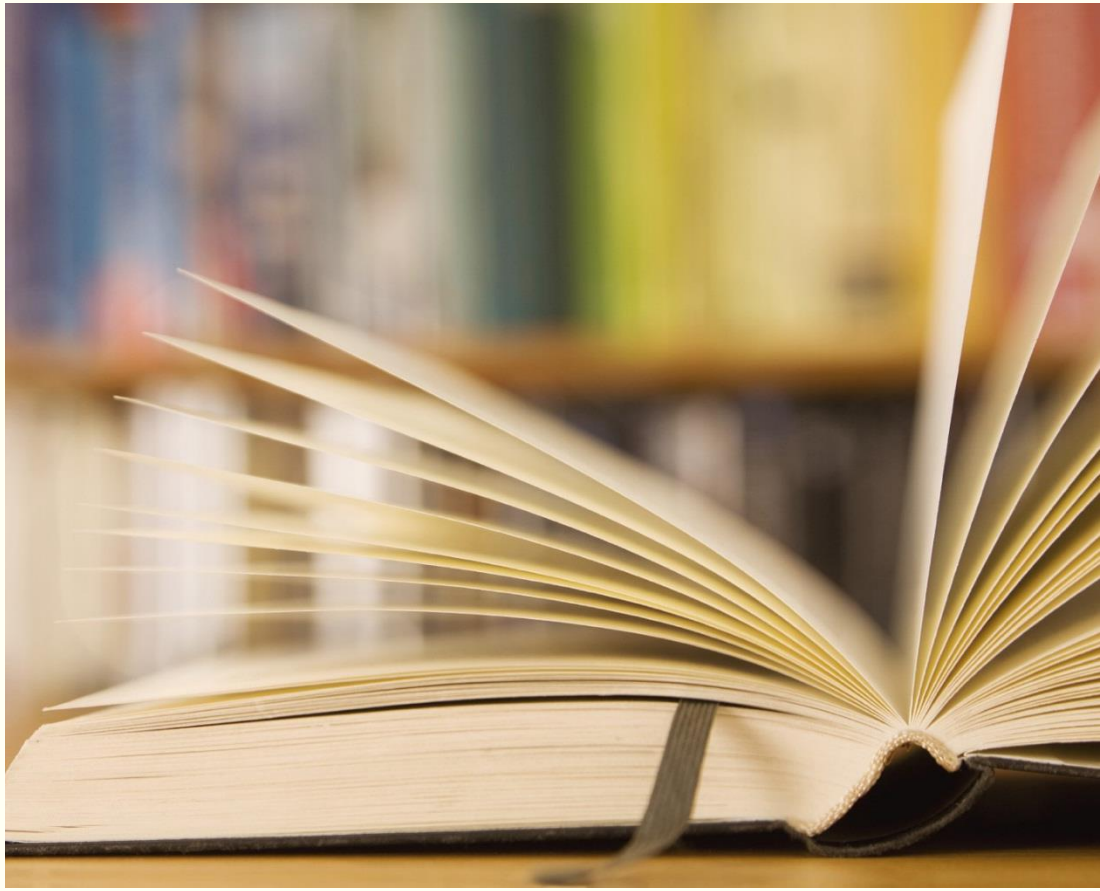
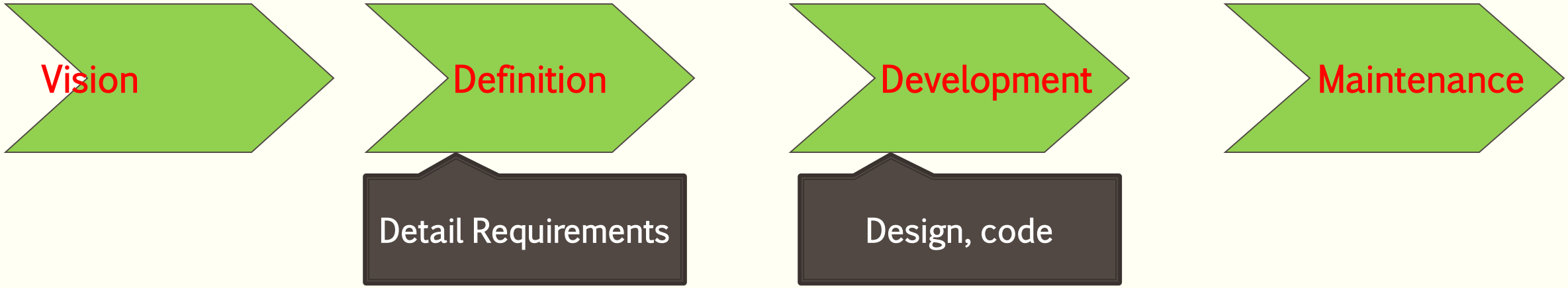
Software Engineering

Dr.Khubaib Amjad Alam,
Tahir Farooq



Software Engineering Phases

1. Vision – focus on *why*
2. Definition – focus on *what*
3. Development – focus on *how (how we can achieve this technically, design, code)*
4. Maintenance – focus on *change*



Embracing Change

- In software projects one thing that is constant:

What is Software Process?

- *“A software process is a set of related activities that leads to the production of a software product ”*

Software Processes Fundamentals Activities

Software Specification:

The functionality of the software and constraints on its operation must be defined

Software Design and Implementation:

The software to meet the specification must be produced

Software Processes Fundamentals Activities

Software Validation:

The software must be validated to ensure that **it does** what the **customer wants**

Software Evolution:

The software must evolve to meet changing customer needs

Software Process Models

- A software process model is a simplified representation of a software process
- Each process model represents a process from a particular perspective
- Provides only partial information about that process

Process Model

- Defines a distinct **set of activities**, **actions**, **tasks**, **milestones**, and **work products** that are required to engineer high-quality software

- The activities may be linear, incremental, or evolutionary

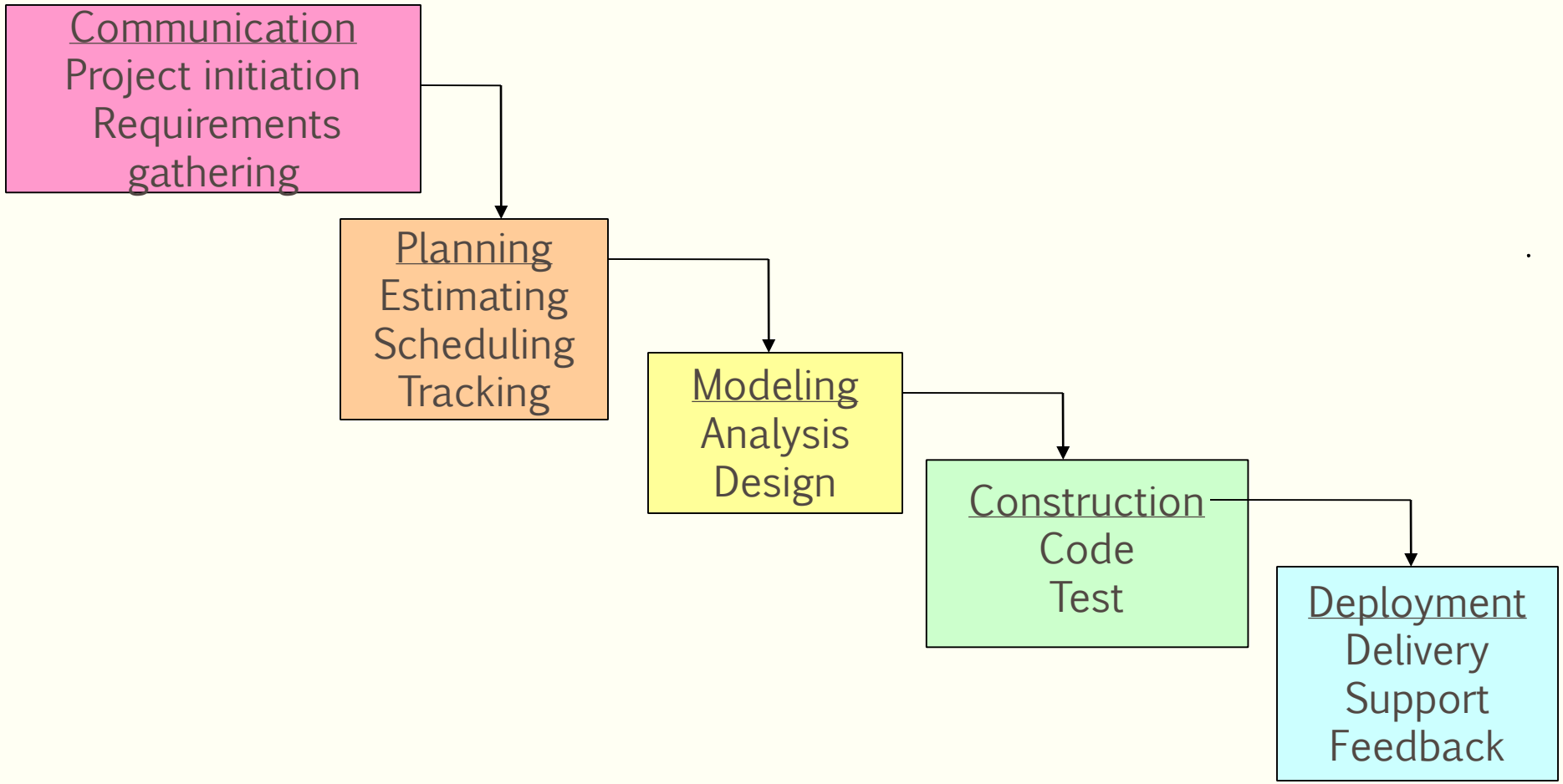
The Waterfall Model

- The first published model of the software development process (Royce, 1970)

- Because of the cascade from one phase to another, this model is known as the 'Waterfall Model'

- **Plan and schedule all of the process activities before starting work on them**

Waterfall Model (Diagram)

