Fundamental software engineering concepts

Please read Chapters 1, 5, 6 & 7

Dr. Abdelkarim Erradi

CSE @ QU

Acknowledgement

Books on which the slides are based:

- Timothy C Lethbridge & Robert Laganiere, Object-Oriented Software Engineering: Practical Software Development using UML and Java, 2nd edition, 2005, McGraw-Hill
- Eric Braude, 2005, Software Design: From Programming to Architecture
- Ian Sommerville, 2010, Software Engineering (9th Edition)

Outline

- Software Engineering Fundamental Concepts
- Software Engineering Process
- Introduction to modeling with UML

Software Engineering Fundamental Concepts

Software engineering

- IEEE definition
  - The application of a systematic, disciplined, quantifiable approach to the **development**, **operation**, **maintenance** of software; that is, the application of engineering to software.
  - Engineering discipline = using **appropriate theories and methods to solve problems** bearing in mind **organizational** and **financial** constraints.
- **Goal** = Produce **high quality software** that **meets the needs of users**, with a given **budget**, before a given **deadline**, while **changes occur.**

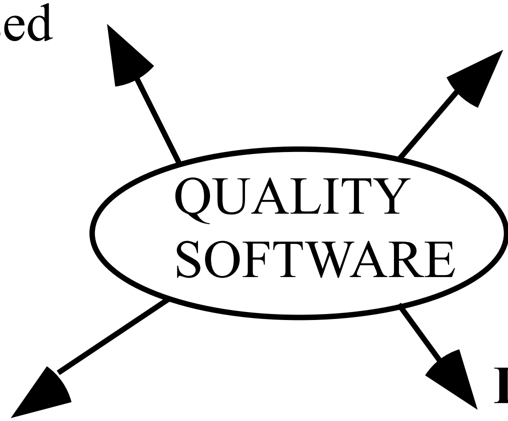
Importance of software engineering

- More and more, individuals and society rely on advanced software systems
  - We need to be able to produce **reliable** and **efficient** systems **economically** and **quickly**.
- It is usually cheaper, in the long run, to use software engineering methods and techniques rather than "rush to code" development process
  - For most types of system, the majority of costs are the **costs of changing the software after it has gone into use**.
- Real companies do it !
- Will help you build better software

Software Quality and the Stakeholders

Customer:
solves problems at
an acceptable cost in
terms of money paid and
resources used

User:
easy to learn;
efficient to use;
helps get work done



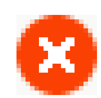
Developer:
easy to design;
easy to maintain;
easy to reuse its parts

Development manager:
sells more and
pleases customers
while costing less
to develop and maintain

Software Desired Quality Attributes

- **Dependability**
  - It does what it is required to do without failing
- **Efficiency**
  - It doesn't waste resources such as CPU time and memory
- **Maintainability**
  - It can be easily changed
- **Usability**
  - Users can learn it fast and get their job done easily
- **Reusability**
  - Its parts can be used in other projects

Software Engineering Process



Main Phases of Software Process

1. ***Requirements Elicitation (gathering)***

   Define what the system must do and the constraints on its operation

2. ***Analysis*** (answers "***WHAT***?")

   Understand the problem and decompose it into smaller, understandable pieces

=> identify use cases + key concepts and their associations & attributes

1. ***Design*** (answers "***HOW***?")

   Specify the system components and how they will work together

2. ***Implementation*** (A.K.A. "***CODING***")

   Write the code = Translate the design into running software

3. ***Testing*** (type of ***VERIFICATION***)

   Verify that the resulting software meets the requirements

4. ***Maintenance*** (***REPAIR*** or ***ENHANCEMENT***)

   Repair defects and add enhancements to meet changing requirements

Example

- ***Requirements Analysis*** => Use cases diagram & Scenarios + Domain Model

  e.g., " … The application shall display the balance in the user's bank account. …"

- ***Design*** => Diagrams (e.g., Class diagram and Sequence diagram)

  e.g., " … The design will consist of the classes *CheckingAccount, SavingsAccount*, …"

