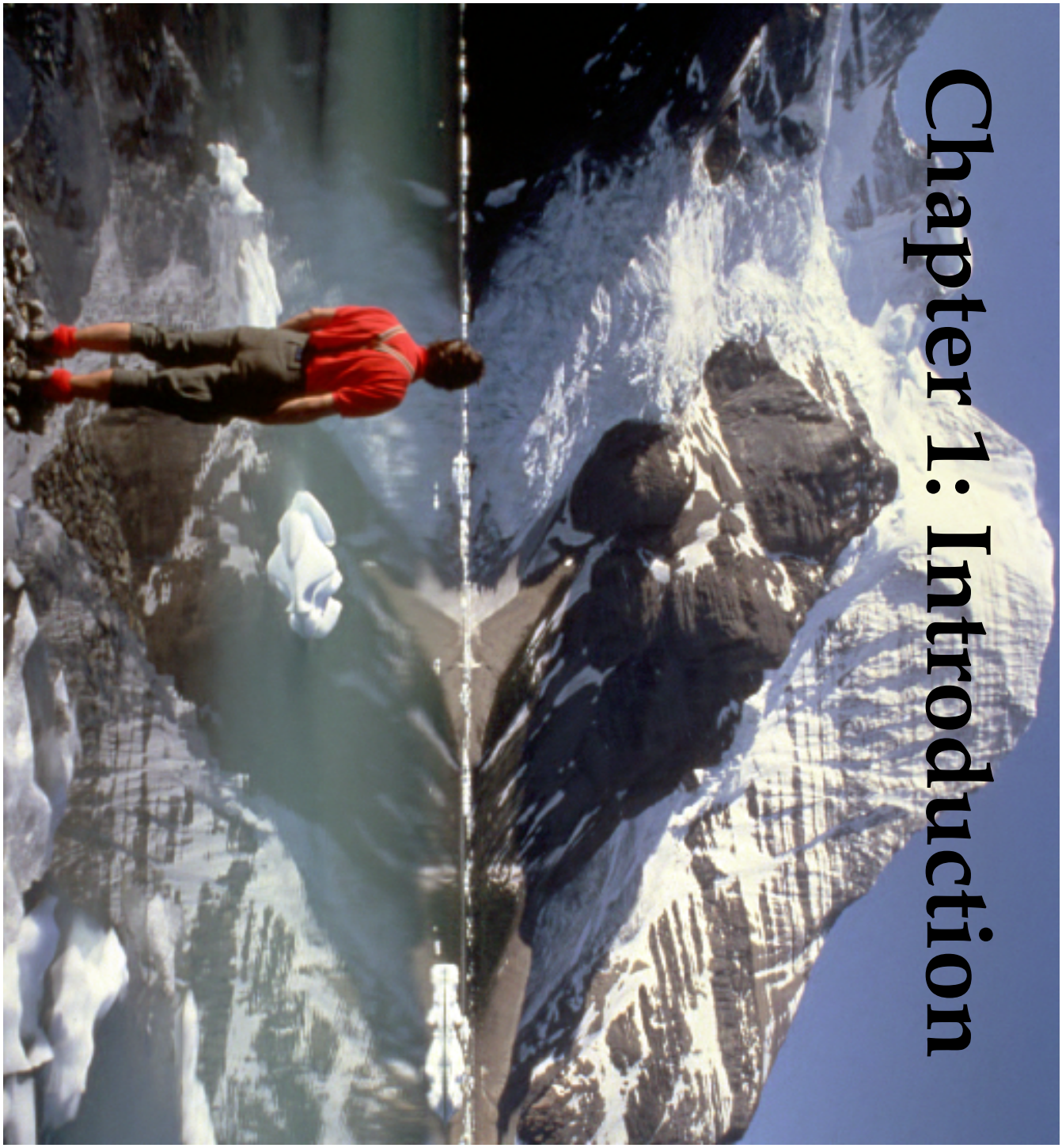
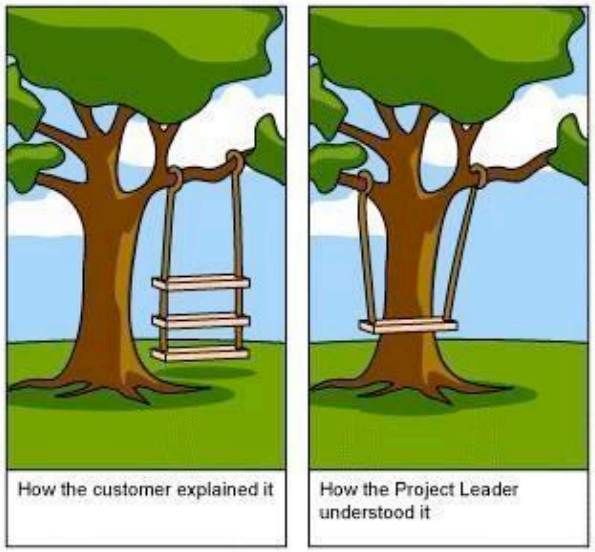
Object-Oriented Software Engineering
Using UML, Patterns, and Java

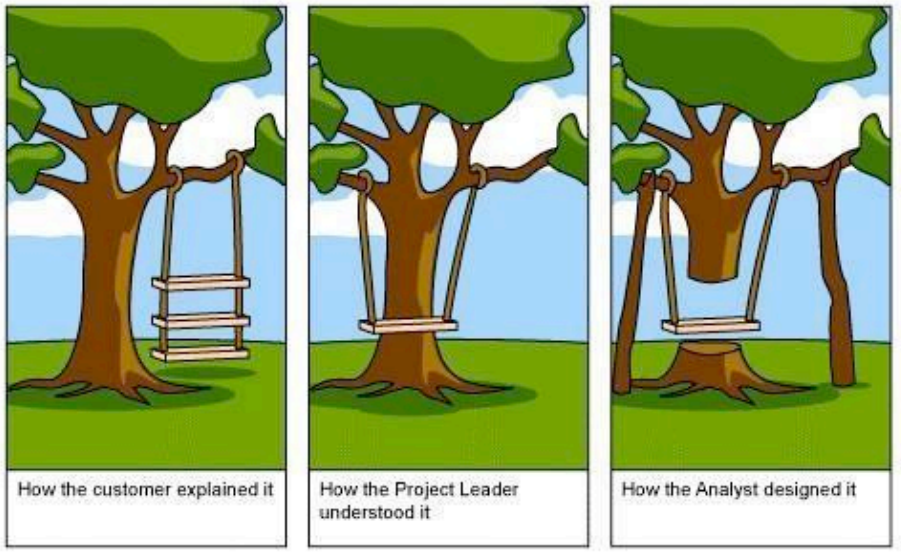


Ingegneria del software: scenario di riferimento

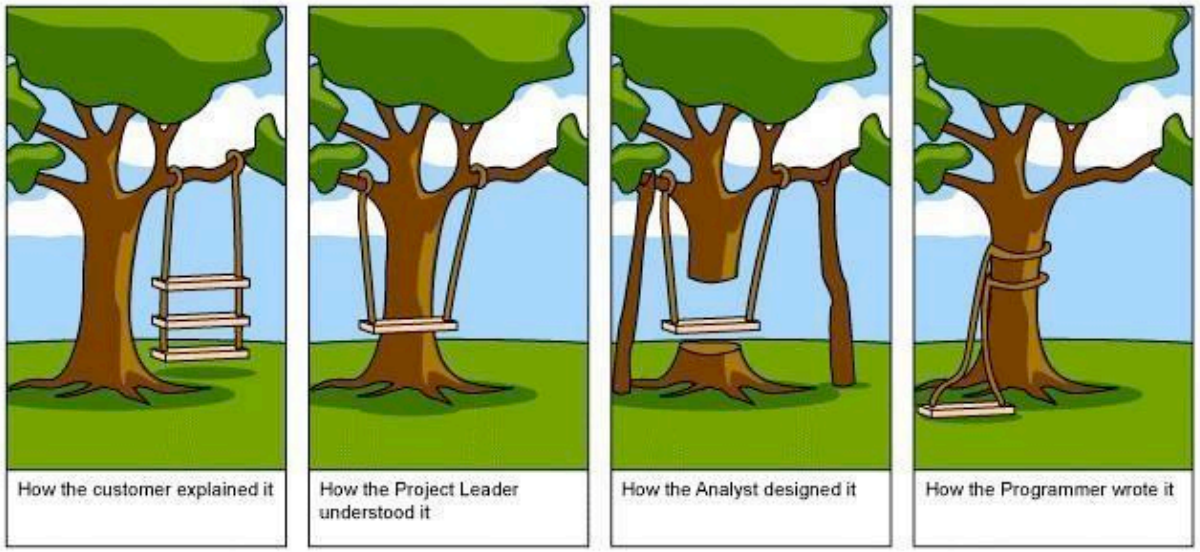
Ingegneria del software: scenario di riferimento



