ITSS SOFTWARE DEVELOPMENT

1. SOFTWARE DEVELOPMENT PROCESS

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

References

[1] ISO/IEC FDIS 12207, *Systems and software engineering — Software life cycle processes*.

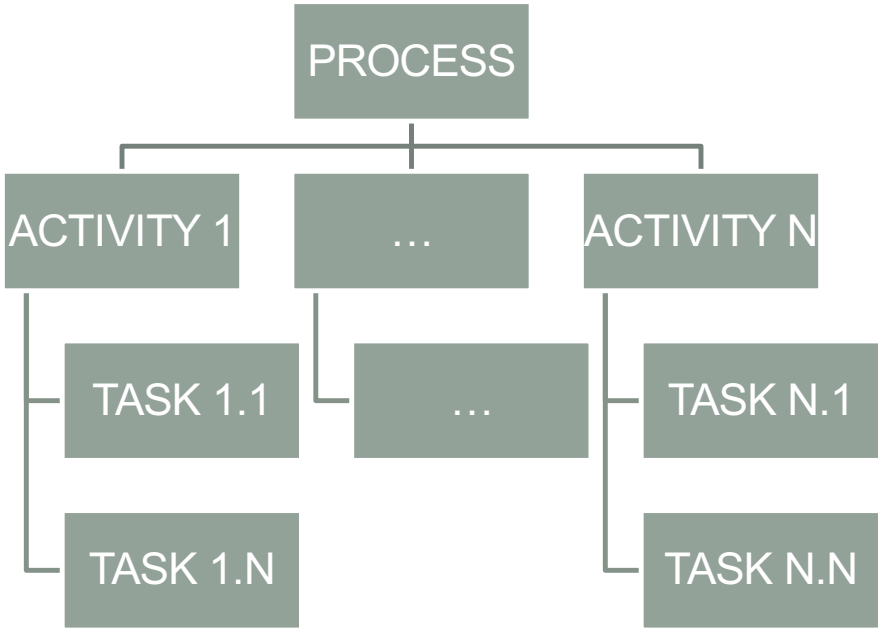
Content

1. Software Life Cycle Process
2. Software Implementation Process
3. Object-Oriented Analysis and Design
4. Software Development Models

Content

1. Software Life Cycle Process

- “ISO/IEC 12207:2008, Systems and software engineering — Software life cycle processes”
  - The latest and International Standard Software Development Process
- “The life cycle begins with an idea or a need that can be satisfied wholly or partly by software and ends with the retirement of the software.”
- Standard implementation works
  - hierarchically as processes



What are International Standards?

- In ISO, all industry standards, including Information Technology, are developed.
- In the field of Information Technology, in ISO/IEC JTC1, international standards are developed.
- ISO/IEC JTC1 has 32 principal member bodies which develop the international standards, and 44 observer member bodies.
- Some abbreviation
  - ISO: International Organization for Standard
  - IEC: International Electrotechnical Commission
  - JTC1: Joint Technical Committee

Why International Standards?

- Standards are important, especially in ICT
  - Basis of common understanding such as frameworks, and terminology / definitions
  - TBT Agreement of WTO recommends the use of ISO Standards for governmental purchase in affiliate countries
  - Based on some standards, the certifications can be got, and they make some appeal points in international transaction

WTO: World Trade Organization

TBT Agreement: Agreement on Technical Barriers to Trade

Software Life Cycle Process

- "This International Standard groups the activities that may be performed during the life cycle of a software system into seven process groups"[1]*:
  1. Agreement Processes: 2 processes
  2. Organizational Project-Enabling Processes: 5 processes
  3. Project Processes: 7 processes
  4. Technical Processes: 11 processes
  5. **Software Implementation Processes: 6 processes**
     - **Purpose: "to produce a specified system element implemented as a software product or service"** [1]**.
  6. Software Support Processes: 8 processes
  7. Software Reuse Processes: 3 processes

[1]*: clause 5.2.1; pp. 13, [1]**: clause 7.1.1.1; pp. 57,

Content

1. Software Life Cycle Process
2. Software Implementation Process
3. Object-Oriented Analysis and Design
4. Software Development Models

2. Software Implementation Process

System Requirements Analysis Process and System Architectural Design Process are achieved just before Software Implementation Process .

Software Implementation Process includes the following lower-level processes:

1. **Software Requirements Analysis Process**
2. **Software Architecture Design Process**
3. **Software Detailed Design Process**
4. **Software Construction Process**
5. **Software Integration Process**
6. **Software Qualification Testing Process**

2.1. Software Requirements Analysis process

- Purpose: "to establish the requirements of the software elements of the system" [1]
- Main items written on the brief requirement description
  - **System environmental conditions** under which the software is to perform.
  - The **functional requirements** and the **interface requirements**.
  - **Data definition and database requirements**.
  - Some **non-functional requirement** items such as reliability, usability, time efficiency
  - **Qualification requirements**: The requirements are used as criteria or conditions to qualify a software product as complying with its specifications.

[1]: Session 7.1.2.1; pp. 59

2.2. Software Architectural Design process

- Purpose: "to provide a design for the software that implements and can be verified against the requirements" [1]
- Software architecture is designed from the software requirements
- Main items
  - a top-level structure of the software and the software components which constructs the software
  - a top-level design for the interfaces external to the software and between the software components
  - a top-level design for the database

[1]: Session 7.1.3.1; pp. 61

2.3. Software Detailed Design process

- Purpose: "to provide a design for the software that implements and can be verified against the requirements and the software architecture and is sufficiently detailed to permit coding and testing" [1]
- A detailed design for each software components are developed. In the detailed design, the following items are developed:
  - **each component** is refined into **software units** that can be coded, compiled, and tested
  - **the interfaces external to the software item**, **between the software components**, and **between the software units**

[1]: Session 7.1.4.1; pp. 62

2.4. Software Construction process

- Purpose: "to produce executable software units that properly reflect the software design" [1]
- Main items to be developed:
  - Each software unit and database
  - Test procedure and test data for software unit and database
  - Unit tests and database test
- The implementer shall evaluate software code and test results considering internal external consistency, test coverage of units and, traceability to the requirements and design of the software.

[1]: Session 7.1.5.1; pp. 63

2.5. Software Integration process

- Purpose: “to combine the software units and software components, producing integrated software items, consistent with the software design, that demonstrate that the functional and non-functional software requirements are satisfied on an equivalent or complete operational platform” [1]
- Main tasks
  - An integration plan, including test requirements, test procedure, and test cases/data.
  - Integration of software units/components
  - Program/software/integration test

[1]: Session 7.1.6.1; pp. 64

2.6. Software Qualification Testing Process

- Purpose: "to confirm that the integrated software product meets its defined requirements" [1].
- Qualification testing in accordance with the qualification requirements for the software item is conducted
  - Tests, test cases, and test procedures
- The implementer supports audit(s) to conform the software meets to the qualification requirements
  - If it is successful completion of the audits, the implementer prepare the deliverable software product for System Construction process

[1]: Session 7.1.7.1; pp. 66

Summary

- "Software Life Cycle Process – SLCP" is **the international standard** processes focused on the development and support of Application Software.
- SLCP can be used as **a common language** among the stakeholders such as acquirers and suppliers. They can communicate or order the software development using SLCP. For example, we can say "To order **the software detailed design process or later software implementation processes** of new library system".