- ***Implementation*** => Source code

  e.g., … class CheckingAccount { double balance; … } …

- ***Testing*** => Test cases and test results

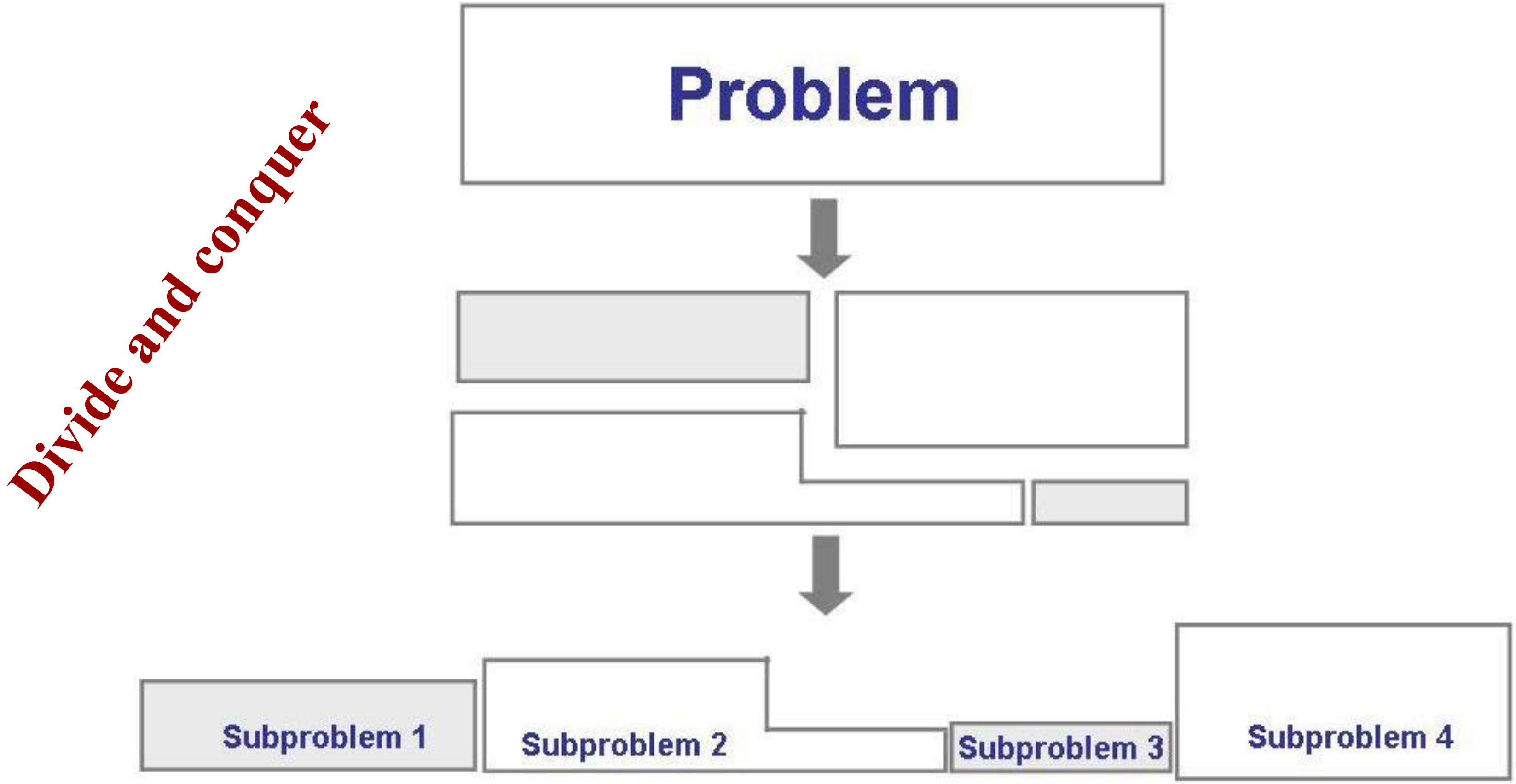
  e.g., "… With test case: *deposit $44.92 / deposit $32.00 / withdraw $101.45 /* … the balance was $2938.22, which is correct. …"

- ***Maintenance*** => Modified design, code, and test

  e.g., Defect repair: "Application crashes when balance is $0 and attempt is made to withdraw funds. …"

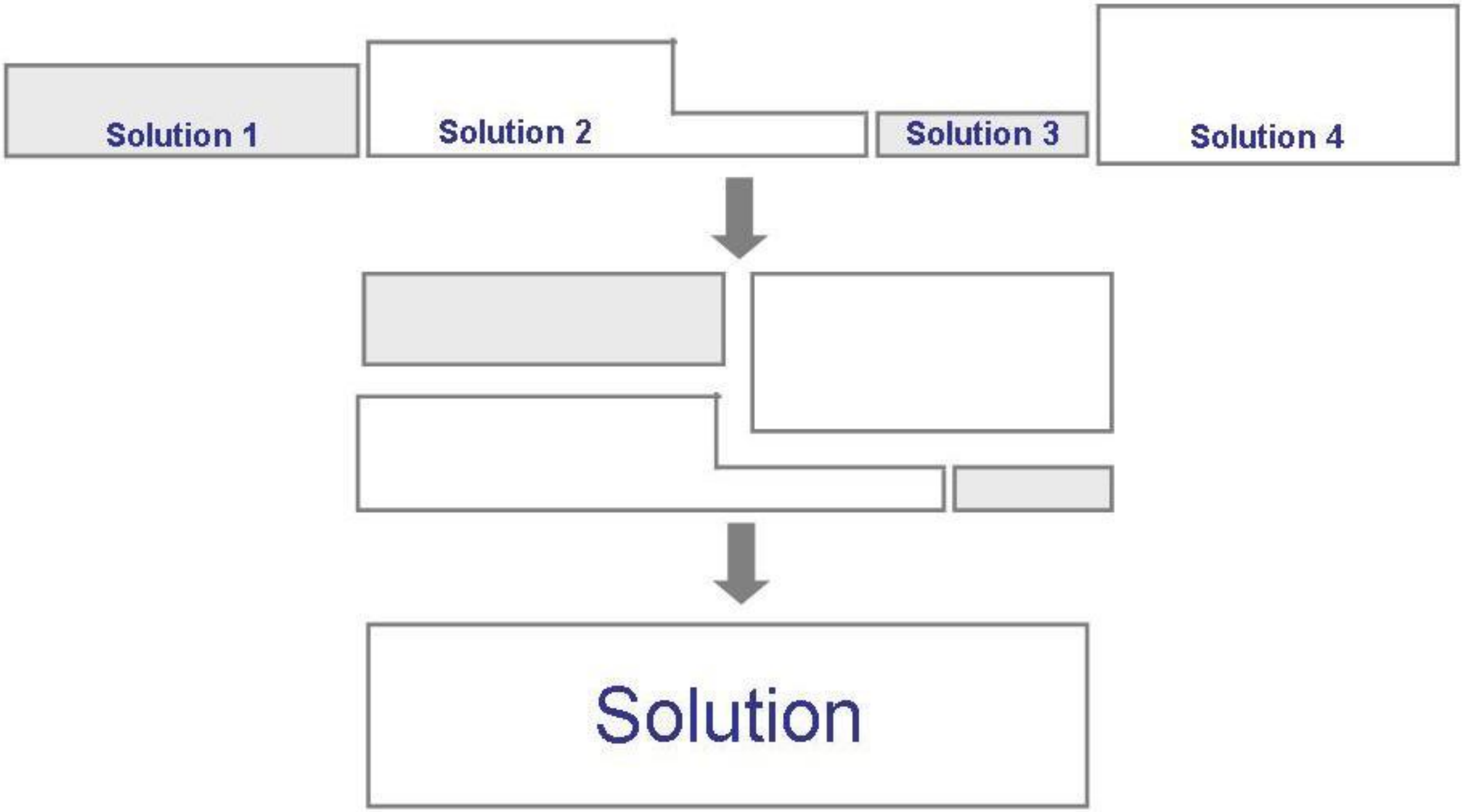
  e.g., Enhancement: "Send SMS message to Customer when balance changes"

The analysis process



Analysis = decompose a large problem into smaller, understandable and manageable pieces

Design = synthesis process



Synthesis: build (compose) a software from smaller building blocks

Analysis vs. Design?

