Object Oriented Analysis & Design

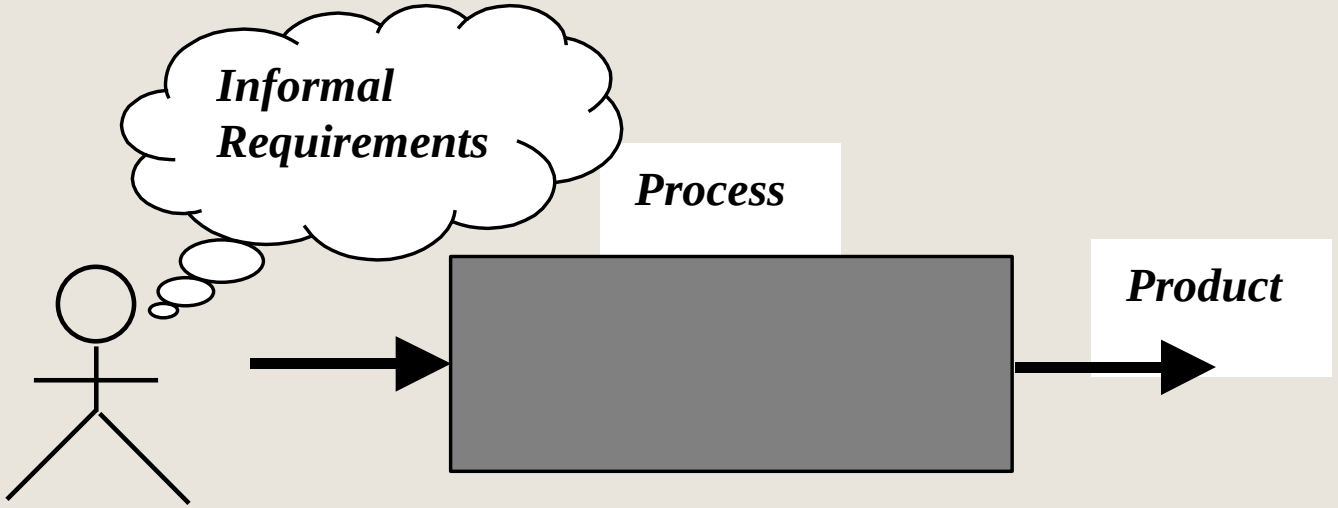
Lecture 02 Process Models

Course Instructor: **SYED SAQLAIN HASSAN**

Course Code: SE321

Spring 2023

Process as a "black box"



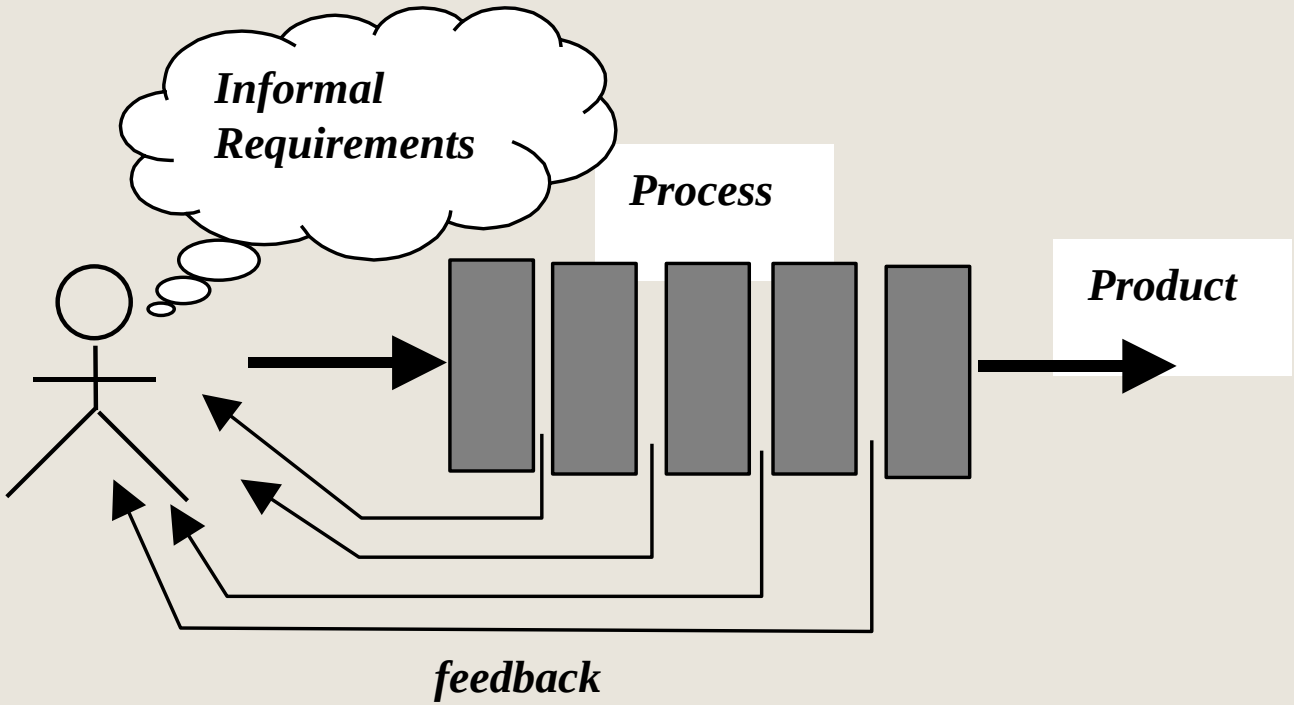
Problems

- The assumption is that **requirements** can be **fully understood** prior to development

- **Interaction** with the customer occurs only at the **beginning** (requirements) and **end** (after delivery)

- Unfortunately the **assumption** almost **never holds**

Process as a "white box"



