Software Engineering

CSE 305

Course Instructor

DR. HEMA KUMAR YARNAGULA

Assistant Professor

Dept. of Computer Sc and Engg., SRM University - AP

Unit – I

SOFTWARE PROCESS AND AGILE DEVELOPMENT


Software Process Models
developers
arriving
technology
design
verification
people
traceability
hours
configuration
program
developed
pursued
decentralized
workers
degree
hardware
support
organizations
activity
abstraction
modularity
test
early
eligible
time
sponsor
middleware
requirement
microcomputer
characteristics
gained
resource
term
complexity
academia
military
human
internships
knowledge
job
globalization
management
massive
science
life
profession
collaborated
debugging
world
system
process
practitioners
experience
newness
legally
elicitation
definition
application
architecture
education
curriculum
popularization
executions
appeared
software
engineer
countries
information
algorithmic
maintenance
manual
development
deal
subfield
impediments
paradigm
disciplined
systematic
partitioning
eyestrain
programmers
offshore
analysts
mid
researchers
designation
outsourcing
cycle
advances
companies
disciplines
controlling
quantified
graduate

Generic Process Model

- A generic process framework for software engineering **defines five framework activities**:

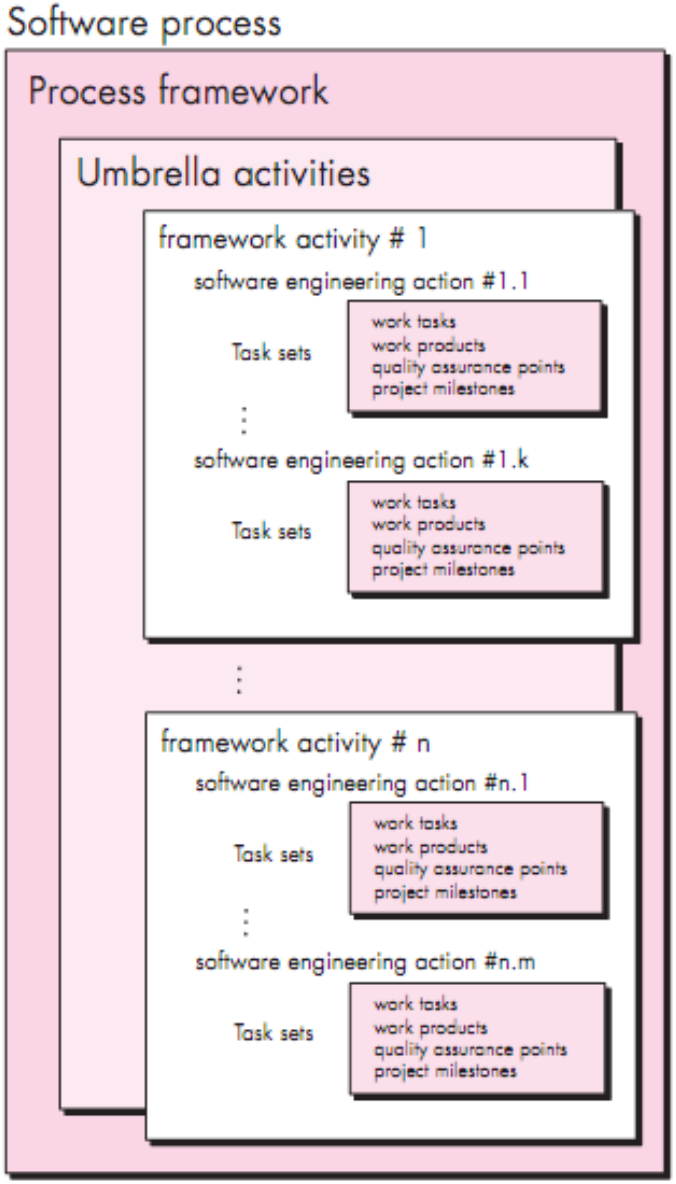
  Communication, Planning, Modeling, Construction, and Deployment.

- In addition, a **set of umbrella activities** are applied throughout the process.

Software Process Framework

FIGURE

A software process framework



Process Flow

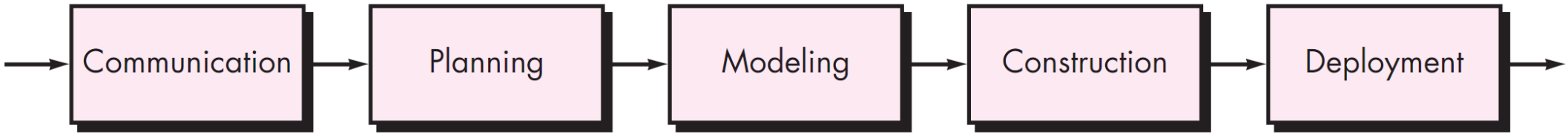
- One important aspect of the software process has not yet been discussed.

- This aspect is called - ***Process Flow***.

- Process Flow describes "**how the framework activities, actions and tasks are organized with respect to Sequence and Time**“.

Linear Process Flow

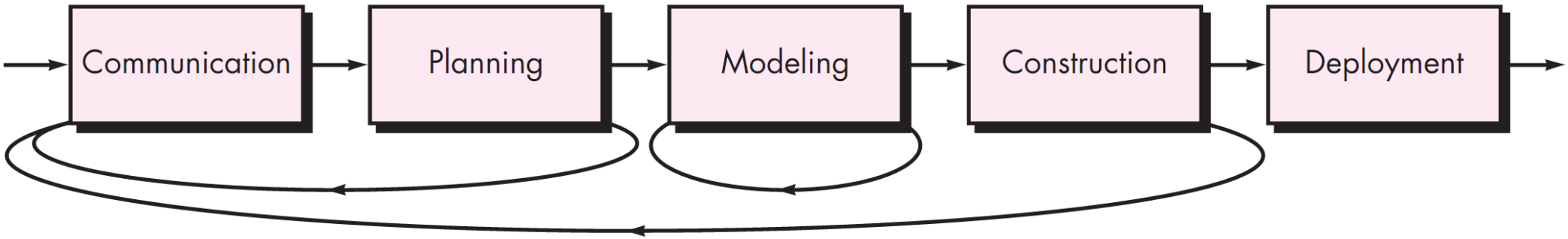
- A *linear process flow* **executes** each of the five framework activities **in sequence**.
  - beginning with communication and
  - culminating with deployment



(a) Linear process flow

Iterative Process Flow

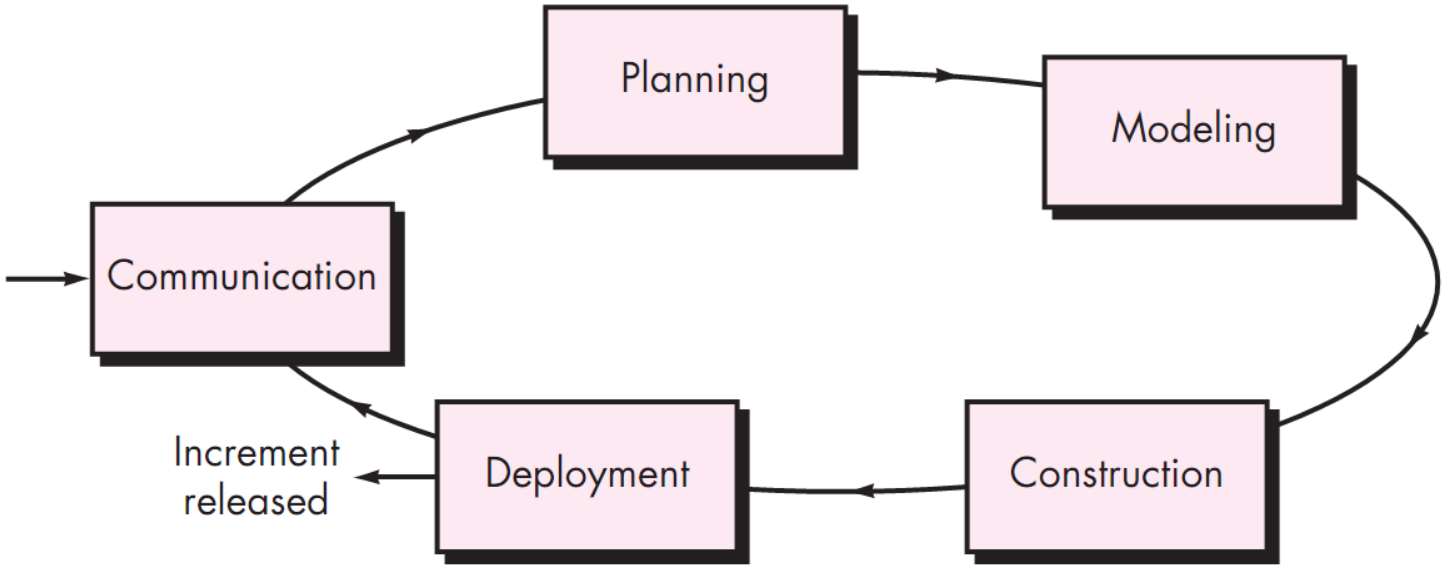
- An iterative process flow **repeats** one or more of **the activities** before proceeding to the next



(b) Iterative process flow

Evolutionary Process Flow

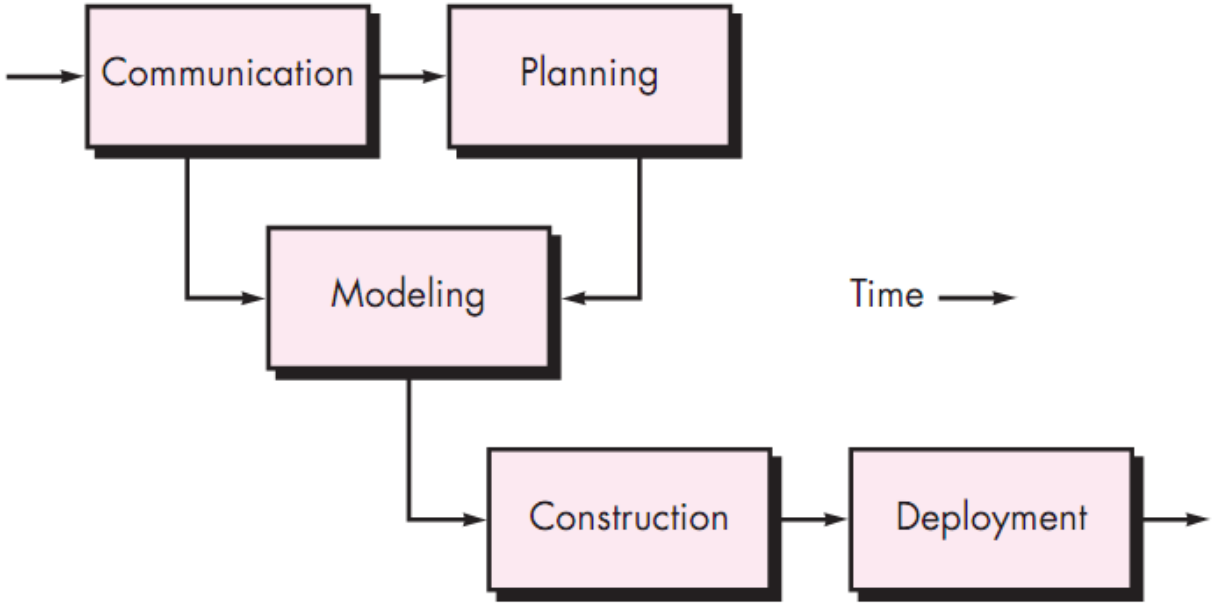
- An evolutionary process flow **executes** the **activities in a "Circular" manner**.
- **Each circuit** through the five activities **leads to** a more **complete/ newer version of the software**.