Learning points

"The customer's business success depends on the system development success."

BA Software is one of the major components of the BA System.

What are the main factors for the system development success?

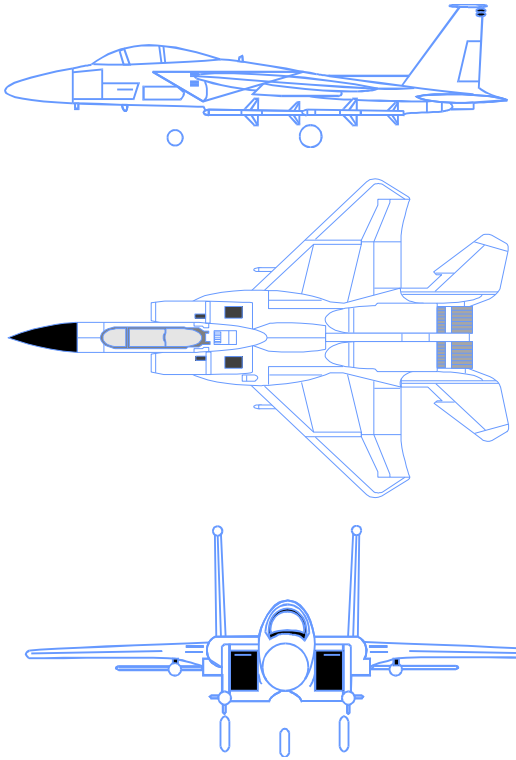
1. The Software to be developed meets to **functional** requirements → ?
2. To keep appointed date of **delivery** → ?
3. Meets the required **quality** such as Reliability, Usability, Performance, Maintainability → ?
4. Necessary to provide **maintenance** activity during the system operation period → ?

Content

1. Software Life Cycle Process
2. Software Implementation Process
3. Object-Oriented Analysis and Design
4. Software Development Models

2.1. Modeling

- A model is a simplification of reality.



Why Model?

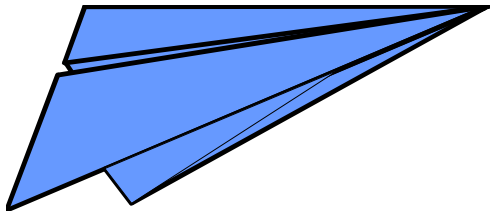
- Modeling achieves four aims [1]:
  - Helps you to “visualize a system as you want it to be”.
  - Permits you to “specify the structure or behavior of a system”.
  - Gives you “a template that guides you in constructing a system”.
  - “Documents the decisions you have made”.
- You build models of complex systems because you cannot comprehend such a system in its entirety.
- You build models to better understand the system you are developing.

[1]: Chapter 1, Section 1.1

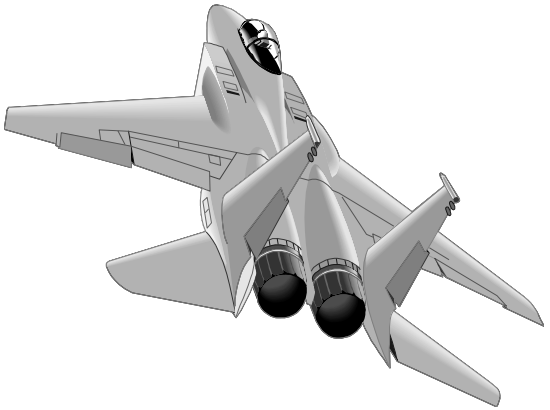
The Importance of Modeling

Less Important

More Important



Paper Airplane



Fighter Jet

2.2. Unified Modeling Language (UML)

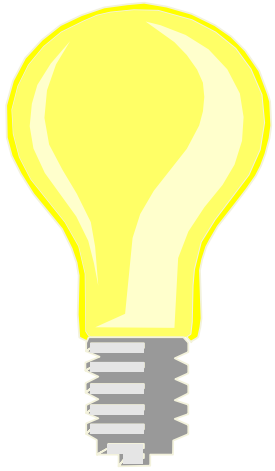
- “The UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

  the artifacts of a software-intensive system” [1].

[1]: Chapter 2, Section 2.1

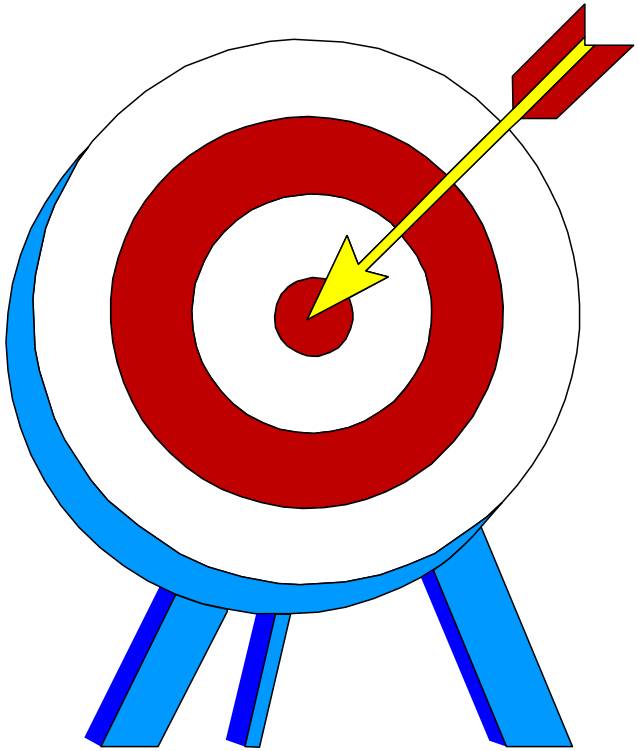
The UML Is a Language for Visualizing

- Communicating conceptual models to others is prone to error unless everyone involved speaks the same language.
- There are things about a software system you can't understand unless you build models.
- An explicit model facilitates communication.



The UML Is a Language for Specifying

- The UML builds models that are precise, unambiguous, and complete.