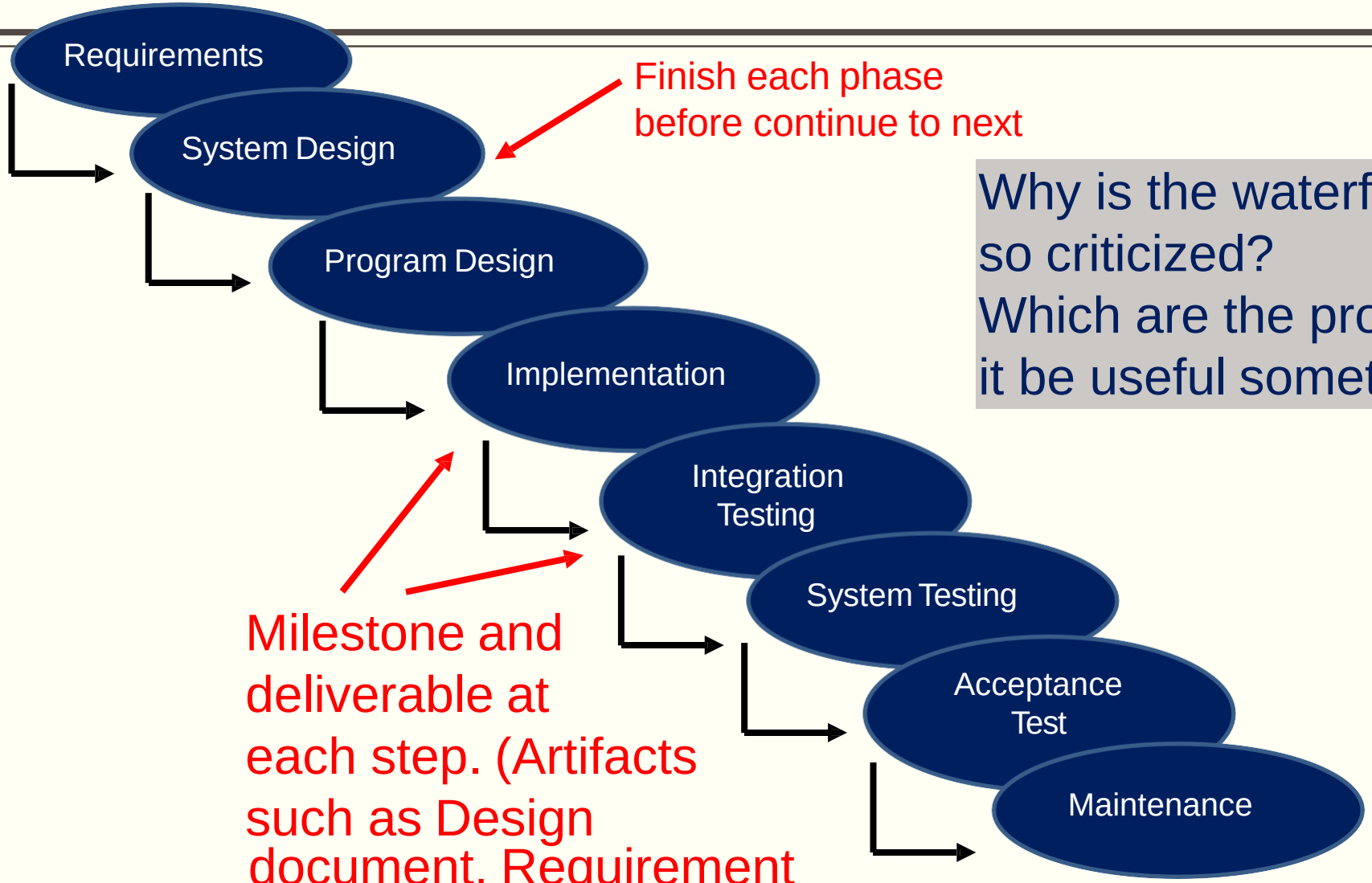


Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.

- The waterfall model is mostly used for large system engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

The Waterfall Model

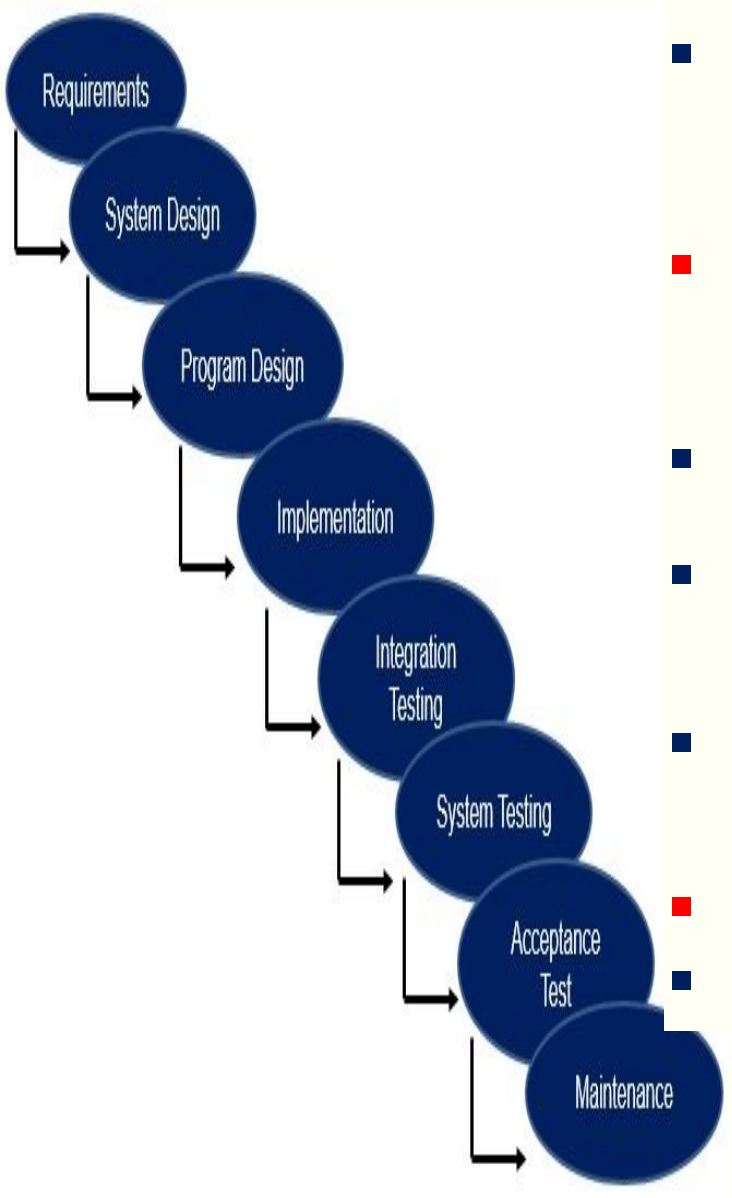


Finish each phase before continue to next

Why is the waterfall model so criticized?
Which are the problems? Can it be useful sometimes?

Milestone and deliverable at each step. (Artifacts such as Design document, Requirement Specification. etc.)

The Waterfall Model - Pros



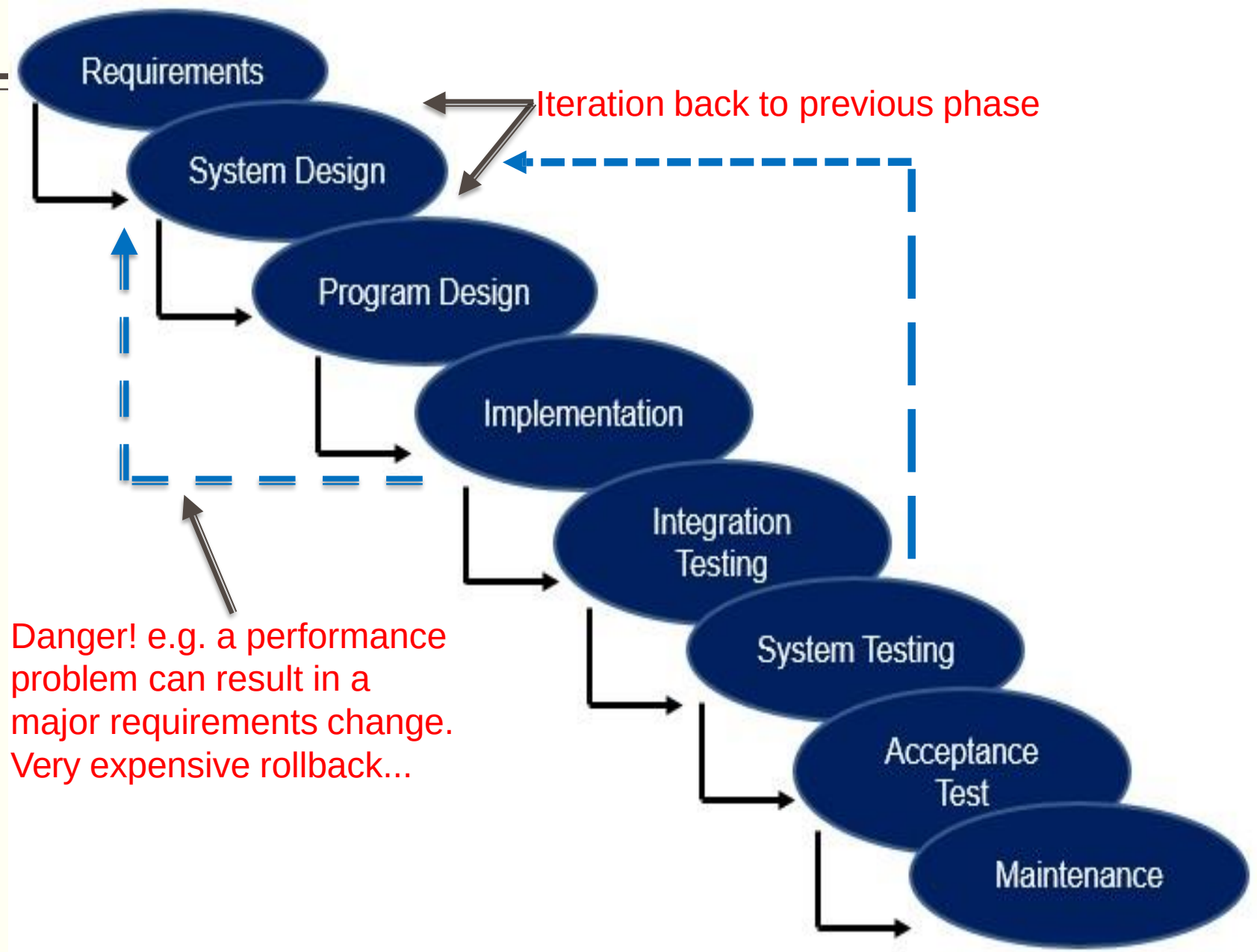
- Simple, manageable and easy to understand
- Fits to common project management practices (milestones, deliverables etc.)
- Focus on requirements and design at beginning, save money and time at the end
- Can be suitable for short projects (some weeks)
- Can be suitable for "stable" projects, where requirements do not change
- Focus on documents, saves knowledge which can be reused by other people.
- Widely used, e.g. US Department of Defense
- Can be suitable for fixed-price contracts


Problem?