(c) Evolutionary process flow

Parallel Process Flow

- A parallel process flow **executes** one or more **activities in parallel** with other activities.



(d) Parallel process flow

e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software.

Defining a Framework Activity

Defining a Framework Activity

- We have described five framework activities and provided a basic definition so far.

  Communication, Planning, Modeling, Construction, and Deployment.

- However, a software team would **need significantly more information**
  - **When:** before it could properly execute any one of these activities
  - **Where:** as part of the software process.

Defining a Framework Activity

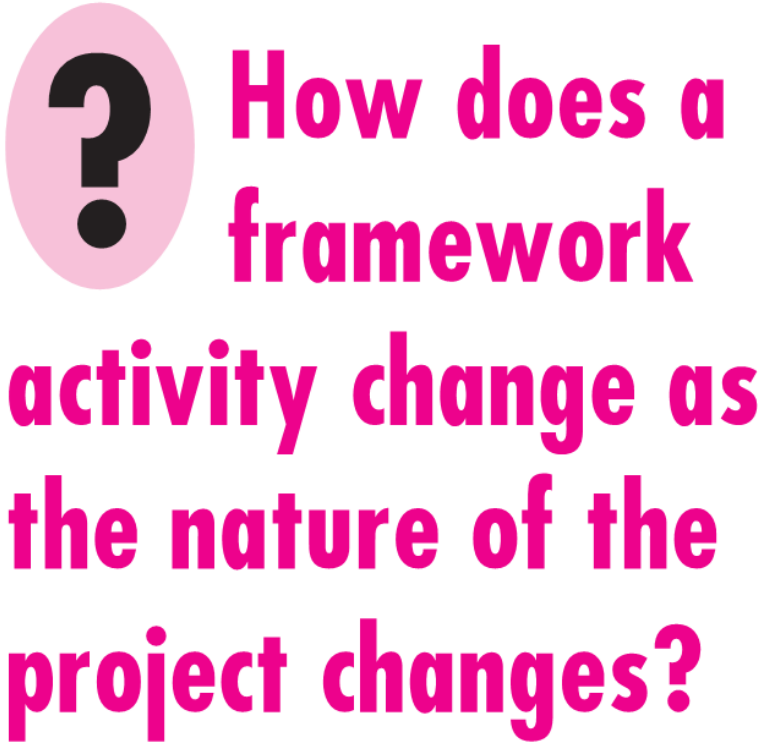
- Therefore, we are faced with a key question:

What actions are appropriate for a framework activity, given:

- the **nature of the problem** to be solved,
- the **characteristics of the people** doing the work, and
- the **stakeholders** who are **sponsoring the project**

Defining a Framework Activity

- In short, we want to know:



Defining a Framework Activity

- For a **small software project**:
  - requested by one person (at a remote location)
  - with simple, straightforward requirements

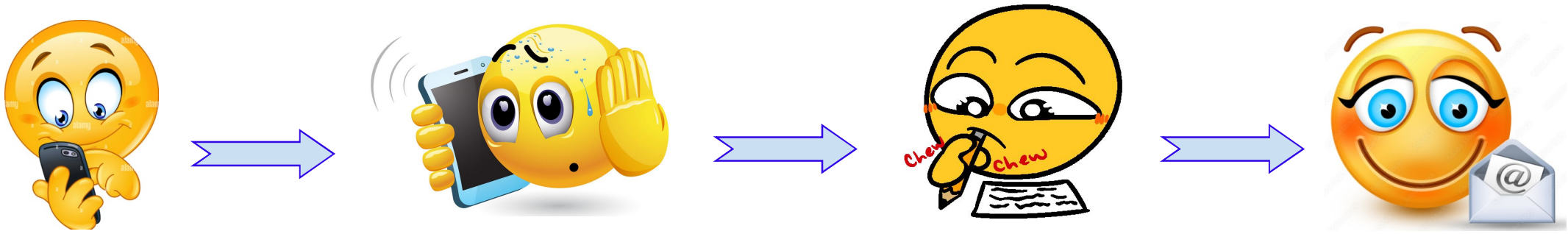
- The communication activity might **encompass** little more than a **phone call.**

- Therefore, the only **necessary Action is a Phone Conversation**

Defining a Framework Activity

- The **work tasks** (the task set) that this action encompasses are:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.



Defining a Framework Activity

- If the **project** was **considerably more complex:**
  - with many stakeholders
  - each with a different set of (*sometime conflicting*) requirements

- Then the communication activity might **have Six distinct Actions**:
  1. **inception**
  2. **elicitation**
  3. **elaboration**
  4. **negotiation**
  5. **specification, and**
  6. **validation**

Defining a Framework Activity

- Each of these Six Actions would have:
  - many work tasks and
  - a number of distinct work products/outcome.

Note: *We will have an insightful discussion on them later.*

Identifying a Task Set

Identifying a Task Set

- Different projects demand different task sets.

- The software team chooses the task set based on problem and project characteristics.

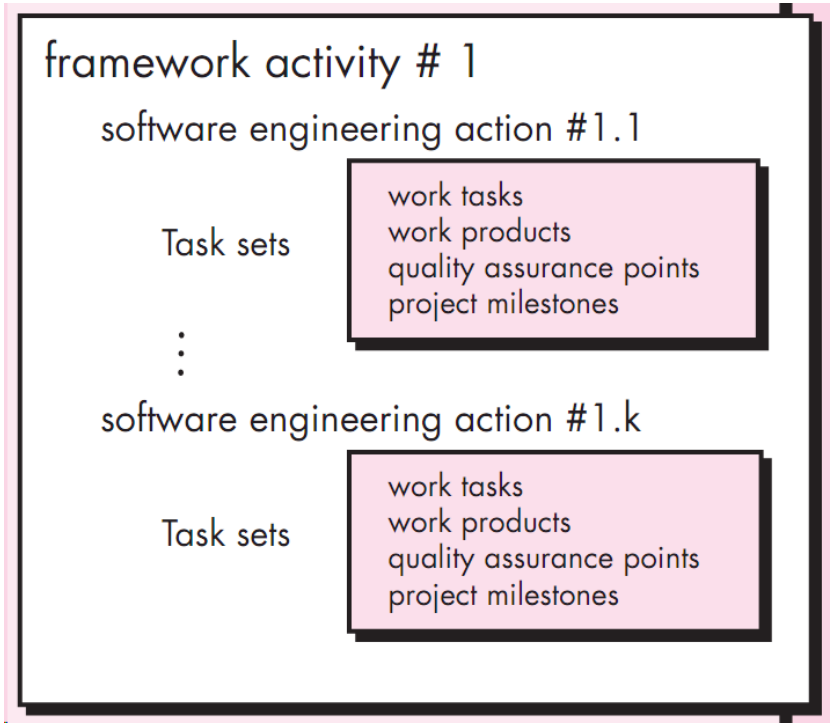
Identifying a Task Set

- Each **Action** can be **represented** by a number of **different task sets**

- We should **choose a task set** that best accommodates:
  - the needs of the software project and
  - the characteristics of the project team

- This implies that:

  "Action can be adapted to the specific needs"



Process Patterns

Process Patterns

A **Process Pattern** describes:

- a **process-related problem** that is **encountered** during software engineering work
- **identifies the environment** in which the problem has been encountered
- **suggests** one or more **proven solutions** to the problem.

Process Patterns

In short,

- Patterns can be used to describe a problem (and solution) associated with:
  - a framework **activity** (e.g., planning) or
  - an **action** within a framework activity (e.g., project estimating)

Process Patterns

- Ambler, in 1998, has **proposed a template** for describing a process pattern:

- It consists of three things:
  1. **Pattern Name**
  2. **Forces**
  3. **Type**

Process Patterns

Pattern Name

- The pattern is given a meaningful name
- Describing it within the context of the software process
- **e.g.,** TechnicalReviews

Process Patterns

Forces describes

- The environment in which the **pattern is encountered**

- The issues that make the problem visible and may affect its solution.

Process Patterns

Type

- The pattern type is specified

- Ambler suggests three types:
  1. **Stage pattern**
  2. **Task pattern**
  3. **Phase pattern**

Stage Pattern

- It defines a problem associated with a **framework activity**
- Since a framework activity encompasses multiple actions and work tasks
- A stage pattern **incorporates multiple task patterns**

Task Pattern

- It defines a **problem associated** with an **Action or Work Task**
- e.g., **RequirementsGathering** is a task pattern.

Phase Pattern

- It define the **sequence of framework activities that occurs** within the process
- Even when the overall flow of activities is iterative in nature.
- An example: **SpiralModel** or **Prototyping**.

Process Assessment and Improvement

Process Assessment and Improvement

The existence of a **Software Process** is **may not guarantee**

- that software will be delivered on time,
- that software will meet the customer's needs,
- that software will exhibit the technical characteristics, or
- that will lead to long-term quality characteristics

Process Assessment and Improvement

- Process itself can be **assessed**

Why:

- to ensure that it meets a set of basic process criteria
- It is essential for a successful software engineering

Process Assessment and Improvement

Assessment attempts to understand the current state of the software process with the intent of improving it.

Process Assessment and Improvement

What formal techniques are available for assessing the software process?

Process Assessment and Improvement

- A number of different approaches to software process assessment and improvement have been proposed:
  1. **Standard CMMI Assessment Method for Process Improvement (SCAMPI)**
  2. **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**
  3. **SPICE (ISO/IEC15504)**
  4. **ISO 9001:2000 for Software**

Standard CMMI Assessment Method for Process Improvement (SCAMPI)

- This provides a five-step process assessment model

- It incorporates five phases:
  1. initiating
  2. diagnosing
  3. establishing
  4. acting, and
  5. learning

- The SCAMPI method uses the SEI CMMI as the basis for assessment

CMM-Based Appraisal for Internal Process Improvement (CBA IPI)

- It provides a diagnostic **technique for assessing the relative maturity** of a software organization

- It uses the SEI CMM as the basis for the assessment

SPICE (ISO/IEC 15504)

- It is an ISO standard
- It defines a **set of requirements** for software process assessment.
- The intent is to **assist organizations in developing an objective evaluation of the efficacy** of any defined software process

ISO 9001:2000 for Software

- This is a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides.