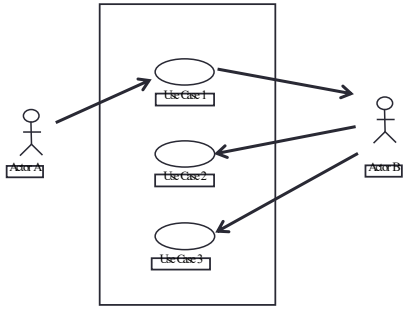
The UML Is a Language for Constructing

- UML models can be directly connected to a variety of programming languages.
  - Maps to Java, C++, Visual Basic, and so on
  - Tables in a RDBMS or persistent store in an OODBMS
  - Permits forward engineering
  - Permits reverse engineering

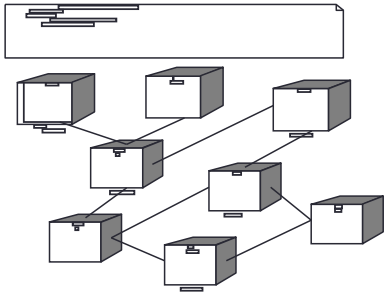
The UML Is a Language for Documenting

- The UML addresses documentation of system architecture, requirements, tests, project planning, and release management

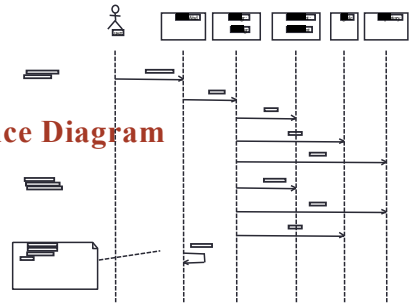
Use Case Diagram



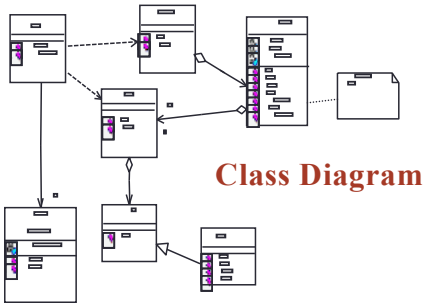
Deployment Diagram



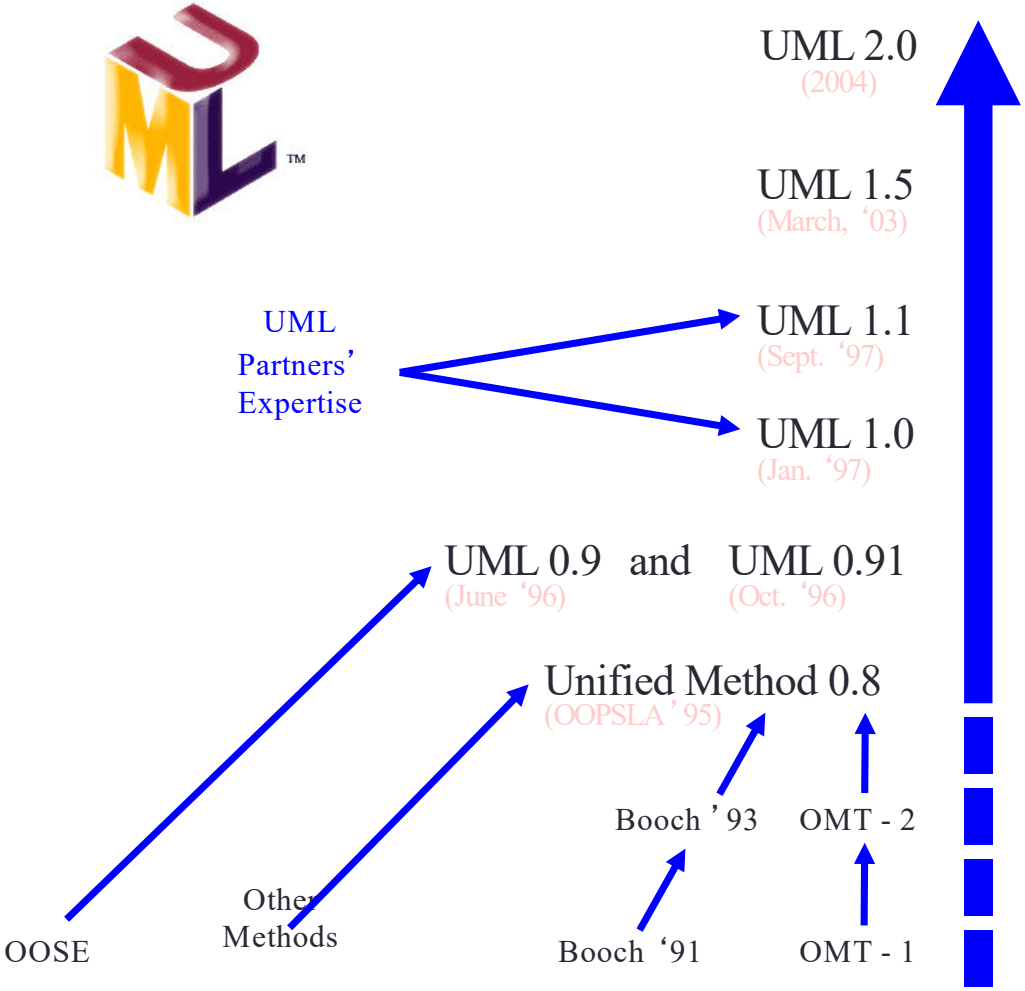
Sequence Diagram

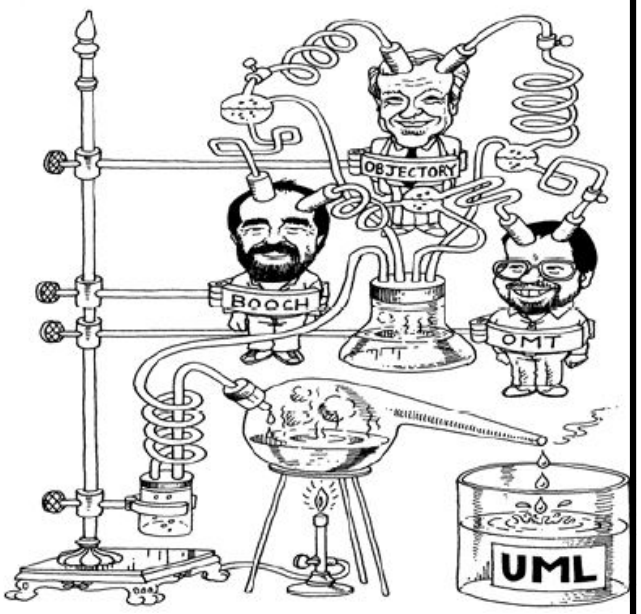


Class Diagram



History of the UML



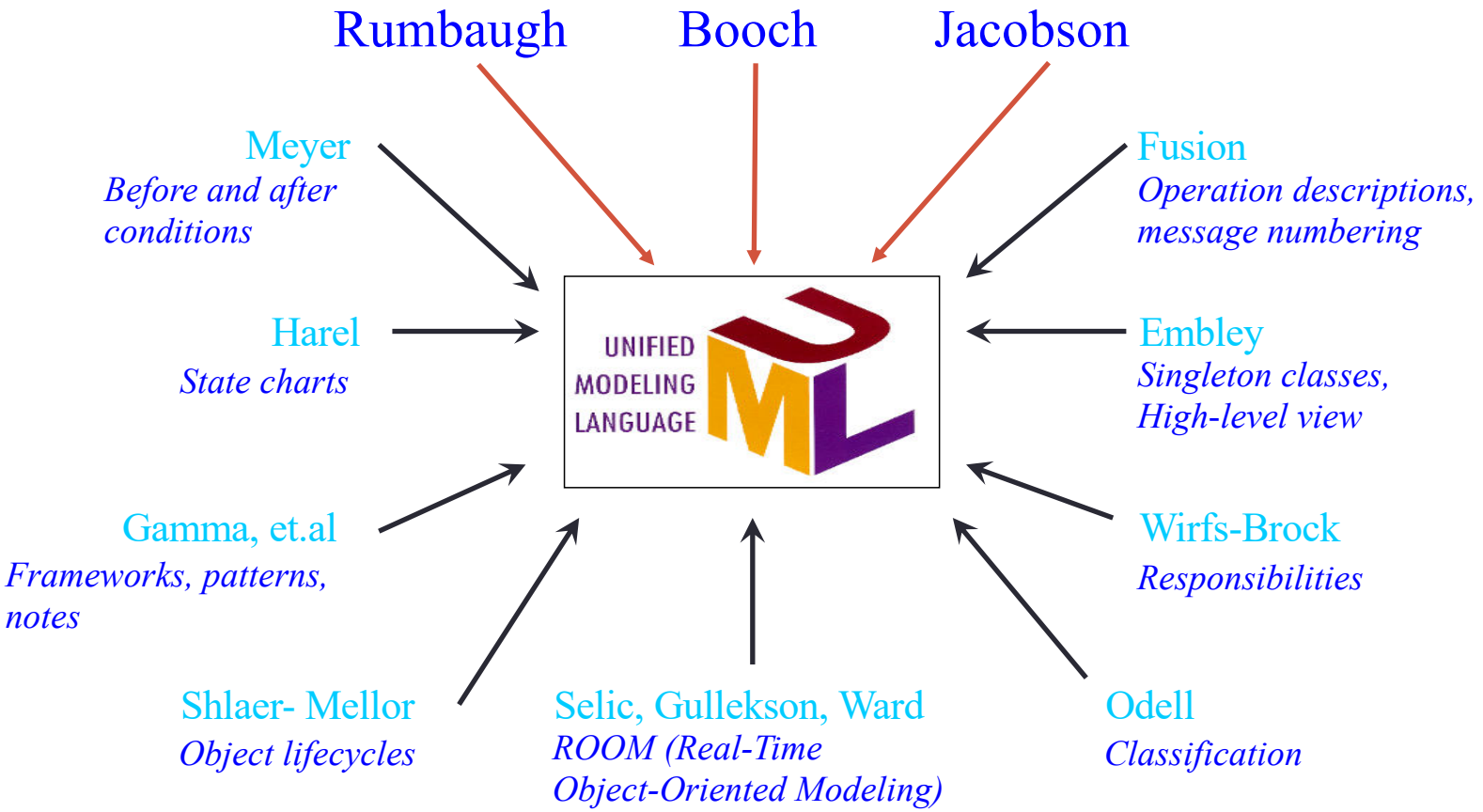


Booch91 (Grady Booch):
Conception, Architecture

OOSE (Ivar Jacobson): Use cases

OMT (Jim Rumbaugh): Analysis

Inputs to the UML



UML Views