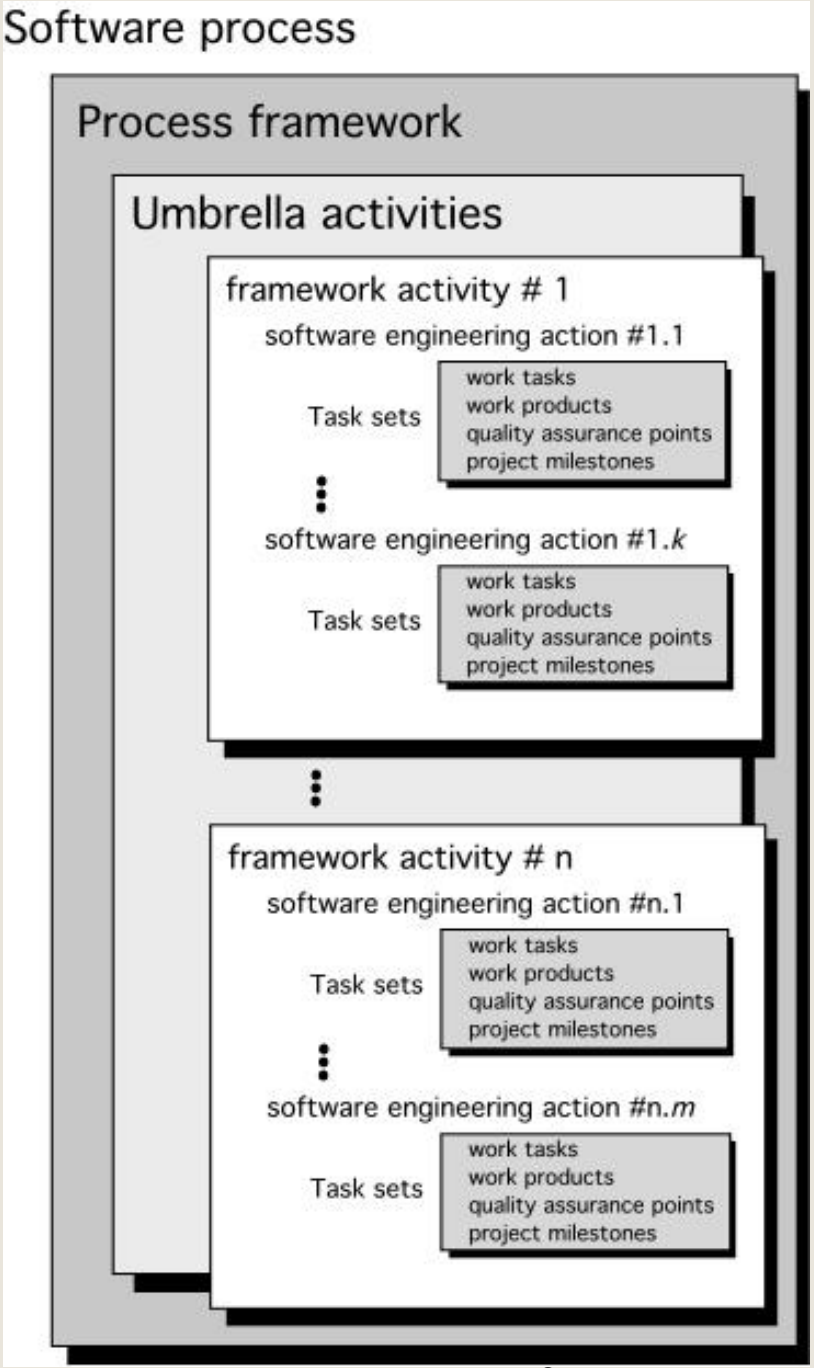
Schematic Software Process Representation

Each activity has:

- Actions
- Tasks
- Work products
- QA Points
- Milestones (indicate progress)

The flow of these activities?

Process Flow!!!

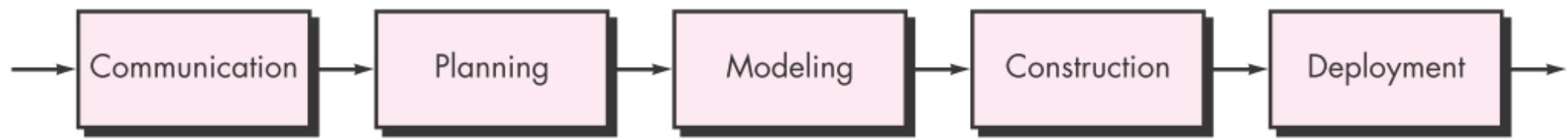


Process Flow

The process flow describes how the framework activities and the actions and tasks that occur within each framework activity are **organized with respect to sequence and time.**

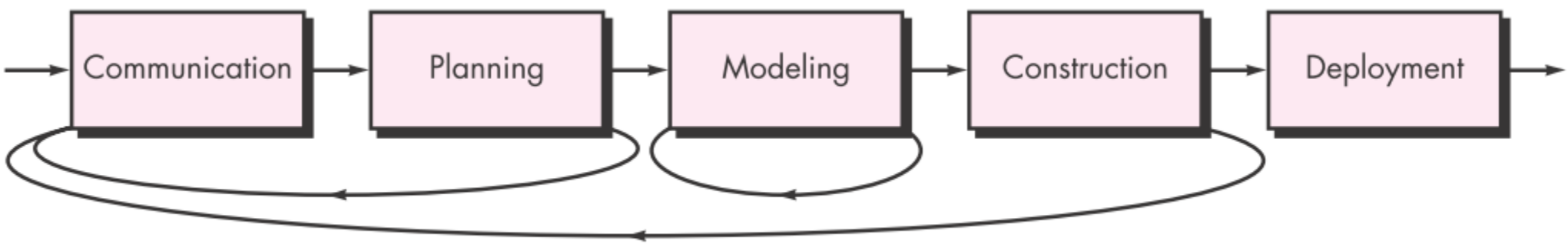
Process Flow

a) **Linear flow** executes each of the five framework **activities in sequence**, beginning with communication and ending with deployment.