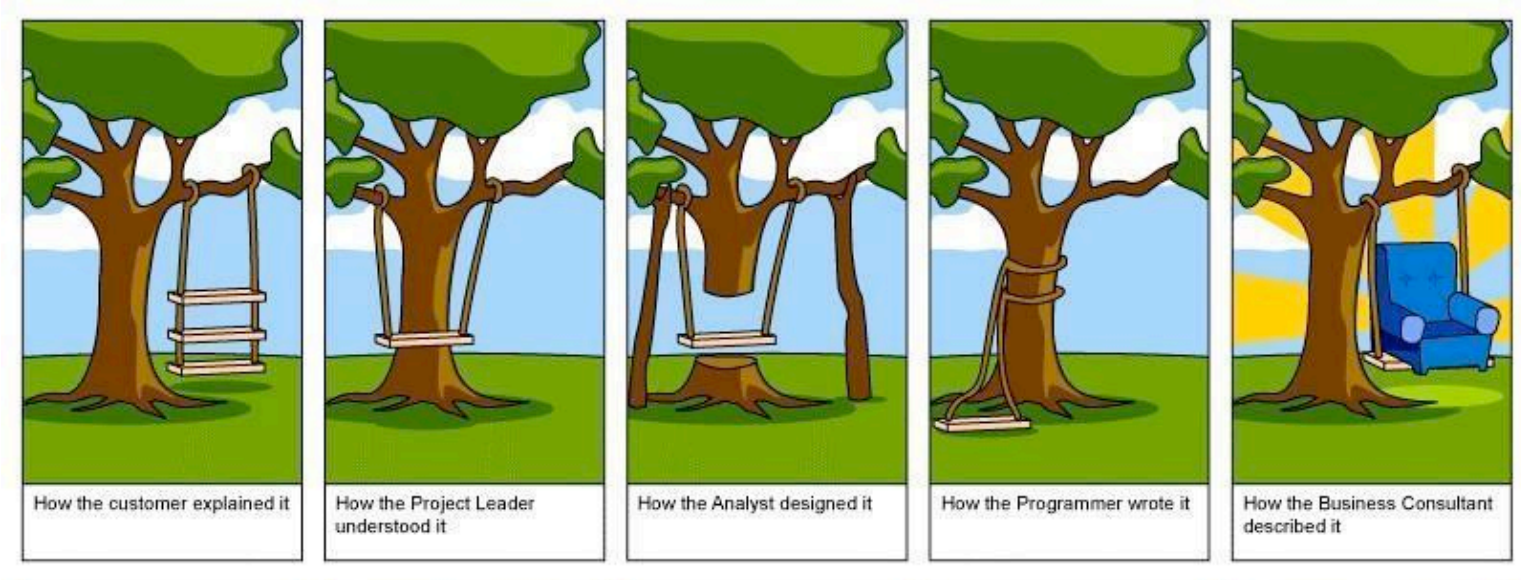
Ingegneria del software: scenario di riferimento



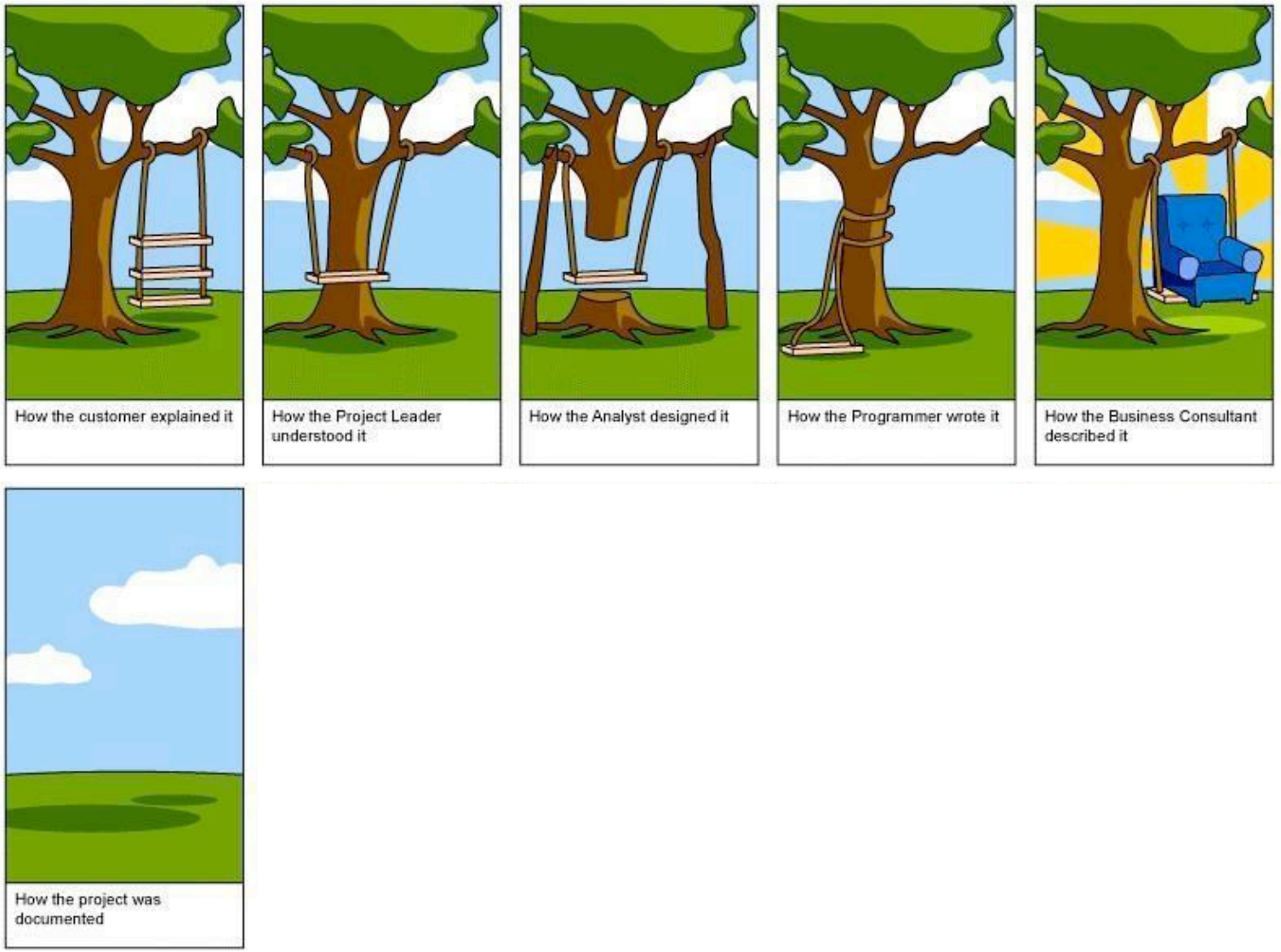
Ingegneria del software: scenario di riferimento



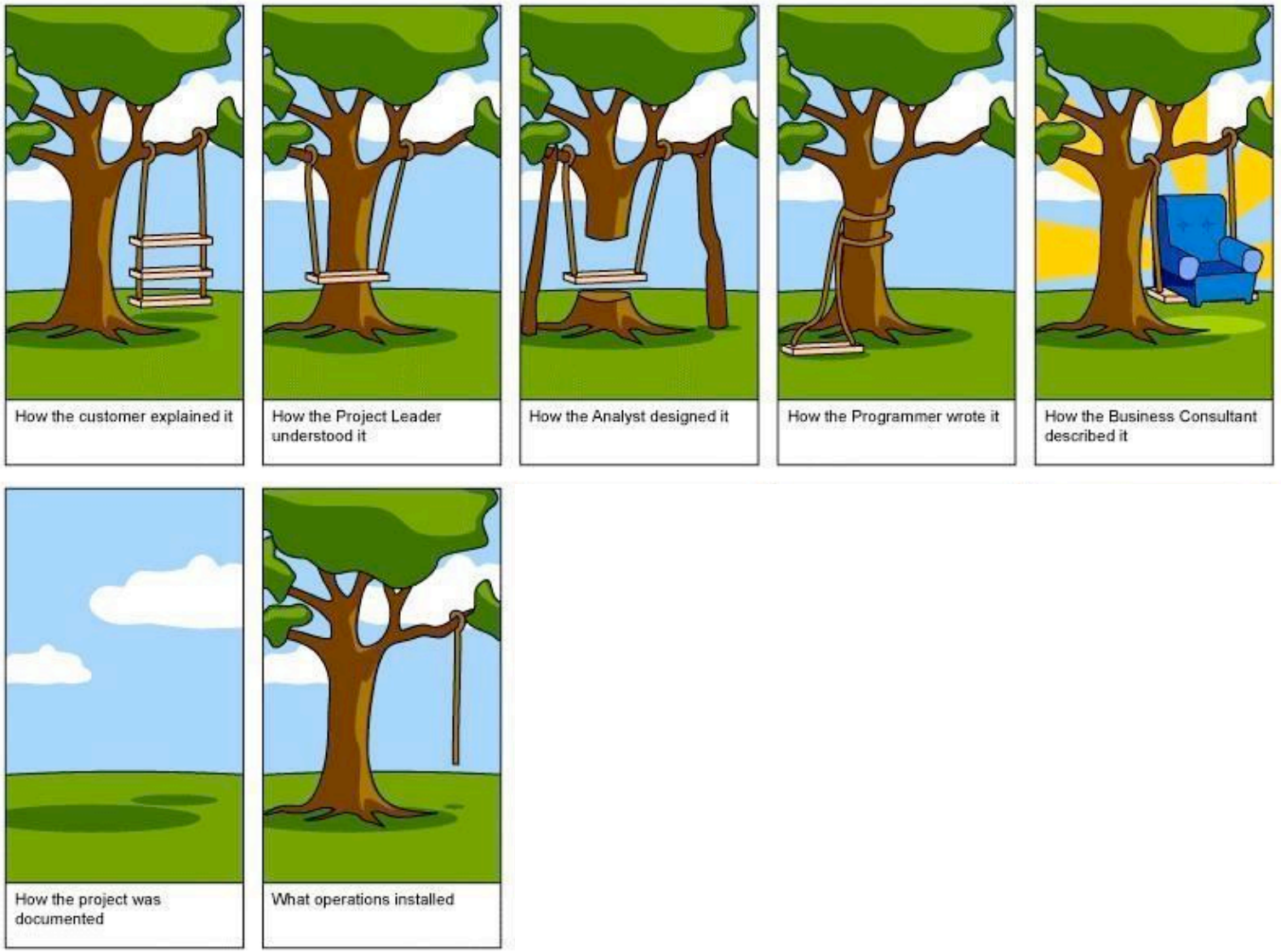
Ingegneria del software: scenario di riferimento