Analysis

- Investigation
- **What?**

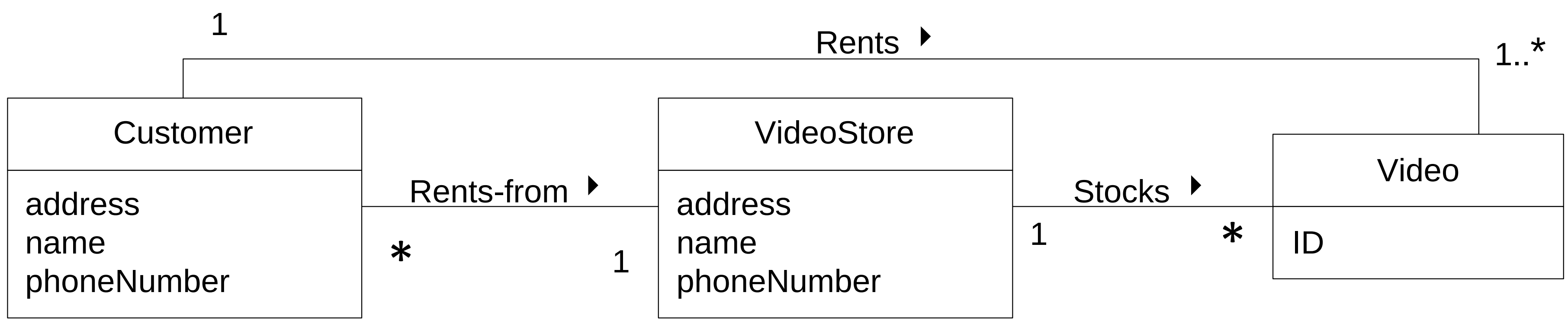
- Includes:
  - Requirements analysis
  - Domain analysis
- Key Questions:
  - How the system will be used (i.e. use cases)?
  - Finding and describing the key concepts – or objects – in the problem domain as well as their attributes and associations (i.e. **conceptual domain model**)

Design

- Solution
- **How?**

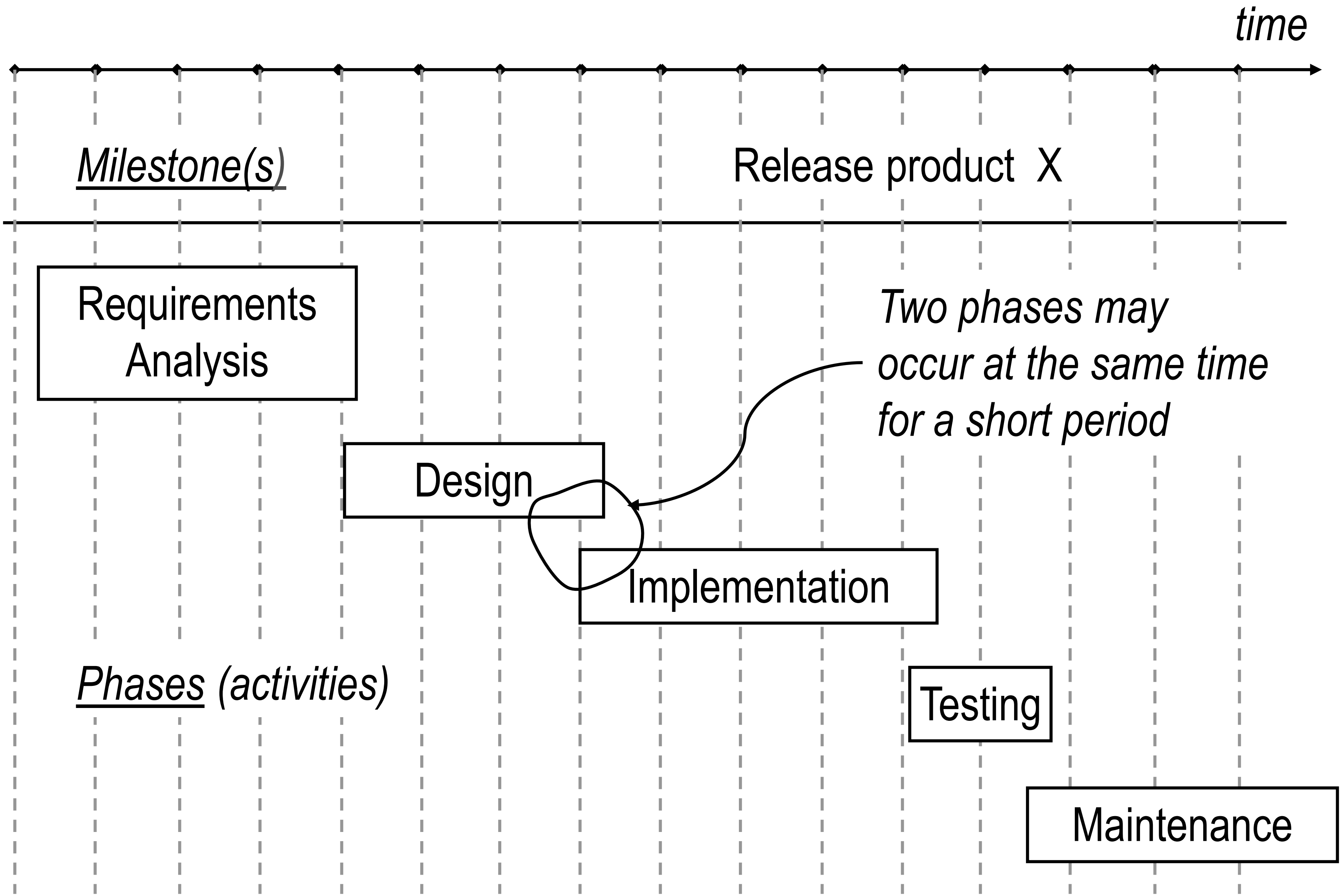
- Includes:
  - Object design
  - Database design
  - User Interface (UI) design
- Key Questions:
  - How should responsibilities be assigned to classes?
  - How should objects interact to fulfill the requirements?