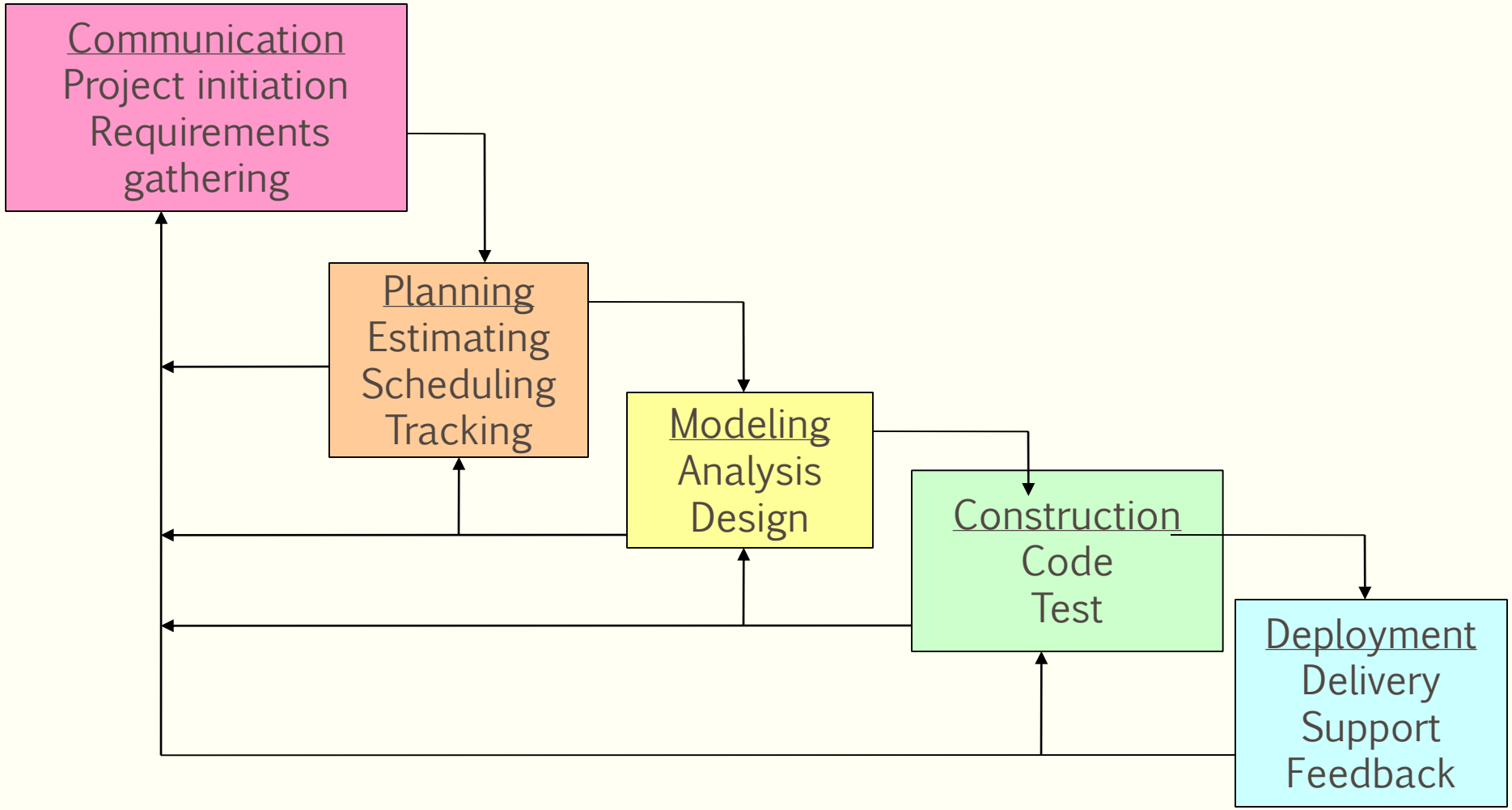
The Waterfall Model - Cons

- Software requirements change, hard to sign-off on a SRS.
- Early commitment. Changes at the end, large impact.
- Feedback is needed to understand a phase. E.g. implementation is needed to understand some design.
- Difficult to estimate time and cost for the phases.
- Handling **risks** are not part of the model. **Pushes the risks forward**.
- Software "is not" developed in such a way. It evolves when problems are more understood.
- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a *working version of the program doesn't occur* until the final phase
- **Problems** can be somewhat **alleviated** in the model through the addition of **feedback loops**

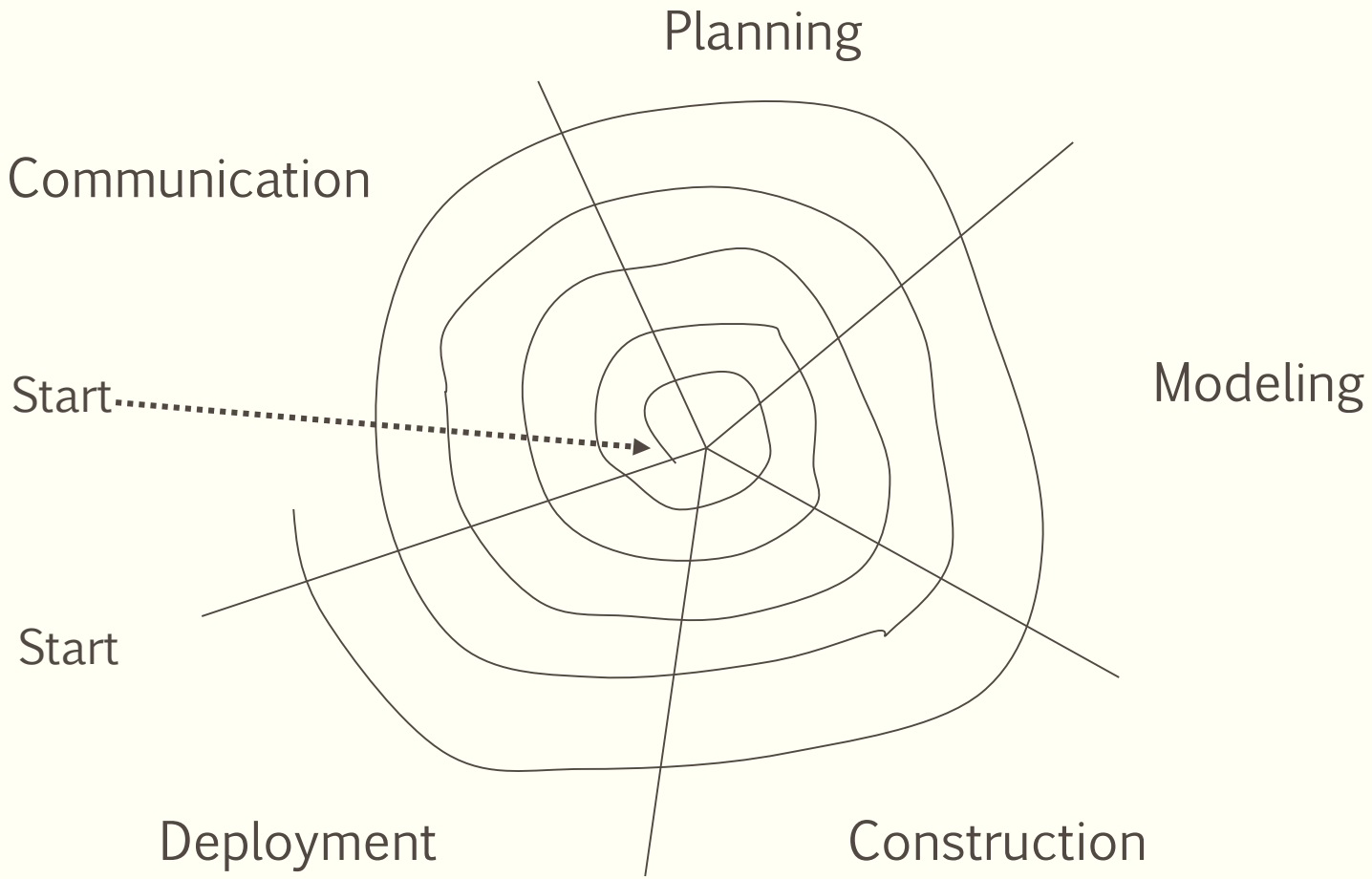
Can We Improve the Model?

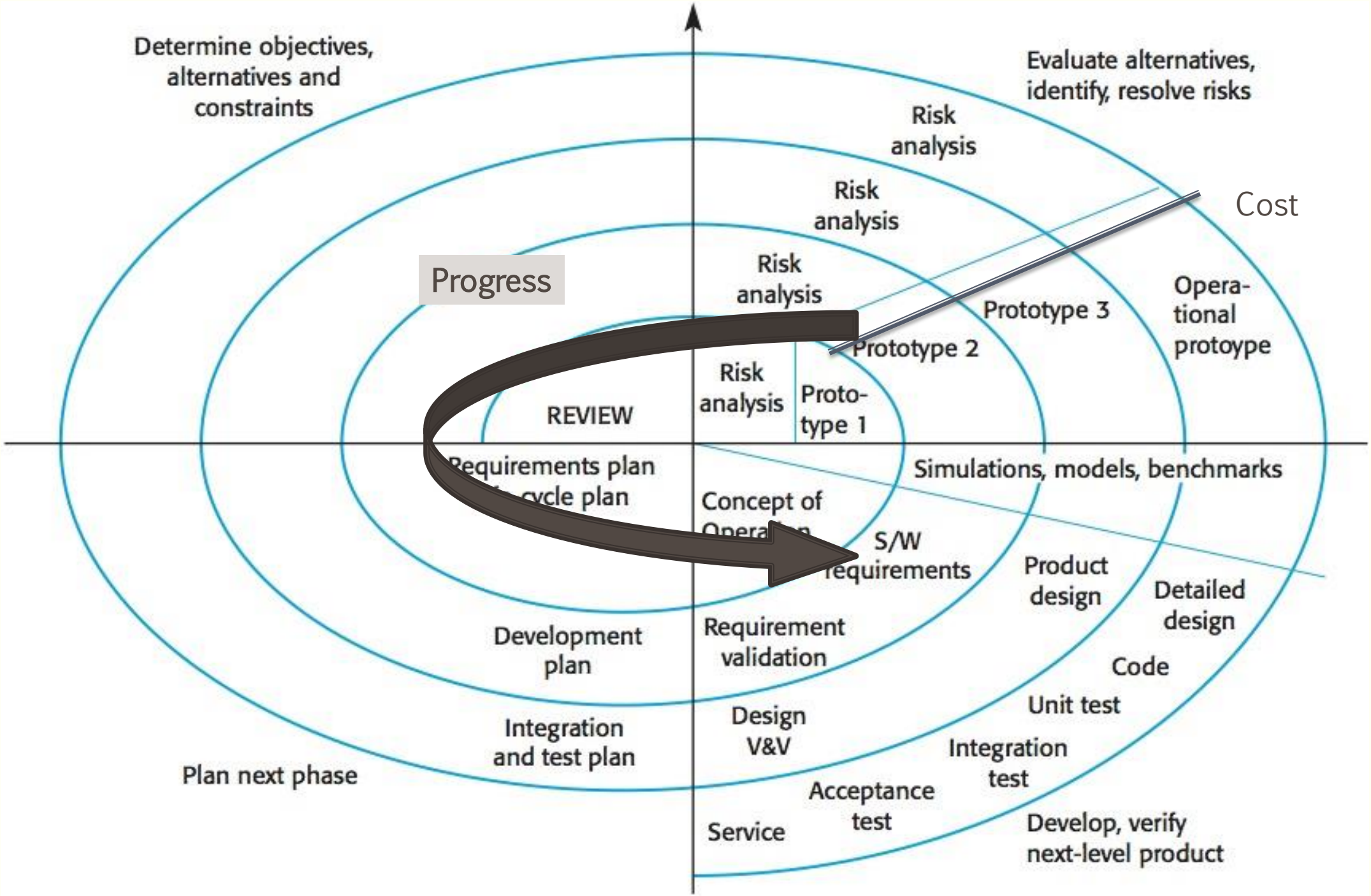


Waterfall Model with Feedback (Diagram)