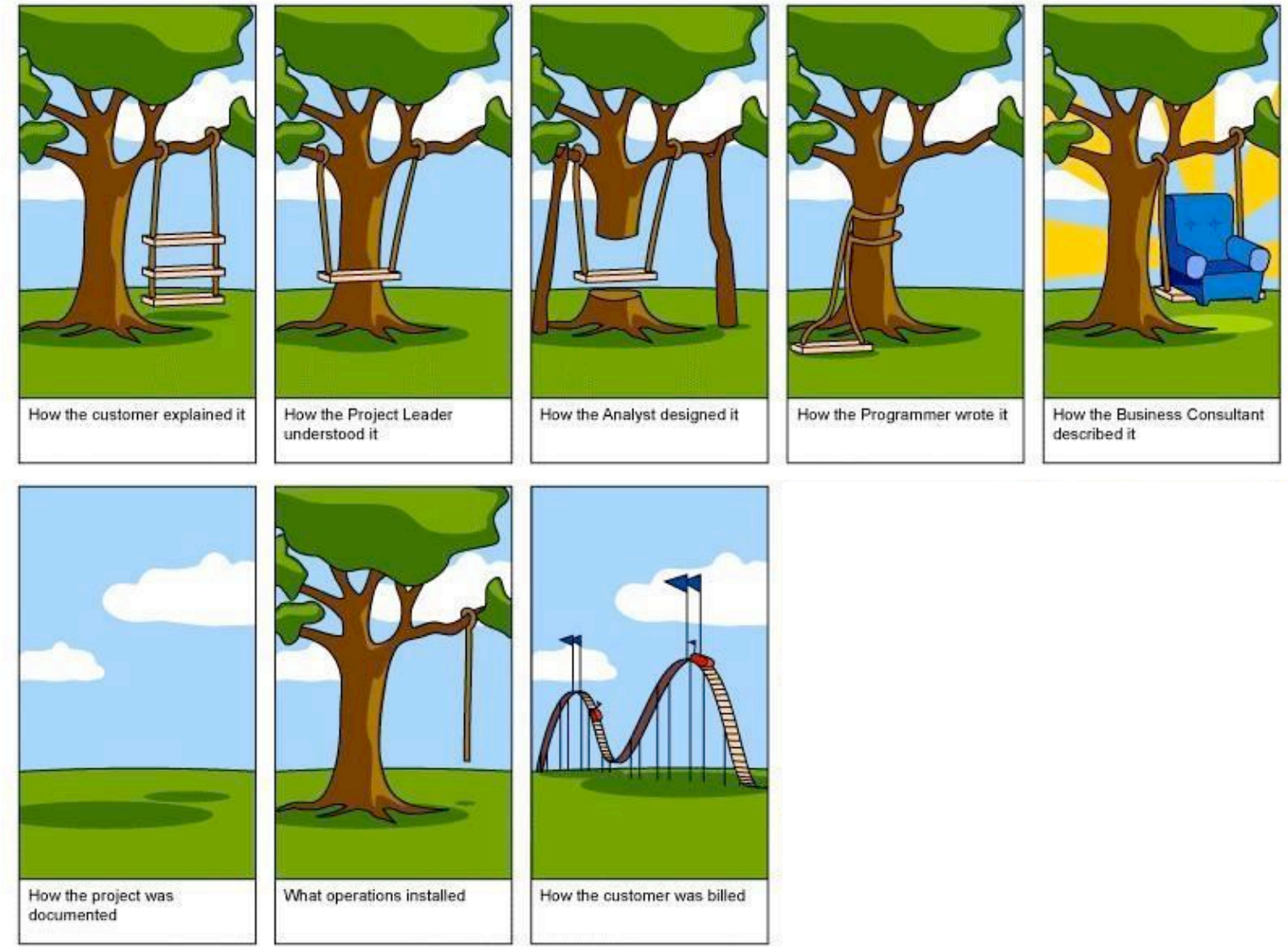
Ingegneria del software: scenario di riferimento



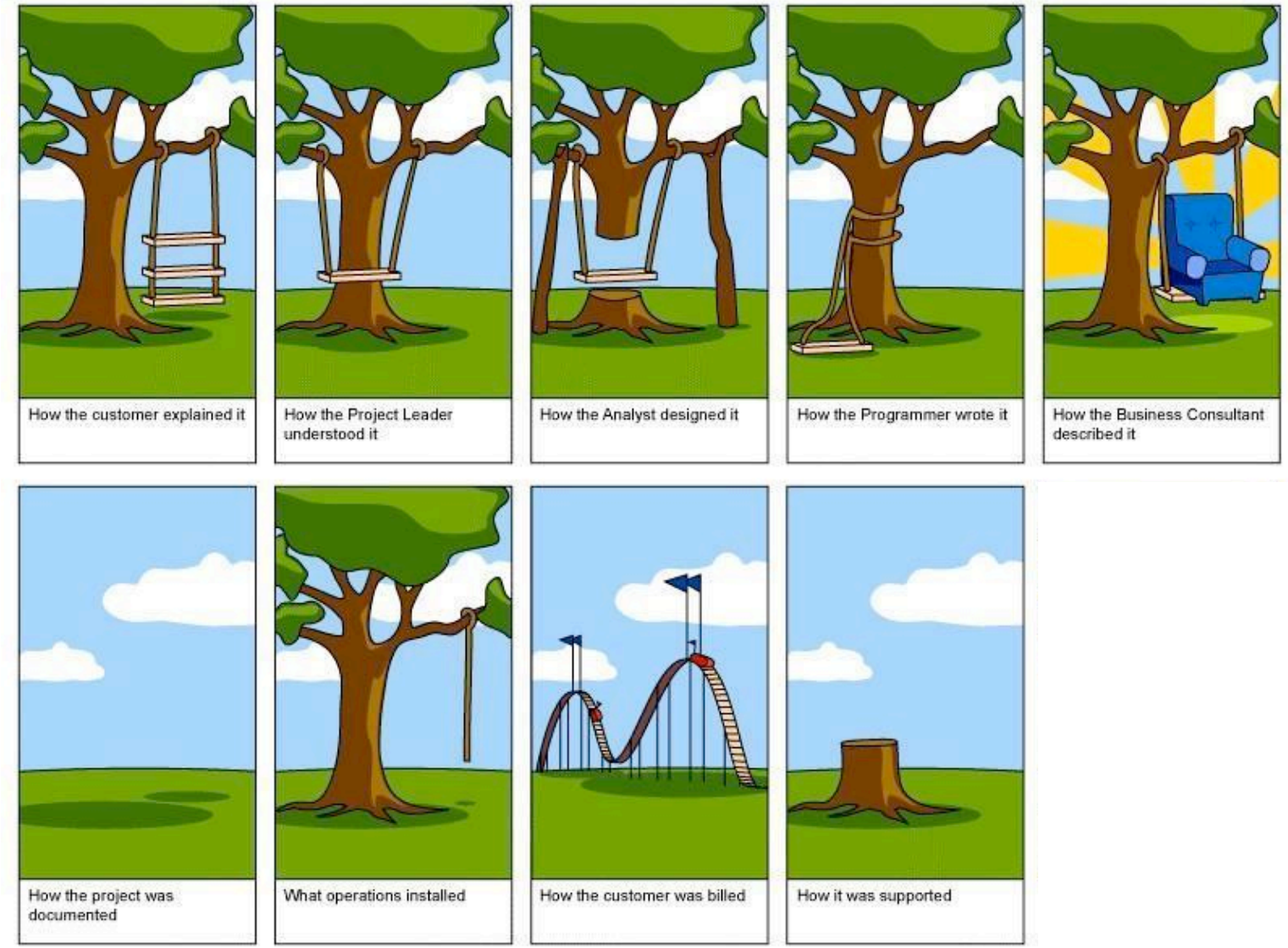
Ingegneria del software: scenario di riferimento