(a) Linear process flow

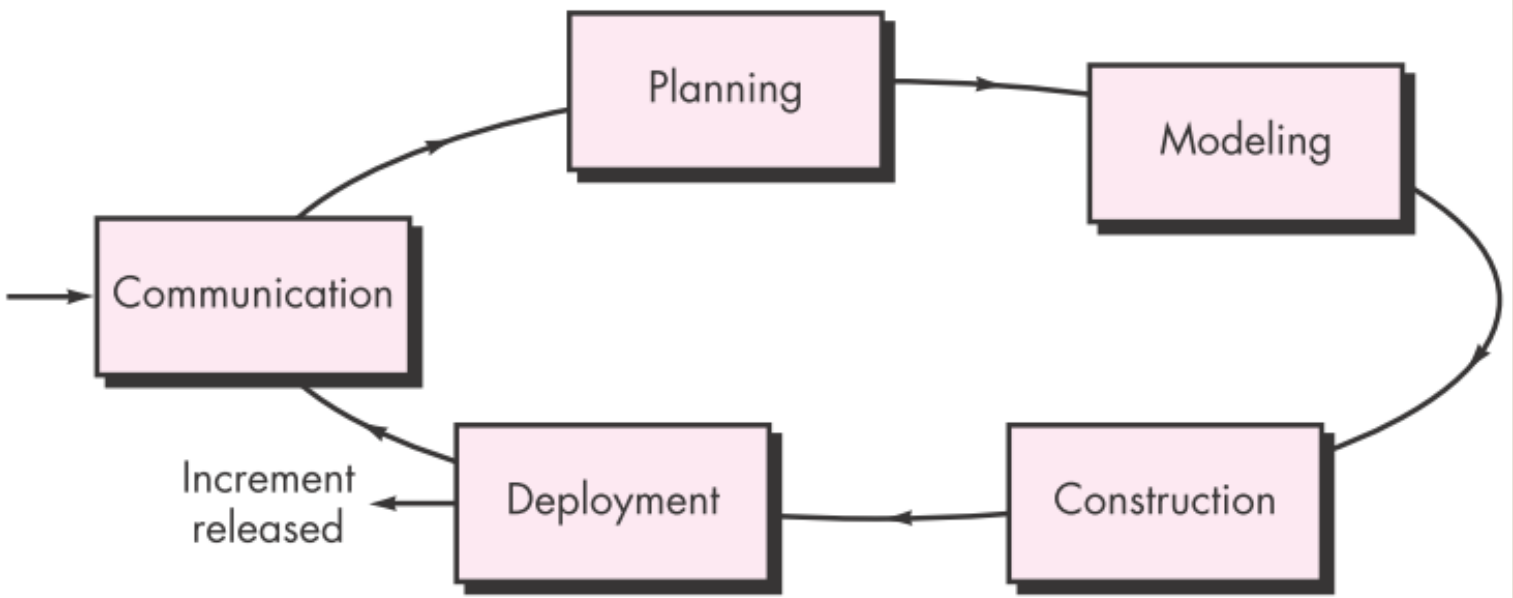
b) **Iterative flow** repeats one or more of the activities before proceeding to the next.



(b) Iterative process flow

Process Flow

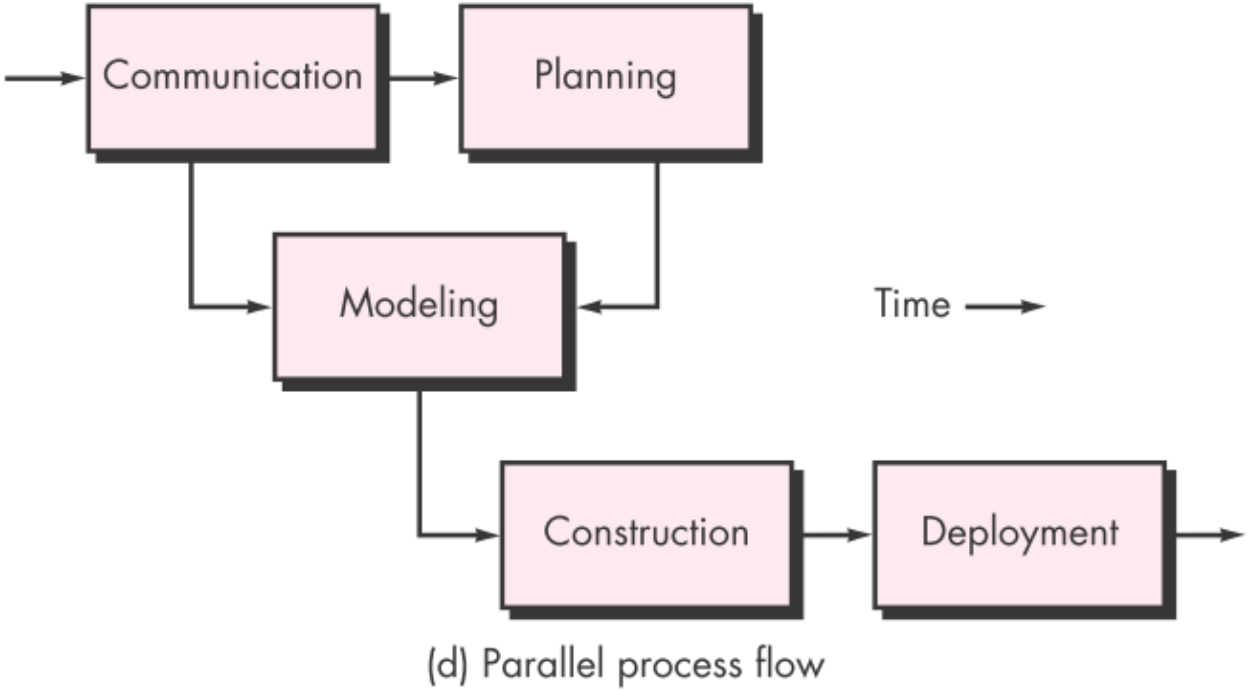
c) **Evolutionary flow** executes the activities in a "**circular**" manner. Each circuit through the five activities **leads to a more complete version** of the software.



(c) Evolutionary process flow

Process Flow

d) **Parallel flow** executes one or more **activities in parallel** with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).



(d) Parallel process flow

Defining Framework Activity

The **actions** to be done in a framework activity must be **decided**.

