Software Engineering

Block 1: Basic topics

BACHELORS DEGREE IN BIOINFORMATICS, SOFTWARE ENGINEERING

ESCI upf.
School of International Studies
BioInformatics-Software Engineering

Outline

In this session we will look at three different aspects of software development:

- The evolution of software and its challenges
- The different software development paradigms
  - Linear model
  - Prototyping model
  - Evolutionary model
  - Agile models
- Process methods and tools

Software engineering principles

In the field of software engineering, these are basic concepts:

- Problem: something that we need to solve.
- Specification: problem description and analysis.
- Algorithm: problem solution (written for humans).
- Program: problem solution (written for computers).

So, the objective is to implement a program from a problem in a correct way.

Software engineering principles: Main Objective

What then is the main objective of the software engineering?

Solving a problem using computing programing in the best possible way
… and in the most efficient way.

Therefore, the key factor is the **efficiency** of:

- The implementation (program).
- And the task of development (software engineering).

Both aspects are fundamental, so we focus on both.

Software engineering Objectives

Operative Objectives

- ***Correctness***: to what extent the program satisfies the initial specifications.
- ***Accuracy***: it is the property that defines with what degree the software complies with the established requirements.
- ***Efficiency***: it is a factor that refers in every way to the execution of the software, and includes factors such as response time, memory requirements and processing capacity.
- ***Integrity***: software accessibility control for unauthorized persons.
- ***Usability:*** the necessary effort to lean and use the software in a correct way