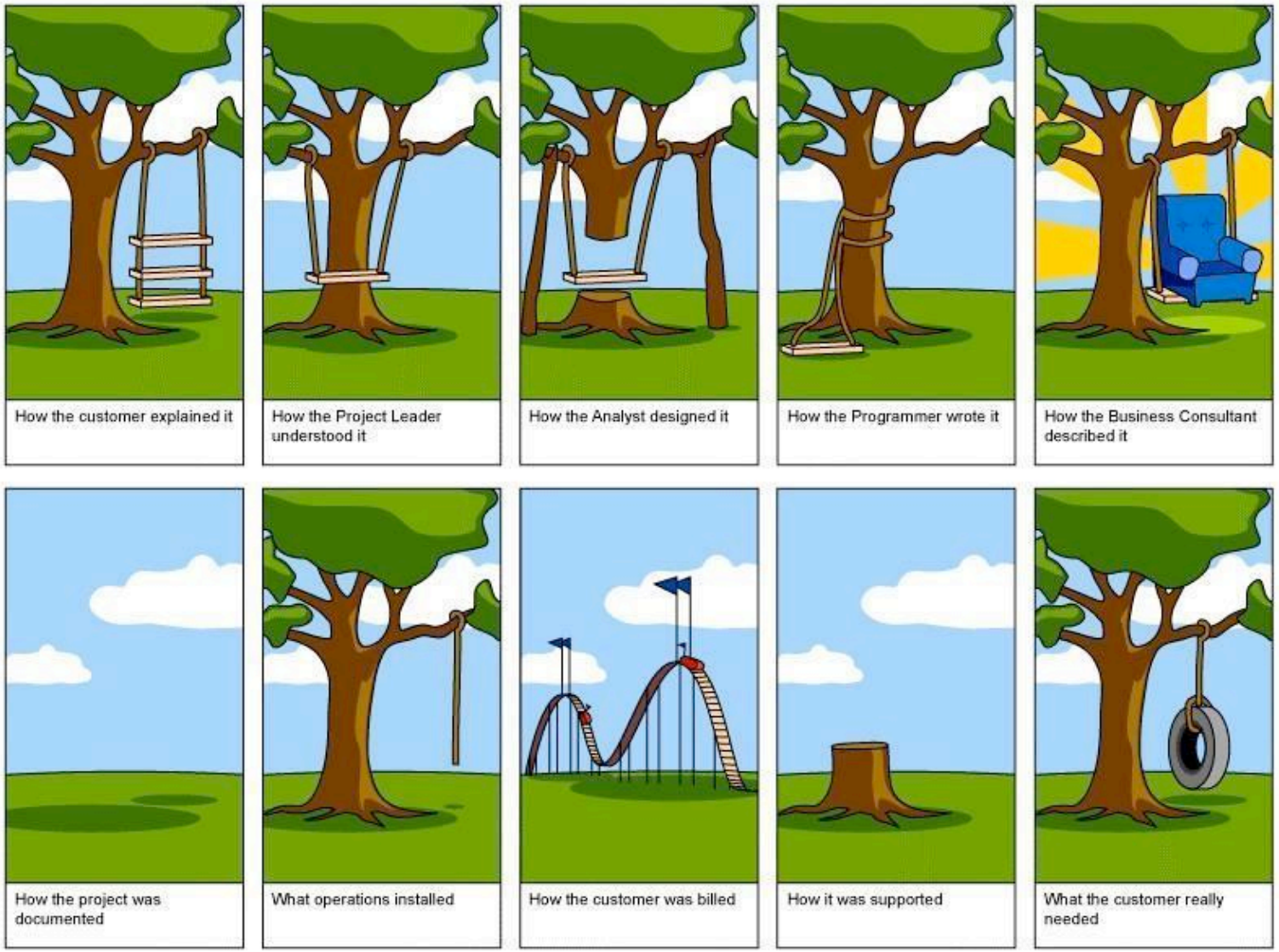
Ingegneria del software: scenario di riferimento



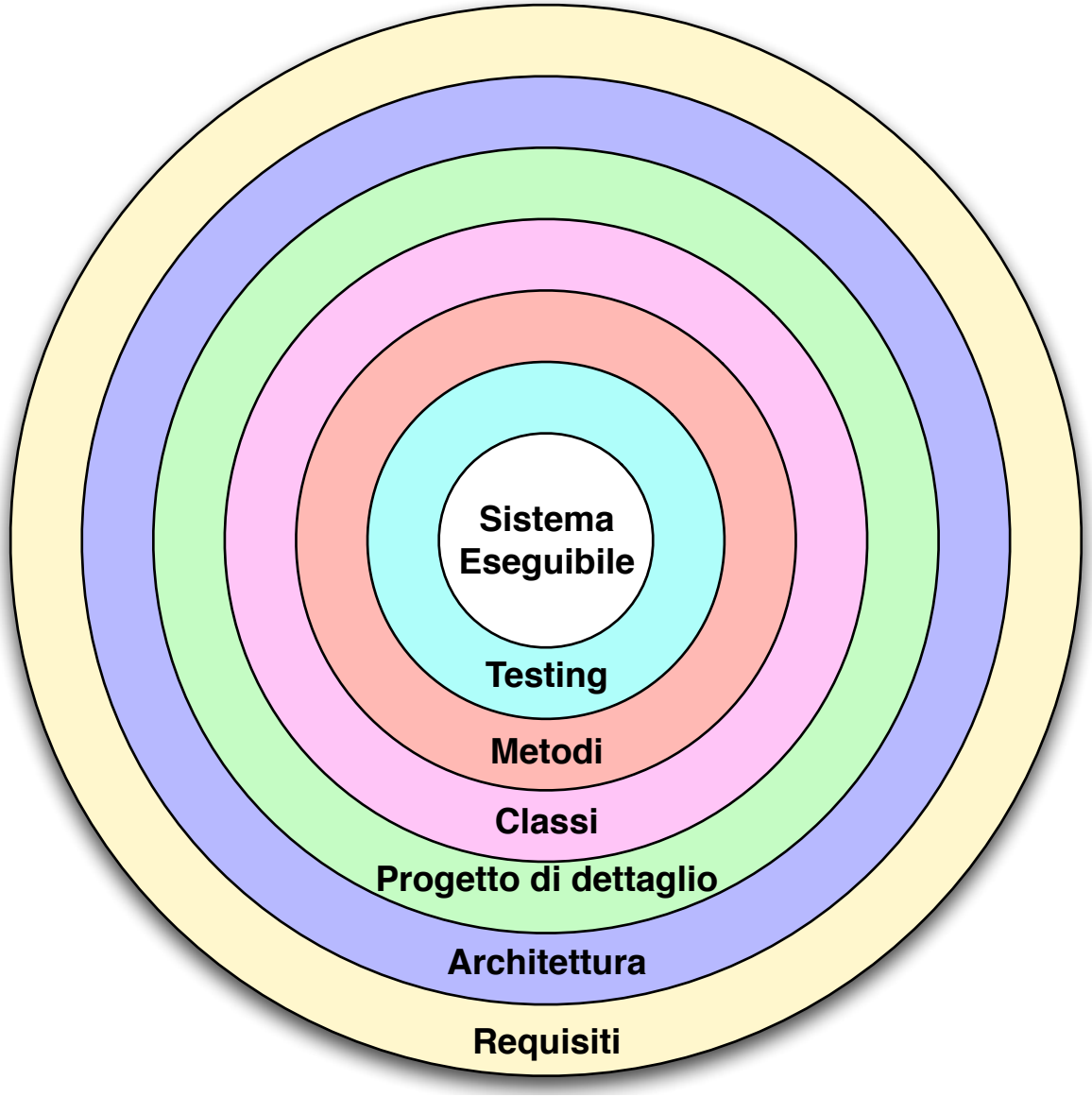
Ingegneria del software: scenario di riferimento