- “A view is simply a subset of UML modeling constructs that represents one aspect of a system” [2]
- Four areas
  - structural classification,
  - dynamic behavior
  - physical layout,
  - and model management.

[2]: Part 2, Chapter 3, Section 3.1

UML Views [2]

Major area	View	Diagram
Structural	Static view	Class diagram
	Design view	Internal structure, Collaboration diagram Component diagram
	Use case view	Use case diagram
Dynamic	State machine view	State machine diagram
	Activity view	Activity diagram
	Interaction View	Sequence Diagram Communication Diagram
Physical	Deployment View	Deployment Diagram
Model Management	Model Management View	Package diagram
	Profile	Package Diagram

[2]: Part 2, Chapter 3, Section 3.1, Table 3.1 (extracted)

Static View vs Dynamic View

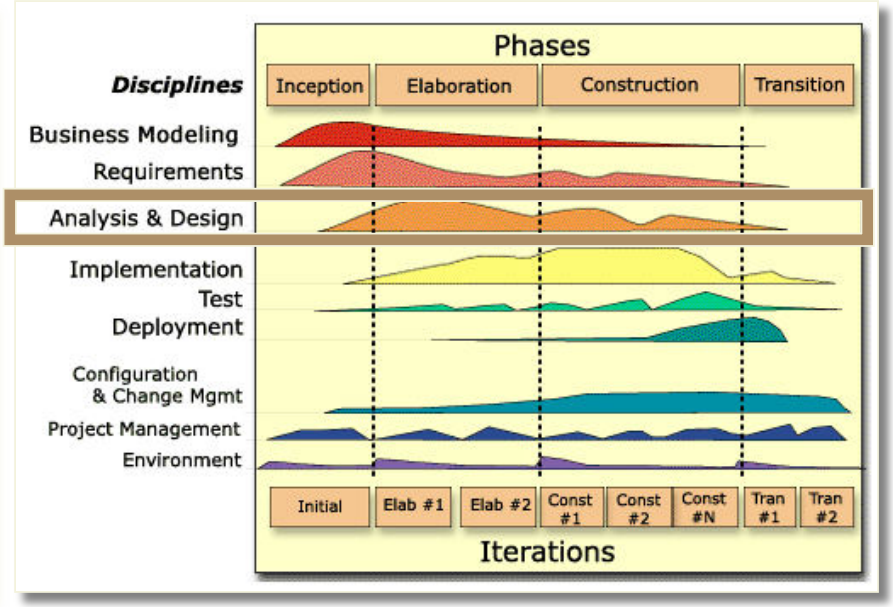
Role of component parts and how they are related to one another

How the components interact with one another and/or change state internally over time