EXAMPLE: Partial Domain Model

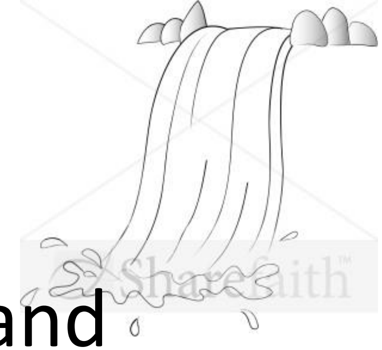


- *Domain Model* helps us **identify**, **relate** and **visualize** important concepts and their associations.

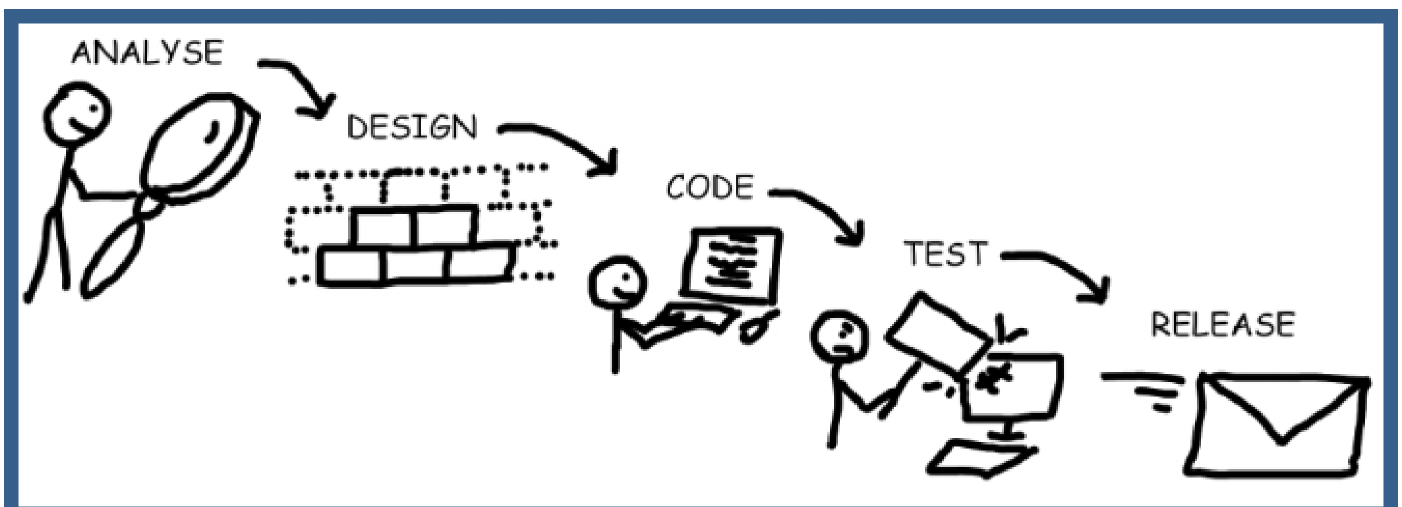
The Waterfall Software Process



Waterfall Process



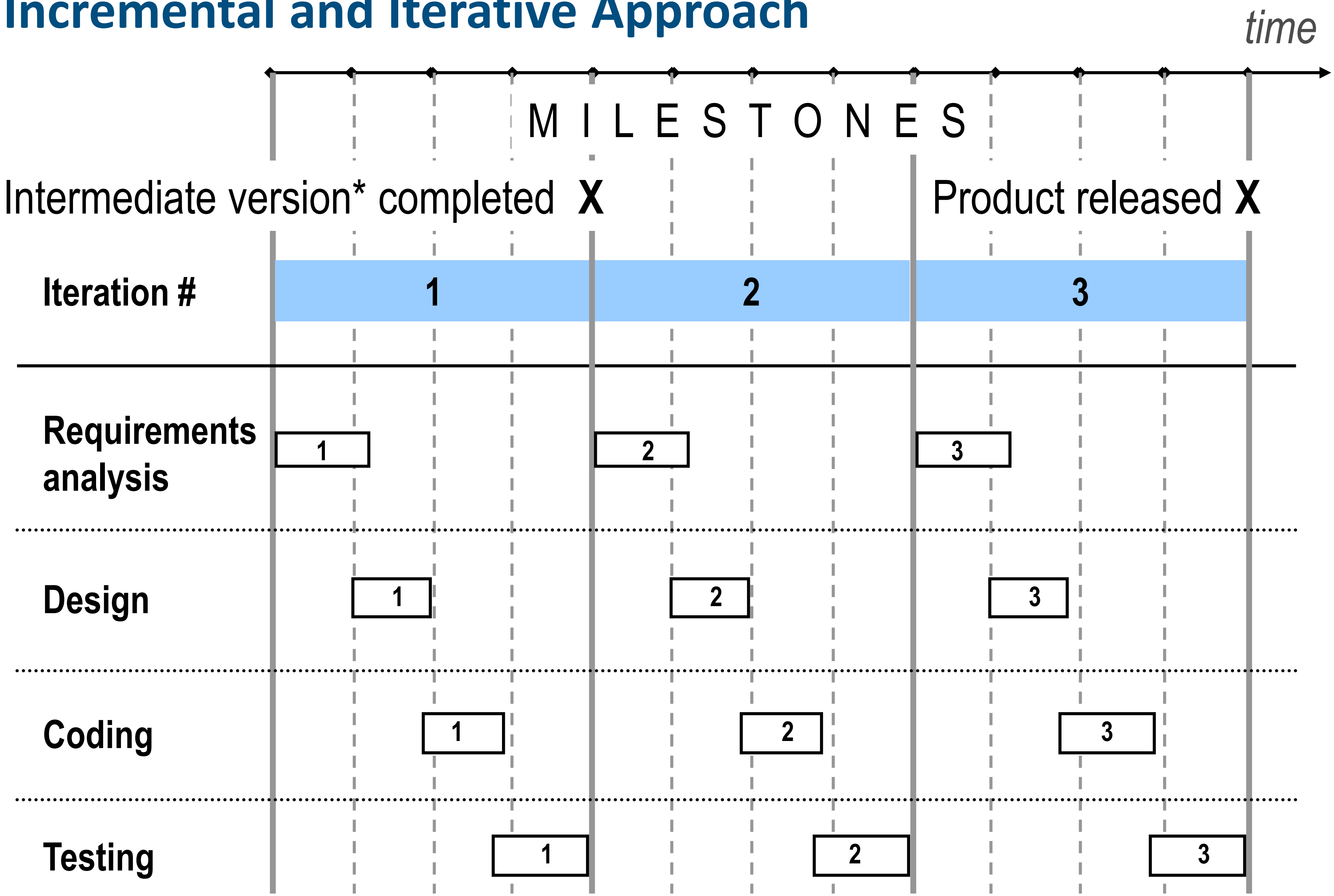
- Requirements analysis, design, coding and testing, are performed in sequence, but with some overlap.
  - Each step cannot begin until the previous step has been completed, documented and signed-off
  - The output of each phase is used by the next phase