The actions may **differ** based upon:

- **Nature** of the **problem**
- Characteristics of the **software team**
- **Stakeholders** of the project

Prescriptive Process Models

- The set of **process elements is prescribed** by a software process model, they include:
  - framework activities,
  - software engineering actions,
  - tasks,
  - work products,
  - quality assurance,
  - change control mechanisms for each project, and
  - **process flow**

Each process model applies **different emphasis** to **each activity** and defines a **process flow** that invokes each framework activity

An Example Project

Word-processing Software

List of desired features

- Basic file management
- File editing,
- Document production
- More sophisticated editing
- Advanced document production in different formats
- Spelling and grammar checking
- Advanced page layout capability

Prescriptive Process Models

- Build/Code and Fix Model
- The Waterfall Model
- The V-Model
- Incremental Process Models
- Evolutionary Process Models
  - Prototyping
  - The Spiral Model
- Concurrent Model

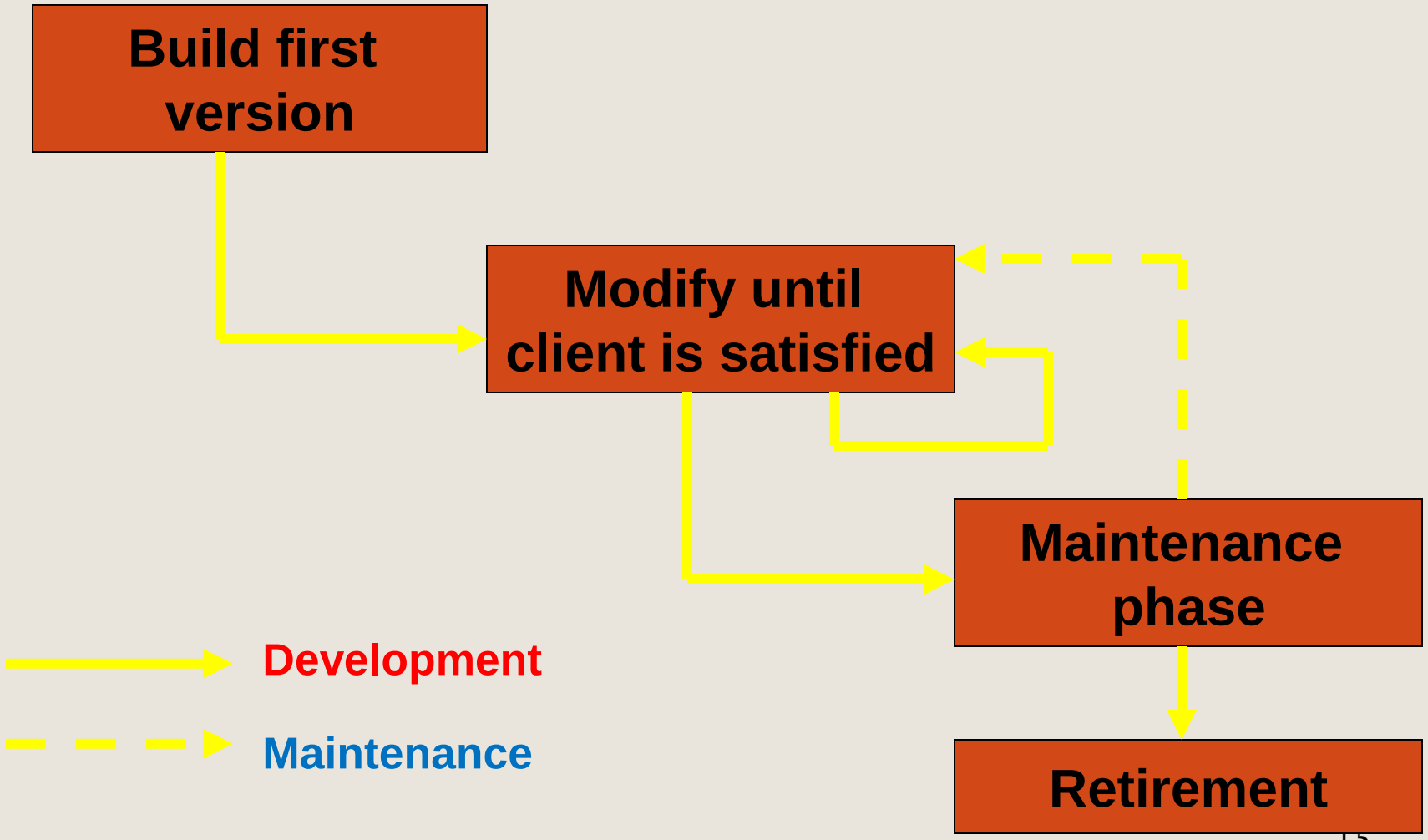
Build and Fix Model (Earliest Software Process)

The earliest approach

- This model starts with an **informal** general **product idea** and just **develops** code until a product is "**ready**" (or money or time runs out).

- **Write code** and **Fix** it to eliminate any **errors** that have been detected, to enhance existing functionality, or to add new features
- Source of difficulties and deficiencies
  - impossible to **predict**
  - impossible to **manage**

Build and Fix Model



Build and Fix Model (Advantages)

1. No administrative overhead
2. Signs of progress (code) early.
3. **Low expertise**, anyone can use it!
4. Useful for **small "proof of concept" projects**, e.g. as part of risk reduction.

