:
  - > by adding new functionalities in successive versions.

Incremental Model






Communication
Planning
Modeling
analysis
design
Construction
code
test
Deployment
delivery
feedback

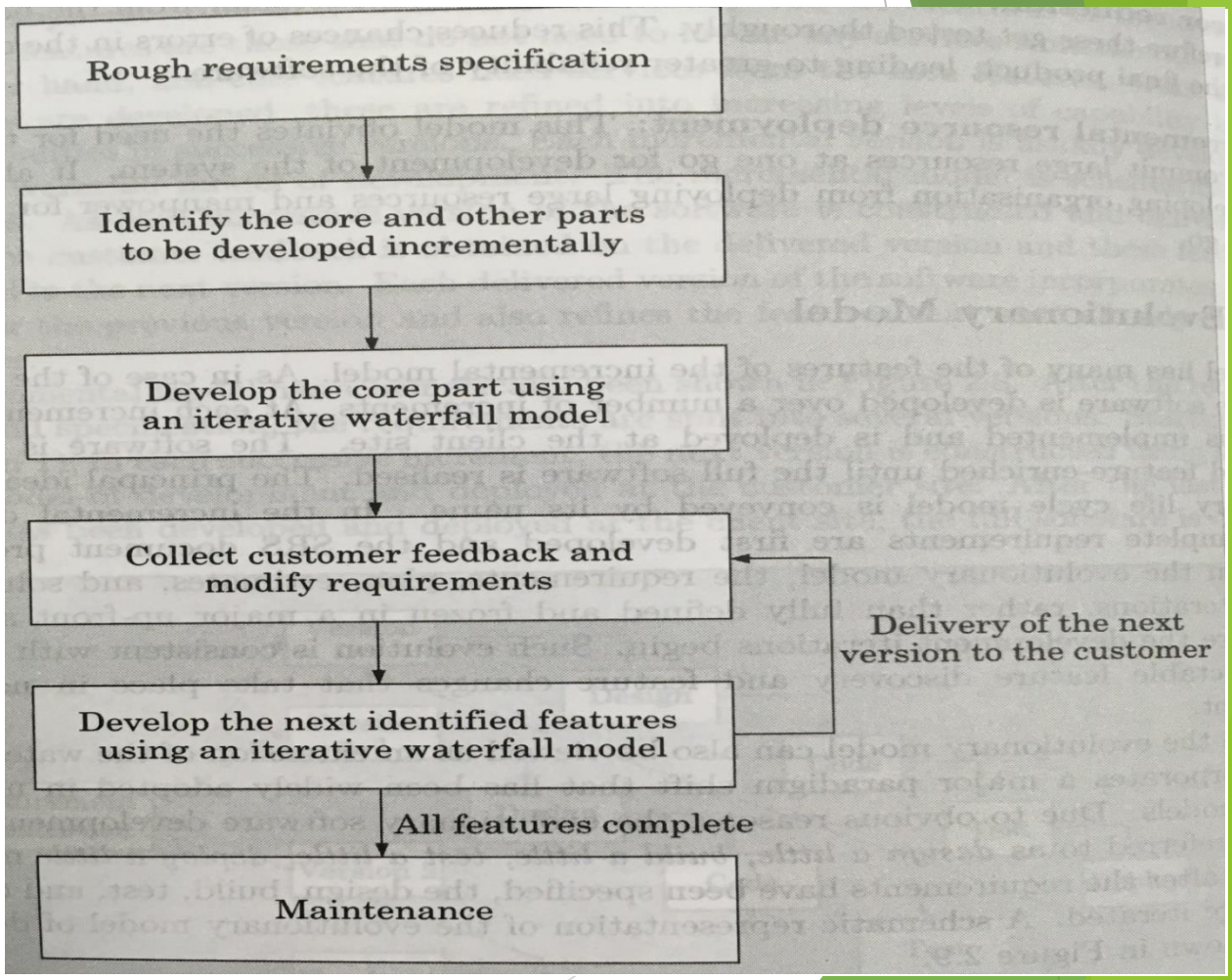


Incremental Life Cycle Model

Evolutionary Development Model

1. Contains many features of incremental model

2. Sometimes referred to as ***design a little***, ***build a little***, ***test a little***, ***deploy a little*** model.