Why a Pure Waterfall Process is Usually Not Practical

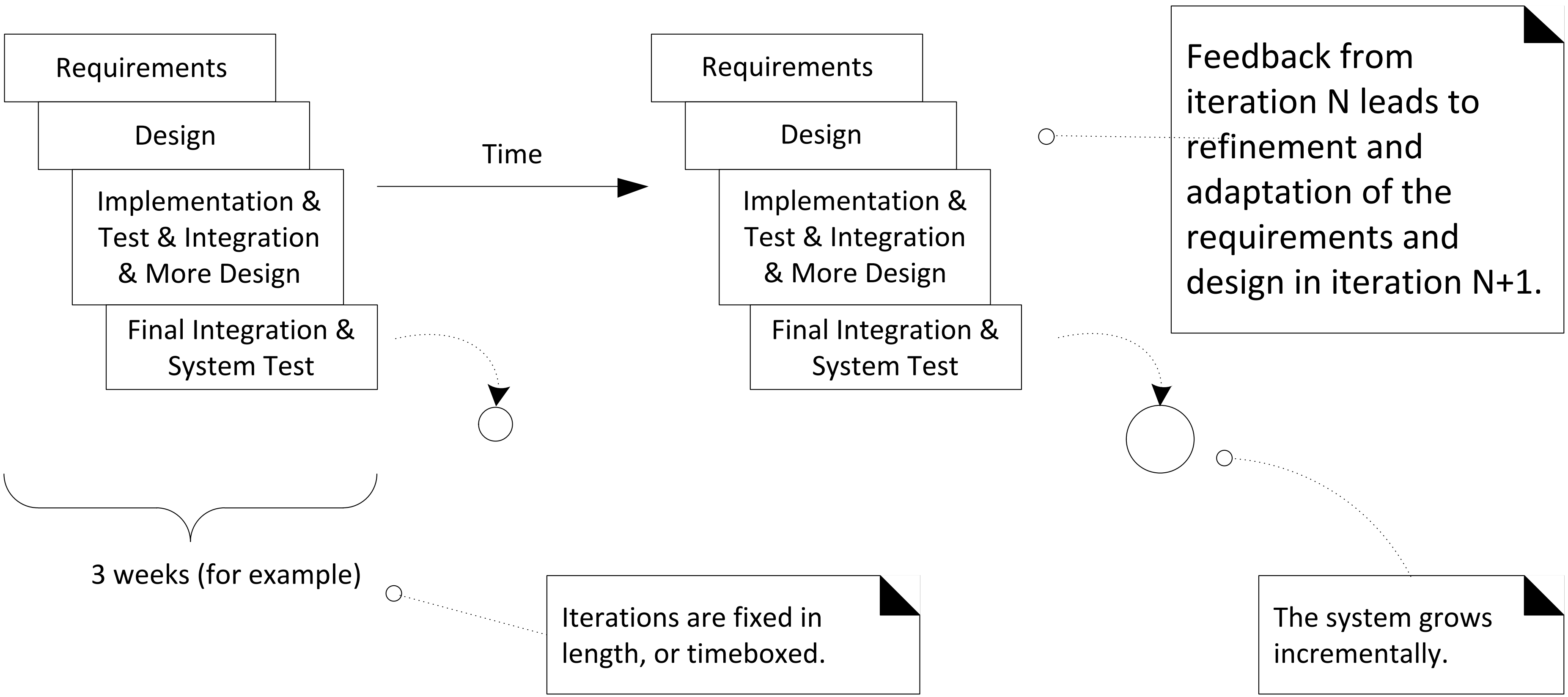
- Hard to fully define complete and correct requirements up front
  - Missed requirements are quite common
- **Difficulty of accommodating *changes*** *in the requirements and users feedback once the requirements are signed-off*
  - Changes in the external environment can result in changes to the requirements
- Hard to estimate the costs of large projects
- Nothing is available for use until the end of the process, which can one or more years from start to finish

Incremental and Iterative Approach



*typically a prototype

Incremental and Iterative Approach



Incremental and Iterative Approach

- After some initial analysis and design, **a subset** of the requirements is implemented, and then additional capabilities are added incrementally
  - Each iteration involves choosing a small subset of the requirements, and quickly designing, implementing and testing
  - The system grows incrementally and converges towards the desired system via a series of '**build-feedback-adapt**' cycles

Key Advantages

- An incremental approach has at least three major advantages:

