Abdelhafid Boussouf
University Center Mila

Faculty of Mathematics and
Computer Science

Department of
Computer Science

Software engineering

Chapter 1

Introduction to Software Engineering

Mrs. S.HEDJAZ

I. Introduction to the SE

01 History, definitions and objectives

02 Principles of Software Engineering

03 Expected quality of software

04 Software Lifecycle

05 Software Lifecycle Models

01 History, definitions and objectives

The term "Software Engineering" was first introduced at a conference in Germany in 1968 in response to the software crisis

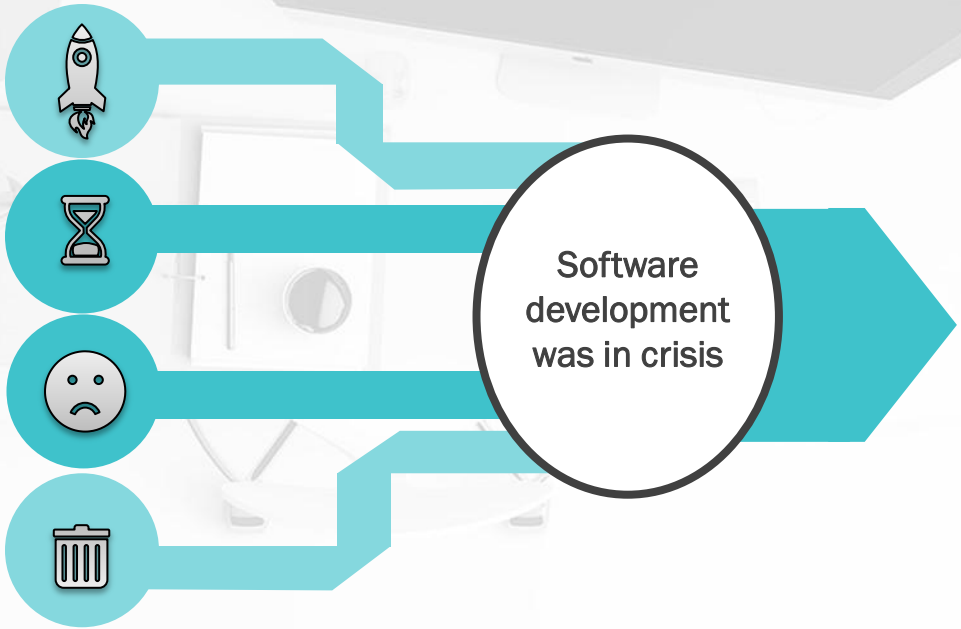
The software crisis refers to the period when the costs, deadlines, and quality of software projects were often unsatisfactory

The construction of the software was very expensive (Average overrun 70%)

Delivery deadlines were not met (Average overrun 50%)

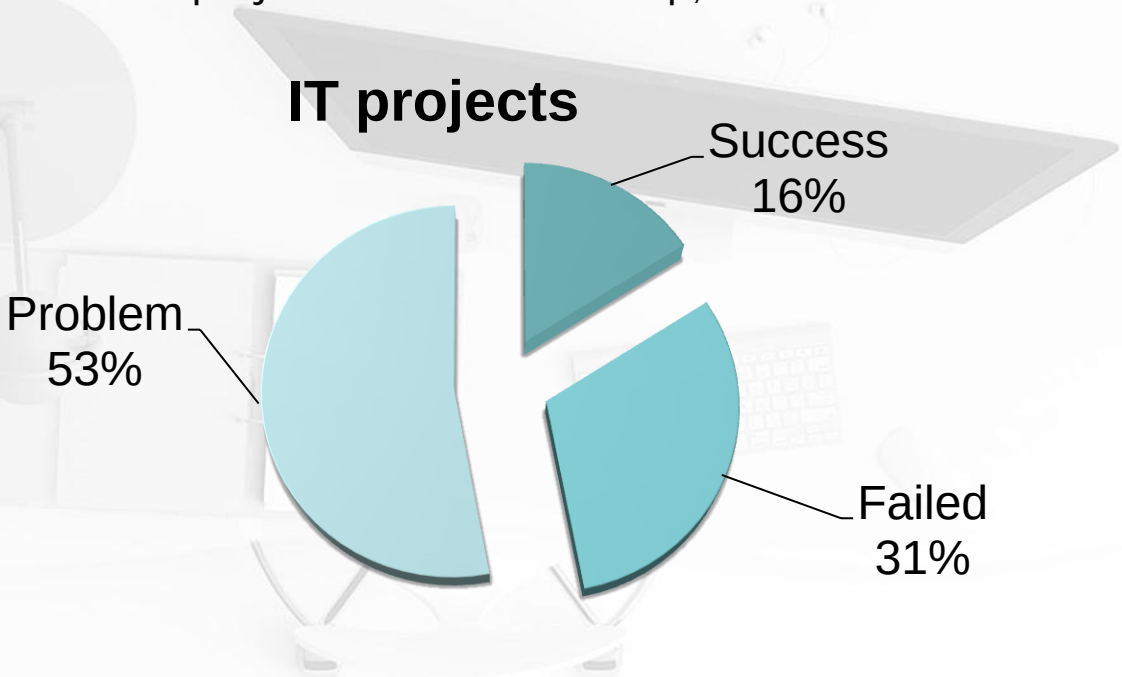
Software does not meet the needs of users