Advantages

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it’s iteration.
- Error reduction
- Incremental resource deployment

Disadvantages

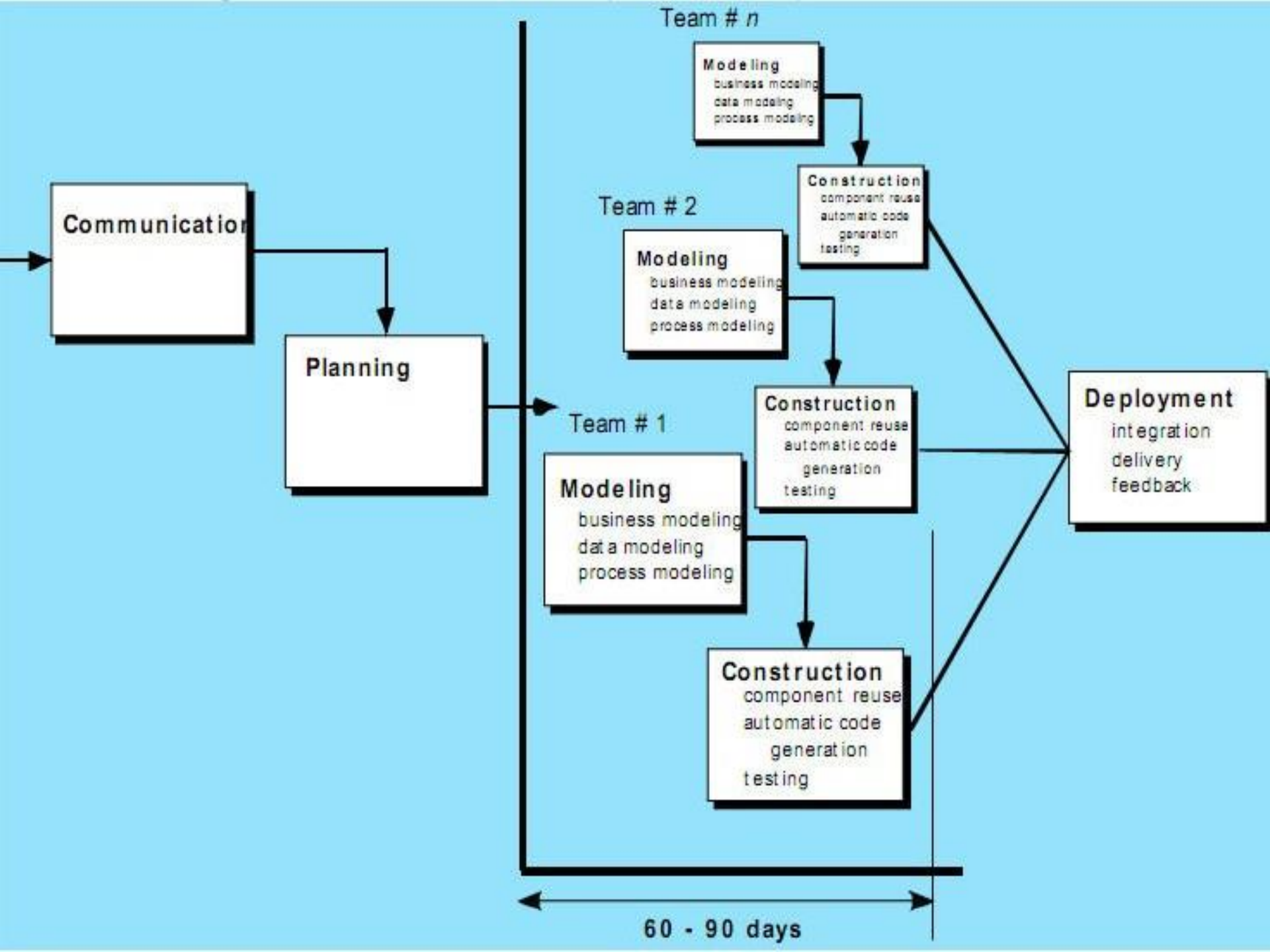
- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than **waterfall**.
- Ad hoc design
- Feature division into incremental parts can be non-trivial