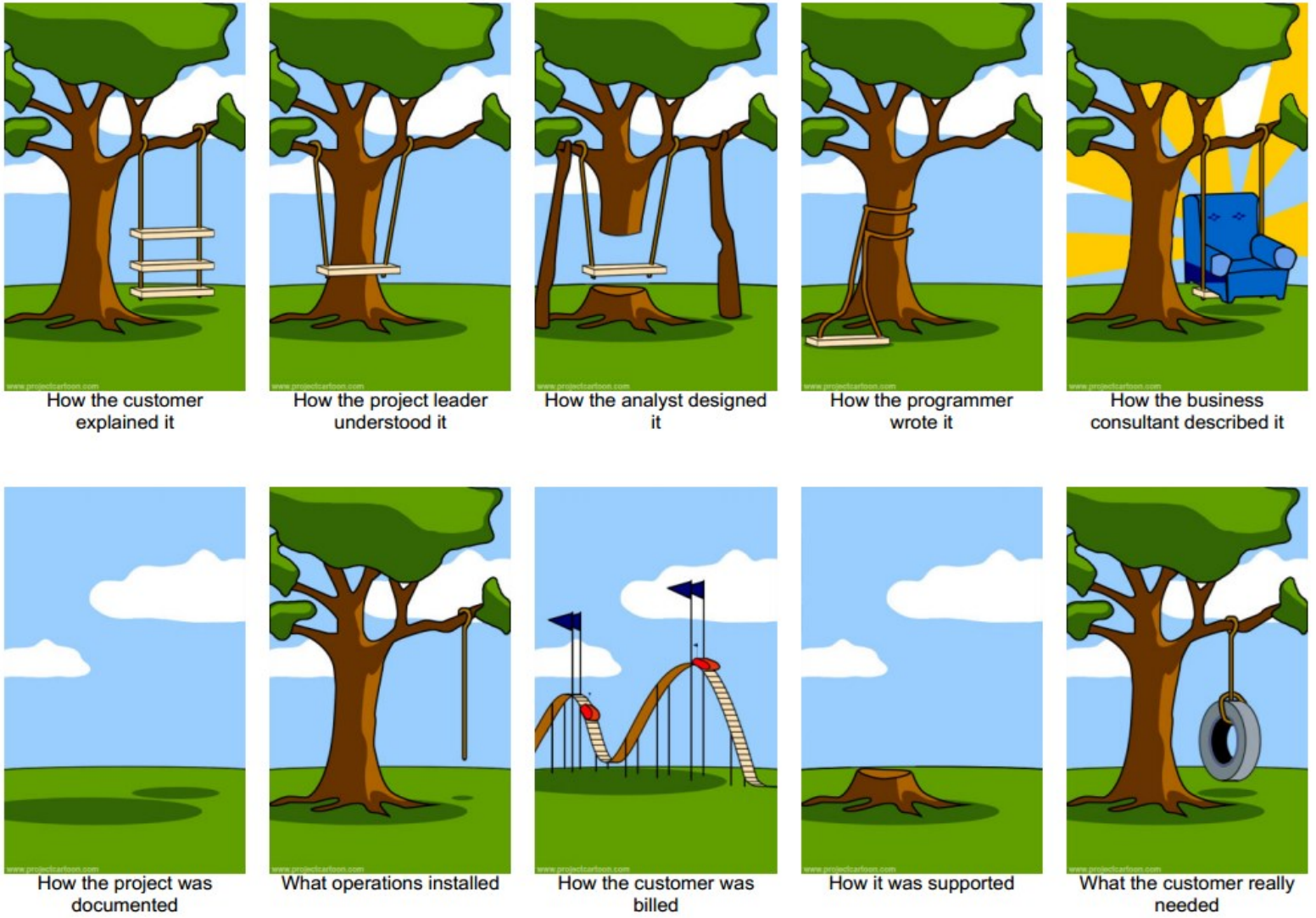
Revision Objectives

- ***Maintenance***: this is the effort required to detect and correct errors in a program that is installed.
- ***Testability***: it is the effort required to prove, to ensure that the system or module performs its purpose.
- ***Flexibility***: it is the necessary effort to modify a program once it is installed.

Transition Objectives

- ***Transportability***: it is the effort required to transfer the software from one hardware configuration to another.
- ***Reusability***: to what extent parts of the software can be reused in other applications.
- ***Interoperability***: it is the necessary effort to connect the system with other systems.

Software engineering Objectives: troubles



Software evolution: a challenge

	1960 ± 5 Programming-any-which-way	*1970 ± 5 Programming-in-the-small*	*1980 ± 5 Programming-in-the-large*	*1990 ± 5 Programming-in-the-world*
Specifi-cations	Mnemonics, precise use of prose	Simple input-output specifications	Systems with complex specifications	Distributed systems with open-ended, evolving specs
Design Empha-sis	Emphasis on small programs	Emphasis on algorithms	Emphasis on system structure, management	Emphasis on subsystem interactions
Data	Representing structure, sym-bolic information	Data structures and types	Long-lived databases	Data & computation independently created, come and go
Control	Elementary understanding of control flow	Programs execute once and terminate	Program systems execute continually	Suites of independent processes cooperate

The complexity of development increases exponentially!

Software development Life Stages

The relevant stages in software development are:

- **Requirements**: The problem defined by the customer, written in human language.
- **Analysis**: Analysis of the requirements, ordered by priority, …
- **Design**: Decide which architecture, what parts, etc. fits better with the analysis.
- **Coding**: Implement what has been designed.
- **Testing**: Quality assurance to ensure the coding has no errors (issues).
- **Acceptance**: The customer must check that the implementation cover its requirements.