- Therefore, the standard is **directly applicable to software organizations** and companies.

Prescriptive Process Models

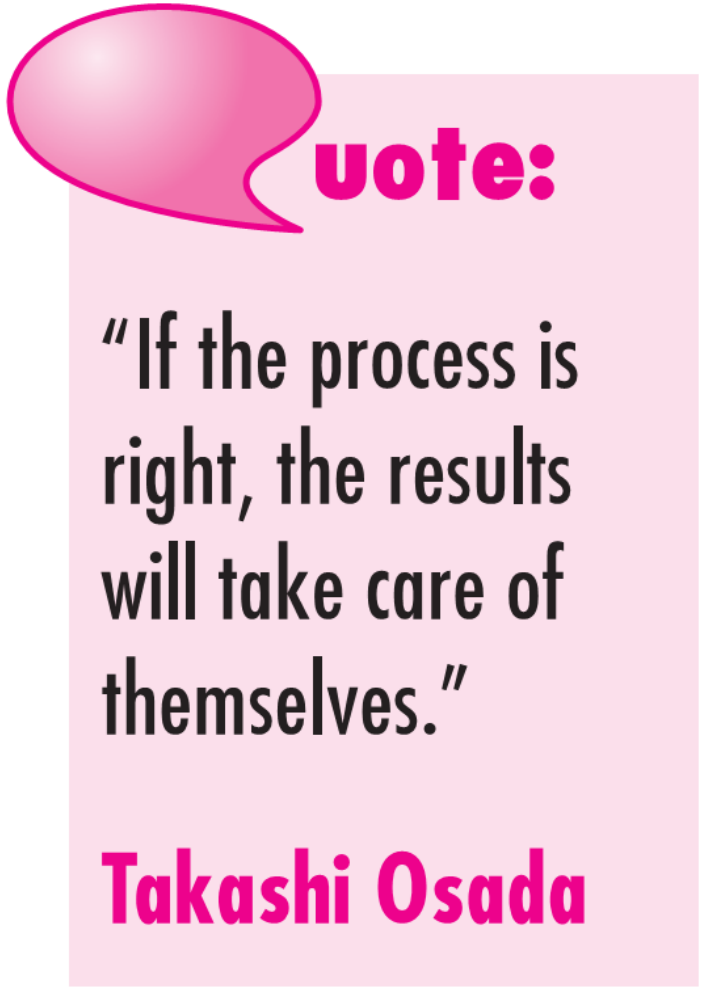
Prescriptive Process Models

- They were originally **proposed to bring order to the chaos** of software development.

These traditional models have:

- **brought** a certain amount of **useful structure** to software engineering work
- **provided a reasonably effective road** map for software teams.

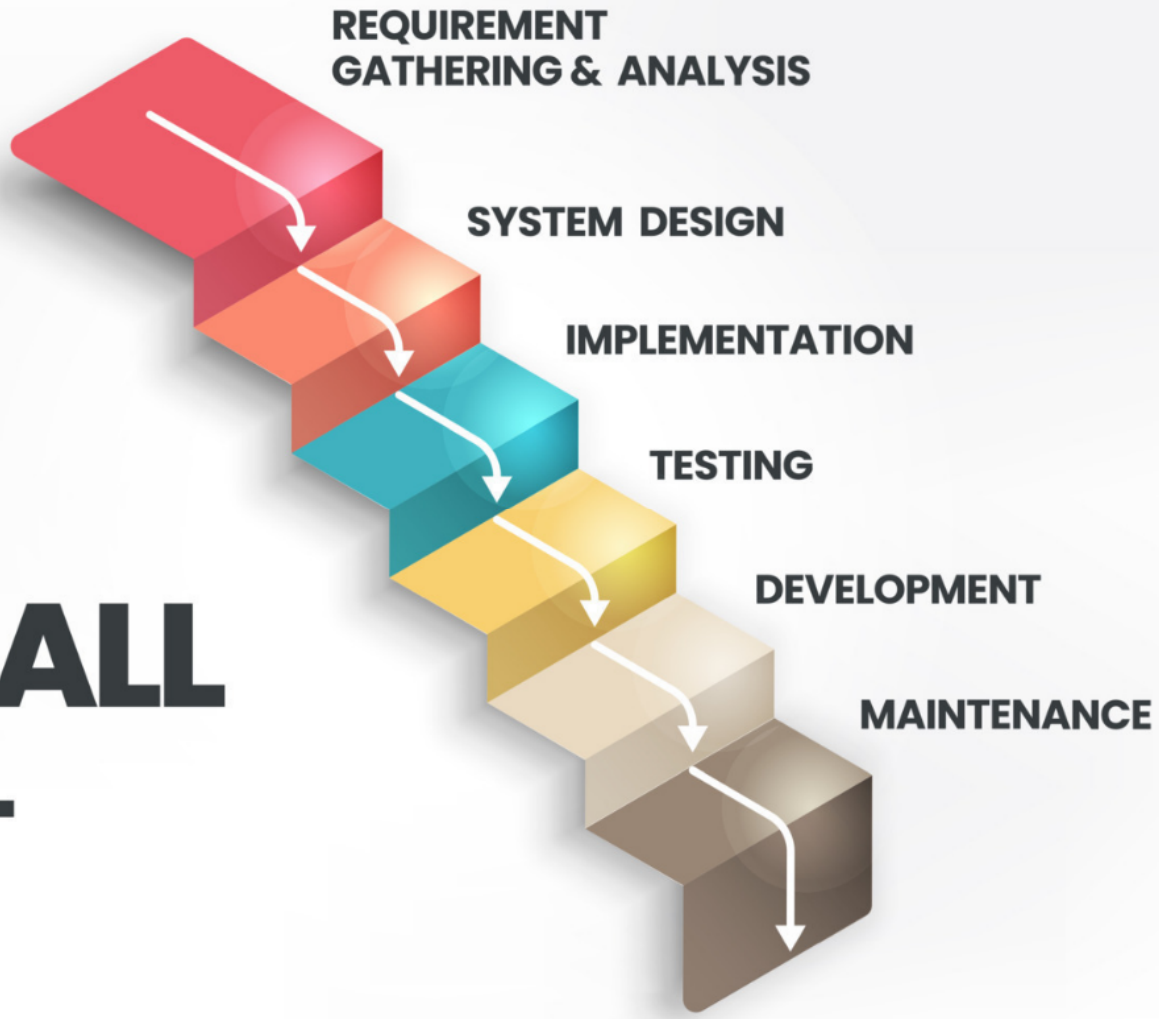
Prescriptive Process Models



Prescriptive Process Models

Prescriptive process models define a prescribed set of process elements and a predictable process work flow.

WATERFALL MODEL



The Waterfall Model

- The waterfall model, sometimes called the ***classic life cycle***

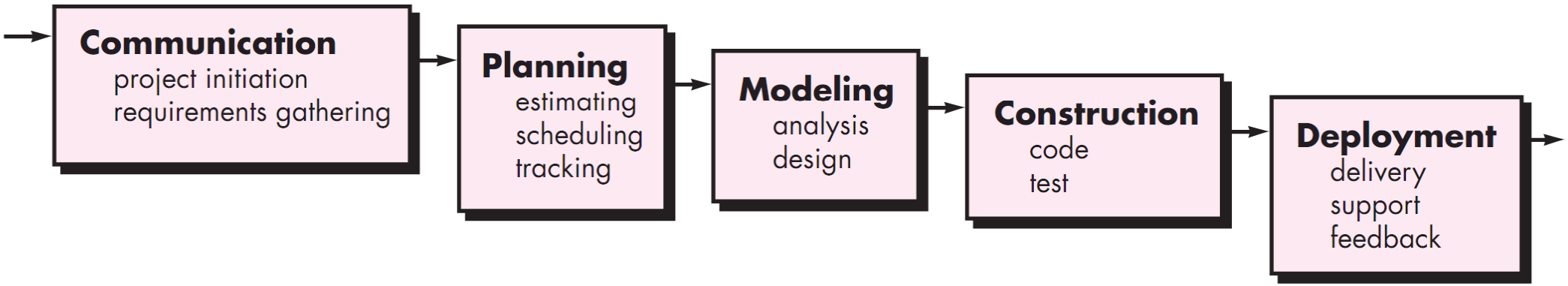
- It suggests a **systematic, sequential approach** to software development.

The Waterfall Model

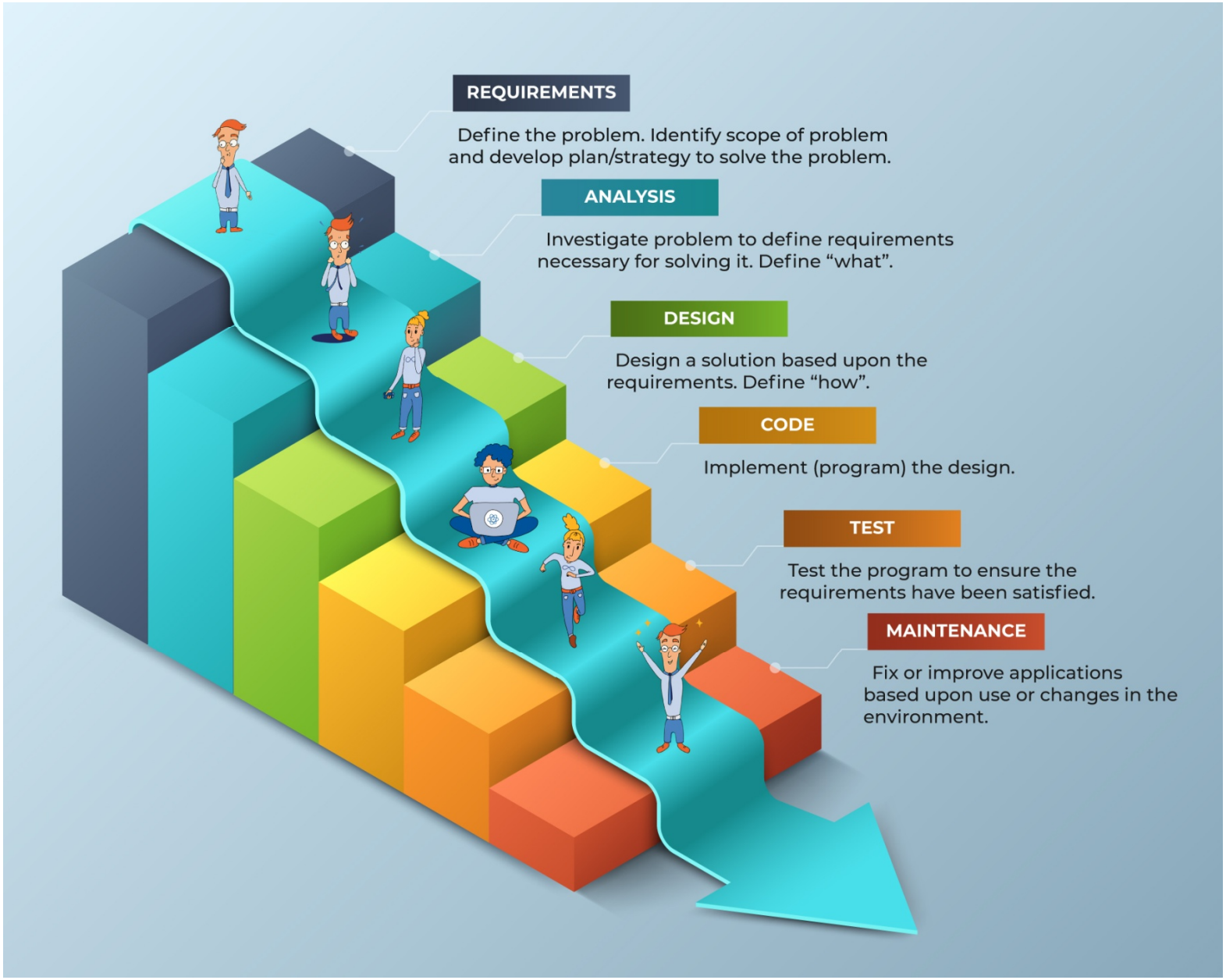
- It begins with customer specification of requirements
- Then progresses through planning, modeling, construction, and deployment,
- Finally, culminating in ongoing support of the completed software