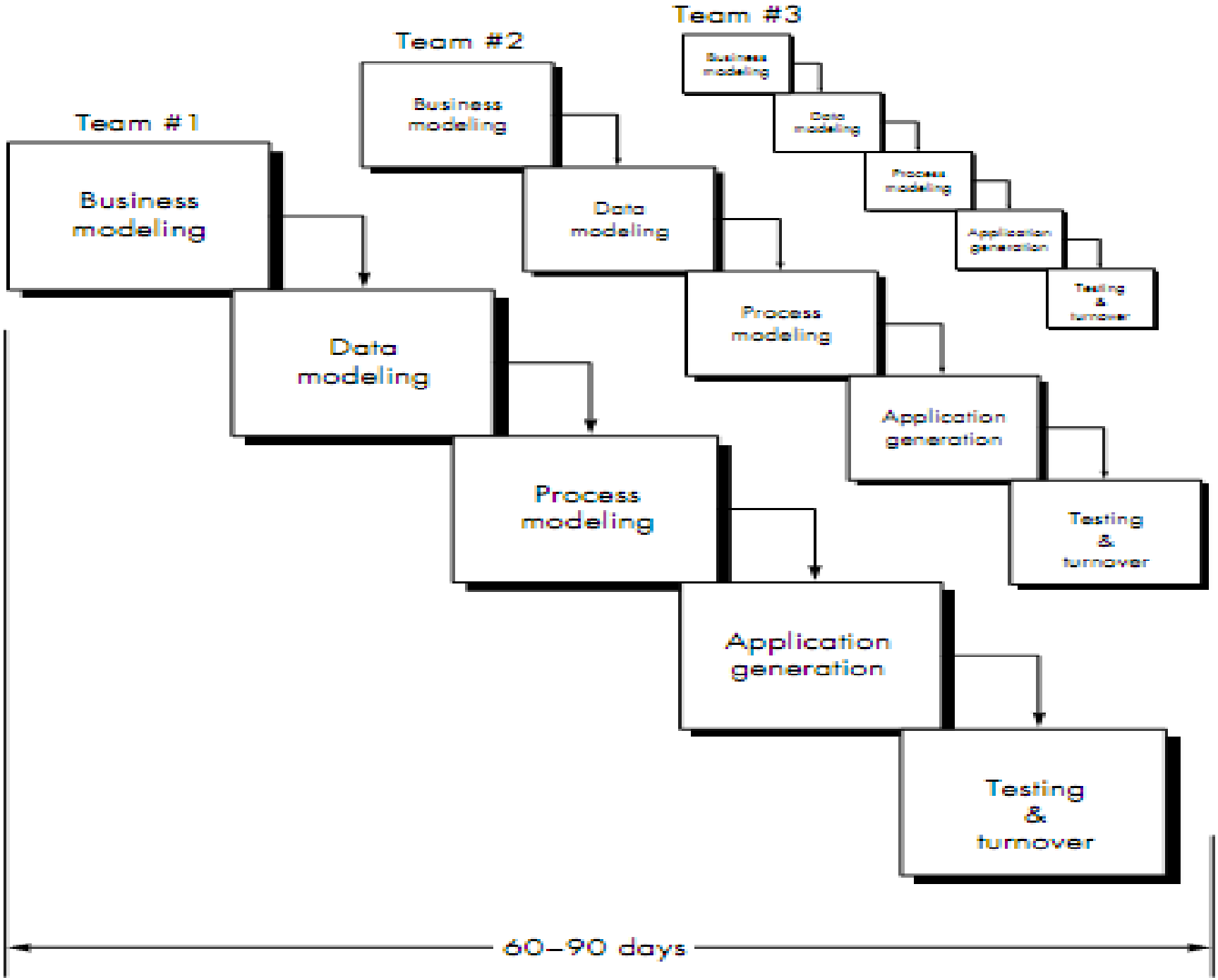
When to use

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.
- **The evolutionary model is well-suited to use in Object-oriented software development projects.**

6.Rapid Application Development(RAD)Model

- Rapid Application Development is a **linear sequential software development process model** that emphasizes an extremely **short development cycle**.
- Rapid application achieved by using a **Component Based Construction** approach.
- If requirements are **well understood and project scope is constrained** the **RAD process enables** a development team to create a **—”fully functional system”** within very short time periods (**e.g., 60 to 90 days**).


Team # n
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code generation
testing
Team # 2
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code generation
testing
Communication
Planning
Team # 1
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code generation
testing
Deployment
integration
delivery
feedback
60 - 90 days


Team #1
Business modeling
Data modeling
Process modeling
Application generation
Testing & turnover
Team #2
Business modeling
Data modeling
Process modeling
Application generation
Testing & turnover
Team #3
Business modeling
Data modeling
Process modeling
Application generation
Testing & turnover
60–90 days

RAD Phases :

- Business modeling.
- Data modeling.
- Process modeling.
- Application generation.
- Testing and turnover.

RAD Phases involve:

- **Business Modeling:**

  The **information flow among business functions** is modeled in a way that answers the following questions-

  - Where does information come from and go ?
  - Who processes it ?
  - What information drives the business process?
  - What information is generated ?

- **Data Modeling:**

  The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business.

  The characteristics (**called attributes**) of each object are identified and the relationships between these objects are defined.