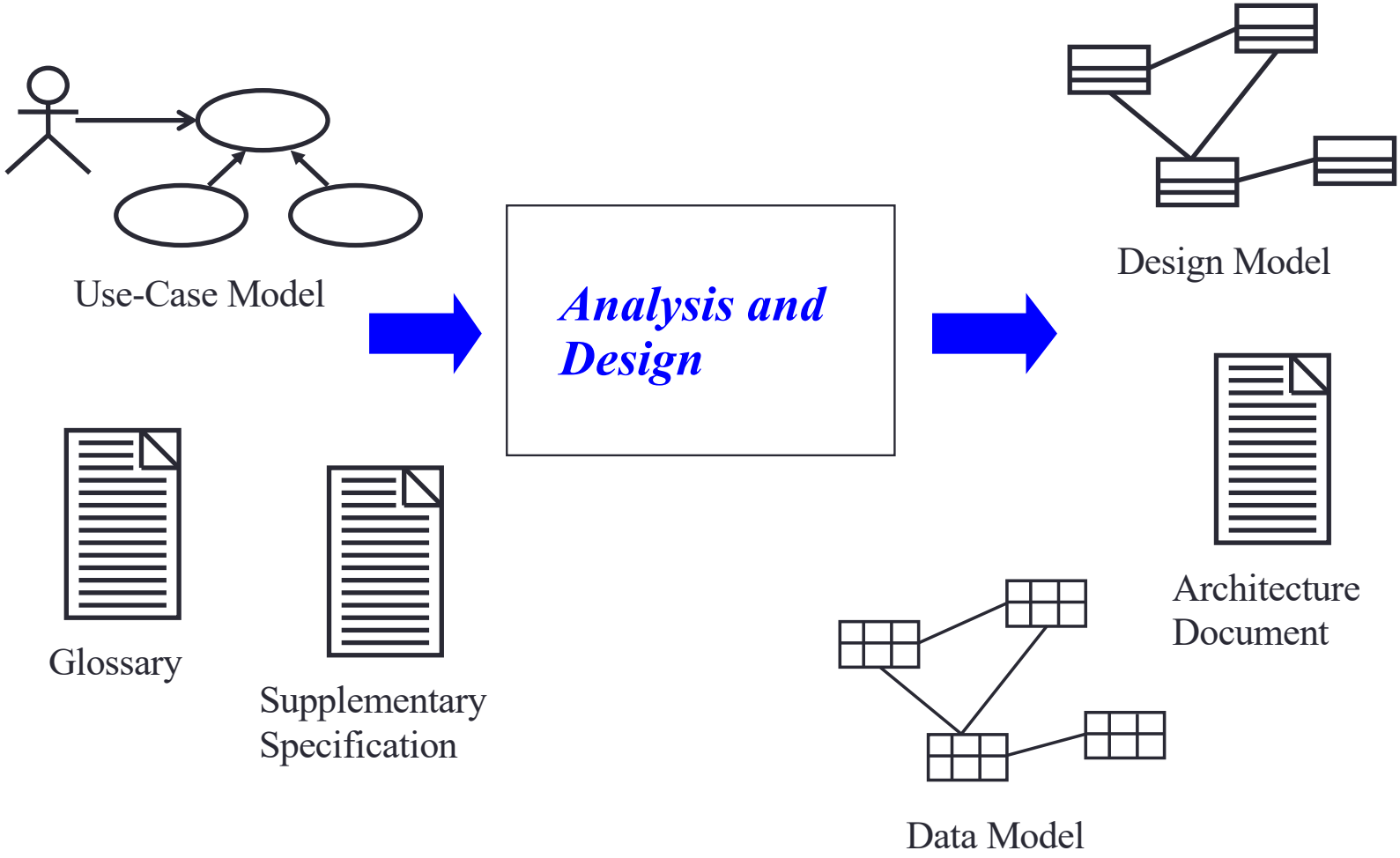
2.3. Analysis and Design

The purposes of Analysis and Design are to:

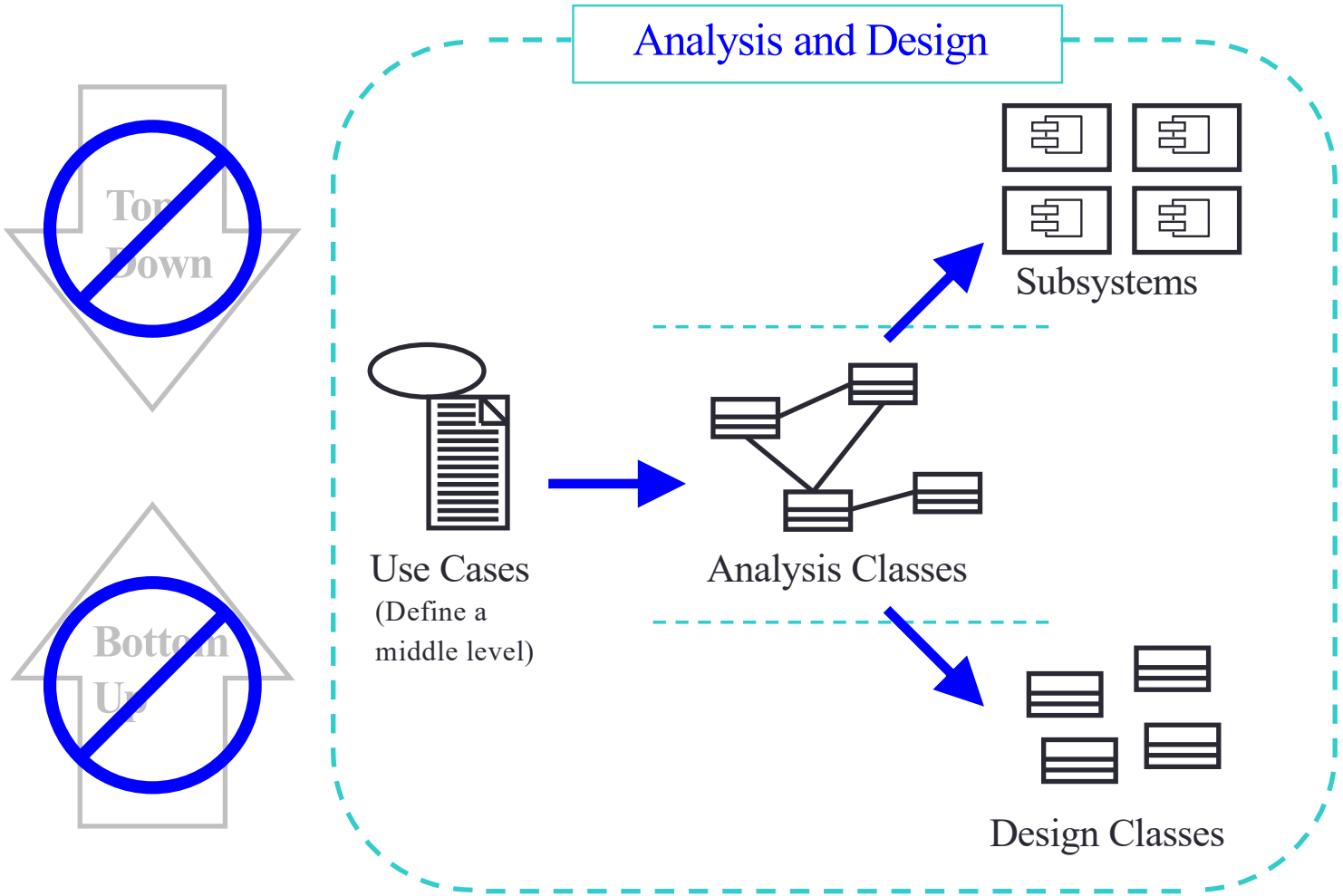
- **Transform the requirements into a design of the system-to-be.**
- **Evolve a robust architecture for the system.**
- **Adapt the design to match the implementation environment, designing it for performance.**