The Waterfall Model

FIGURE The waterfall model



Waterfall Model



When to use Waterfall Model?

- Due to its nature, it is **hard to get back** to the previous phase once completed.

- This is the list of things to consider when using the waterfall:
  - The requirements are clear and frozen.
  - Technology is understood and used by the team in different projects.
  - The project cannot be delivered in an iterative manner.
  - Documentation is essential.
  - Professional Project management skills.
  - The project cost is defined.

Advantages of Waterfall Model

- Disciplined and Structured Approach
- Stages and activities are well defined.
- Clear goals - Each phase has specific deliverables.
- Logical and intuitive progression
- Easier for project managers to plan, schedule the project
- No special methodology training

Disadvantages of Waterfall Model

- **Inflexible:** It is very difficult to go back to any phase after it finished.
- **Lack of client input:** It assumes that the requirements of a system can be frozen without any changes or enhancements.
- It **takes the full lifecycle** to deliver a workable solution to the customer.
- A little **flexibility and adjusting scope is difficult and expensive**.
- It delays the testing phase which can discover a lot of issues in requirements, design, and implementation as well.

V Model
In Software Development

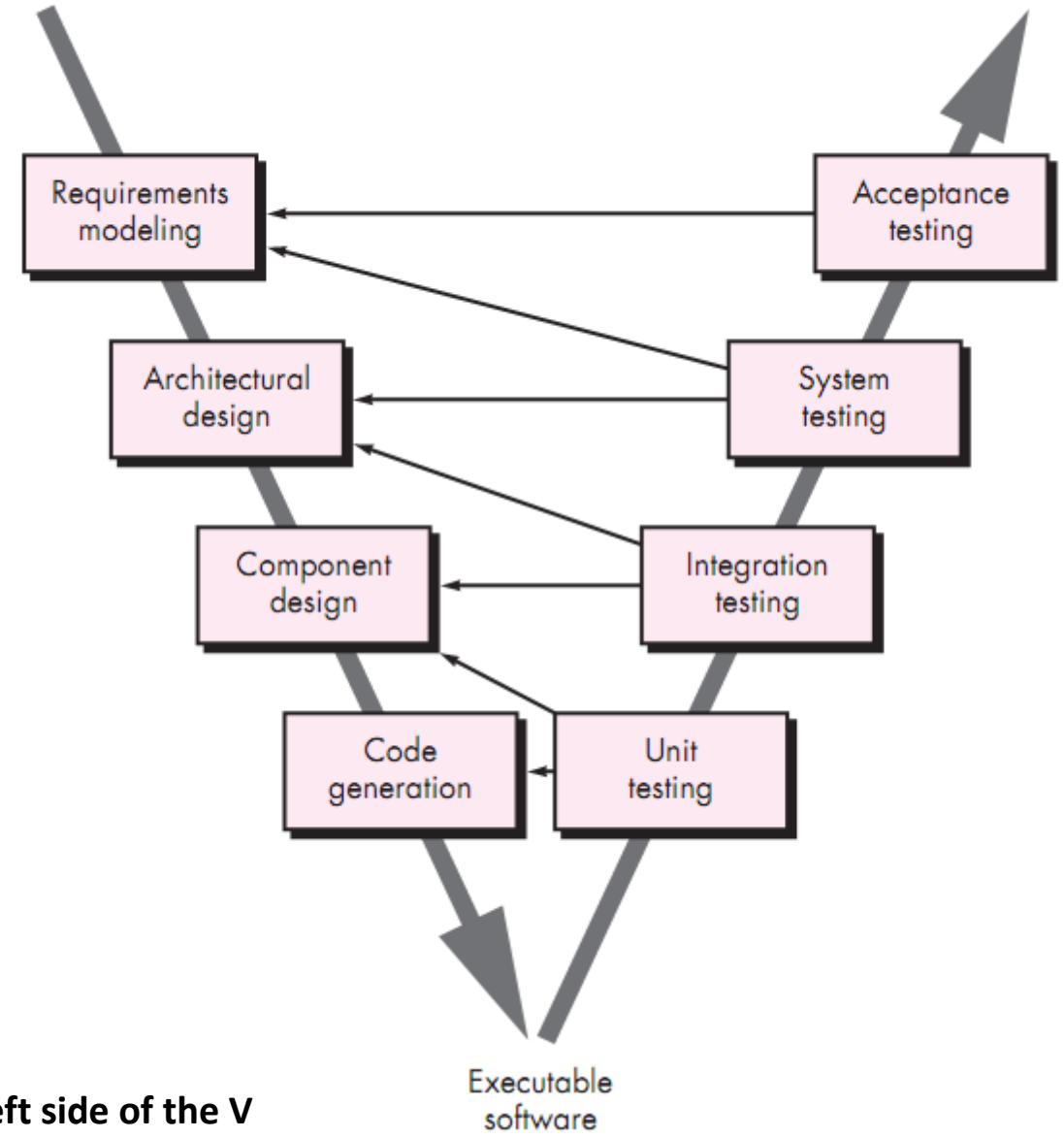
The V-model

- A **variation in the representation** of the waterfall model is called the **V-model**.
- It is also known as the **Verification and Validation model**.
- It is based on the **association of a testing phase** for each corresponding development stage.

The V-model

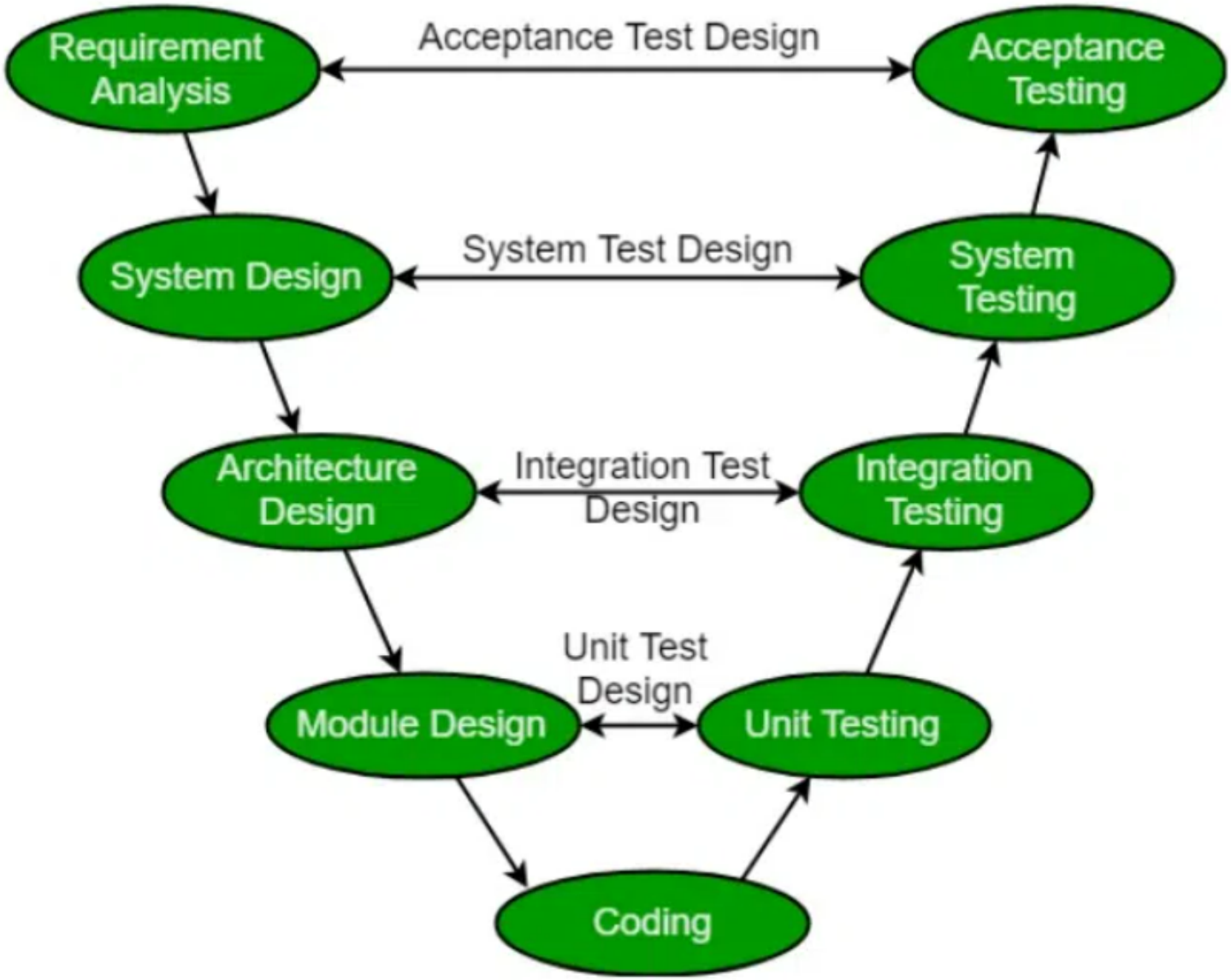
- The development of each step is directly associated with the testing phase.
- The next phase starts only after completion of the previous phase
- But, for **each development activity, there is a testing activity** corresponding to it.