RAD Phases involve:

- **Process modeling:**

  The **data modeling phase are transformed to achieve the information flow** necessary to **implement a business function.**

  To add, modify, delete, or retain a data object, there is a **need for description** which is done in this phase.

- **Application generation:**

  RAD assumes the use of 4 generation techniques.

  Rather than creating software using conventional 3 generation programming languages, the **RAD process works to reuse existing program components** (when possible) or created reusable components (when necessary).

RAD Phases involve:

- **Testing and Turnover:**

  Since the **RAD process emphasizes reuse**, many of the program components have already been tested.

  So we use some these **already existing programs** which are already tested.

  This **reduces over all testing time**.

  However, **new components must be tested** and all interfaces must be **fully exercised**.

Advantages & Disadvantages of RAD:

- **Advantages-**

1. Reduced **development time.**
2. Uses **component-based construction** and emphasizes **reuse** and **code generation.(**Increases reusability of components)
3. Quick initial reviews occur
4. Encourages customer feedback
5. Integration from very beginning solves a lot of **integration issues**.

- **Disadvantages-**

- For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- Requires strong commitment between developers and customers for “rapid-fire” activities
- High performance requirements maybe can’t meet (requires tuning the components).
- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD **SDLC model** should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).
- **For customized software**
- **For Non-critical software**
- **For highly constraint project schedule**
- **For large software**

Selection of a Life Cycle Model

- **Selection of a model is based on:**
  - Requirements.
  - Development team.
  - Users.
  - Project type and associated risk.

Based On Characteristics Of

Requirements	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Are requirements easily understandable and defined?	Yes	No	No	No	No	Yes
Do we change requirements quite often?	No	Yes	No	No	Yes	No
Can we define requirements early in the cycle?	Yes	No	Yes	Yes	No	Yes
Requirements are indicating a complex system to be built	No	Yes	Yes	Yes	Yes	No

Based On Status Of Development Team

Development team	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Less experience on similar projects?	No	Yes	No	No	Yes	No
Less domain knowledge (new to the technology)	Yes	No	Yes	Yes	Yes	No
Less experience on tools to be used	Yes	No	No	No	Yes	No
Availability of training if required	No	No	Yes	Yes	No	Yes

Based On User's Participation

Involvement of Users	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
User involvement in all phases	No	Yes	No	No	No	Yes
Limited user participation	Yes	No	Yes	Yes	Yes	No
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No
Users are experts of problem domain	No	Yes	Yes	Yes	No	Yes

Based On Type Of Project With Associated Risk

Project type and risk	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Project is the enhancement of the existing system	No	No	Yes	Yes	No	Yes
Funding is stable for the project	Yes	Yes	No	No	No	Yes
High reliability requirements	No	No	Yes	Yes	Yes	No
Tight project schedule	No	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes
Are resources (time, money, people etc.) scare?	No	Yes	No	No	Yes	No

Comparison/Selection of various Software Models

Parameter / Process Model→	Waterfall Model	Incremental Model	Prototype Model	Rad Model	Spiral Model	Agile Model	Xp programming
Clear Requirement Specifications	Initial level	Initial level	At medium level	Initial level	Initial level	Change incrementally	Initial level
Feedback from user	No	No	Yes	No	No	No	Yes
Speed to change	Low	High	Medium	No	High	High	High
Predictability	Low	Low	High	Low	Medium	High	High
Risk identification	At initial level	No	No	No	Yes	Yes	Yes
Practically implementation	No	Low	Medium	No	Medium	High	High
Loom	Systematic sequence	Iterative sequence	Priority on customer feedback	Use readymade component	Identification of risk at each stage	Highly customer satisfaction and incremental development[09]	Customer satisfaction and incremental development
Any variation done	Yes-v model	No	No	No	Yes-win win spiral[6]	No	No
Understandability	Simple	Intermediate	Intermediate	Intermediate	Hard	Much complex	Intermediate
Precondition	Requirement clearly defined	Core product should clearly define	Clear idea of Quick Design	Clean idea of Reuse component	No	No	No
Usability	Basic	Medium	High	Medium	Medium	Most use now a days	medium
Customer priority	Nil	Nil	Intermediate	Nil	Intermediate	High	Intermediate
Industry approach	Basic	Basic	Medium	Medium	Medium	High	Medium
Cost	Low	Low	High	very high	Expensive	Much Expensive	High
Resource organization	Yes	Yes	Yes	Yes	No	No	Yes
Elasticity	No	No	Yes	Yes	No	Very high	Medium

END OF Unit-II