Object-Oriented Analysis and Design



Analysis and Design Are Not Top-Down or Bottom-Up



Analysis and Design Steps

Activity	Step	Description	Doer
Define a candidate architecture	1. Architectural Analysis	• Once at early Elaboration • Skip if architectural risk is low	Architect
Analyze behavior	**2. Use case Analysis**	• **Per Use case**	**Designer**
Refine the architecture	**3. Identify Design Elements**	• **Coupling and cohesion** • **Reusability**	**Architect**
	4. Identify Design Mechanisms	• Design patterns	
	5. Describe Run-time Architecture	• Skip if not multi-threading • Process View	
	6. Describe Distribution	• Physical Architecture	
Design components	**7. Use case Design**	• **Per Use case**	**Designer**
	8. Subsystem Design		
	9. Class Design		
Design DB	**10. Database Design**		

Content

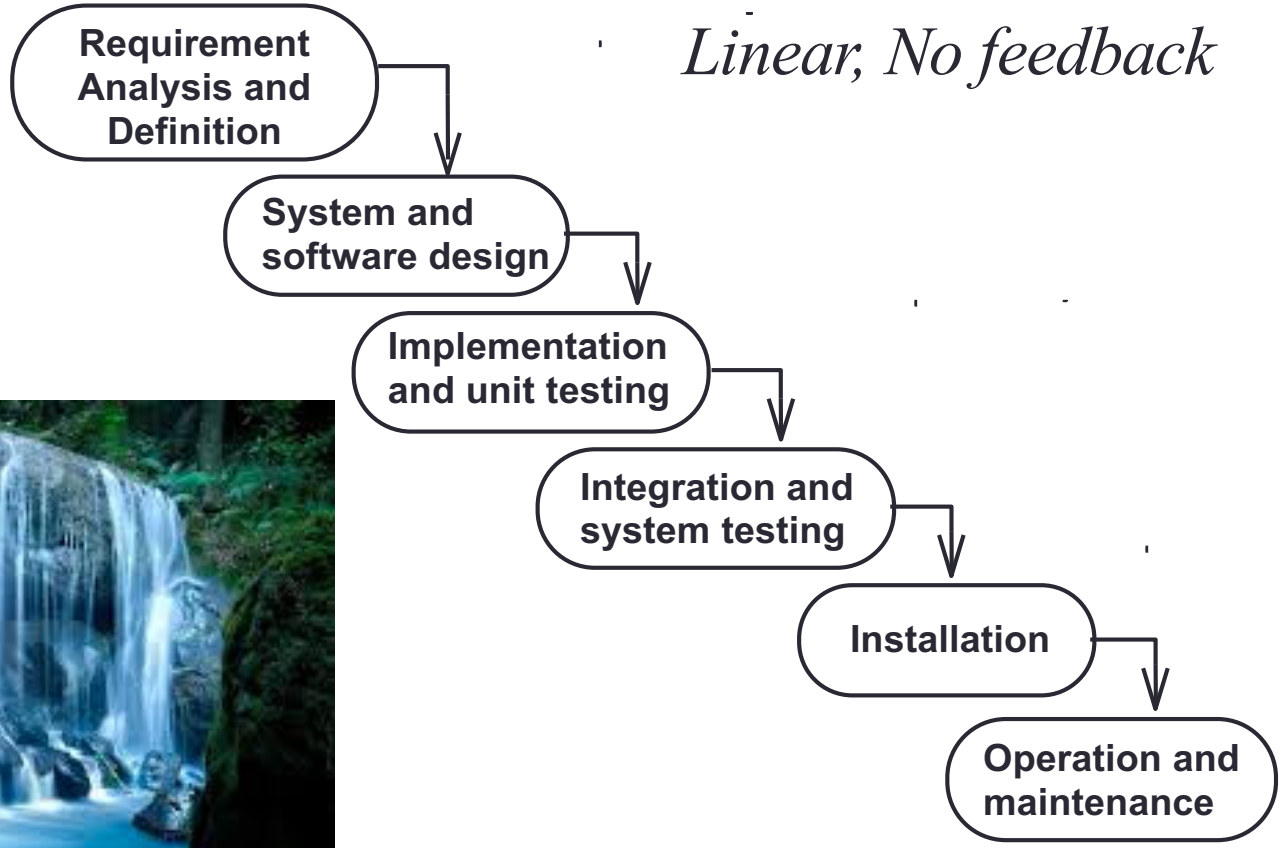
1. Software Life Cycle Process
2. Software Implementation Process
3. Object-Oriented Analysis and Design
4. Software Development Models

Homework

- Compare some common software development models
  - Waterfall model
  - Iterative model
  - Prototype model
  - Spiral model
  - Ration Unified Process (RUP)
  - Agile methodology