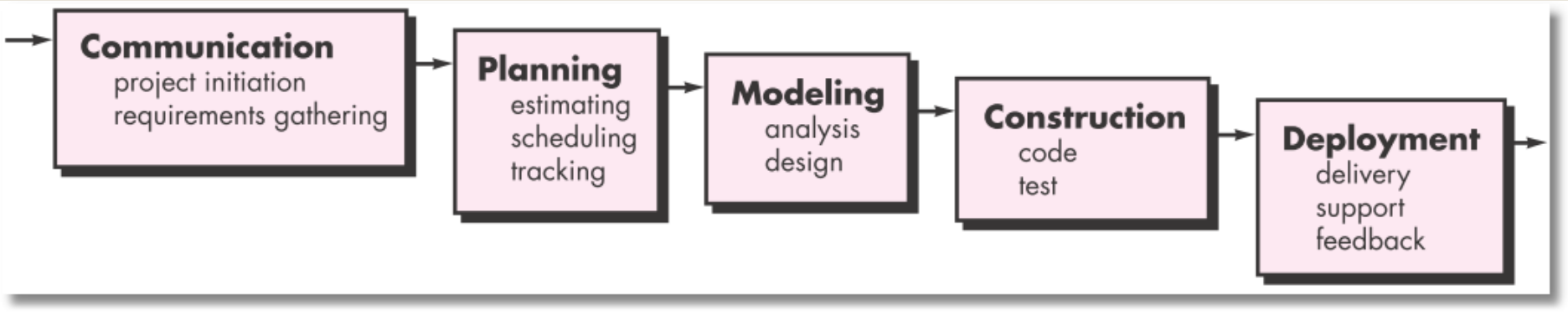
Build and Fix Model (Disadvantages)

1. Dangerous!
2. No visibility/control
3. No resource planning
4. No deadlines
5. Mistakes hard to detect/correct
6. Impossible for large projects, communication breakdown, chaos.

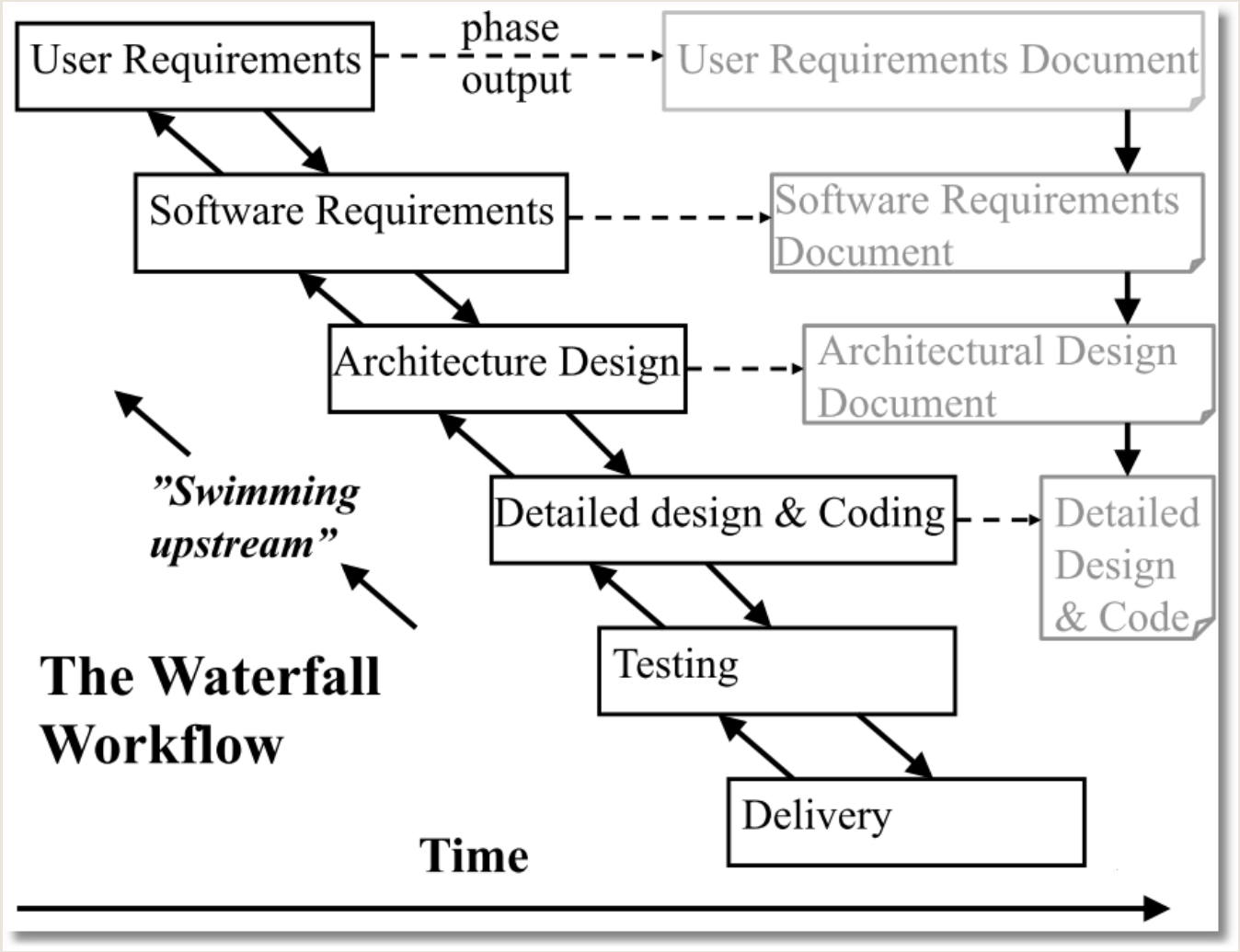
The Waterfall Model

- The waterfall model is the **classic process model** – it is widely known, understood and used.
- In some respect, waterfall is the ”**common sense**” approach.
- Cascades from one stage down to the next, in **stately**, **lockstep**, glorious order.
  - Gravity only allows the waterfall to **go downstream**;
  - it’s very **hard** to **swim upstream**
- Exists in many **variations**

The Waterfall Model



The Waterfall Model



The Waterfall Model (Advantages)

1. **Easy** to understand and implement.
2. **Widely used** and known (in theory!)
3. Fits other engineering process models: **civil**, **mech** etc.
4. Reinforces **good habits**: define-before-design, design before-code
5. Identifies **deliverables** and milestones
6. **Document driven**: People leave, documents don't
7. Works well on large/**mature products** and **weak teams**.