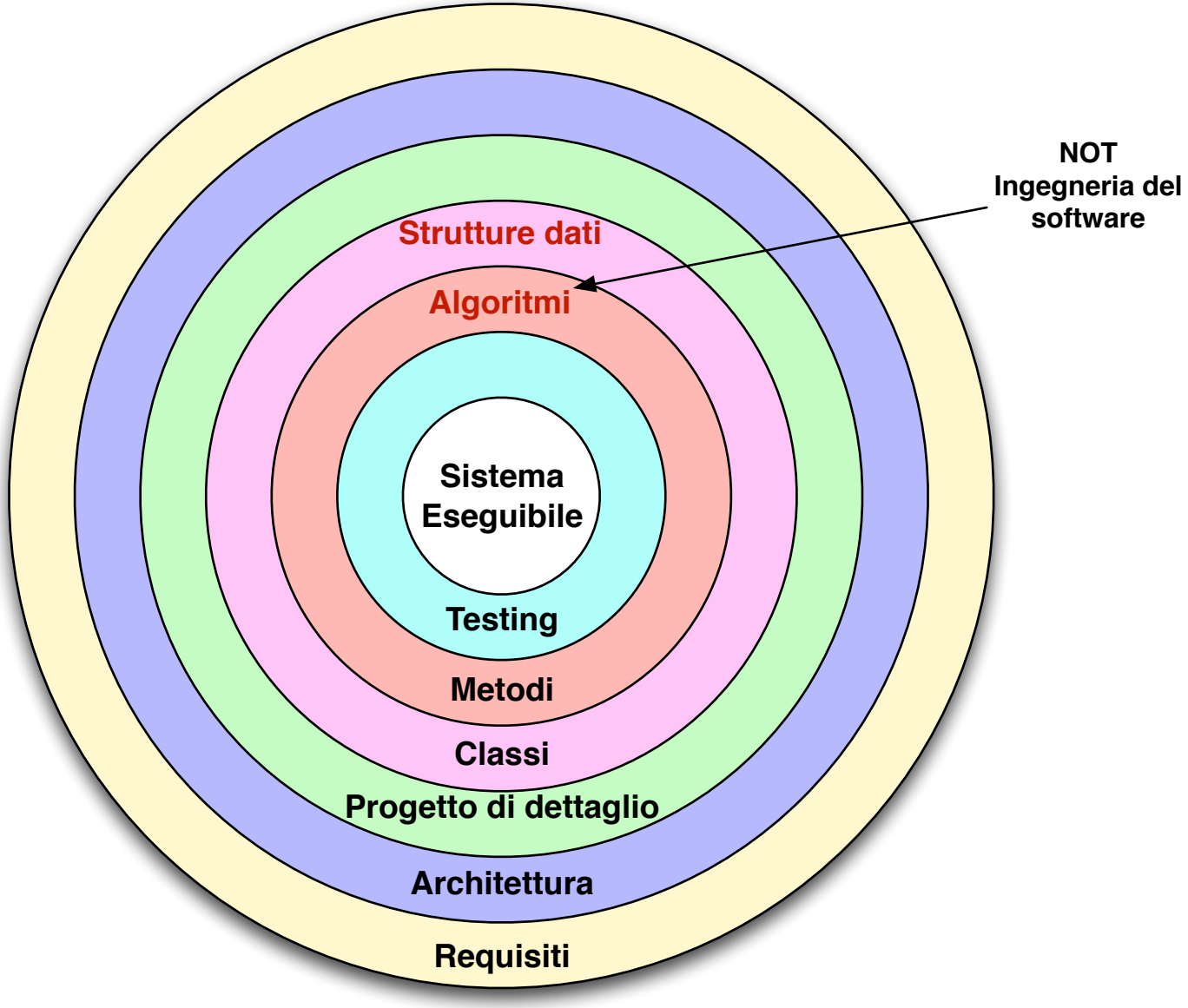
Ingegneria del software: scenario di riferimento



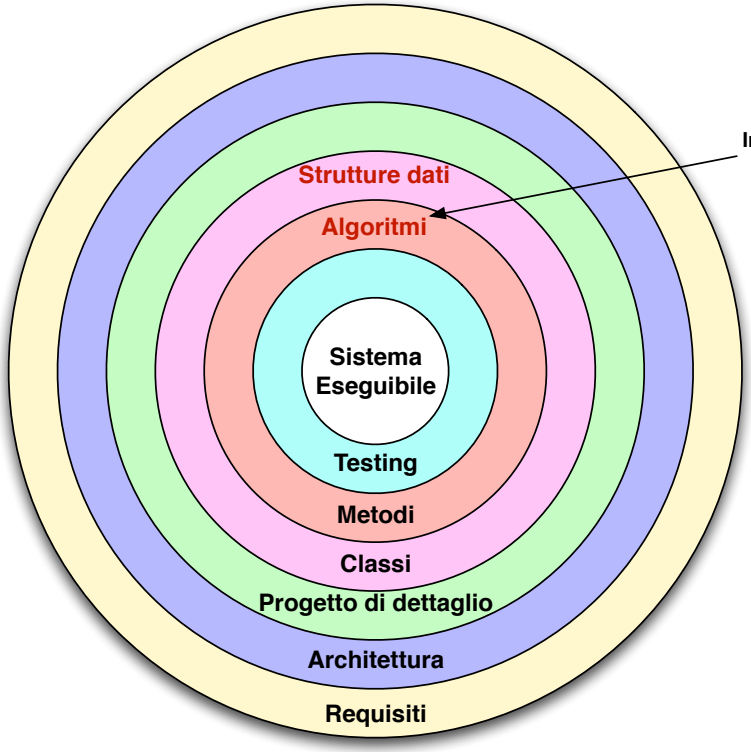
Lo sviluppo del Software: il modello a cipolla