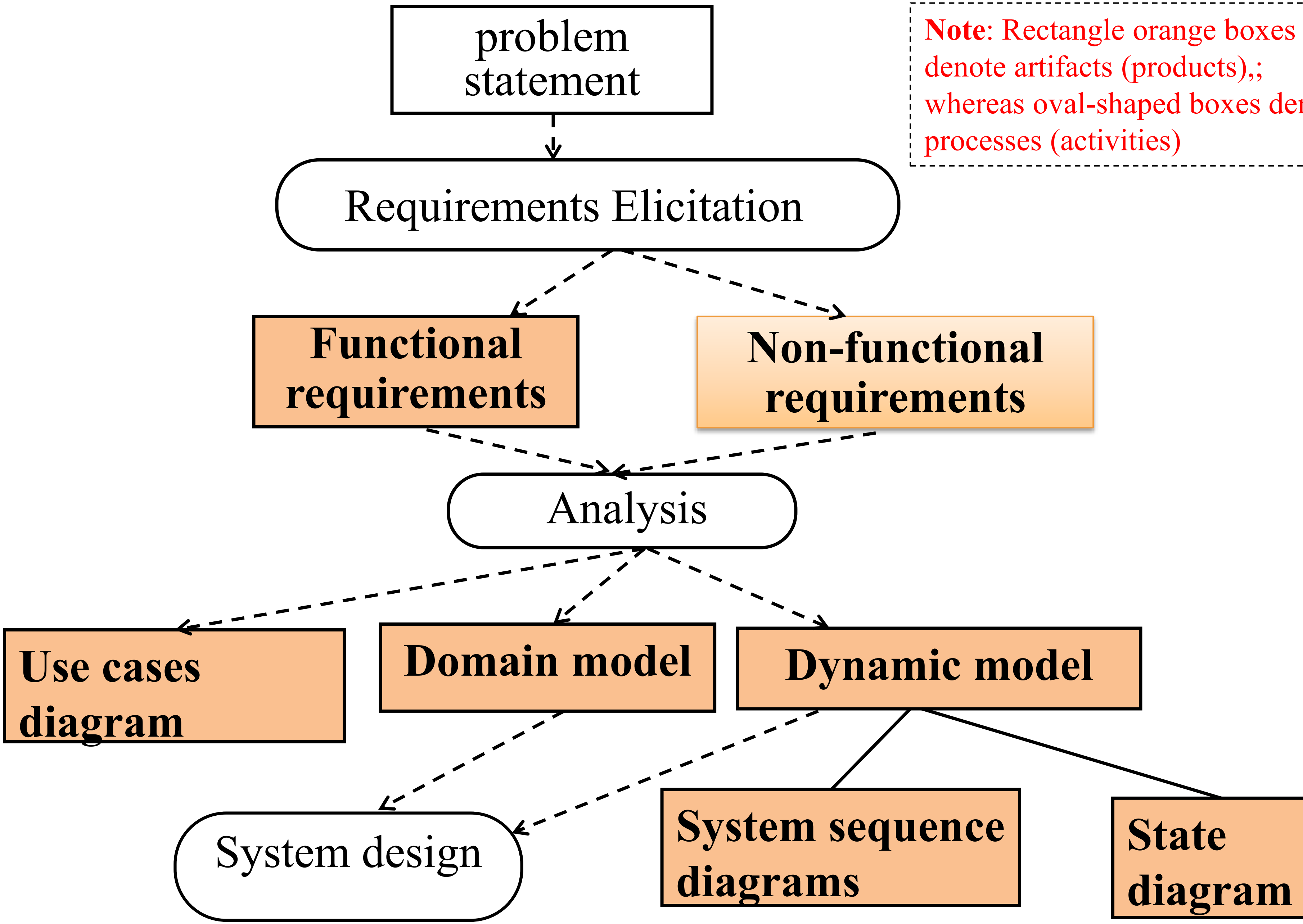
  (1) The client gets to begin making some use of the software fairly early, rather than having to wait for everything to be completed.

  (2) Users have a chance to quickly experience with using a partial system and **provide valuable feedback** that can help to refine the requirements for subsequent parts (**build-feedback-adapt cycle**)

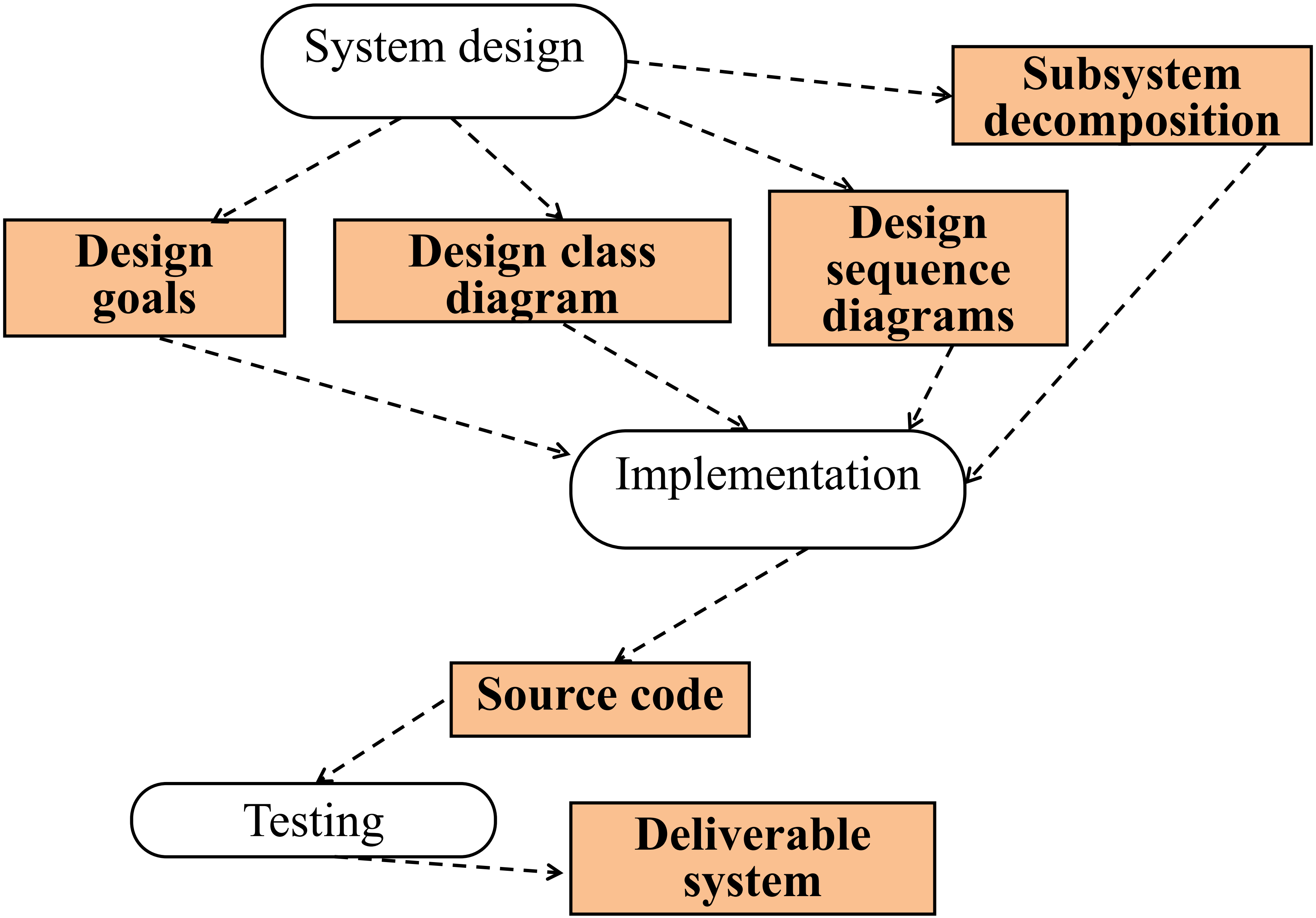
  (3) Continuously engage users for evaluation, feedback, and requirements