Spiral Model (Diagram)



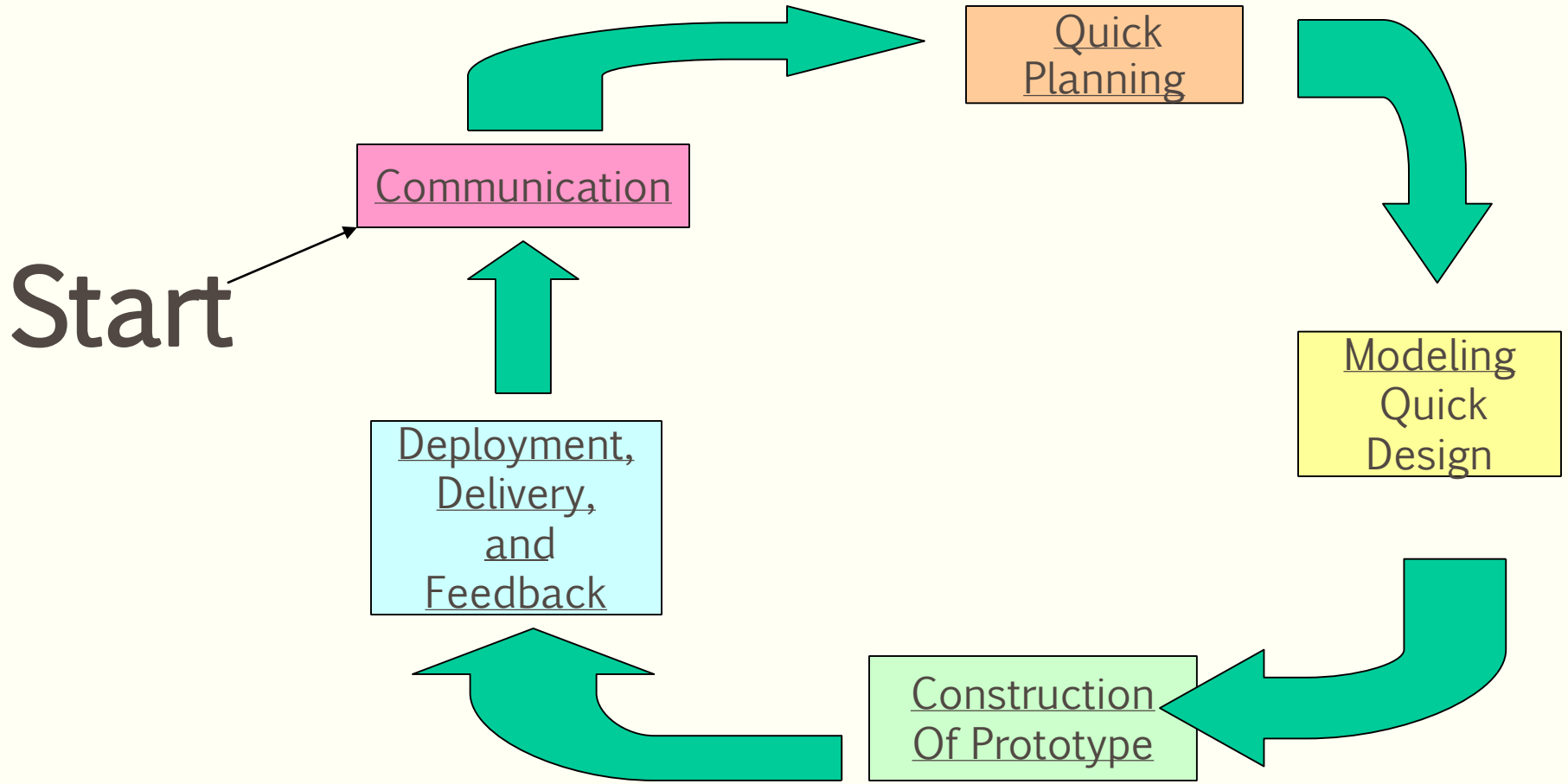

Determine objectives, alternatives and constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Cost
Progress
Prototype 3
Opera-tional protoype
Prototype 2
Risk analysis
Proto-type 1
REVIEW
Requirements plan
cycle plan
Simulations, models, benchmarks
Concept of
S/W requirements
Product design
Detailed design
Development plan
Requirement validation
Code
Unit test
Design V&V
Integration and test plan
Integration test
Plan next phase
Acceptance test
Service
Develop, verify next-level product

Spiral Model (Description)

- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software

- Operates as a risk-driven model…a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development

Prototyping Model(Diagram)



Prototyping Model (Description)

- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- Feedback is used to refine the prototype

Prototyping Model(Potential Problems)

- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- Lesson learned
  - Define the rules up front on the final disposition of the prototype before it is built
  - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality

Prototyping Model

- A customer defines a set of general objectives for software but does not identify detailed **input**, **processing**, or **output** requirements

- The **developer** may be **unsure** of the **efficiency** of an algorithm, the **adaptability** of an operating system

Prototyping Model

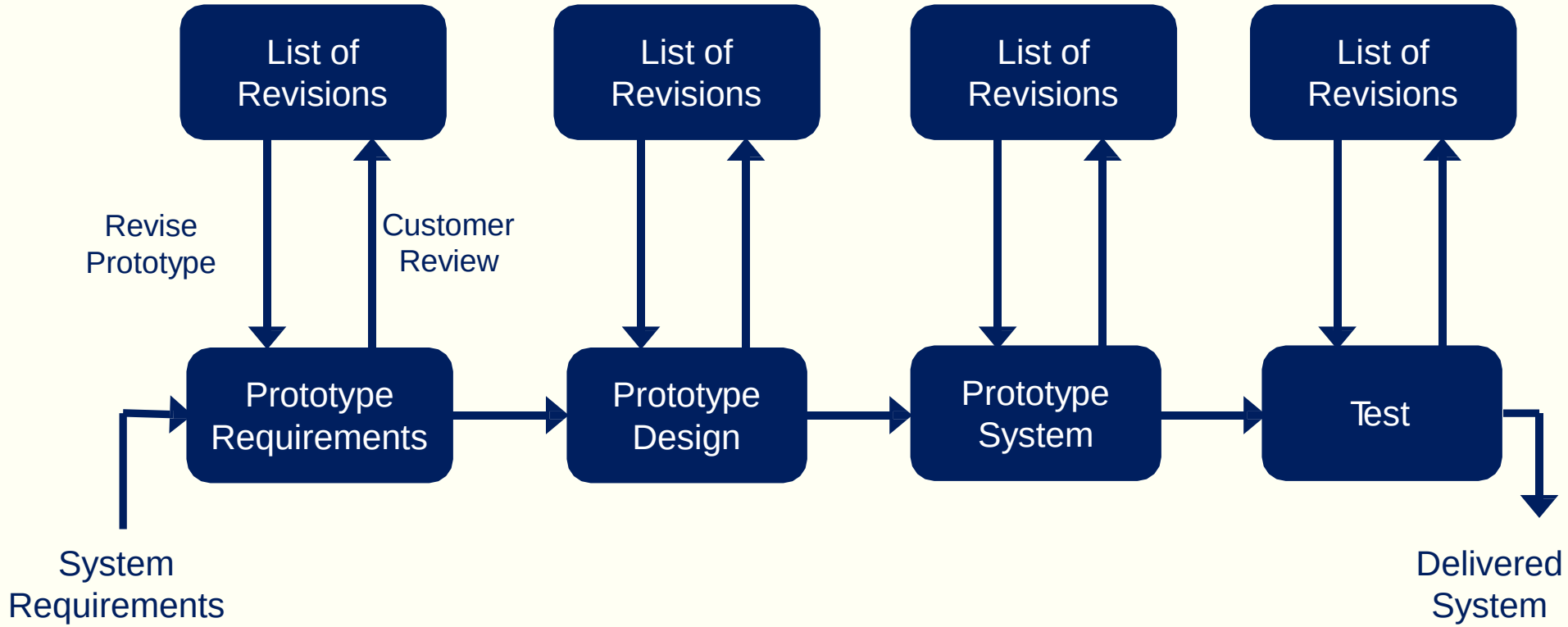
- This model adds prototyping as sub-process
- A prototype is a **partially developed** product that enables customers and developers to examine some aspect of a proposed system and decide if it is suitable for a finished product

Prototyping Model

- Used to explore the risky aspects of the system
  - Risk of developing the “wrong” system
  - User interface without functionality
  - Technical risks