The V-model



A Software Team moves down the left side of the V

The V-model


Requirement Analysis
Acceptance Test Design
Acceptance Testing
System Design
System Test Design
System Testing
Architecture Design
Integration Test Design
Integration Testing
Unit Test Design
Module Design
Unit Testing
Coding

The V-model

In short,

- V-model **depicts the relationship** of quality assurance actions to the actions associated with communication, modeling, and early construction activities.

Incremental Model

Incremental Process Models

The incremental model delivers a series of releases, called increments, that provide progressively more functionality for the customer as each increment is delivered.

Incremental Process Models

When this model is Suitable?

- Initial software requirements are reasonably well defined
- But the overall scope of the development effort **precludes a purely linear process**.

- There may be a compelling need:
  - to **provide a limited set** of software functionality to users quickly
  - then **refine and expand** on that functionality in later software releases.

Incremental Process Models

- Incremental model **combines** elements of both:
  - linear process flows
  - parallel process flows

- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.

- Each linear sequence **produces deliverable "increments"** of the software

- The increments produced by an evolutionary process flow.
- For example, ***word-processing software***

Incremental Process Models

- In incremental model, the first increment is often a **Core Product**.

- Only the basic requirements are addressed
- But many supplementary features (some known, others unknown) remain undelivered.

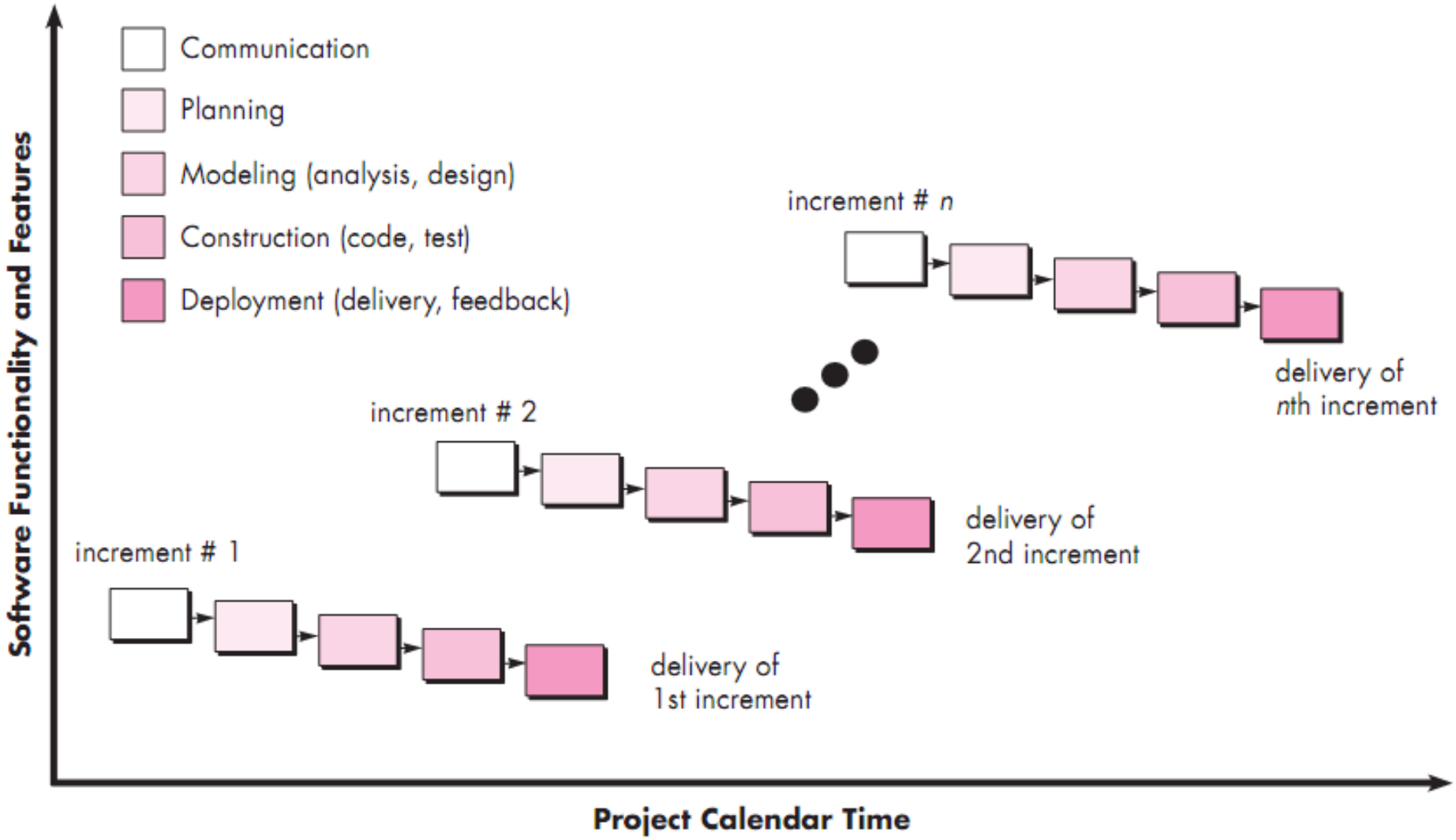
Incremental Process Models

- The core product is used by the customer
  (or undergoes detailed evaluation)

- As a result of evaluation, a plan is developed for the next increment.

- This **process is repeated** following the delivery of each increment, until the **complete product is produced**

The incremental model



Evolutionary Model in Software Engineering

Evolutionary Process Models

Evolutionary process models produce an increasingly more complete version of the software with each iteration.

Evolutionary Process Models

- Software, like all **complex systems**, **evolves over a period of time**.

In Complex Software Systems:

- Requirements **often change** as development proceeds;
- Making a straight line path to an end product **unrealistic**;
- Tight market deadlines make completion of a comprehensive software **impossible**;

Evolutionary Process Models

- But a **limited version must be introduced** to meet competitive or business pressure

In this case:

- A set of core product or system requirements is well understood
- But the details of product or system extensions have yet to be defined.

Evolutionary Process Models

In these situations, we need a process model that:

- **Explicitly designed to accommodate a product that evolves over time.**

- **Hence, Evolutionary models are iterative.**

EVOLUTIONARY PROCESS MODEL

01 Software evolves over a period of time, and hence evolutionary models are more suited to software development.

02 They enable the software developer to develop more complex versions of the software increasingly.

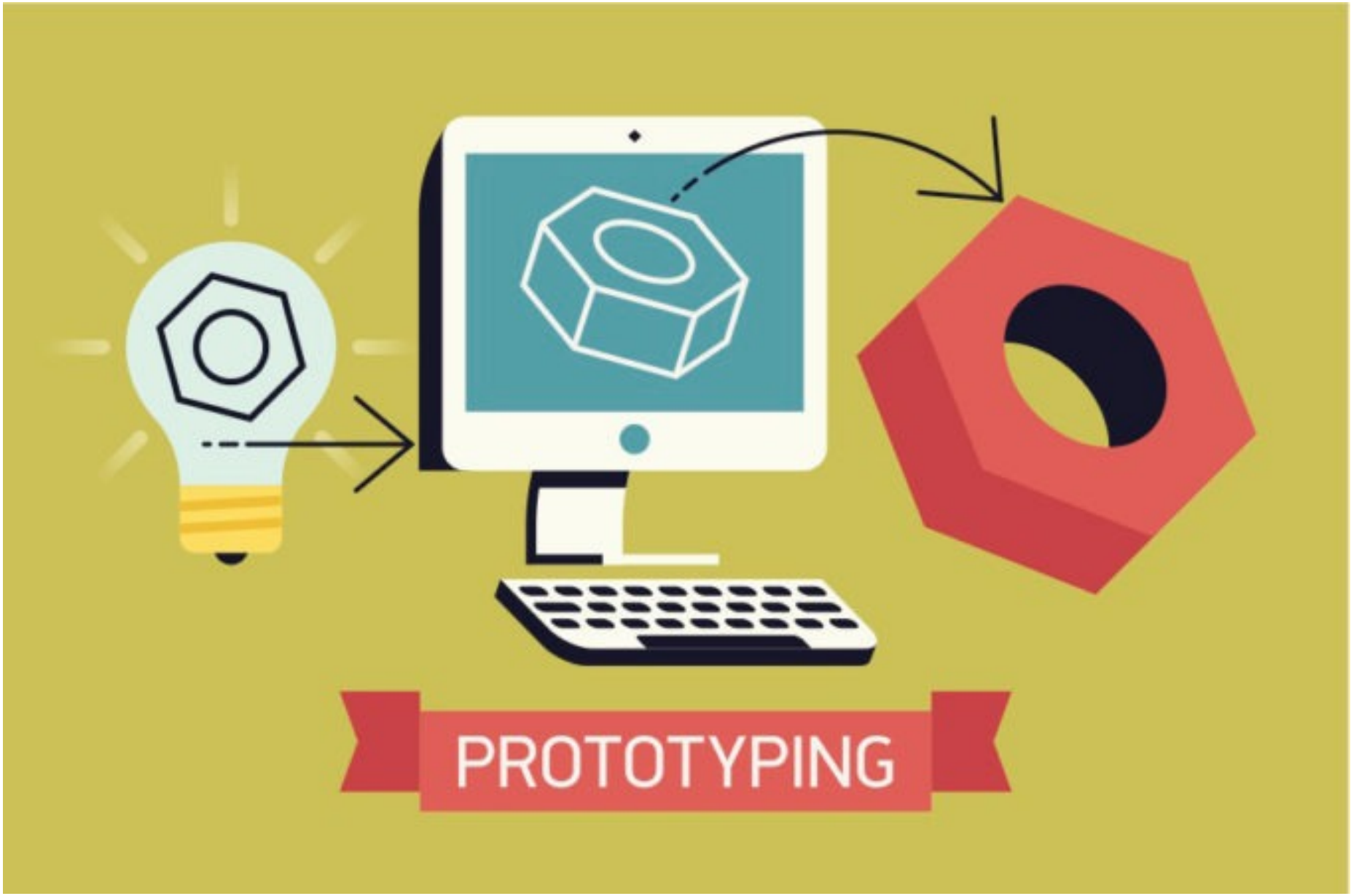
03 These models are more suited to object-oriented systems.

04 Requirements change while software gets developed.

05 They are iterative.

Evolutionary Process Models

- We will present two common evolutionary process models.
  1. **Prototyping.**
  2. **The Spiral Model.**


PROTOTYPING

Prototyping

When your customer has a legitimate need, but is clueless about the details, develop a prototype as a first step.

Prototyping

Case 1:

- Customer defines a set of general objectives for software
- But does not identify detailed requirements (functions and features).