Software engineering process I

Note: Rectangle orange boxes denote artifacts (products),; whereas oval-shaped boxes denote processes (activities)



Software engineering process II



Introduction to modeling with UML

UNIFIED MODELING LANGUAGE ™

What is UML?

U ~ **Unified**:

- Unifies all existing previous Notations

M ~ **Modeling:**

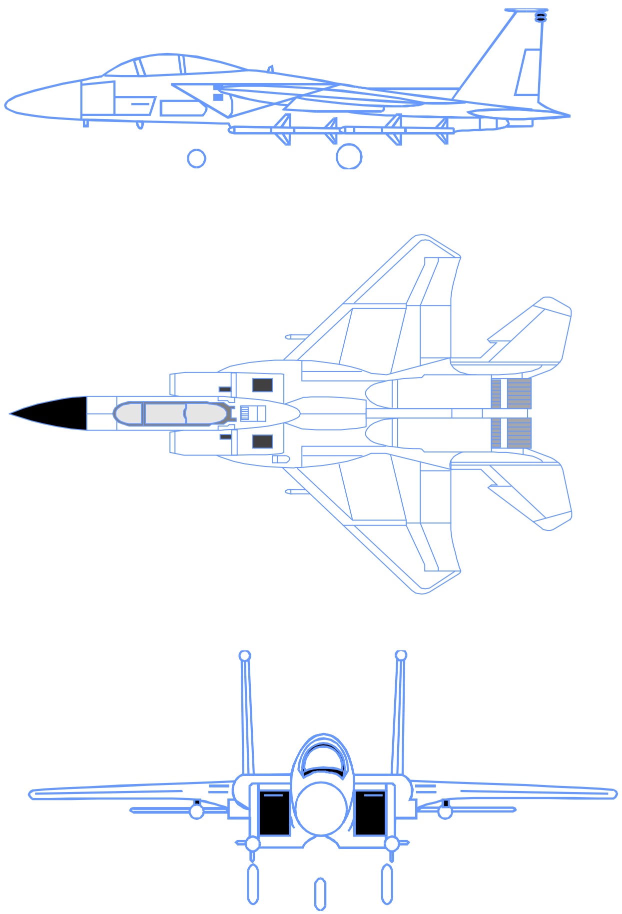
- Used for Modeling Software Artifacts

L ~ **Language:**

- Means of Communication

What Is a Model?

- A model is a simplification of reality.



What Is UML?

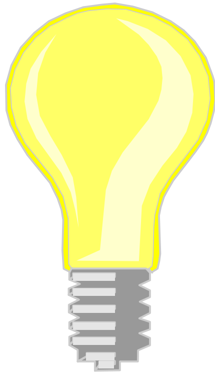
- UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

  the artifacts of a software

UML Is a Language for Visualizing

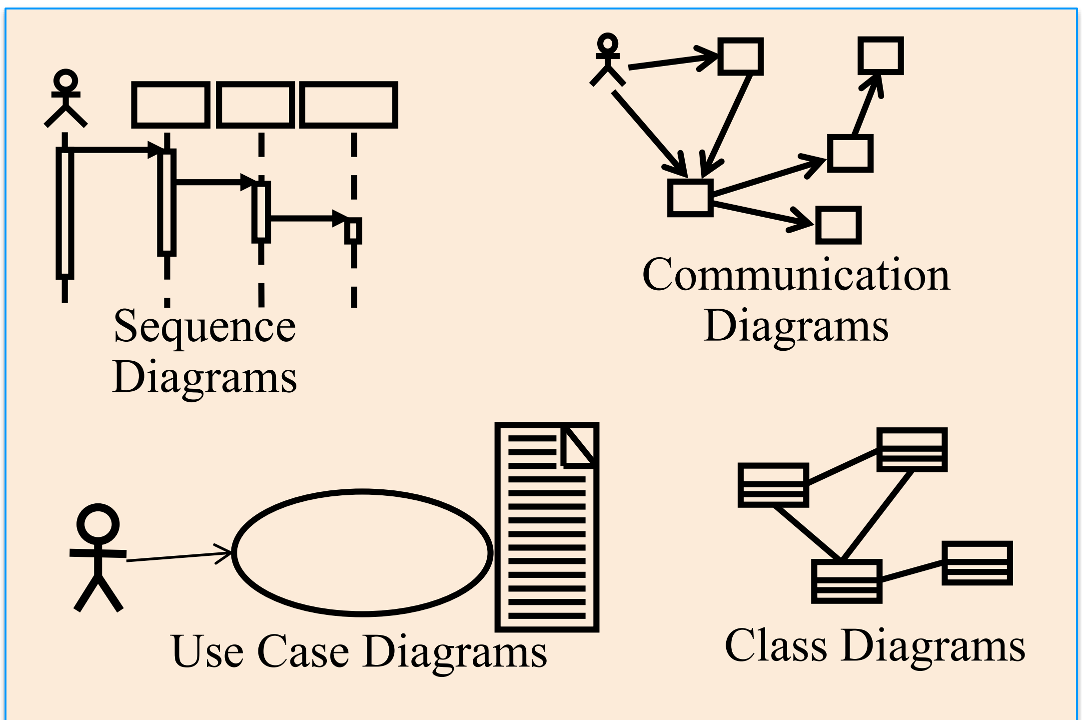
A picture is worth a thousand words!