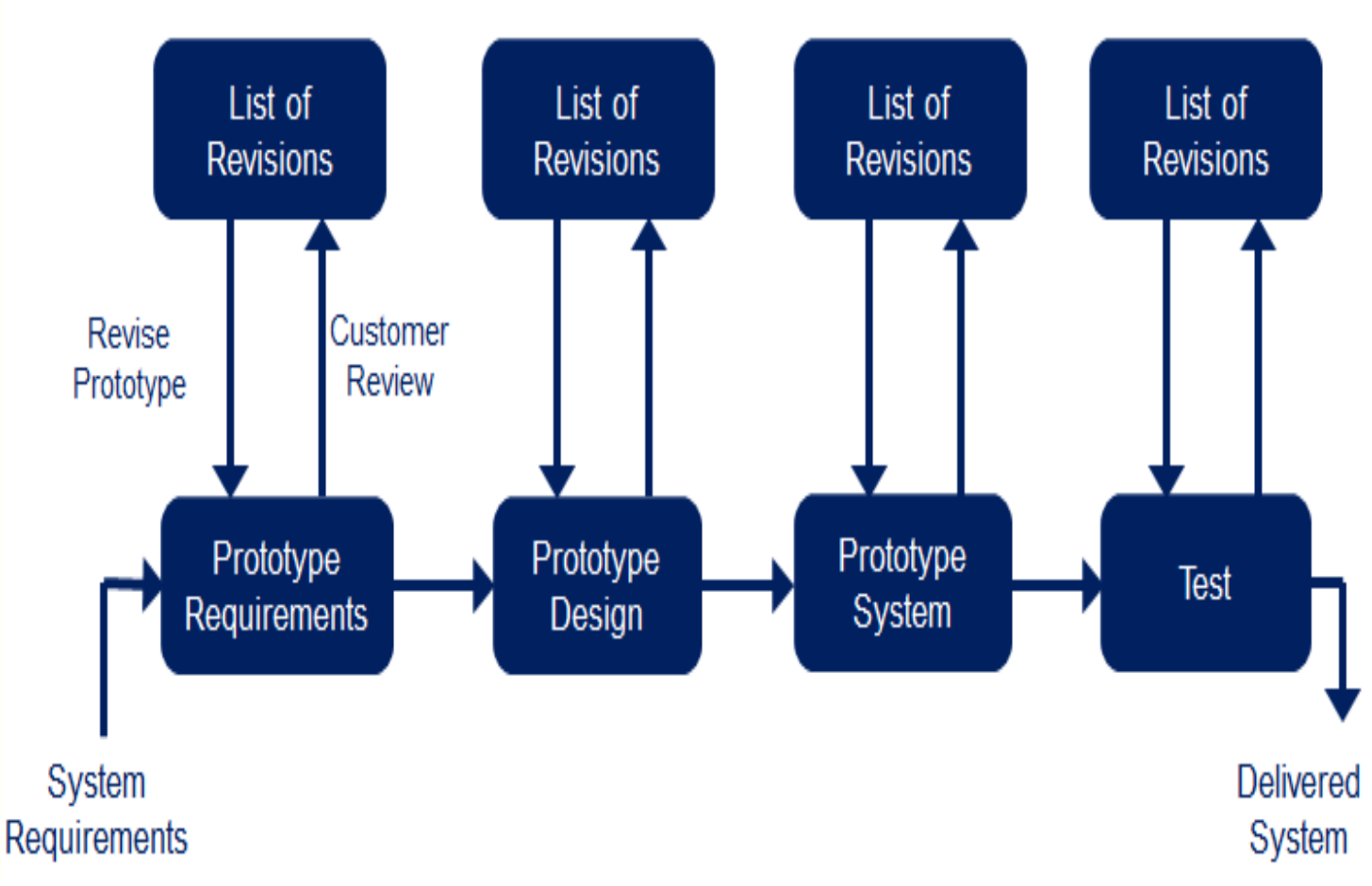
- Prototype may be thrown away or evolve into product

Prototyping Model



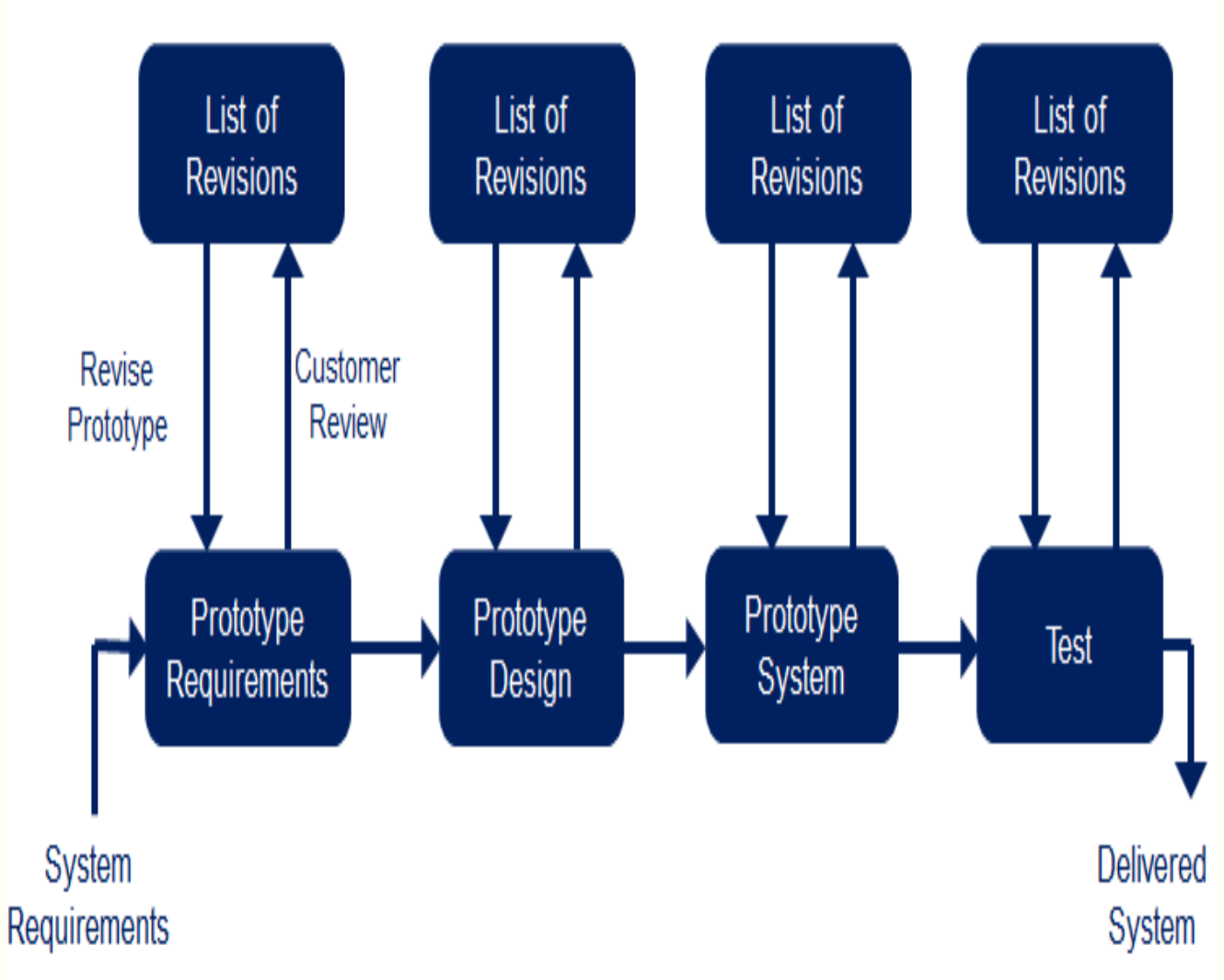
Prototyping Model (Pros)

- Suitable for large systems for which there is no manual process to define the requirements
- Much feedback from customer



Prototyping Model (**Cons**)

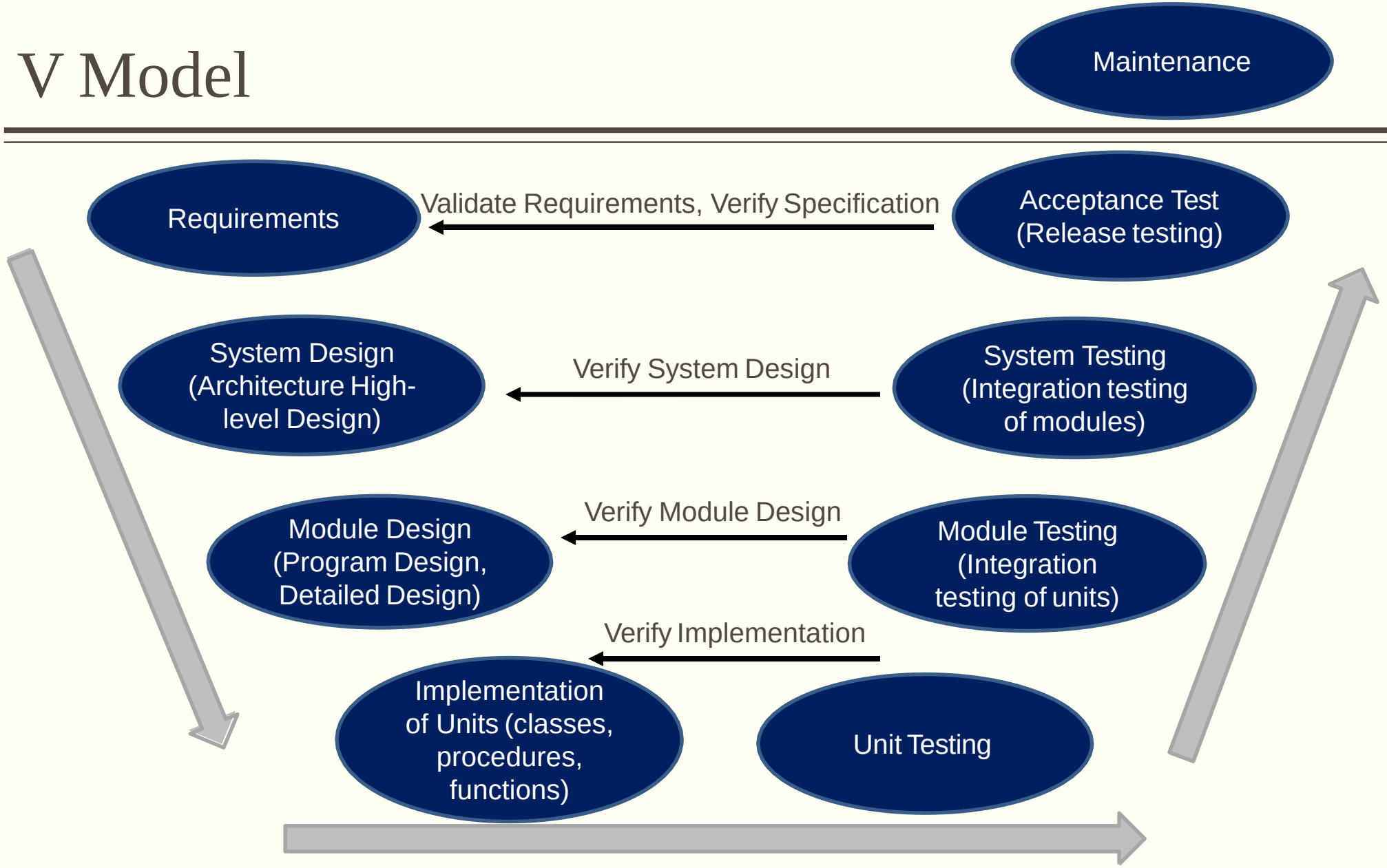
- It may be impossible to tune the prototype to meet non- functional requirements
- Lot of time at customer
- The changes made during prototype development will probably have degraded the system structure