Case 2:

- The developer may be unsure of:
  - the efficiency of an algorithm;
  - the adaptability of an operating system; or
  - the form that human-machine interaction should take.

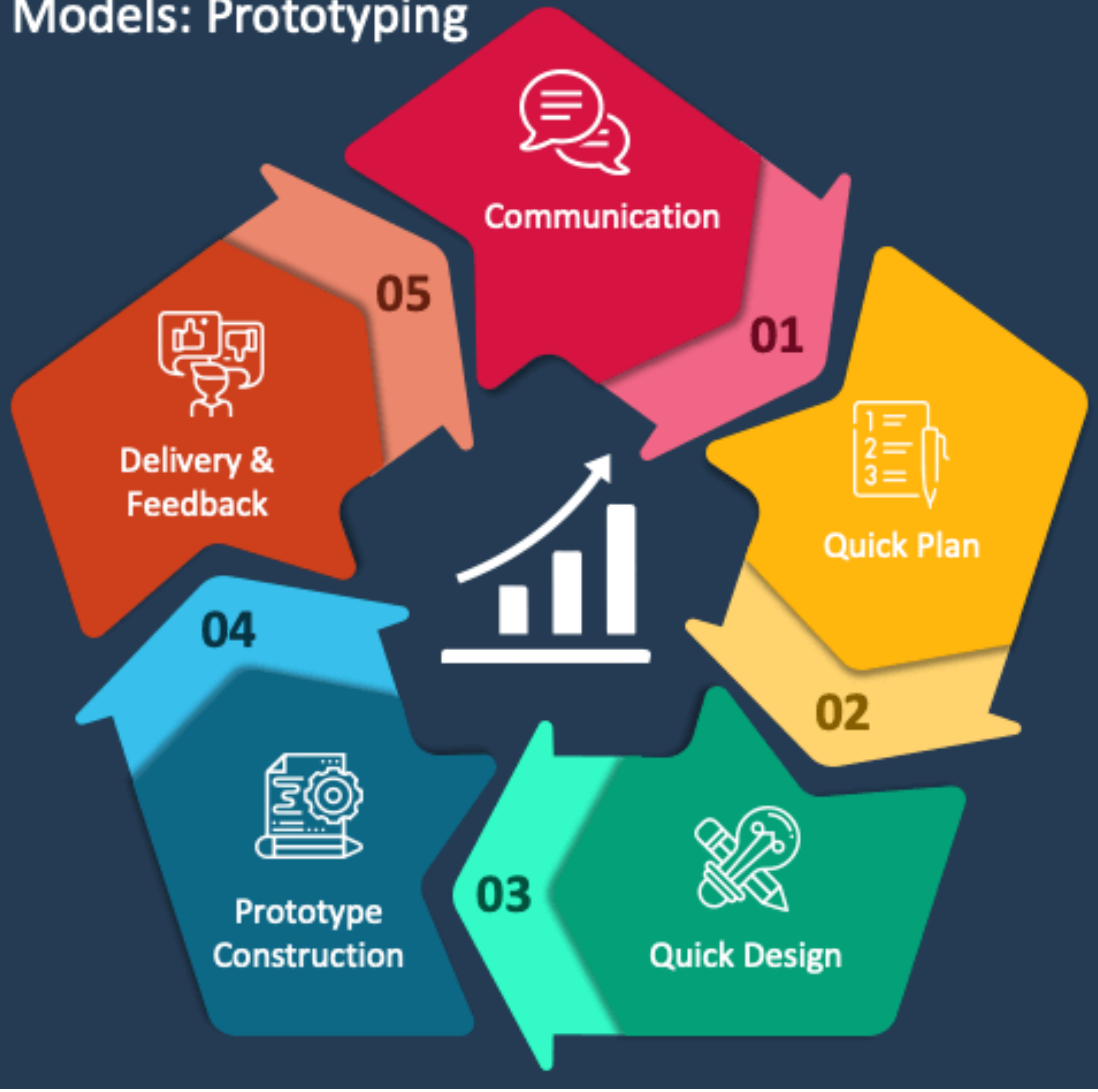
Prototyping

- In these situations, a **prototyping paradigm** may offer the best approach.

- The prototyping paradigm **assists to better understand** what is to be built when **requirements are fuzzy**.

EVOLUTIONARY PROCESS MODEL

Evolutionary Process Models: Prototyping



Prototyping

- The prototyping paradigm begins with communication.
  - Meet with stakeholders to define the overall objectives
  - Identify whatever requirements are known
  - Outline areas where further definition is mandatory.

- Then a quick design leads to the construction of a prototype.

Prototyping

- The prototype is **deployed and evaluated** by stakeholders
- Stakeholders **provide feedback** that is **used to further refine** requirements.
- **Iteration occurs** as the prototype is **tuned to satisfy the needs** of stakeholders

Problems with Prototyping

Prototyping can be **problematic** for the following reasons:

1. Stakeholders see what appears to be a working version of the software
   - unaware that the prototype is held together haphazardly
   - unaware that in the rush to get it working
   - haven't considered overall software quality or long-term maintainability.

2. As a software engineer, we often make implementation compromises in order to get a prototype working quickly.

Prototyping

- Although problems can occur, prototyping can be an effective paradigm for software engineering.

“The key is to define the rules of the game at the beginning;”

SPIRAL MODEL IN SDLC

The Spiral Model

- Originally proposed by **Barry Boehm in 1988**.

- The spiral model is an evolutionary software process model

- It **couples the iterative nature of prototyping** with the **controlled and systematic aspects** of the waterfall model.

The Spiral Model

- A spiral model is divided into a set of framework activities
  - defined by the software engineering team.

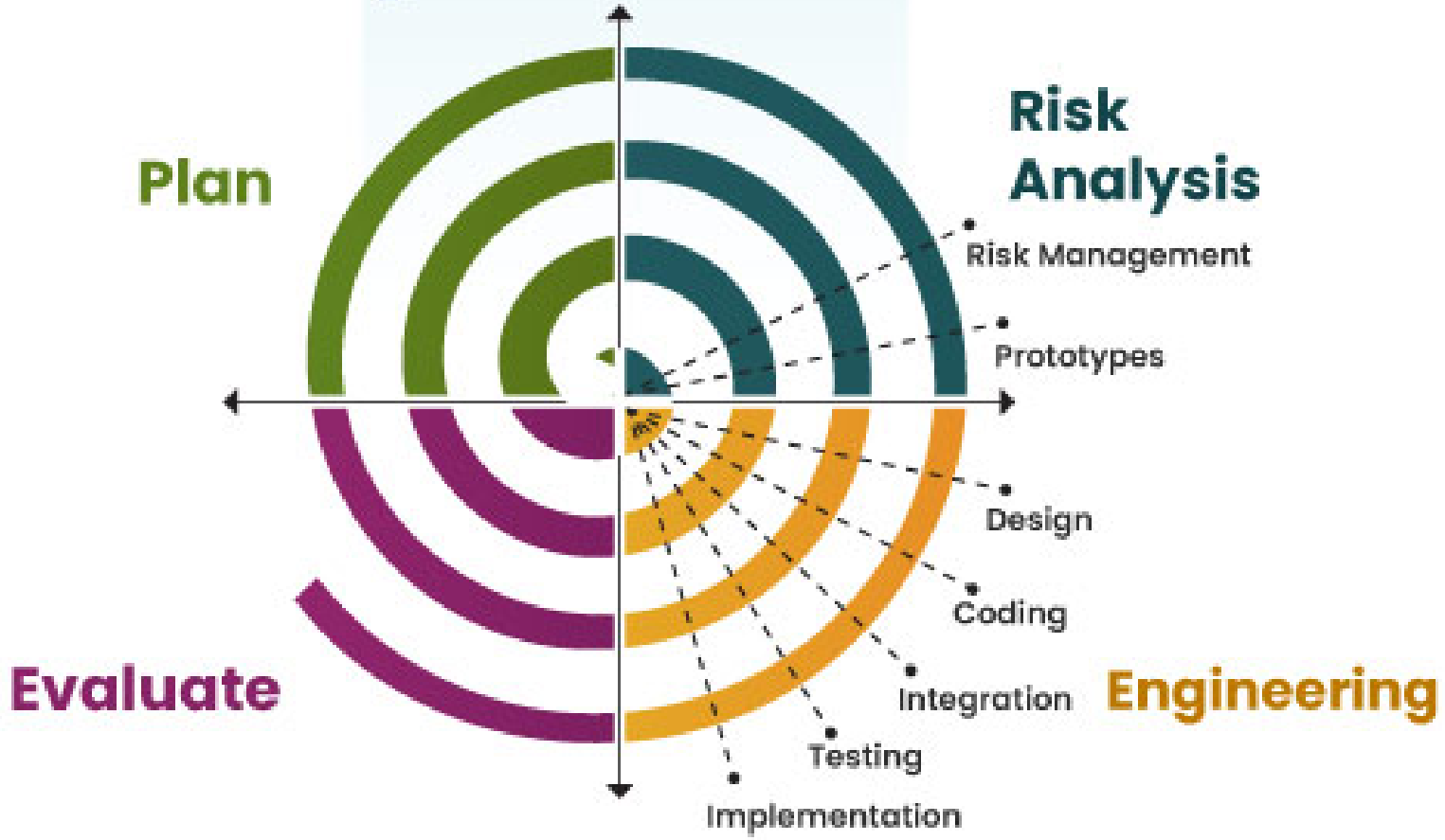
- Each of the framework activities represent one segment of the spiral path