- UML allows creating graphical models with precise semantics
  - It facilitates **communication** since everyone involved uses a common vocabulary
  - Helps you to **visualize** a system as you want it to be => **captures the design graphically**
  - Permits you to **specify the structure** or **behavior** of a system.



UML Is a Language for Documenting

- UML addresses documentation of system requirements, architecture, detailed design, and deployment.



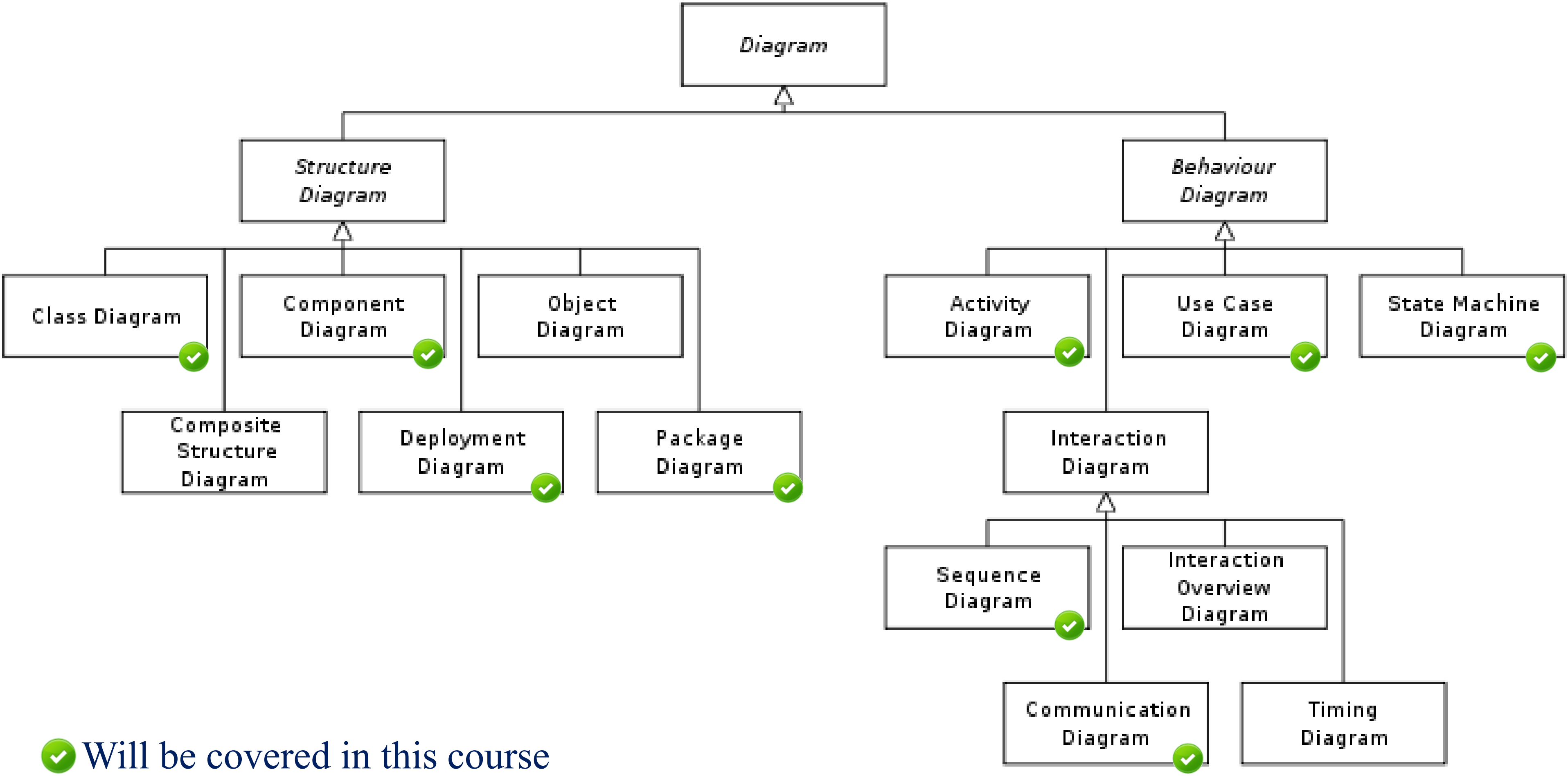
Static vs. Dynamic Diagrams

- A diagram provides a partial representation of the system

2 main categories of diagrams:

- **Structural diagrams** - model static or structural aspects (**irrespective of time**)
  - Nouns: Conceptual or physical elements
- **Behavioral diagrams** - model dynamic or behavioral aspects
  - Verbs: interactions or collaborations among elements
  - Capture "behavior over time"

UML 2.0 - 13 Types of Diagrams



Will be covered in this course

Most Important UML Diagrams

- **Use case diagrams**
  - model the **system’s intended functionality** (use cases) and **its environment** (actors)
- **Class diagrams**
  - describe classes and their relationships
- **Sequence diagrams**
  - show the behaviour of systems in terms of how objects interact with each other
- **State diagrams**
  - show how systems behave internally
- **Component and deployment diagrams**
  - show how the various components of systems are arranged logically and physically

Summary (1 of 2)

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Essential software quality attributes are: dependability, efficiency, maintainability, usability and reusability
- The main activities of software engineering process: Requirements Elicitation, Analysis, Design, Implementation, Testing and Maintenance

Summary (2 of 2)

- Modeling is describing a system at a high level of abstraction
- UML is involved in each phase of the software development life cycle
- UML is a *modeling language*, not a *method*, as it does not comprise a *process*
  - It is primarily a graphical communication mechanism for developers and customers