Il modello a cipolla/2

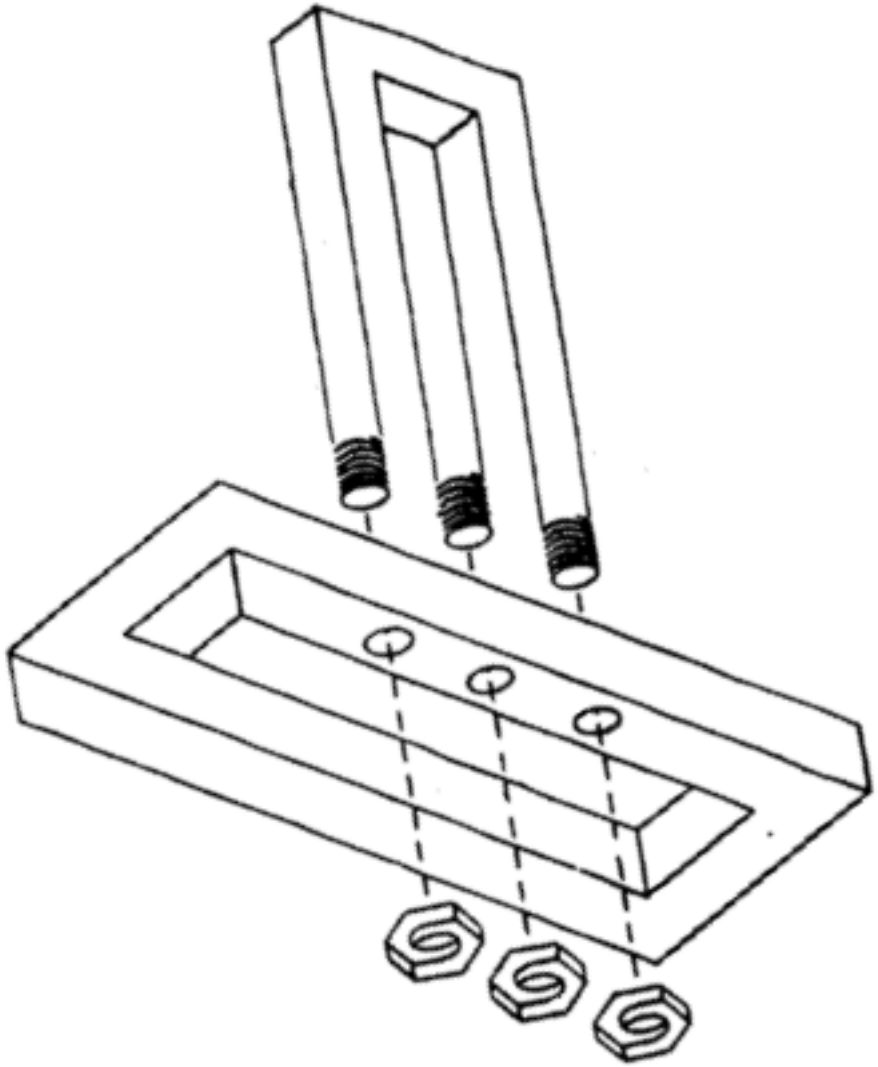


Il modello a cipolla/2

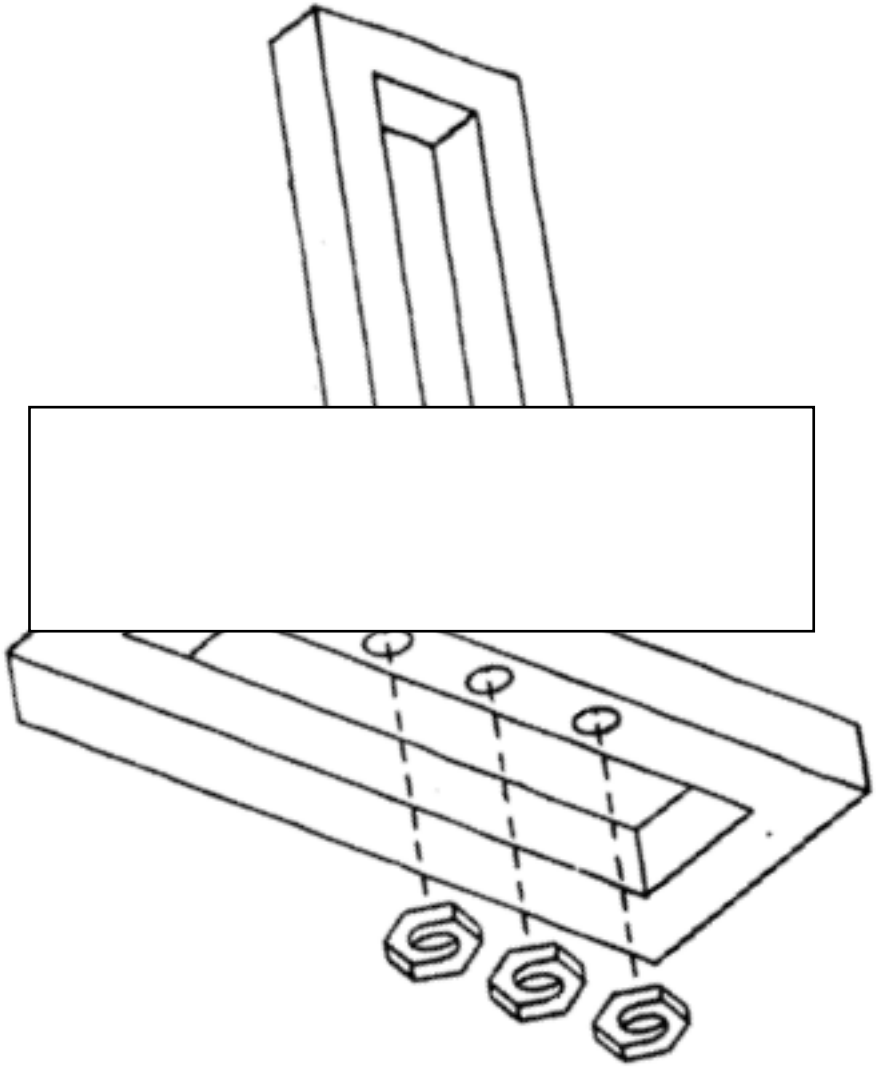


- Fino ad oggi avete studiato:
  - Algoritmi
  - Strutture dati
- Agli inizi dell'informatica bastavano a fare i 'programmi'
  - Wirth: Algoritmi +strutture dati = programmi
- L'ingegneria del software aggiunge altri 4 FONDAMENTALI strati e completa il modello

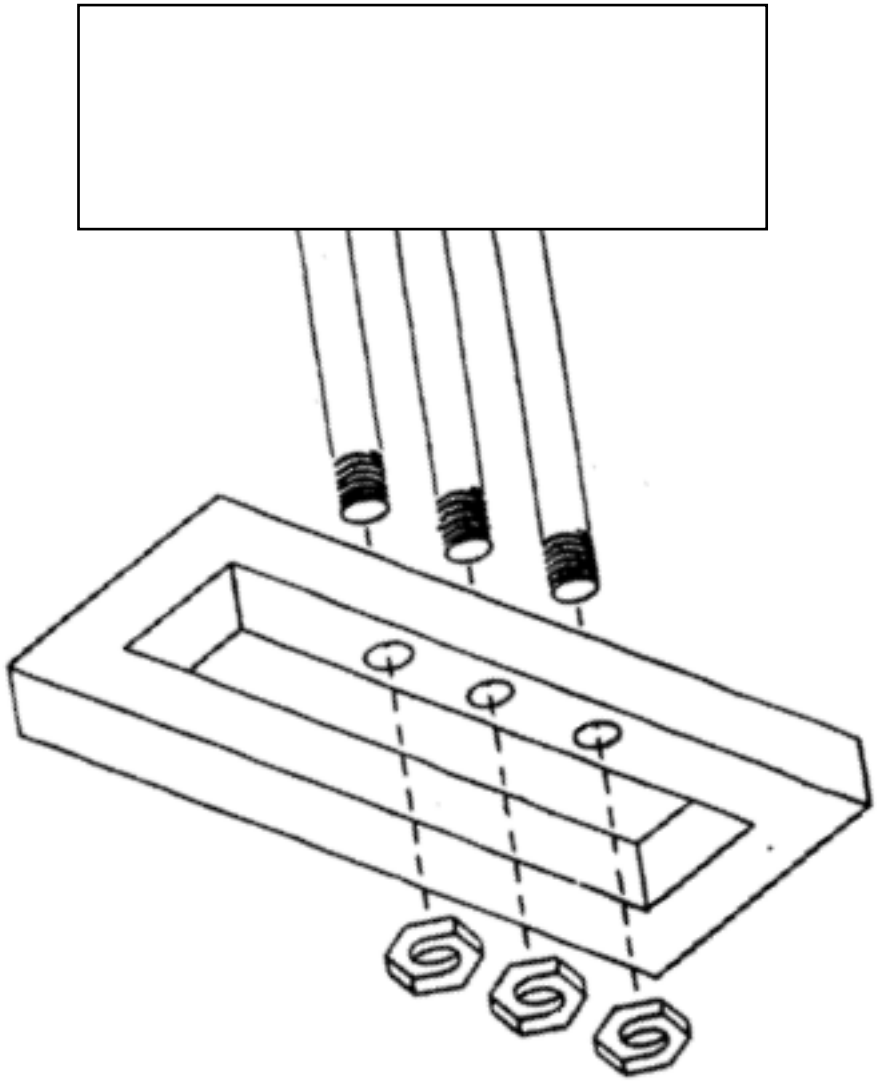
Can you develop this system?



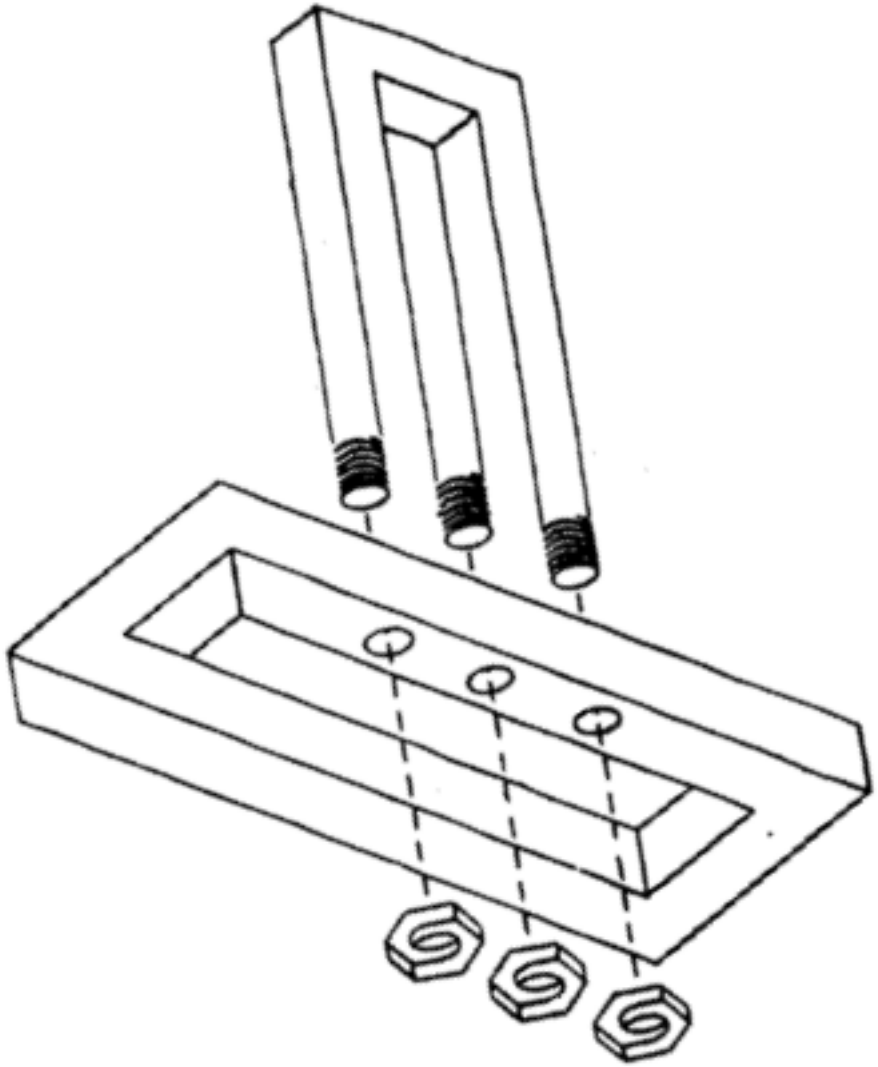
Can you develop this system?



Can you develop this system?



Can you develop this system?



The impossible Fork

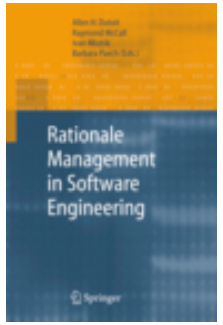
Why is Software Development difficult?

- The problem is usually ambiguous
- The requirements are usually unclear and changing when they become clearer
- The problem domain (called application domain) is complex, and so is the solution domain
- The development process is difficult to manage
- Software offers extreme flexibility
- Software is a discrete system
  - Continuous systems have no hidden surprises
  - Discrete systems can have hidden surprises!

David Lorge Parnas - an early pioneer in software engineering who developed the concepts of modularity and information hiding in systems which are the foundation of object oriented methodologies.

Software Development is more than just Writing Code

- It is problem solving
  - Understanding a problem
  - Proposing a solution and plan
  - Engineering a system based on the proposed solution using a ***good*** design
- It is about dealing with complexity
  - Creating abstractions and models
  - Notations for abstractions
- It is knowledge management
  - Elicitation, analysis, design, validation of the system and the solution process
- It is rationale management
  - Making the design and development decisions explicit to all stakeholders involved.