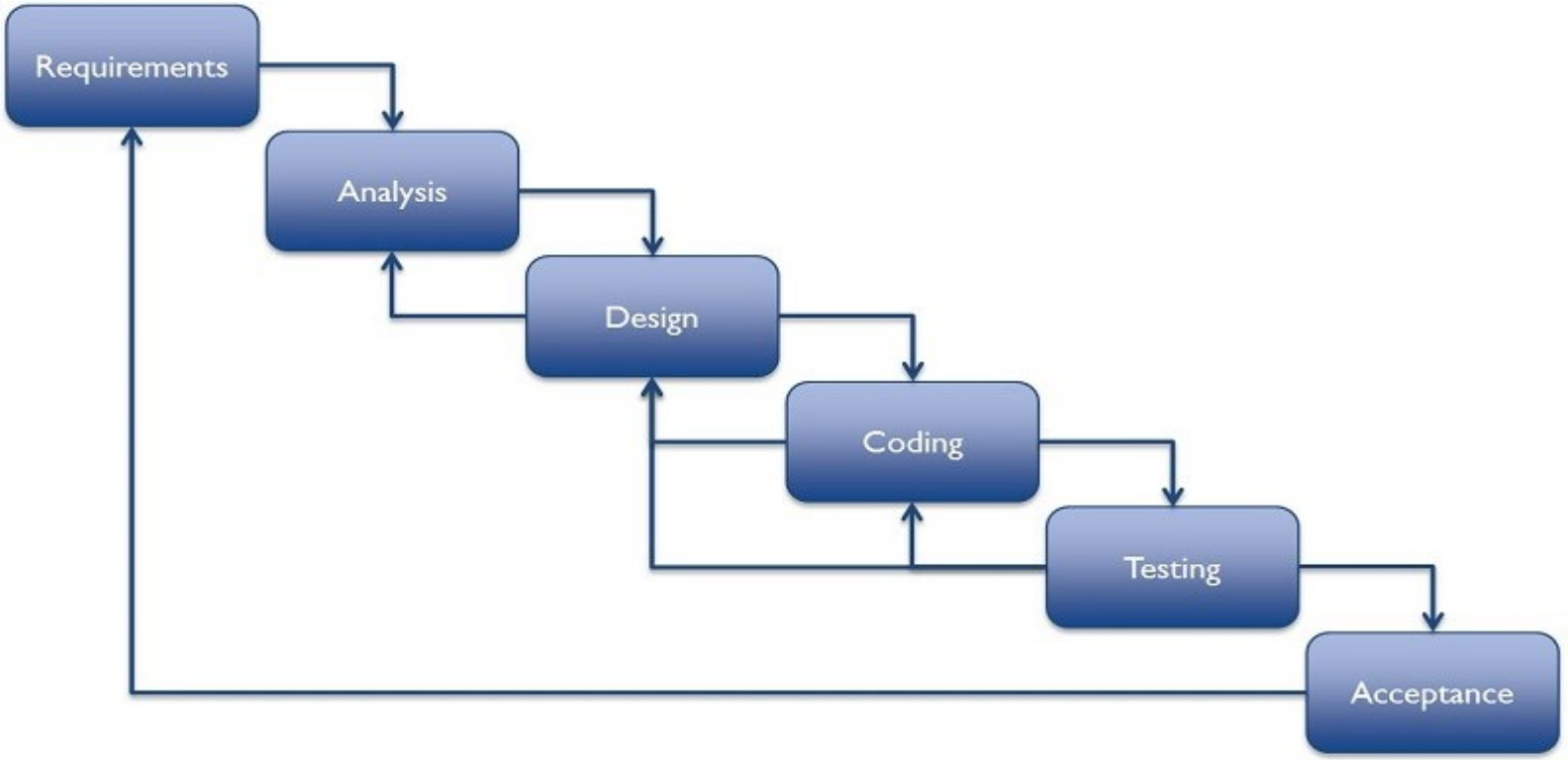
This is what we call the Software Life Cycle.

Software development paradigms

There are different ways (paradigms) of completing the software lifecycle:

- **Sequential linear model**: The classical life cycle of waterfall or cascading.
- **Prototype construction model**: Centered on the UI for identify the functionalities.
- **Rapid Application Development model (RAD)**: Construction based on short features additions into final project. It's a sequential linear model but with an extremely short time.
- **Evolutionary model**: Takes into consideration the evolutionary nature of the software. It's an iterative model that allows the development of more and more complete versions:
  - **Assembling of Components**: Starting from an initial class library, combine those that are useful and add new ones that are added to the bookstore.
  - **Incremental/Concurrent development model**: Modifies the stages of the ES process as a set of states and transitions between them.
  - **Spiral**: Multiple cascading iterations choosing to implement some part in each iteration.
  - **Model of formal methods**: Allows a mathematical specification of the software.