V Model

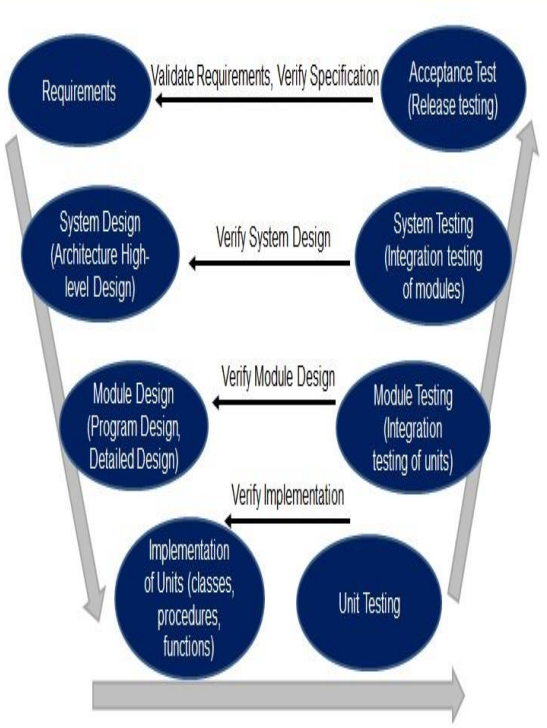
- The V-model is a variation of the waterfall model that demonstrates how the testing activities are related to analysis and design
- Developed by the German Ministry of Defense
- **Unit and system testing** verify the program **design**, ensuring that parts and whole work correctly
- Acceptance testing, conducted by the customer rather than developers, validates the requirements, trying each system function meets a particular requirement in the specification

V Model


Maintenance
Requirements
Validate Requirements, Verify Specification
Acceptance Test (Release testing)
System Design (Architecture High-level Design)
Verify System Design
System Testing (Integration testing of modules)
Module Design (Program Design, Detailed Design)
Verify Module Design
Module Testing (Integration testing of units)
Verify Implementation
Implementation of Units (classes, procedures, functions)
Unit Testing

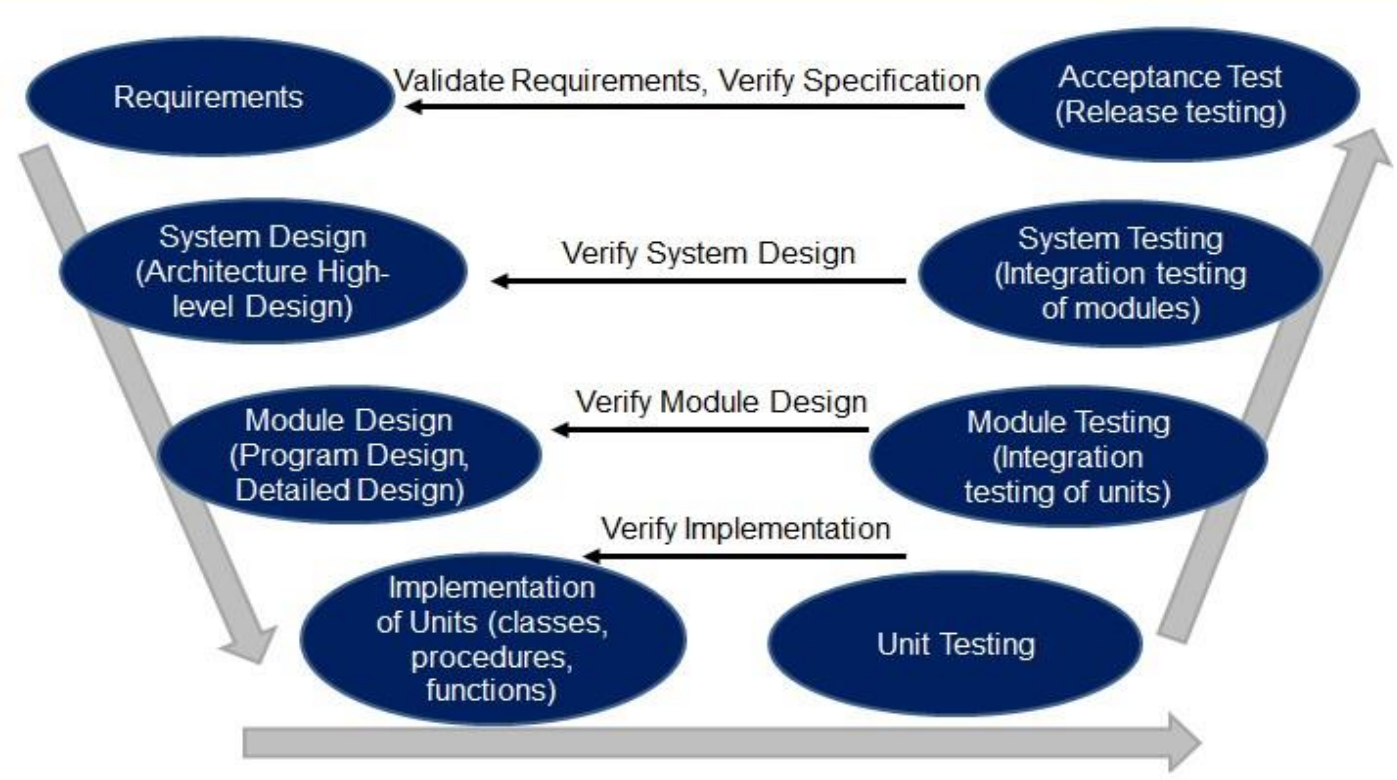
V Model (Pros)

- It defines tangible phases of the process, and proposes a logical sequence in which these phases should be approached
- It also defines logical relationships between the phases
- It demands that testing documentation is written as soon as possible
- It gives equal weight to development and testing
- It provides a simple and easy to follow map of the software development process

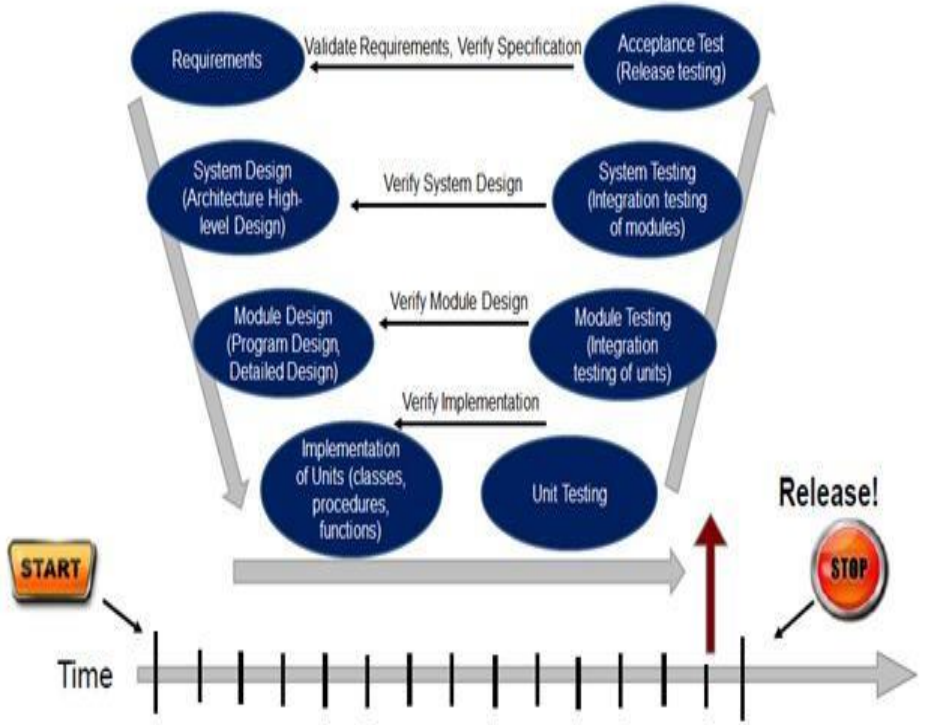


V Model (Cons)

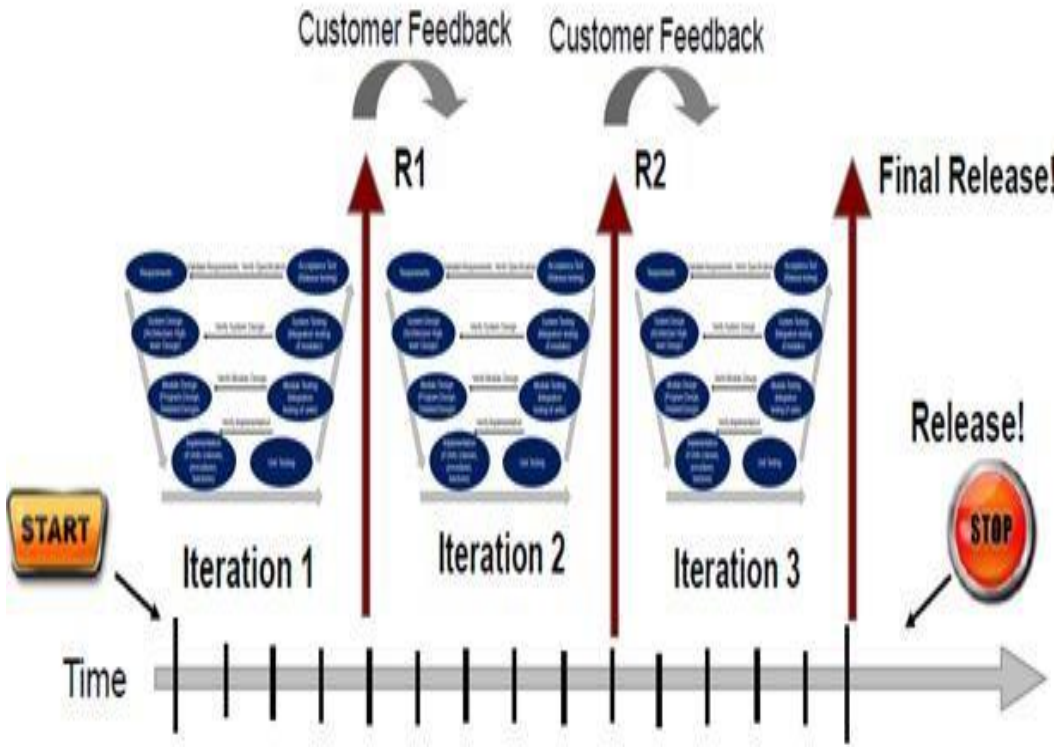
- It is too simple to accurately reflect the software development process
- It is inflexible; it has no ability to respond to change
- It produces inefficient testing methodologies