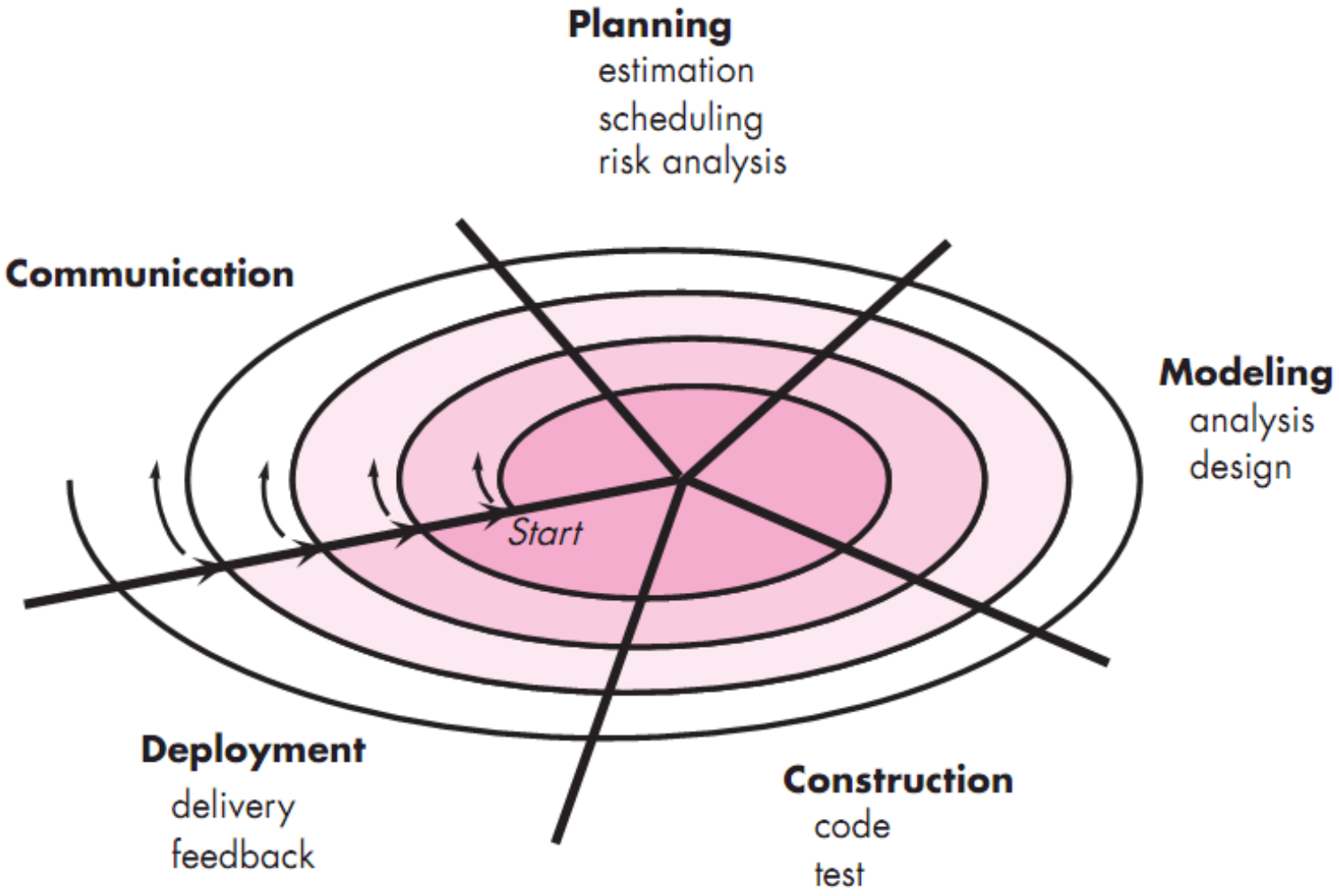
[*illustrated in Figure, next slide*] →

Phase Of
Spiral Method

What Is

Spiral Model





Note: If we use generic framework activities, for illustrative purposes.

The Spiral Model

- Unlike other process models that end when software is delivered,
- The spiral model **can be adapted to apply throughout the life** of the computer software.

The spiral model can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance.

The Spiral Model

If your management demands fixed-budget development (generally a bad idea), the spiral can be a problem. As each circuit is completed, project cost is revisited and revised.

The Spiral Model - Cons

- The spiral model is a **realistic approach** to the development of **large-scale software**

- But like other paradigms, the **spiral model is not a panacea**.

Panacea: a universal remedy

- It may be **difficult to convince customers**
  - particularly in contract situations

Concurrent Process Models

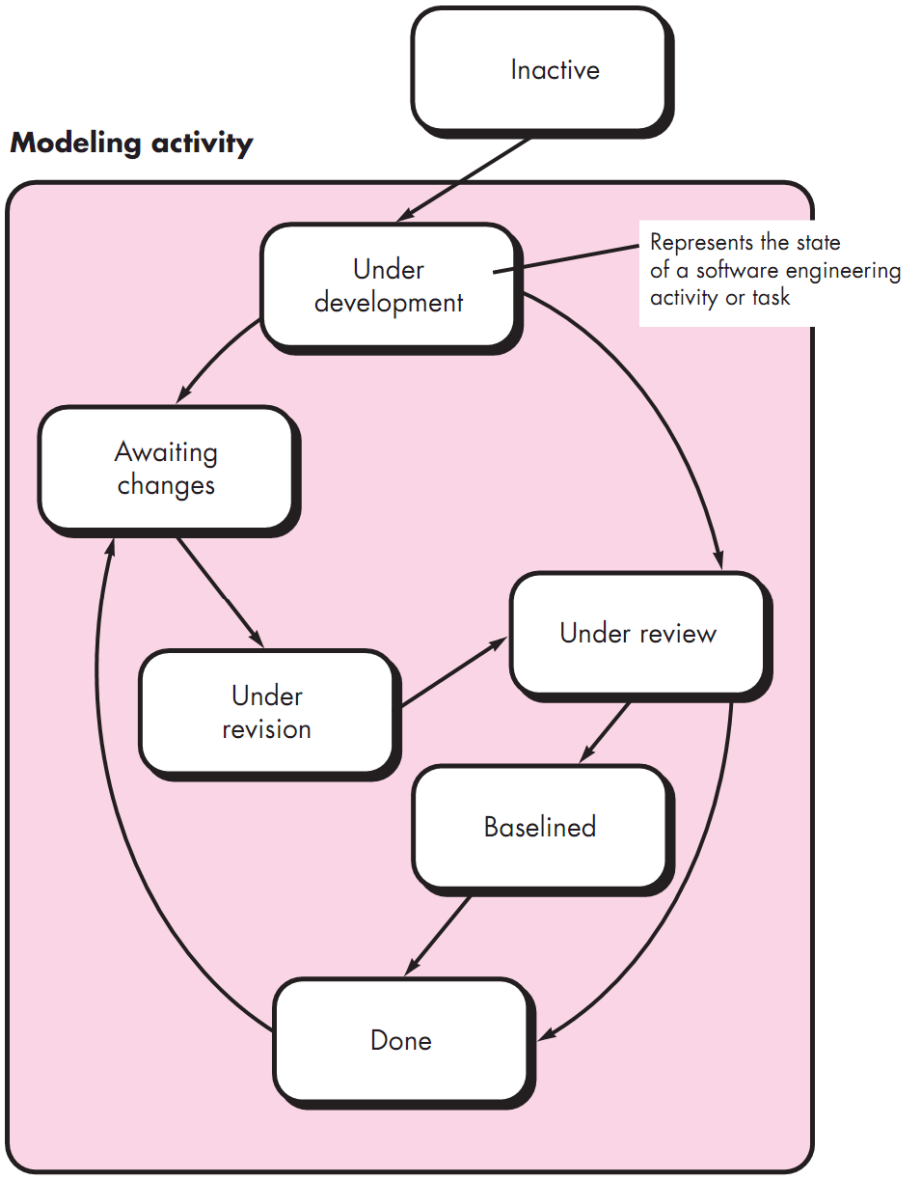
Concurrent Process Models

The concurrent model is often more appropriate for product engineering projects where different engineering teams are involved.

Concurrent Process Models

- This is sometimes called ***concurrent engineering***
- This **allows** a software team to **represent iterative and concurrent elements** of any of the process models
- All software engineering **activities exist concurrently** but **reside in different states**.

Concurrent Process Models



Concurrent Process Models

In short,

- Concurrent modeling is applicable to all types of software development

- Provides an **accurate picture of the current state** of a project.

Concurrent Process Models

- Rather than confining activities, actions, and tasks to a **sequence of events**
- It defines a **process network**.
- Events generated at one point in the process network **trigger transitions** among the states.

A Final Word on Evolutionary Processes

A Final Word on Evolutionary Processes

- Modern computer software is **characterized by**:
  - continual change;
  - very tight time lines; and
  - an emphatic need for customer–user satisfaction.

A Final Word on Evolutionary Processes

- In many cases, **time-to-market** is the most important management requirement.

- If a **market window is missed**, the software **project itself** may be **meaningless**.

A Final Word on Evolutionary Processes

- Evolutionary process models were conceived to address these issues

- Yet they too have **weaknesses**.

A Final Word on Evolutionary Processes

- The intent of evolutionary models is to develop high-quality software

- The **challenge for software teams and their managers** is to **establish a proper balance** between:
  - Critical project/product parameters
  - Customer satisfaction (the ultimate arbiter of software quality).

Specialized Process Models

Specialized Process Models

- They take on **many of the characteristics** of one or more of the traditional models

- However, they tend to be applied when a **specialized or narrowly** defined software engineering approach is chosen.

Specialized Process Models

- Some of the Specialized Process Models are:
  - **Component-Based Development**
  - **The Formal Methods Model**
  - **Aspect-Oriented Software Development**

Component-Based Development

Component-Based Development

- **Commercial off-the-shelf (COTS)** software components
- They are developed by vendors who offer them as products
- They **provide targeted functionality** with well-defined interfaces

Component-Based Development

- Component-based development model constructs applications from **prepackaged software components**.

- These components can be **designed** as either:
  - **conventional software modules or**
  - **object-oriented classes or**
  - **packages of classes.**

Component-Based Development

- It incorporates many of the characteristics of the spiral model.
- It is evolutionary in nature

Component-Based Development

Advantages:

- The model leads to software **reuse**
- Reusability provides software engineers with a number of measurable benefits.
- If component reuse becomes part of the culture, team can achieve:
  - **a reduction in development cycle time**
  - **a reduction in project cost**

The Formal Methods Model

The Formal Methods Model

- It **encompasses a set of activities** that leads to **formal mathematical specification** of computer software.

- Formal methods enable you to **specify, develop, and verify** a computer-based system **by applying** a rigorous, **mathematical notation**.

The Formal Methods Model

- A variation on this approach, called:

  Cleanroom Software Engineering

- It is currently applied by some software development organizations.

Advantage of Cleanroom Software Engineering

- It **allows errors to be found earlier** in the lifecycle
  - Which minimizes expensive rework later on
  - Speeds time to market.

- **How:** through the application of mathematical analysis.

- Designs are simplified, straightforward, and verifiable
- This resulting in less "**spaghetti**" code.

- **Quality is achieved by design and verification, not testing.**

The Formal Methods Model

If formal methods can demonstrate software correctness, why is it they are not widely used?

The Formal Methods Model

- The formal methods model **NOT a mainstream approach**
  - Although it offers the **promise of defect-free software**.

- Yet, there is **concern about its applicability**:
  - The development is currently quite time consuming and expensive.
  - Extensive training is required, to apply formal methods.
  - Difficult to use as communication mechanism for technically unsophisticated customers.

The Formal Methods Model

- However, these **concerns notwithstanding**
- The approach has **gained adherents** among software developers
- Who must build safety-critical software
  - developers of aircraft avionics
  - developers of medical devices
- They would **suffer severe economic hardship** should software errors occur.

Aspect-Oriented Software Development (AOSD)

Aspect-Oriented Software Development (AOSD)

- As modern computer-based systems become more sophisticated (and complex)
- Certain ***concerns*** span the entire architecture.
- Some concerns are high-level properties of a system
- Other concerns affect functions

Aspect-Oriented Software Development (AOSD)

- When concerns cut across multiple system functions, features, and information, they are often referred to as **Crosscutting Concerns**.