Software development paradigms: Waterfall



Software development paradigms: Waterfall

ADVANTAGES

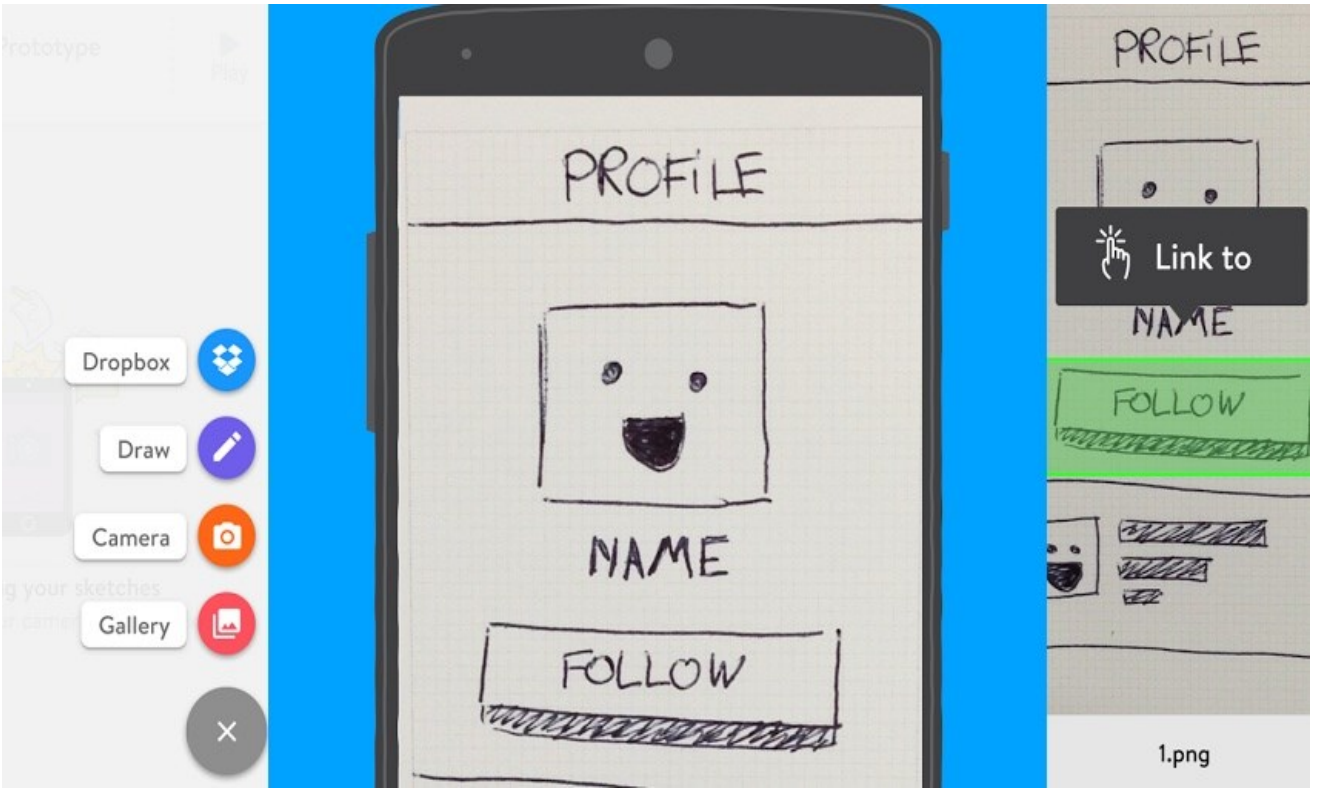
- Start-to-End of each defined stage → Easily measurable project progress.
- Promote detailed documentation.
- Signed agreement of system requirements.