Computer Science vs. Engineering

- Computer Scientist
  - Assumes techniques and tools have to be developed.
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…
- Engineer
  - Develops a solution for a problem formulated by a client
  - Uses computers & languages, techniques and tools
- Software Engineer
  - Works in multiple application domains
  - Has only 3 months...
  - …while changes occurs in the problem formulation (requirements) and also in the available technology.

Software Engineering: A Working Definition

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

A high quality ***software*** system developed with a given *budget* before a given *deadline* while *change* occurs

Challenge: Dealing with complexity and change

Course Outline

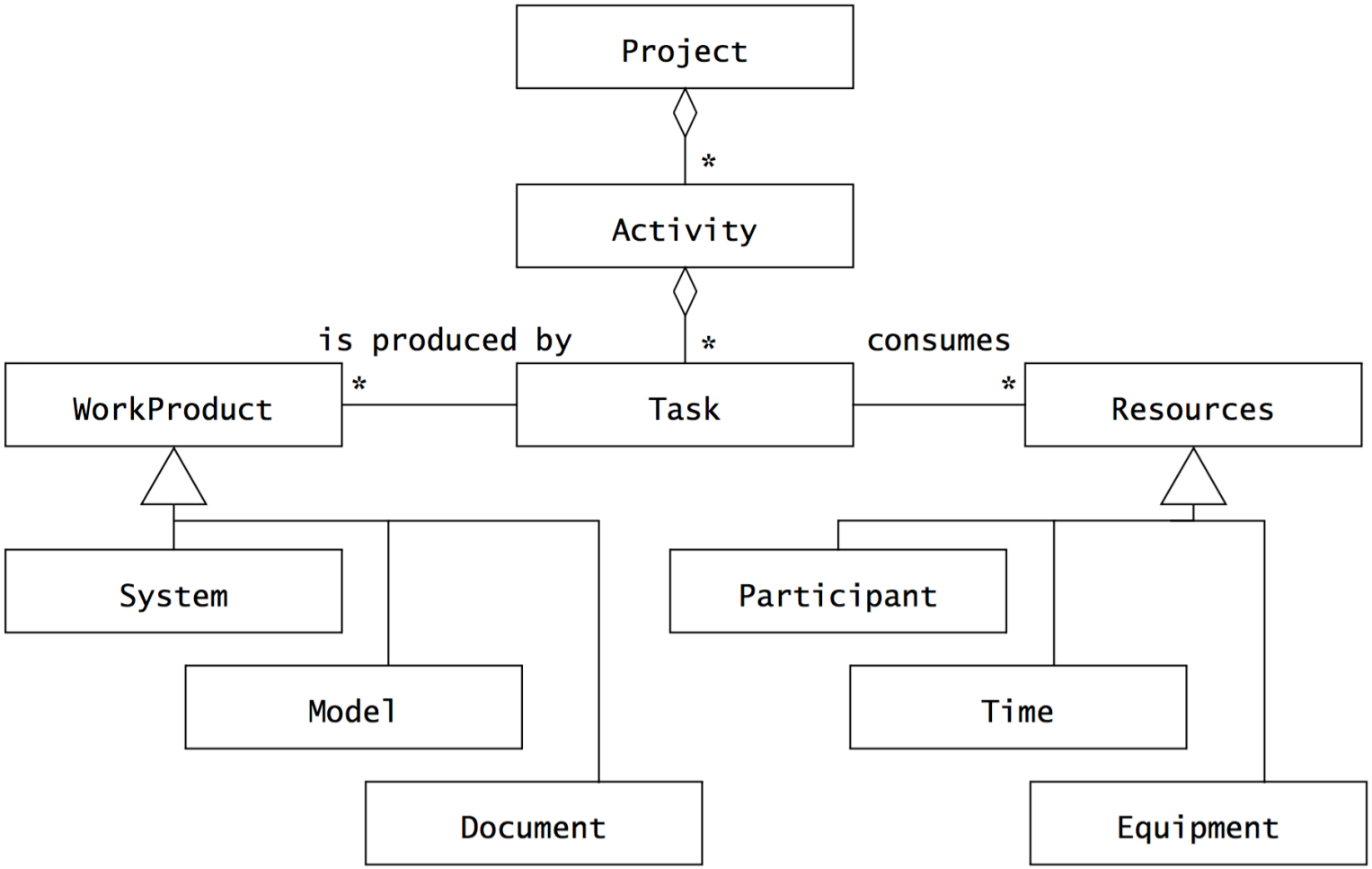
Dealing with Complexity

- Notations (UML, OCL)
- Requirements Engineering, Analysis and Design
  - OOSE, SA/SD, scenario-based design, formal specifications
- Testing
  - Vertical and horizontal testing

Dealing with Change

- Rationale Management
  - Knowledge Management
  - ~~Patterns~~
- Release Management
  - Configuration Management, Continuous Integration
- Software Life Cycle
  - Linear models
  - Iterative models
  - Activity-vs Entity-based views
- ~~Project Management~~

Software Engineering Concepts



Participants and Roles

- Participants
  - We refer to all the persons involved in the project as participants.
  - E.g.: Client, Developer, Project Manager
- Roles
  - A role is associated with a set of **tasks** and is assigned to a participant. The same participant can fill multiple roles.

Role	Responsibilities
Client	The client is responsible for providing the high-level requirements on the system and for defining the scope of the project (delivery date, budget, quality criteria).
User	The user is responsible for providing domain knowledge about current user tasks. Note that the client and the user are usually filled by different persons.
Manager	A manager is responsible for the work organization. This includes hiring staff, assigning them tasks, monitoring their progress, providing for their training, and generally managing the resources provided by the client for a successful delivery.

Systems and Models

- **System** is a collection of interconnected parts.
  - Modeling is a way to deal with complexity by ignoring irrelevant details.
- The term **model** refers to any abstraction of the system.
- A development project is itself a system that can be modeled.
  - The project schedule, its budget, and its planned deadlines are models of the development project.

Work Products

- A **work product** is an artifact that is produced during the development,
  - E.g.: a document or a piece of software for other developers or for the client.
- **Internal work product** - a work product for the project's internal consumption
- **Deliverable** - a work product that must be delivered to a client.
  - Deliverables are generally defined prior to the start of the project and specified by a contract binding the developers with the client.

Examples of work products

Work product	Type	Description
Specification	**Deliverable**	The specification describes the system from the user's point of view. It is used as a contractual document between the project and the client. The `TicketDistributor` specification describes in detail how the system should appear to the traveler.
Operation manual	**Deliverable**	The operation manual for the `TicketDistributor` is used by the staff of the train company responsible for installing and configuring the `TicketDistributor`. Such a manual describes, for example, how to change the price of tickets and the structure of the network into zones.
Status report	**Internal work product**	A status report describes at a given time the tasks that have been completed and the tasks that are still in progress. The status report is produced for the manager, Alice, and is usually not seen by the train company.
Test manual	**Internal work product**	The test plans and results are produced by the tester, Zoe. These documents track the known defects in the prototype `TicketDistributor` and their state of repair. These documents are usually not shared with the client.

Activities, Tasks, and Resources

- An **activity** is a set of tasks that is performed toward a specific purpose.
  - Activities can be composed of other activities.
  - Activities are also sometimes called **phases**.
- A **task** represents an atomic unit of work that can be managed
  - for instance assigned by a project manager, performed by a developer, …)
  - Tasks consume resources, result in work products, and depend on work products produced by other tasks.
- **Resources** are assets that are used to accomplish work.
  - Resources include time, equipment, and labor.

Functional and Nonfunctional Requirements

- **Requirements** specify a set of features that the system must have.
- A **functional requirement** is a specification of a function that the system must support
  - The user must be able to purchase tickets
  - The user must be able to access tariff information
- A **nonfunctional requirement** is a constraint on the operation of the system that is not related directly to a function of the system.
  - The user must be provided feedback in less than one second
  - The colors used in the interface should be consistent with the company colors