Cost Ratio of Errors

Requirements : Maintenance

=

1 : 200

The Waterfall Model (Disadvantages)

1. **Doesn't** reflect **iterative** nature of exploratory development.
2. Sometimes **unrealistic** to expect accurate **requirements early** in a project
3. Software is **delivered late**, delays discovery of serious errors.
4. **No** inherent **risk management**
5. **Difficult** and **expensive** to change decisions, "**swimming upstream**".
6. Significant administrative overhead, **costly** for **small teams** and **projects**.
7. **Late testing** paradigm and **late feedback** to customer and engineer

Analysis of the V-Shaped Life-Cycle

- **Improves** the testing paradigm

  ==> Quality Assurance

- Does **NOT** really improve:
  - sequentially
  - feedback
  - risk management (during development)

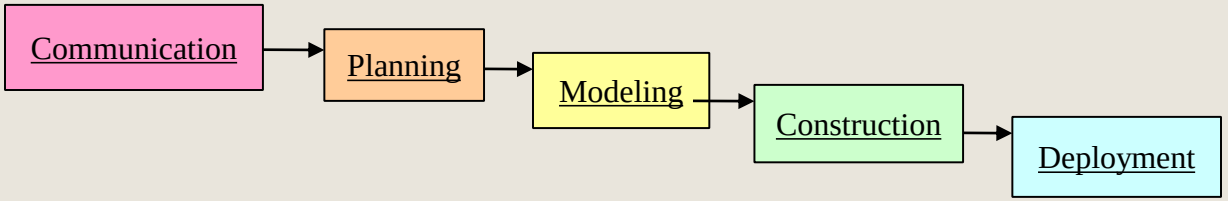
Validation vs Verification

Validation: Did we build the right system?

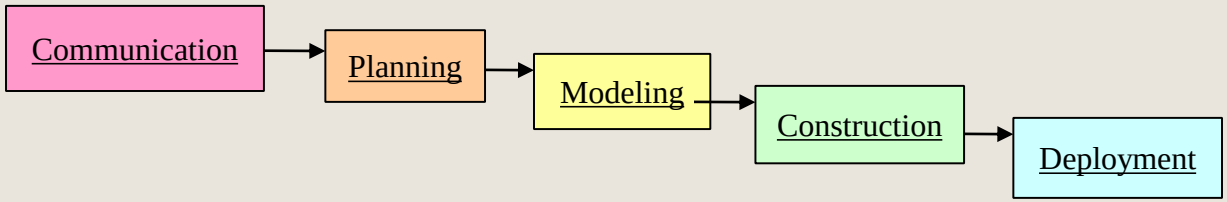
Verification: Did we build the system right?

Incremental Model (Diagram)

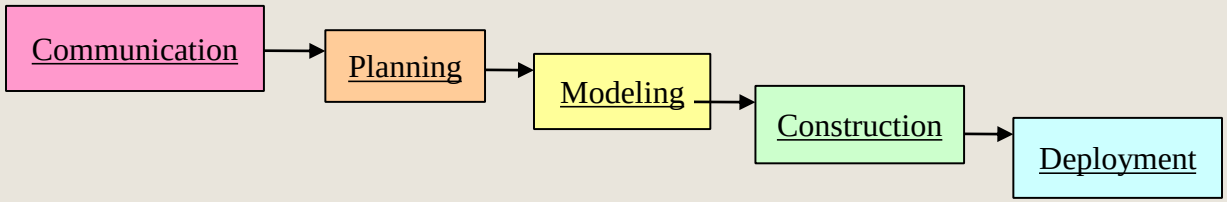
Increment #1



Increment #2

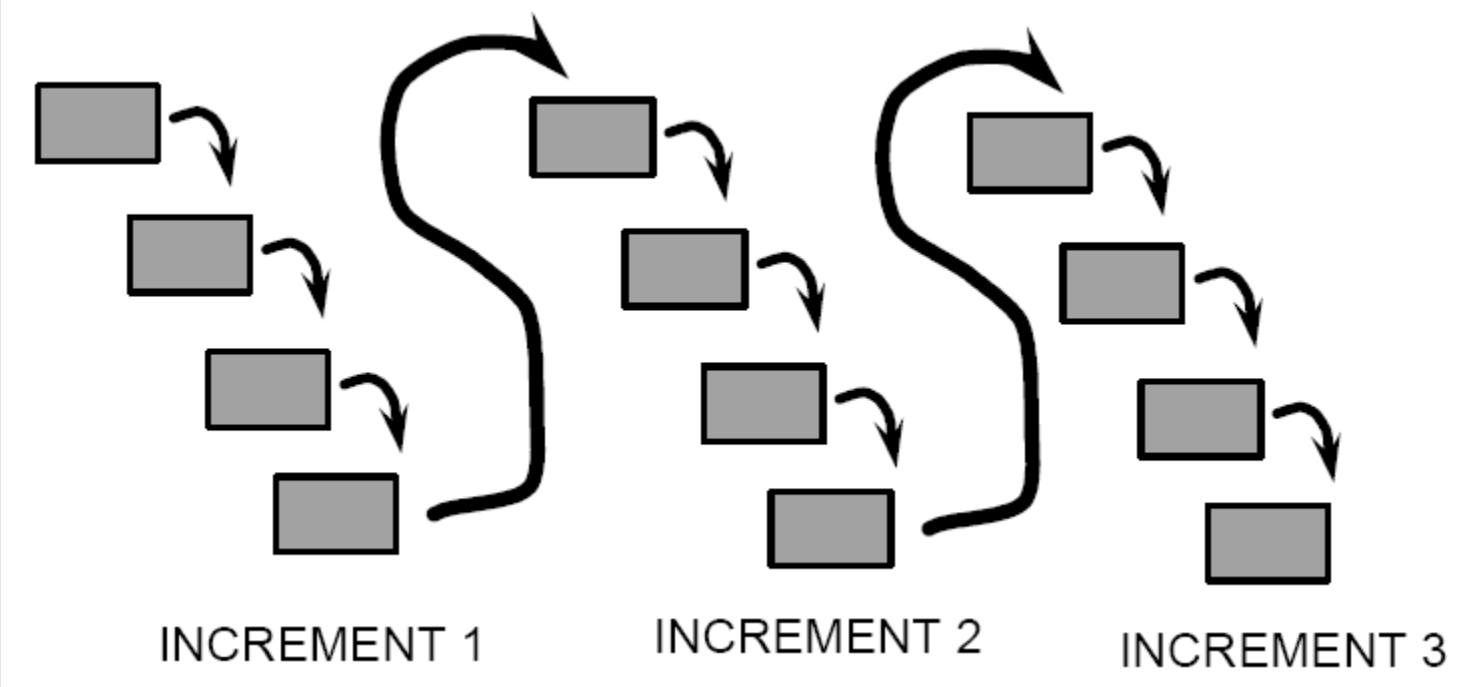


Increment #3



Incremental Model

- Each increment is a mini-waterfall.