Software development paradigms: Waterfall

DISAVANTAGES

- A real project is never so markedly sequential.
- It’s necessary to consider the possibility of returning to previous stages of development.
- Difficulty in explicitly establishing all the requirements at the beginning of the project.
- The client may not have requirements clear or may change during the process.
- The client must wait to see the result at the end.
- All planning is geared to a single day of delivery.
- It does not allow a stage development.
- Sometimes it would be better to do a part of the system and then improve it and complete it, but this can not be done if all the requirements must be known a priori.

Software development paradigms: Prototyping



Software development paradigms: Prototyping

ADVANTAGES

- Improvement of the first three drawbacks of the classical life cycle.
- It is developed from a set of known requirements (general objectives): collection of global objectives, rapid design, construction of the prototype, prototype test.
- Enable two forms:
  - In width: all functions for limited or inefficient algorithms.
  - In depth: a subset of the final product, problematic functions.

Software development paradigms: Prototyping

DISAVANTAGES

- The client may see it as a definitive software.
- Prototypes can be made with inadequate resources: so the developer should not "get carried away" for the comfort.

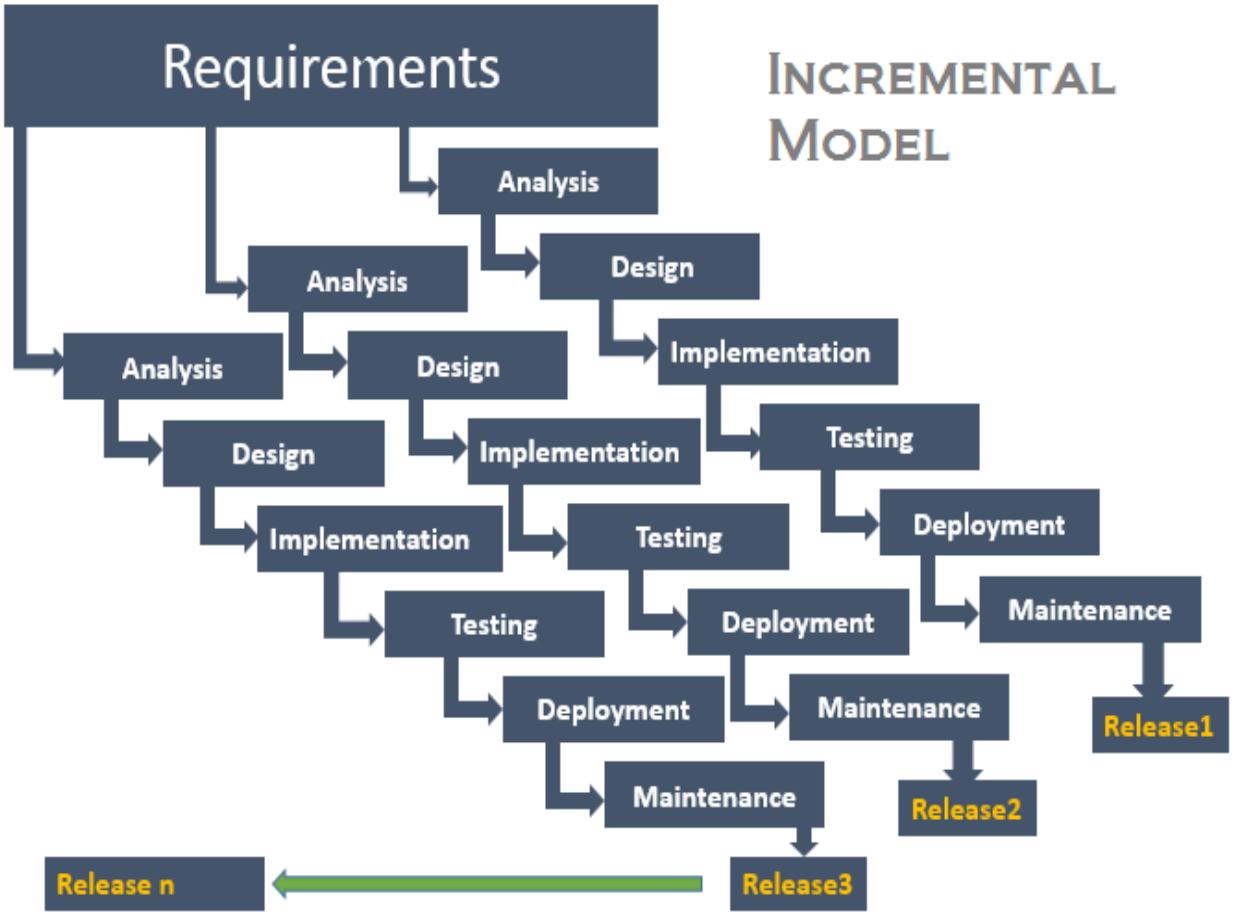
Software development paradigms: RAD

Today, the Rapid Application Development model is not widely used because:

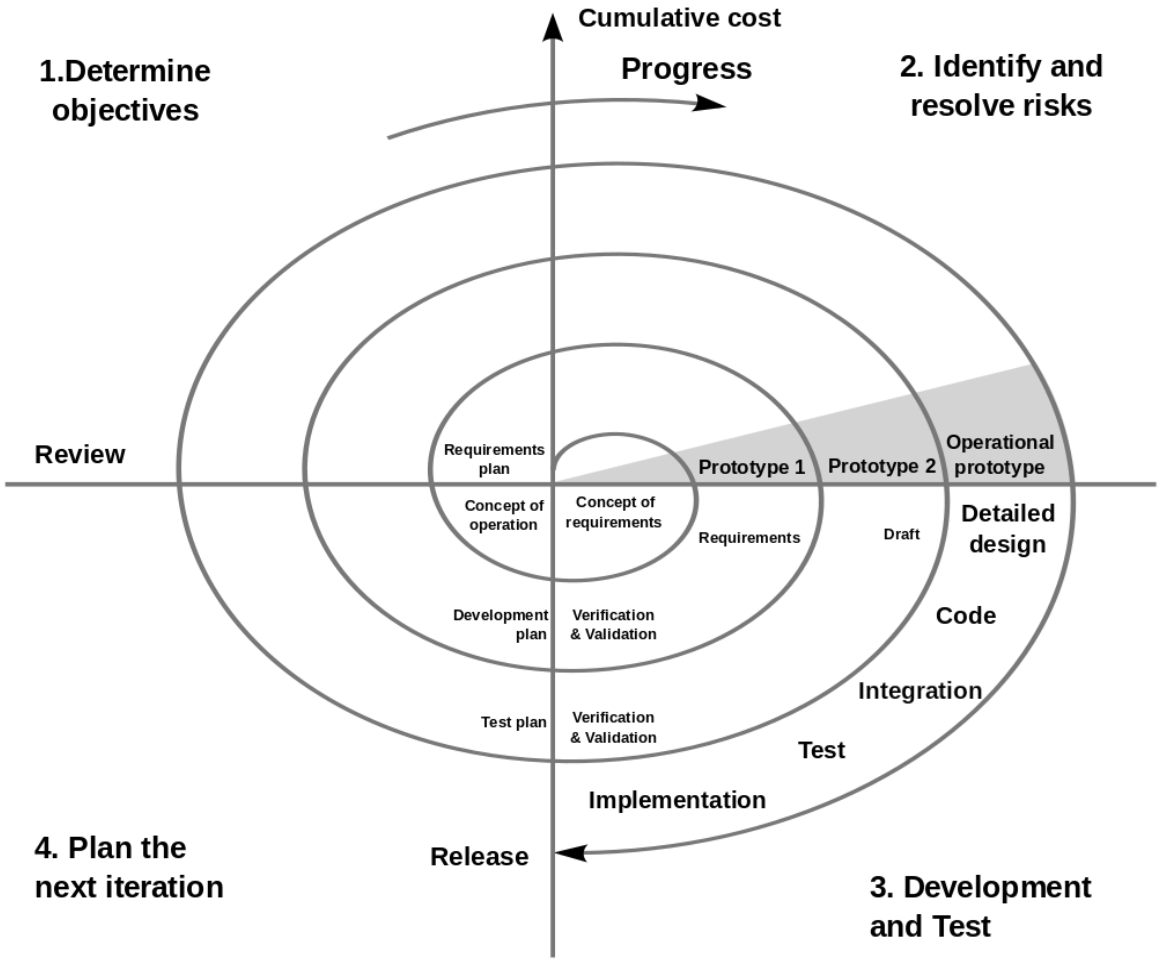
- It is in fact the same as the waterfall model with a short-term delivery.
- It was the seed of evolutionary models.