Notations, Methods, and Methodologies

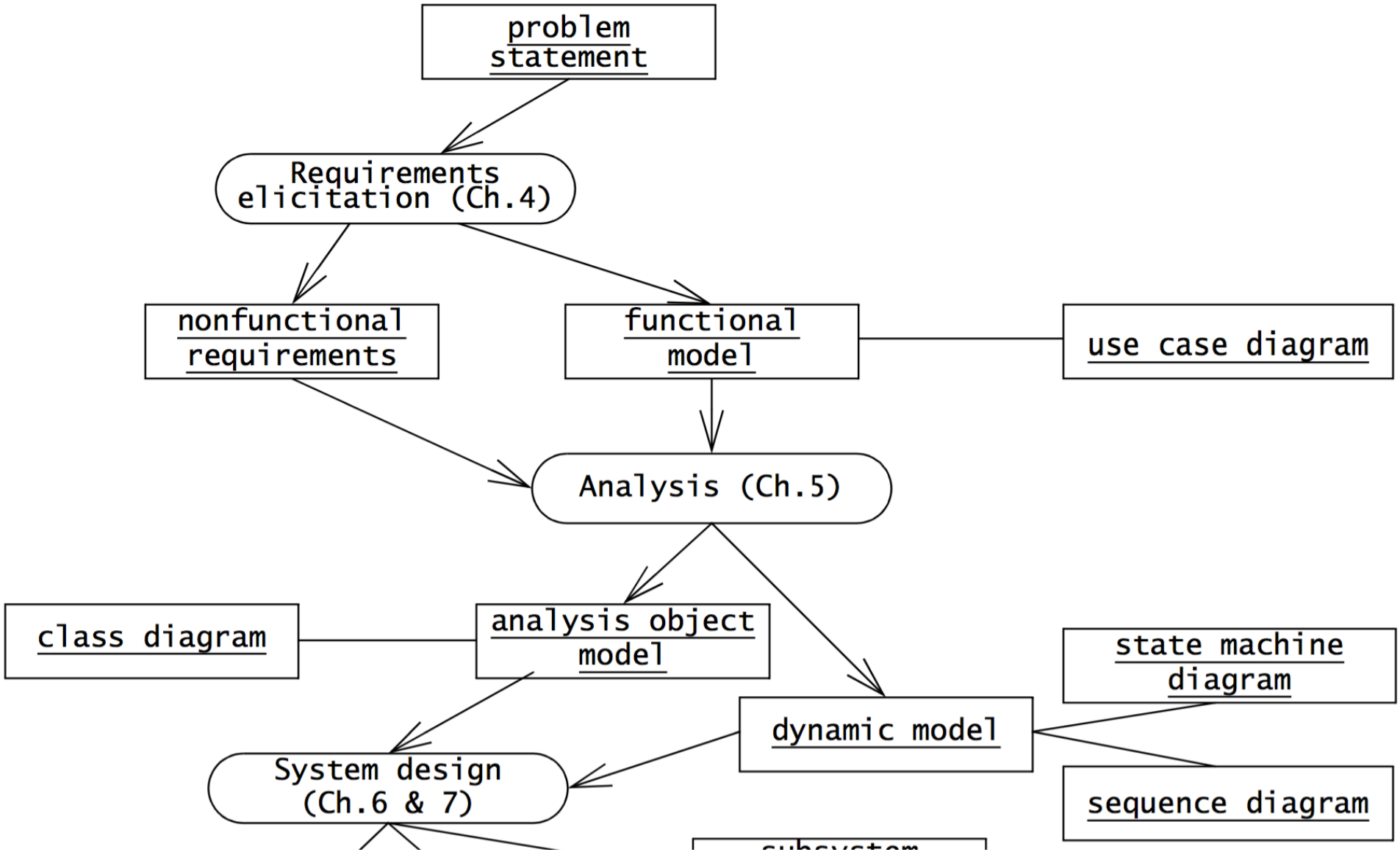
- A **notation** is a graphical or textual set of rules for representing a model.
  - The Roman alphabet is a notation for representing words.
  - UML (Unified Modeling Language) is a notation for representing object-oriented models.
- A **method** is a repeatable technique that specifies the steps involved in solving a specific problem.
  - A recipe is a method for cooking a specific dish.
  - A sorting algorithm is a method for ordering elements of a list.
  - Rationale management is a method for justifying change.
  - Configuration management is a method for tracking change.

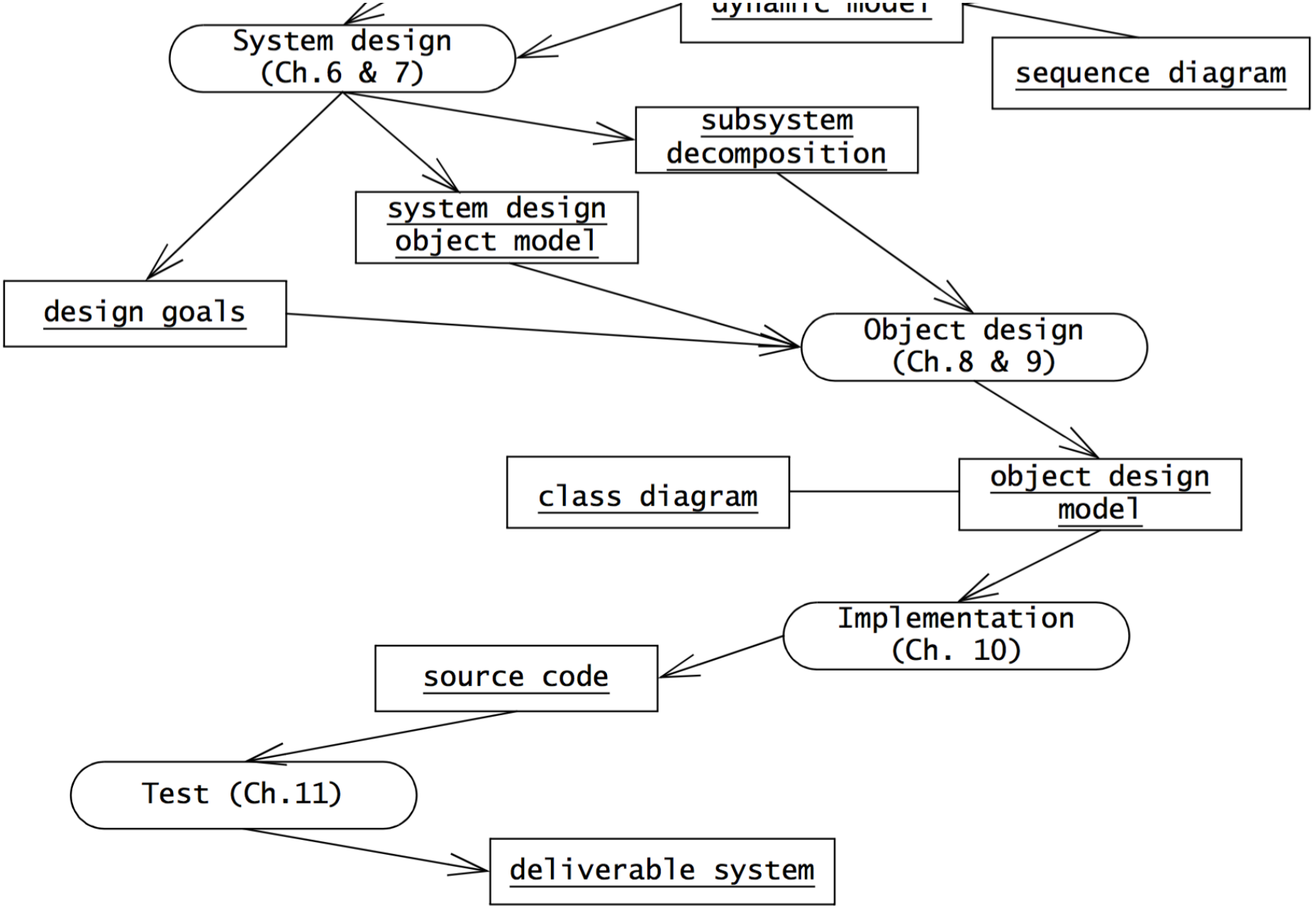
Notations, Methods, and Methodologies/2

- A **methodology** is a collection of methods for solving a class of problems and specifies how and when each method should be used.
  - A seafood cookbook with a collection of recipes
    - it also contains advice on how ingredients should be used and what to do if not all ingredients are available.
- Software development methodologies decompose the process into activities.
  - Examples of such activities are:
    - Analysis, which focuses on formalizing the system requirements into an object model,
    - System Design, which focuses on strategic decisions,
    - Object Design, which transforms the analysis model into an object model that can be implemented.

Software Engineering Development Activities

- Requirements Elicitation (Section 1.4.1)
- Analysis
- System Design
- Object Design
- Implementation
- Testing


problem statement
Requirements elicitation (Ch.4)
nonfunctional requirements
functional model
use case diagram
Analysis (Ch.5)
class diagram
analysis object model
state machine diagram
dynamic model
System design (Ch.6 & 7)
sequence diagram


System design
(Ch.6 & 7)
dynamic model
sequence diagram
subsystem
decomposition
system design
object model
design goals
Object design
(Ch.8 & 9)
class diagram
object design
model
Implementation
(Ch. 10)
source code
Test (Ch.11)
deliverable system

Requirements Elicitation

- Client and developers define the purpose of the system.
  - **Deliverable**: a description of the system in terms of actors and use cases.
  - **Actors** represent the external entities that interact with the system.
    - Actors include roles such as end users, other computers the system needs to deal with and the environment (e.g., a chemical process).
  - **Use cases** are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality.

Analysis

- Developers produce a model of the system that is correct, complete, consistent, and unambiguous.
  - Developers transform use cases into an object model that describes the system.

System Design

1. Developers define the design goals of the project
2. Developers decompose the system into smaller subsystems that can be realized by individual teams.
3. Developers select strategies for building the system
   - the hardware/software platform on which the system will run,
   - the persistent data management strategy,
   - …

Object Design

- Developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design.
  - It means describing object and subsystem interfaces, selecting off-the-shelf components, …

Implementation

- Developers translate the solution domain model into source code.

Testing

- Developers find differences between the expected system behaviour and its actual one by executing the system (or parts of it) with sample input data sets.