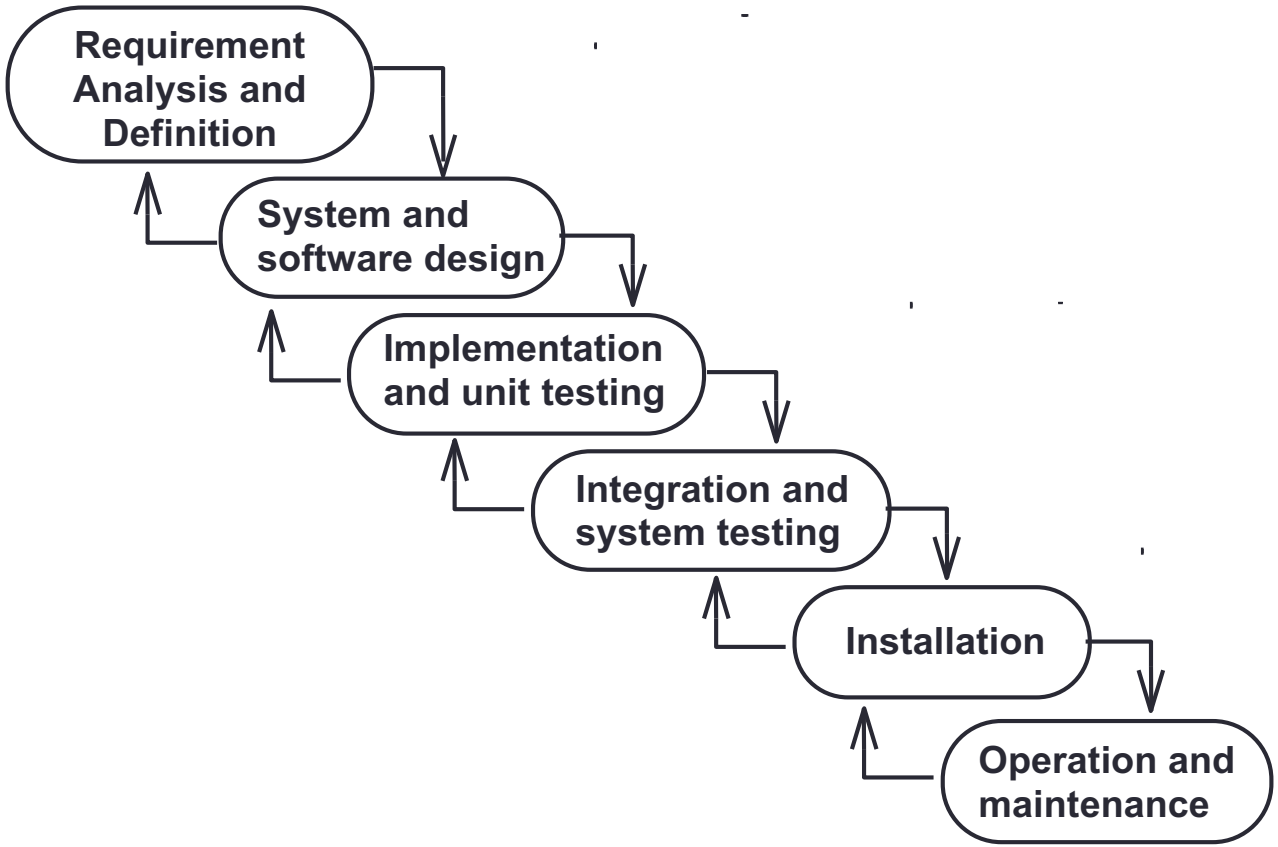
➔ Submit report individually

The Waterfall/Linear Model



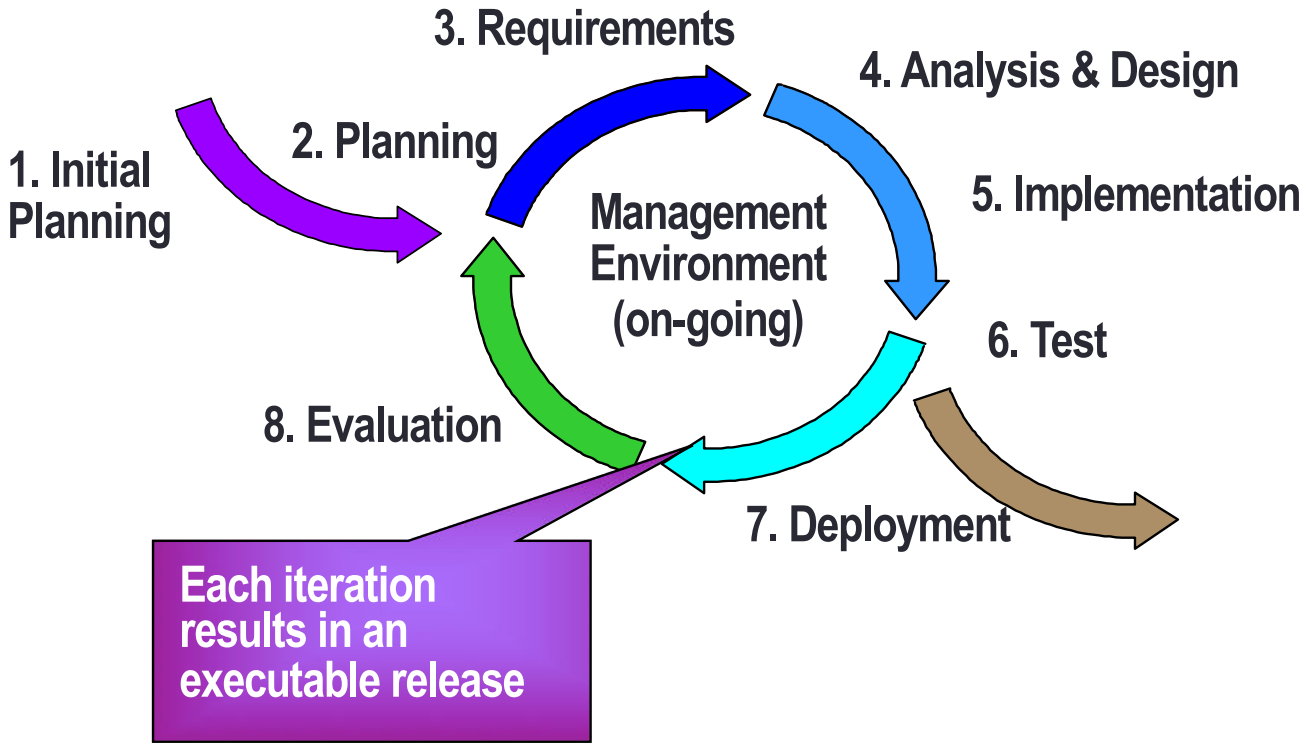
Linear, No feedback

Iterative Waterfall/Linear Model

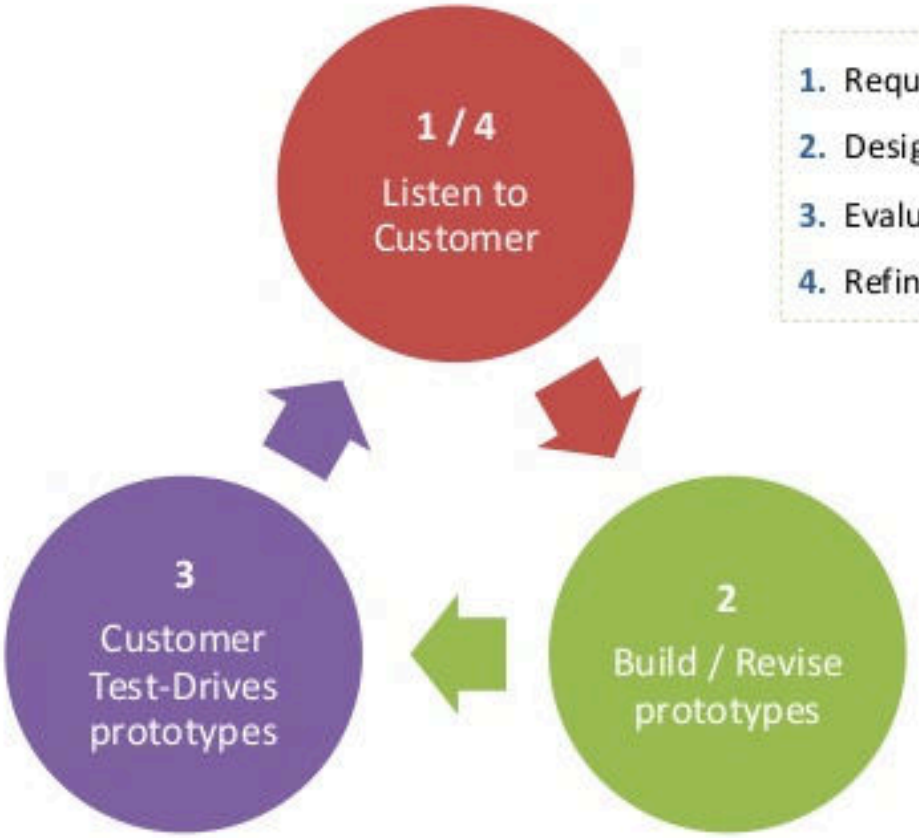


Iterative Model

- Each iteration produces an executable

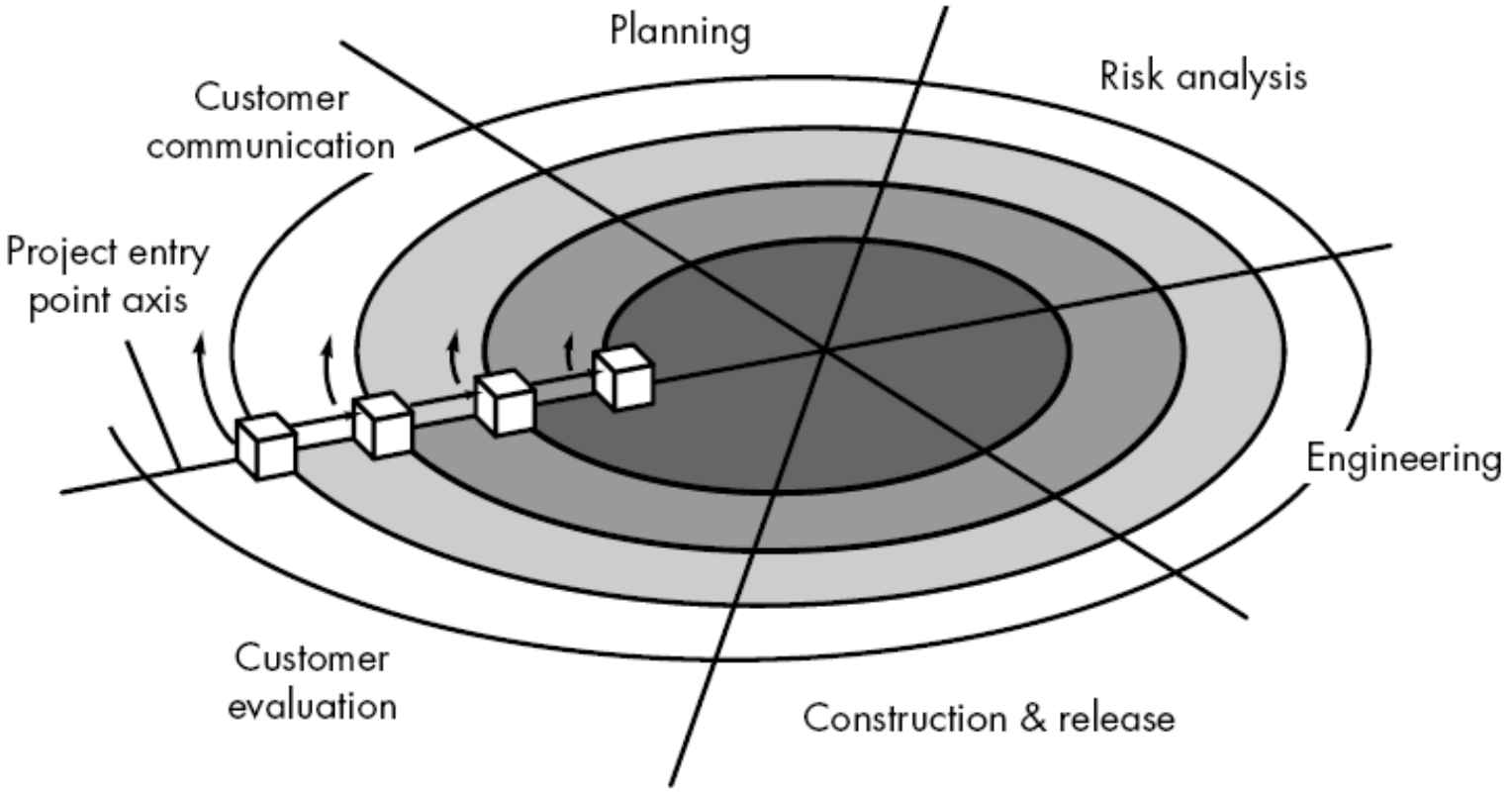


Prototype model



1. Requirements gathering.
2. Design and build SW prototype.
3. Evaluate prototype with customer.
4. Refine requirements.

Spiral model

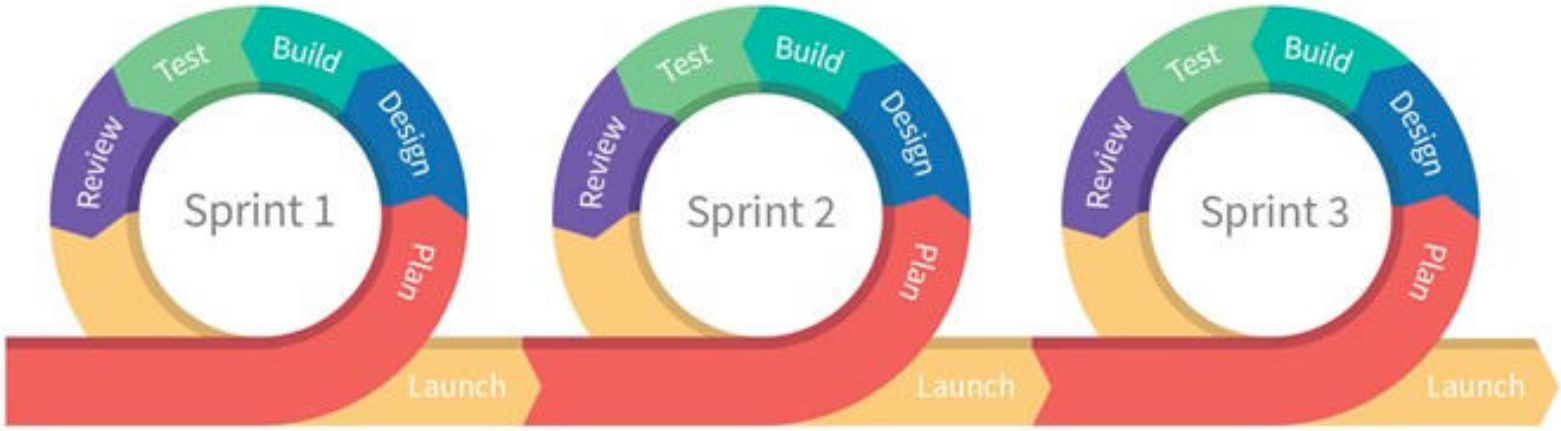


Agile methodology

- “Agility is the ability to both create and respond to change in order to profit in a turbulent buisness environment.”
  - [Jim Highsmith, Agile Software Development Ecosystems, Preface XXIII]

=> Goal: Outline values and principles to allow software teams to:

- **develop quickly** and
- **respond to change.**



The Agile manifesto

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Agile principles (1)

- 1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- 2. Welcome changing requirements, even late in the development. Agile processes harness change for the customer's competitive advantage.
- 3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to a shorter time scale.

The Agile principles (2)

- 4. Business people and developers must work together daily throughout the project.
- 5. Build projects around motivated individuals. Give them the environment and support their need, and trust them to get the job done.
- 6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

The Agile principles (3)

- 7. Working software is the primary measure of progress.
- 8. Agile processes promote sustainable development.
- 9. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- 10. Continuous attention to technical excellence and good design enhances agility.

The Agile principles (4)

- 11. Simplicity – the art of maximizing the amount of work not done – is essential.
- 12. The best architectures, requirements, and designs emerge from self-organising teams.
- 13. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.