- **Aspectual requirements define** those crosscutting concerns

Aspect-Oriented Software Development (AOSD)

- AOSD often referred to as **Aspect-Oriented Programming (AOP)**
- AOSD is a relatively **new software engineering paradigm**
- **AOSD provides a process and methodological approach** for defining, specifying, designing, and constructing aspects

Aspect-Oriented Software Development (AOSD)

- A distinct aspect-oriented process has **not yet matured**.
- However, it is likely that such a process will **adopt characteristics** of both **evolutionary and concurrent process models**.
- *A detailed discussion of AOSD is best left to students, if you have further interest.*

Rational Unified Process

The Unified Process

- Unified Process is sometimes called **Rational Unified Process (RUP)**

- Named after **Rational Corporation** (subsequently acquired by IBM)
  - an early contributor to the development and refinement of the UP
  - a builder of complete environments (tools and technology)

- The **Unified Process** is an attempt to draw on the **best features and characteristics** of traditional software process models

The Unified Process

- The Unified Process recognizes the importance of:
  - customer communication *and*
  - streamlined methods

Recognizes for what:

- For describing the customer's view of a system
- We call them as the ***Use Case***

Use Case *describes a system function or feature from the user's point of view.*

The Unified Process

In short,

- It **helps** the architect **to focus on the right goals**, such as:
  - understandability,
  - reliance to future changes, and
  - reuse

The Unified Process

A Brief History

- During the early 1990s, a team began working on a “unified method” for object-oriented modeling.

- The result was **UML - a Unified Modeling Language**

The Unified Process

- **UML contains a robust notation** for the modeling and development of object-oriented systems.

- By 1997, UML became a **de facto industry standard** for object-oriented software development.

- Today, the **Unified Process (UP) and UML are widely used** on **object-oriented projects** of all kinds.

Phases of the Unified Process

Phases of the Unified Process

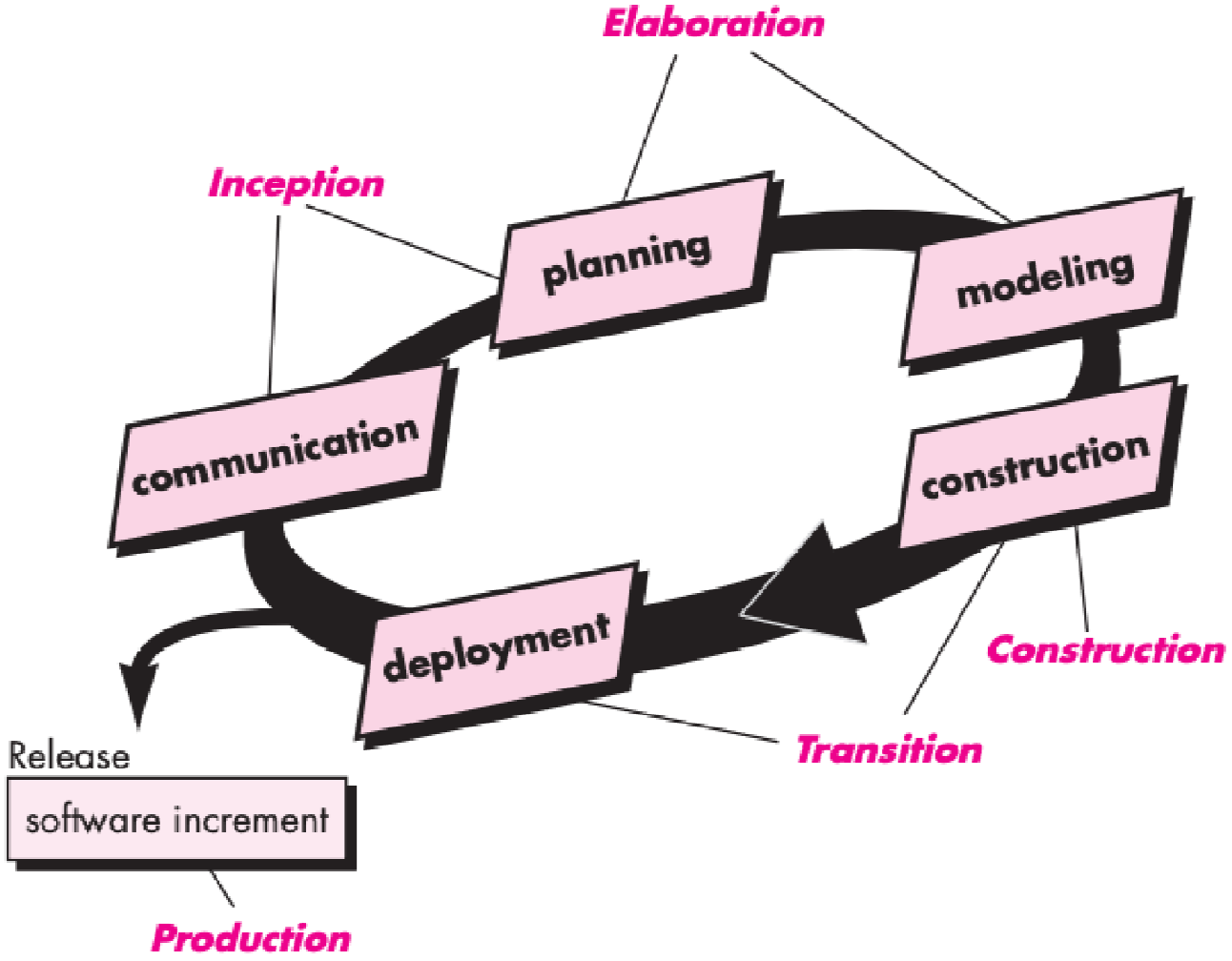
- The **Five Generic Framework Activities** may be used to describe any software process model.
- **The Unified Process is no exception.**
- Next, we will list the **“phases” of the UP** and relates them to the generic activities

Phases of the Unified Process

The phases of the Unified Process are:

- The inception phase
- The elaboration phase
- The construction phase
- The transition phase
- The production phase

Phases of the Unified Process



The inception phase

- The inception phase of the UP encompasses both
  - customer communication and
  - planning activities

What is Done:

- By collaborating with stakeholders, business requirements are identified;
- A rough architecture for the system is proposed;
- A plan for iterative, incremental nature of the ensuing project is developed.

The inception phase

Make a Note:

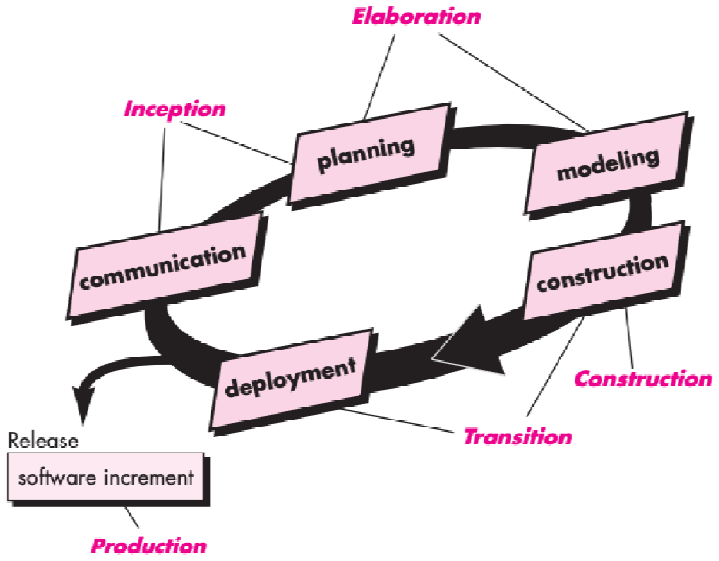
- Fundamental business requirements are described through a set of preliminary use cases

- Architecture at this point is nothing more than a tentative outline of major subsystems
- Later, the architecture will be refined and expanded

- Planning identifies resources, assesses major risks, defines a schedule
- It also establishes a basis for the phases that are to be applied as the software increment is developed.

The elaboration phase

- The elaboration phase encompasses both:
  - communication activities
  - modeling activities



What is Done:

- Elaboration refines and expands the preliminary use cases
- Expands architectural representation to include 5 different views of software
- The plan is carefully reviewed at the culmination of the elaboration phase
- Modifications to the plan are often made at this time.

The elaboration phase

- The five different views of software are:
  - the use case model
  - the requirements model
  - the design model
  - the implementation model
  - the deployment model

- The plan is carefully reviewed to ensure that scope, risks, and delivery dates remain reasonable.

The construction phase

- The construction phase of the UP is identical to the construction activity

What is Done:

- Using the architectural model as input, necessary and **required features are then implemented as source code**.
- As components are being implemented, **unit tests are designed** and executed for each.
- In addition, **integration activities** (component assembly and integration testing) are conducted.

The transition phase

- The transition phase of the UP encompasses
  - The latter stages of the generic construction activity
  - The first part of the generic deployment (delivery and feedback) activity.

What is Done:

- Software is given to end users for **beta testing**
- User feedback **reports** both **defects and necessary changes**.
- Software team **creates the necessary support information** required for release.

  (e.g., user manuals, troubleshooting guides, installation procedures)
- At the conclusion, the software becomes a usable software release.

The production phase

- The production phase **coincides with deployment activity** of the generic process.

What is Done:

- During this phase, the ongoing use of the **software is monitored**
- Support for the operating environment (infrastructure) is provided
- **Defect reports** and requests for changes are **submitted and evaluated**.

Phases of the Unified Process

- It should be noted that **not every task identified for a UP** workflow is **conducted** for every software project.

- The **team adapts the process** (actions, tasks, subtasks, and work products) **to meet its needs**.

Process Technology

Process Technology

- The process technology also called
  - **Process modeling tools**
  - **Process management tools**

- **Objective:** If an organization works to improve a software process, it must first understand it.

Process Technology

- The Tools allow a team to **define elements of unique process model**
- The elements include: *activities*, *actions, tasks, work products, QA points*
- Tools can be **used to allocate, monitor, and even control** all software engineering activities defined as part of the process model.
- They can also be used to coordinate the use of other software engineering tools that are appropriate for a particular work task.

Process Technology

Representative Tools:[25]

Igrafx Process Tools—tools that enable a team to map, measure, and model the software process (**www.micrografx.com**)

Adeptia BPM Server—designed to manage, automate, and optimize business processes (**www.adeptia.com**)

SpeedDev Suite—a collection of six tools with a heavy emphasis on the management of communication and modeling activities (**www.speedev.com**)

The Duality of Product and Process

- If the process is weak, the end product will undoubtedly suffer.
- But an obsessive over-reliance on process is also dangerous.

thank you