Incremental Development



Iterative Development

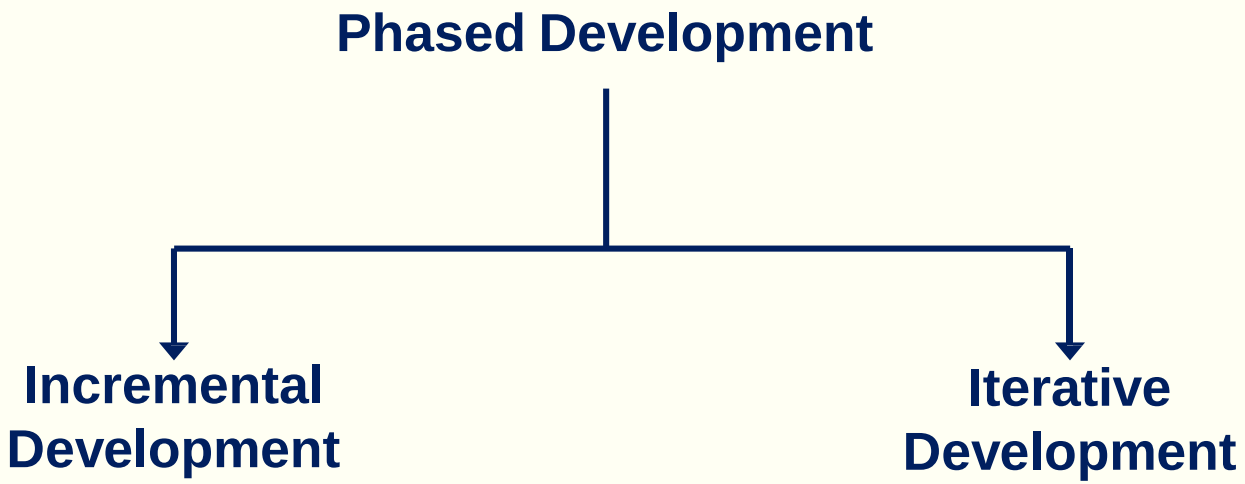


Phased Development

- Design a system so it can be delivered in pieces, letting users have some functionality while the rest is under development

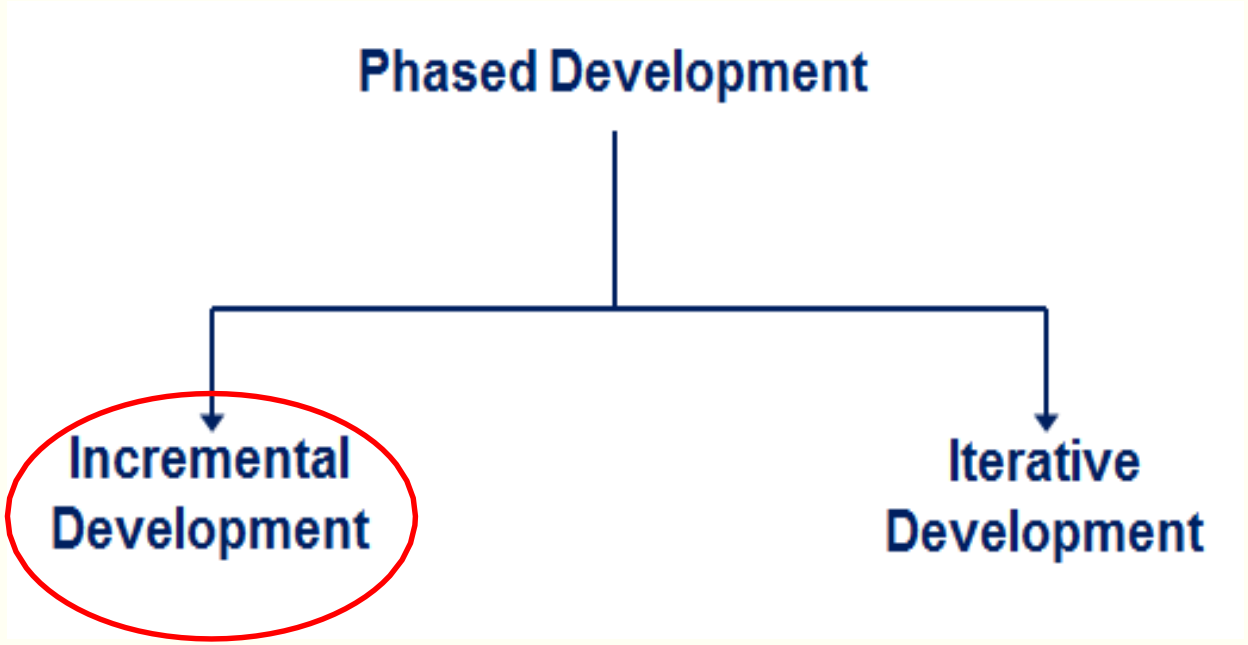
- There are usually two or more systems in parallel:
  a) The operational or production system in use by customers
  b) The development system which will replace the current release

Phased Development

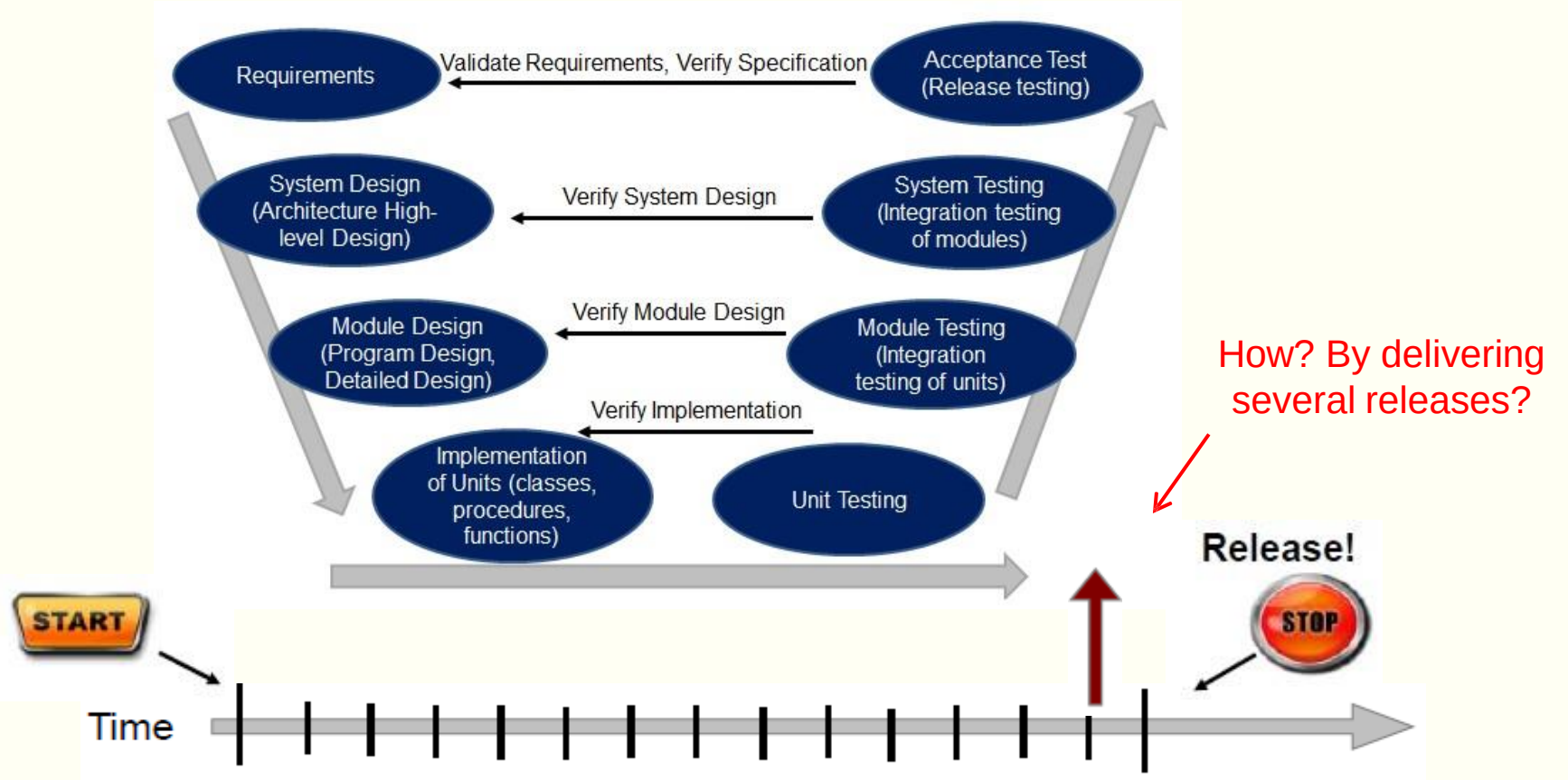


Incremental Development

- Incremental development partitions a system by functionality
- Early release starts with small, functional subsystem, later releases add functionality

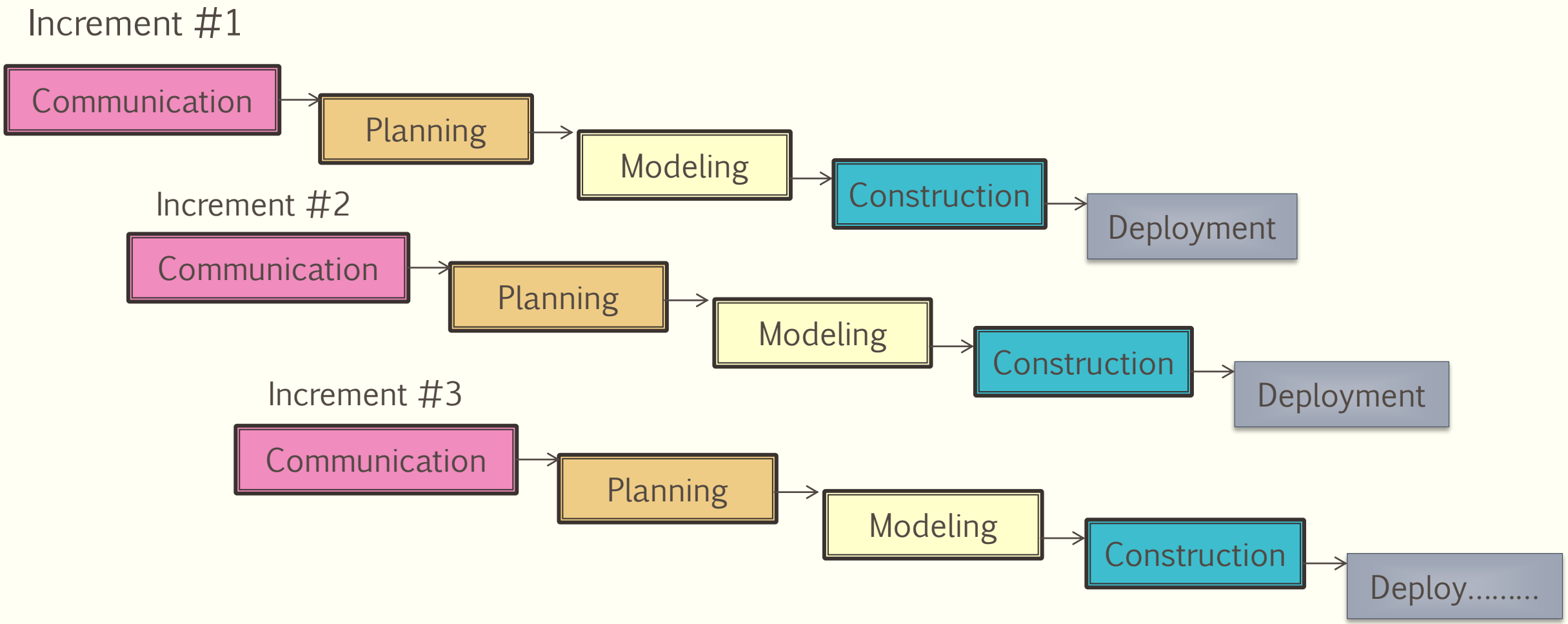


Incremental Development



How? By delivering several releases?