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 rolling 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

Evolutionary Models

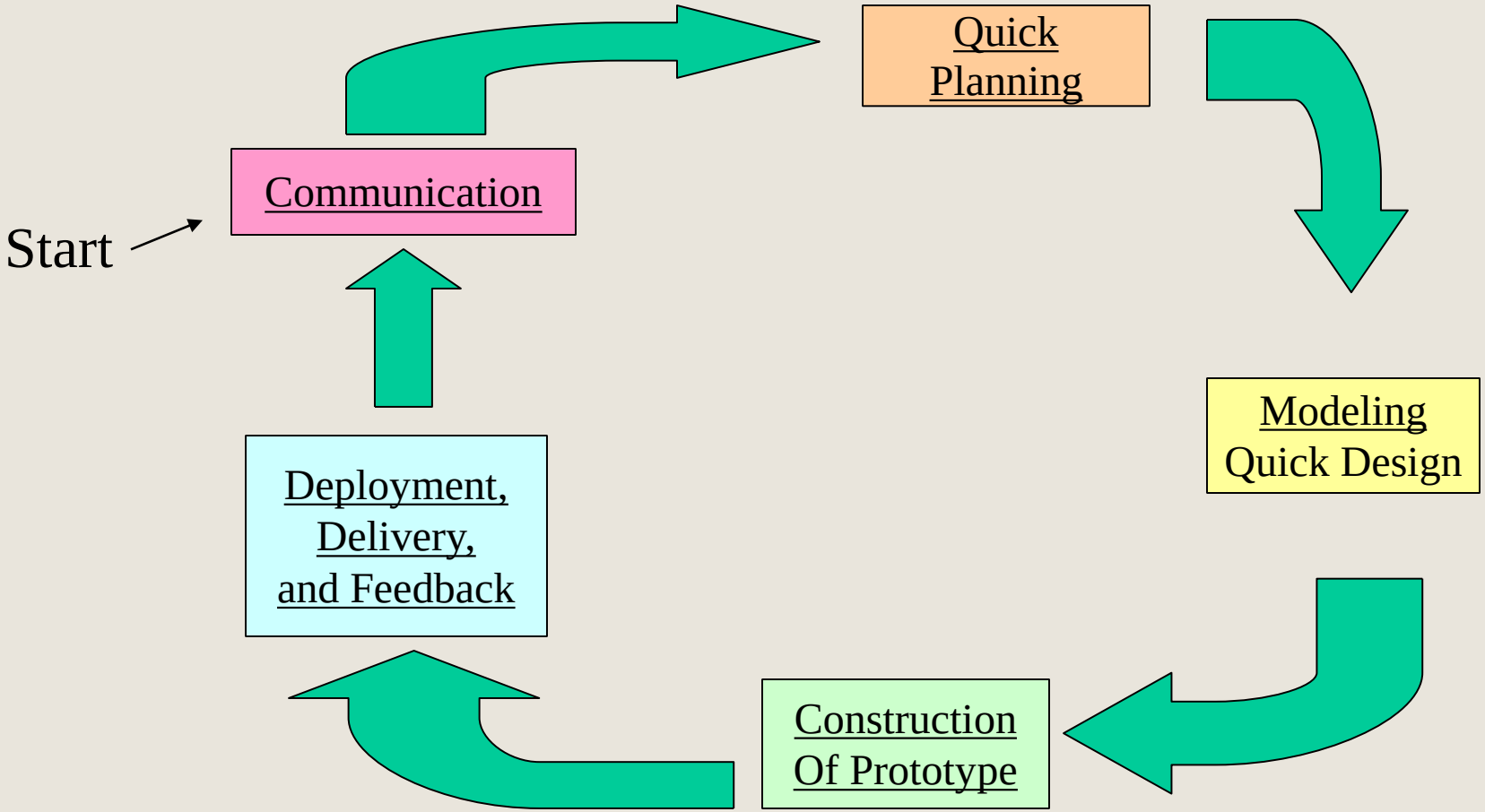
- Software **system evolves** over time as **requirements** often **change** as development proceeds. Thus, a straight line to a complete end product is not possible.

- Usually a set of **core product** or system requirements is well **understood**, but the details and **extension** have yet to be **defined**.

- You need a process model that has been explicitly designed to **accommodate** a **product that evolved** over time.

- Two types are introduced, namely **Prototyping** and **Spiral models**.

Prototyping Model



Prototyping Model

- 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

Types of prototyping-

Throwaway prototyping

- Also called **close-ended** prototyping.
- **Throwaway** or Rapid Prototyping refers to the creation of a model that will eventually be **discarded** rather than becoming part of the final delivered software
- The most obvious reason for using Throwaway Prototyping is that it can be **done quickly**
- Another strength of Throwaway Prototyping is its ability to **construct interfaces** that the users can test.

Types of prototyping-

Throwaway prototyping

- In this approach the prototype is constructed with the idea that it will be **discarded** and the **final system** will be **built from scratch**.

The steps in this approach are:

- Write **preliminary requirements**
- **Design** the **prototype**
- User **experiences/uses** the **prototype**, specifies new requirements
- **Repeat** if necessary
- Write the **final requirements**

Types of prototyping-
Evolutionary Prototyping

- The main goal when using Evolutionary Prototyping is to **build a very robust prototype** in a structured manner and **constantly refine** it.