Experience has shown that it is not really feasible.

Software devel. paradigms: Evolutionary-Incremental



Software devel. paradigms: Evolutionary-Spiral



Software development paradigms: Evolutionary

ADVANTAGES

- The iterative model allows the development of progressively more complete versions, combining the advantages of waterfall and prototyping.
- In each iteration a new feature is added, but always keeping the overall vision in mind.

DISAVANTAGES

- The end of the phases is not defined.
- The list of requirements is not closed until the end of the project.
- Risk control: a new change to be implemented may not be compatible with previous work, and this could cause a redesign.

Software devel. paradigms: Evolutionary >> Agile

Manifest (principles) of the Agile Software Development:

- People and iterations over processes and tools.
- Intuitive and functional software over extensive documentation.
- Collaboration with the client over contract negotiation.
- Response to the change over following the planification.

Different methodologies following Agile principles:

SCRUM **KANBAN**
LEAN **XP**

Process methods: Main Phases

The three main phases of the software lifecycle:

- **Analysis**: Understand the problem and represent its basic elements (requirements), describing each one of them and creating the relationship links between these elements.
- **Design**: Produce a model or representation of the system architecture that can be used, at a later stage, to implement the system.
- **Implementation**: Generate the code that represents the system and validate it.

Process methods: Analysis

During analysis, the characteristics of the problem are described in terms of **requirements** and **use cases**. In this phase the ***what*** is described.

Special concepts:

- **Requirement**: Set of ideas that the client has about what the software to develop should be. They represent the **behavior of the system**.
- **Use Cases**: A use case is a list of actions or event steps typically defining the **interactions** between a role (an actor) and a system to achieve a goal.
- **Actor**: Entity that will use the software interactively or causing **inputs** to them.
- **Software Specification**: Official **technical document** listing and categorizing the Requirements and Use cases.

Process methods: Design

During the design, we use our experience and intellectual skills to generate a model of the system. In this phase we describe the ***how***.

Definition:

- The process of applying different techniques and principles in order to define a device, process or system with sufficient levels of detail to allow its **physical realization**.
- It is at the technical core of the Software Engineering process. And it applies regardless of the development paradigm used.
- **Problem domain → Solution domain**

Objective:

- Produce one model or representation of the **system architecture** that can be used to implement it.

Process methods: Design levels

The results of the design phase can be depicted at different levels.

Preliminary:

- Focus on requirements transformation and software architecture (classes, sequences).

Detailed:

- Data structures refinement and algorithmic representation.

Process methods: Main Design objectives

A **good design** requires achieving a sufficient degree of these objectives:

- **Verifiability**: To be able to evaluate the correctness of the design.
- **Completeness**: All components (data structures, modules, external interfaces, etc.) must be specified.
- **Consistency**: That there are no inherent inconsistencies.
- **Efficiency**: Proper use of system resources. The resources are not infinite.
- **Traceability**: All design elements must be able to "map" towards the analysis of requirements. → verification matrix
- **Simplicity / Comprehensible**: It should be noted that the design document, like the requirements specification, will be used as the basis for the subsequent stages of the design process.