Incremental Model (Diagram)



Incremental Model (Description)

- Used when **requirements** are **well understood**
- Multiple independent deliveries are identified
- Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- Iterative in nature; focuses on an operational product with each increment
- Provides a **needed set of functionality** sooner while delivering optional components later
- Useful also when staffing is too short for a full-scale development

Iterative Development

- Iterative development improves overall system in each release
- Delivers a full system in the first release, then changes the functionality of each subsystem with each new release

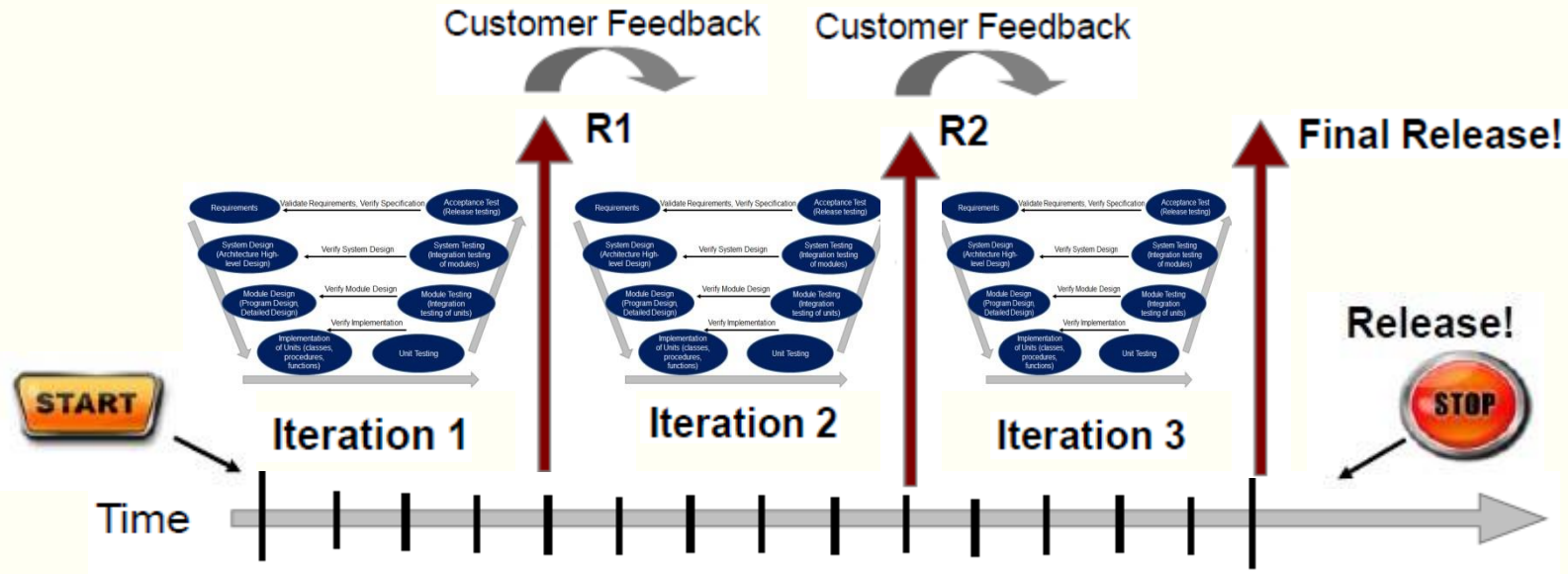
Iterative Development

When should the releases take place?

Time-boxing - The time period is fixed for each iteration.

When should the releases take place?

Prioritized functionality - Do the most important parts first.



Iterative vs. Incremental Development

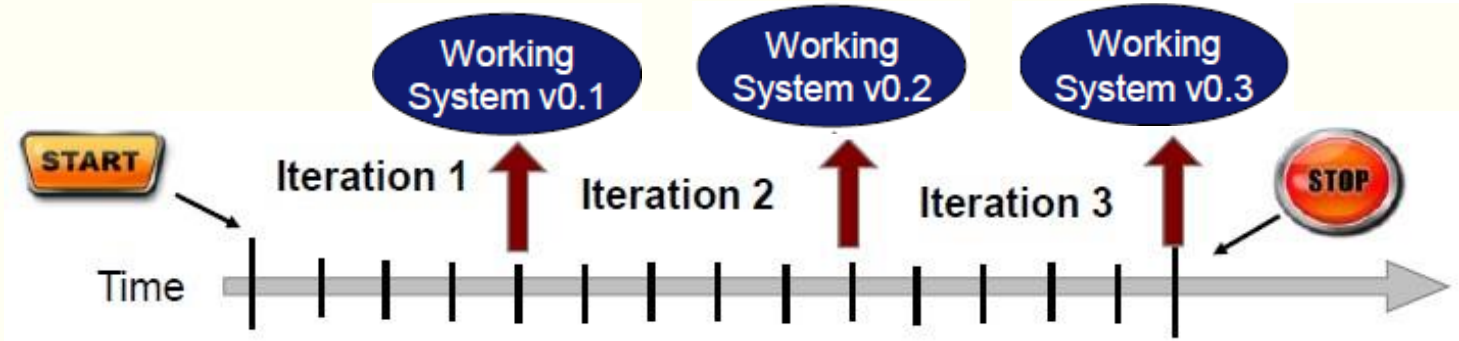
Incremental Development

Add a new "part" at each increment

Iterative Development

Improve a "working system" at each iteration

Note. Both concepts are often combined and sometimes misleading called just iterative development.

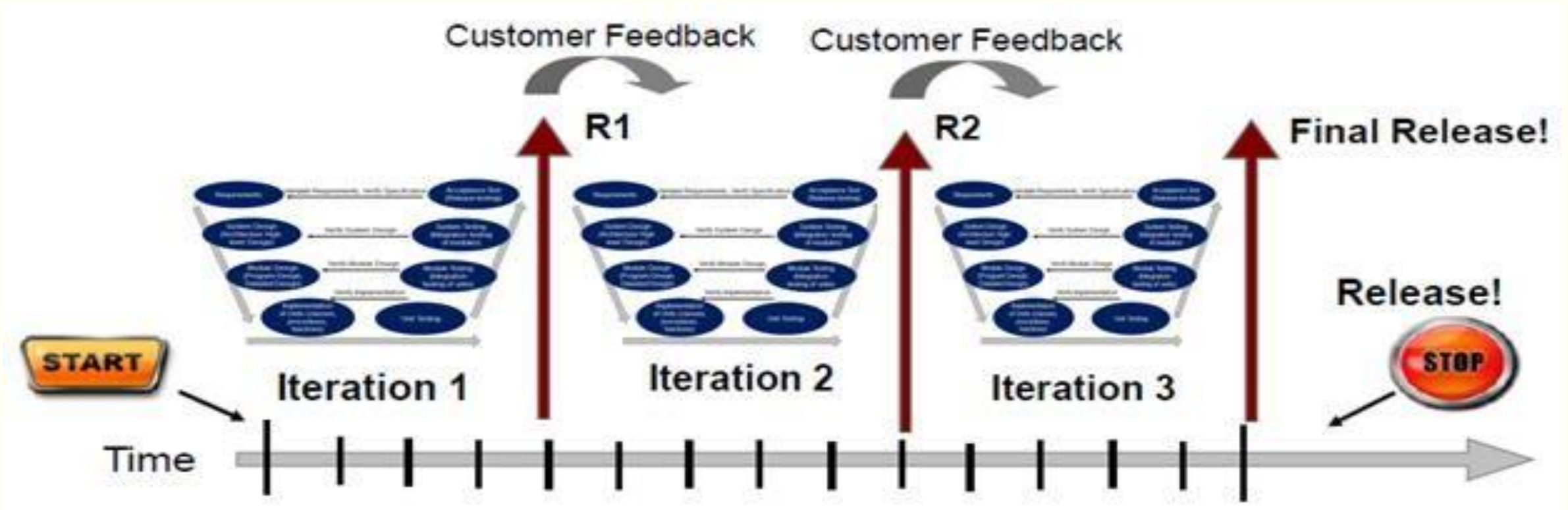


Iterative Development - Pros

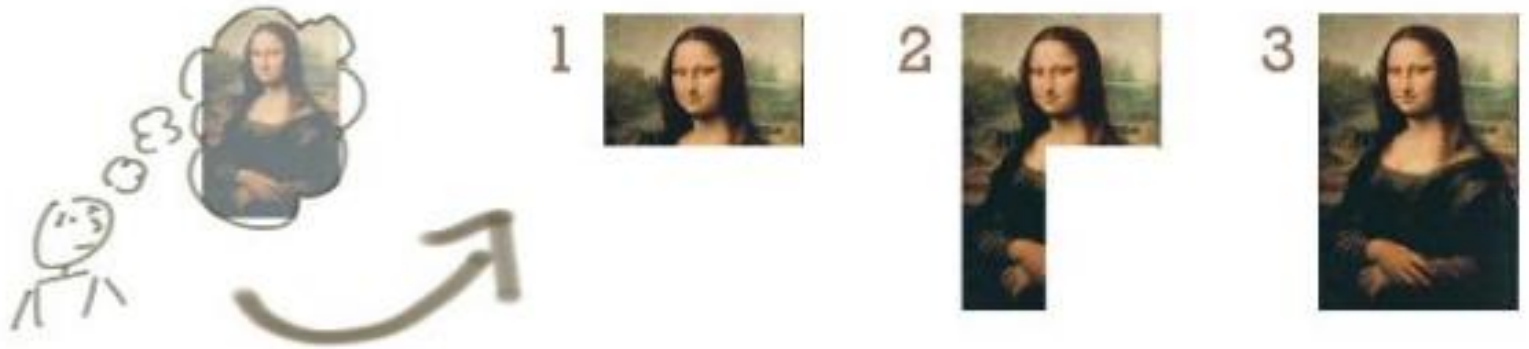
- Misunderstandings and inconsistency are made clear early (e.g. between requirement, design, and implementation)
- Encourage to use feedback -> elicit the real requirements
- Forced to focus on the most critical issues
- Continuous testing offers project assessment
- Workload is spread out over time (especially test)
- The team can get "lesson learned" and continuously improve the process
- Stakeholders gets concrete evidence of progress

Iterative Development - Cons

- Problem with current business contracts, especially fixed-price contracts
- With short iterations it can be hard to map customer requirements to iterations



Incremental Development:



Iterative Development:

Activity

A customer wants to develop a word processing software package. It is estimated that the software package will be completed within one and half year. The customer wants that the software package should be develop before estimated time or he/she could be able to start working on the software package as soon as possible. What process model company should follow?