- The reason for this is that the Evolutionary prototype, when built, forms the **heart of the new system**, and the improvements and further requirements will be built.

- When developing a system using Evolutionary Prototyping, the system is **continually refined and rebuilt**.

  "…evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood

Types of prototyping-

Incremental Prototyping

- The final product is built as **separate prototypes**. At the end the separate prototypes are **merged** in an **overall design**.

- By the help of incremental prototyping we can reduce the time gap between user and software developer

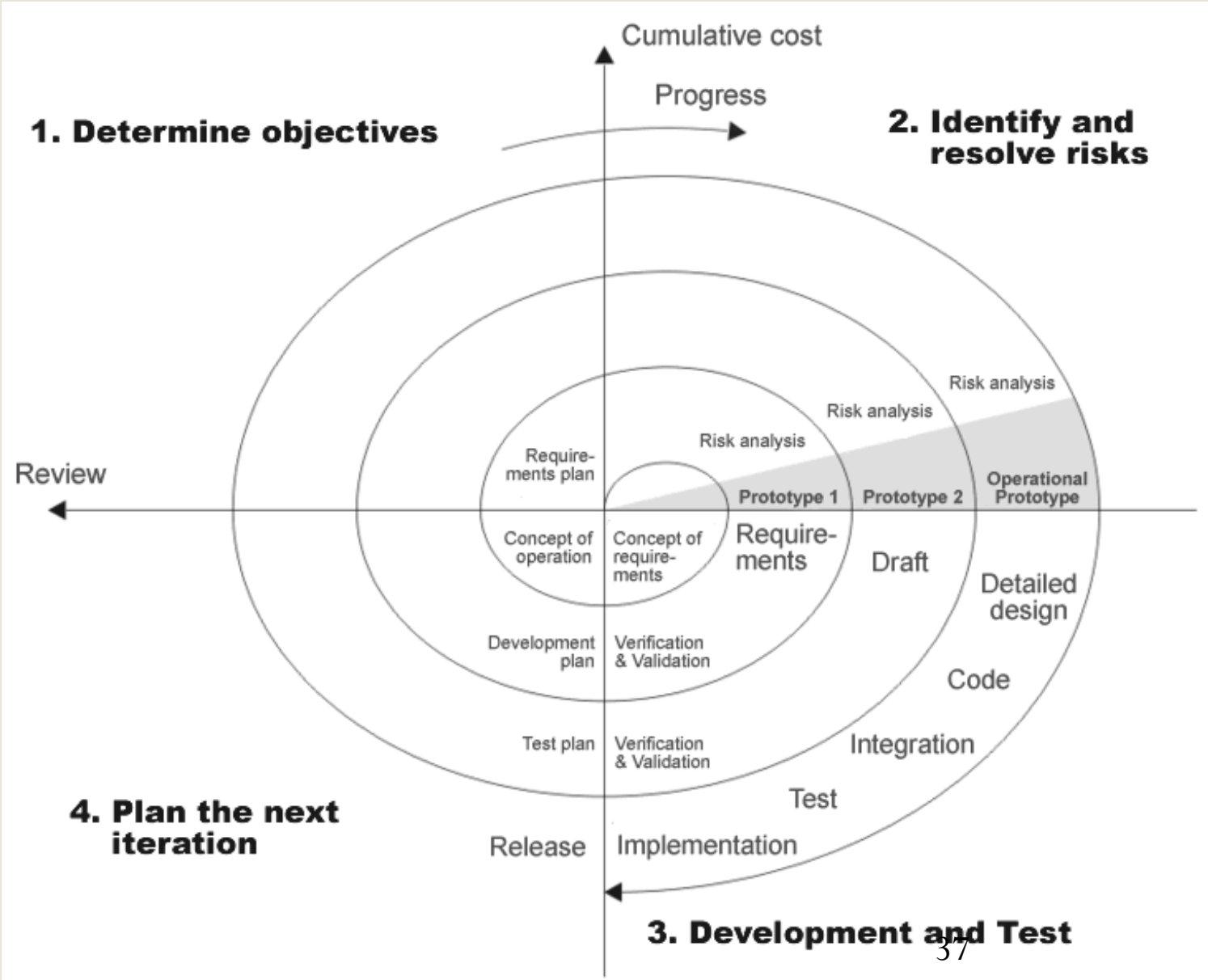
Prototyping Model (Potential Problems)

- 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

Spiral Model



Spiral Model

Extends waterfall model by **adding iteration** to explore /**manage risk**

Project risk is a moving target. Natural to progress a project cyclically in four step phases

1. Consider **alternative scenarios**, constraints
2. **Identify** and resolve **risks**
3. **Execute** the **phase**
4. **Plan next** phase:e.g. user req, software req, architecture…then goto 1

Key idea: on each iteration identify and solve the sub-problems with the ***highest risk***.

Spiral Model

- Invented by Dr. **Barry Boehm in 1988**
- 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**

Spiral Model-Applications

- For **medium** to **high-risk** projects.
- **Long-term project** commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is **not sure of their requirements** which is usually the case.
- **Requirements are complex** and need evaluation to get clarity.
- Significant **changes are expected** in the product during the development cycle.

Spiral Model (Advantages)

1. **Realism**: the model accurately reflects the iterative nature of software development on projects with unclear requirements

2. **Flexible**: incorporates the advantages of the waterfall and evolutionary methods

3. **Comprehensive** model **decreases risk**

4. **Good** project **visibility**.

Spiral Model (Disadvantages)

1. Needs **technical expertise** in risk analysis and risk management to work well.

2. Model is **poorly understood** by **non-technical management**, hence not so widely used.

3. **Complicated** model, needs competent professional management. High administrative overhead.

4. End of project may not be known early.
5. Not suitable for small or low risk projects and could be expensive for small projects.

Recommended Readings

- Chapter # 2: Process Models, **Page 30-44,** from Roger S. Pressman, Software Engineering (A Practitioners Approach), 7th Edition.

- Chapter # 2: Software Processes, **Page 48-50** from Ian Sommerville, Software Engineering, 9th Edition.