Process methods: Design goals

What design does and what it does not include:

- The design process should not be unique, alternatives should be evaluated.
- Each element of the design model must be able to go back to the analysis to find out what requirements it satisfies.
- Design does not have to invent anything new. Reuse as much as possible.
- Minimize the distance between the problem domain and the solution domain.
- It must be uniform (a constant style).
- It must be structured to admit changes.
- …

Process methods: Design goals

What design does and what it does not include (II):

- It must be robust. Accept unusual circumstances.
- The design is not writing code and writing code is not designed.
- The quality of the design is evaluated while it is being made, not afterwards.
- It must be reviewed to avoid conceptual (semantic) errors: omissions, ambiguities, inconsistencies, etc.

Process methods: Design concepts

Specific concepts included when designing:

- **Abstraction**: Generic level vs. Concrete level.
- **Modularity**: Subdivision of the system into elements.
- **Refinement**: Formal verification of the specification implementation.
- **Software Architecture**: Global structure of the software and the way in which this structure provides conceptual integrity to a system.
- **Structural partition**: The program structure can be divided horizontally or vertically.
- **Data structure**: Logical relationship between individual data elements.
- **Software procedure**: Processing details of each module individually.
- **Principle of concealment (of information)**: The modules must be designed so that the information (data and processes) is inaccessible to other modules that do not need it.

Process methods: Design abstraction models

Level of abstraction has a major impact on the design phase.

- **Abstraction**: Allows components to be defined in abstract form at different levels, without worrying about their implementation details.
  - An abstraction describes the external behavior of the component, without having to pay attention to the internal details that produce the behavior.
  - The abstraction is an aid to modularization, since it allows to see the external behavior of the modules so that they can be interconnected.

- Three models:
  - **Functional abstraction**: From the point of view of the function it performs. A sequence of instructions that has a specific and limited function.
  - **Data abstraction**: A collection of data that describes a data object.
  - **Control abstraction**: Implies a program control mechanism without specifying the internal details.

Process Tools: Software Specification Document

The analysis is provided for in what is known as the **Software Specification Document** (also called the S.R.S. or *Software Requirements Specification*):

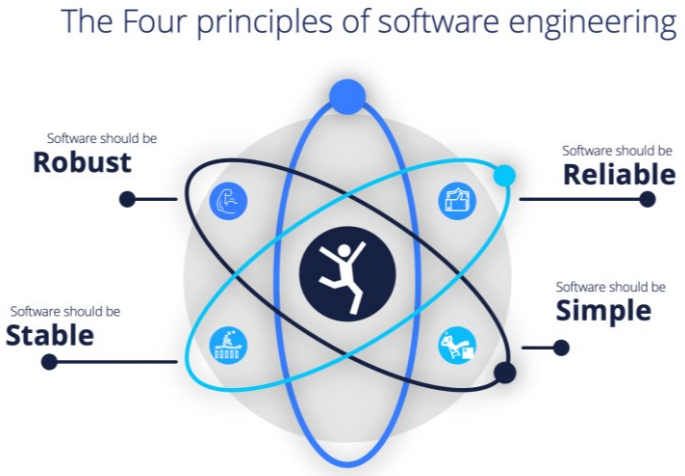
- This well organized and written document includes all the system requirements as well as the use cases.
- It acts as a contract to be validated by the client with respect to all the functionalities to be provided by the system.
- It also includes all the necessary constraints that are due to causes of the problem domain and not of the design.

Process Tools: Design Document

The design is provided for in what is known as the Design Document.

- This document outlines the overall vision of the design: the elements/components, their characteristics and relationships between them. Both at a preliminary level and at a detailed level.
- It defines how the overall structure and its behavior will be implemented → keeping in mind that everything must be derived from the requirements that appear the Software Specification Document!
- To perform this design task and part of the analysis we need to use some methodologies and tools: UML is the most widely used at present.

CONCLUSIONS



- Software engineering principles help to create quality software in a systematic way.
- They can be applied at different stages and for different goals.
- Each software engineering methodology has its own steps to follow the principles.

Recommended reading to find out more:
https://www.geeksforgeeks.org/software-paradigm-and-software-development-life-cycle-sdlc/