Difficult to use, maintain, or scale (40% - 70% of the software cost)



01 History, definitions and objectives

According to a study of 8380 projects "Standish Group, 1995":



01 History, definitions and objectives

Famous bugs:

1962: The Mariner 1 spacecraft is destroyed in flight shortly after its flight

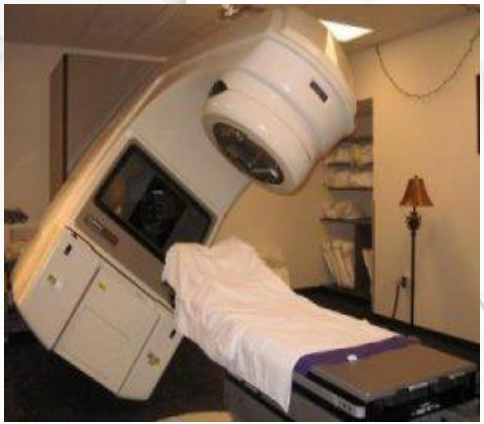
- Cost: $18.5 millions
- Mission: flyby of the planet Venus
- Cause: Error in manually transcribing a mathematical symbol in the specification

01 History, definitions and objectives

Famous bugs:

1985/1987: Therac-25 problem, overdose in radiotherapy "therapeutic irradiation device"

- At least 5 deaths from massive radiation doses
- Cause: Design flaws in hardware and software



01 History, definitions and objectives

Famous bugs:

1994: The Pentium split bug, an error was introduced due to a design flaw in the floating division algorithm

1996: Ariane 5, explosion of the first flight of Arian 5 due to an overflow error during the conversion of a 64-bit float number to a 16-bit integer.
Cost $500 million. (The costliest computer bug in history)

2000: The bug of, malfunctions when the dates are after December 31, 1999

2011: PlayStation Network, millions of personal and banking data hacked, financial losses of billions of dollars, due to network vulnerabilities

2014: Encryption tool, Open SSL, 500000 web servers affected by the flaw. Vulnerability that could read a portion of the memory of a remote server

01 History, definitions and objectives

Idea:

The idea of **software engineering** is to apply classical engineering methods to the software field.

Engineering : A set of functions ranging from **design** and **studies** to responsibility for the **construction** and **control of equipment** in a **technical or industrial installation.**

Definitions:

As defined by Pollice, 2005 and IEEE 1993, software engineering is the application of a systematic quantifiable disciplined approach to the development, operation, and maintenance of software, as well as the study of these approaches

Software engineering is a set of **methods, techniques**, and **tools** dedicated to the **design, development,** and **maintenance** of computer systems.

01 History, definitions and objectives

SE objective (CQFD criterion):

The SE is concerned with software manufacturing processes to ensure that the following four criteria are met:

- Cost (C): Costs remain within the original limits.
- Quality (Q): Quality is the original service contract.
- Feature (F): The system that is manufactured meets the needs of users.
- Deadline (D): The deadlines remain within the limits provided at the outset.

02 Principles of Software Engineering

Rigor:

The main sources of software failures are human-caused. At all times, you have to question the validity of your action.
Verification tools that accompany development can help reduce errors. This family of tools is called CASE (Computer Aided Software Engineering) such as Sparx Systems Enterprise Architect , Microsoft Visio, Rational Software Architect, Eclipse Modeling Framework (EMF).

Abstraction:

Extract general concepts to reason about, and then instantiate solutions on particular cases.

Decomposition into sub-problems:

Treat each aspect separately, each sub-problem simpler than the overall problem.

02 Principles of Software Engineering

Modularity:

Partition of the software into interacting modules, fulfilling a function and having an interface hiding the implementation from the other modules.

Incremental construction:

Step-by-step construction, gradual integration.

Genericity:

Propose solutions that are more general than the problem so that they can be reused and adapted to other cases. Reusable software is much more valuable than a dedicated component.

02 Principles of Software Engineering

Anticipation of developments:

Linked to genericity and modularity, plan for possible additions/modifications of functionalities.

Documentations:

Essential for project monitoring and communication within the project team.

03 Expected quality of software

Definition 1:

A set of entities necessary for the operation of an automatic information processing process.
Among these entities are: **Programs, User documentation, configuration information, … etc.**

Definition 2:

A set of **programs** that enables a **computer system to perform a particular task or function.**

Example: a game, website, mobile application, … etc.

03 Expected quality of software

Software quality factors have been given by B.MEYER in (design and object programming)

External

Validity: the ability of a software product to perform exactly the tasks defined by its specification.
Robustness: the ability of software to operate even under abnormal conditions.
Extensibility: Ease of adapting software to changes in specification.
Reusability: the ability of software to be reused in whole or in part for new applications.
Compatibility: the ability of a software to be combined with each other.

03 Expected quality of software

Other factors of software quality are less crucial:

Efficiency: good use of equipment resources.
Portability: the ease with which the product can be adapted to different hardware or software environments.
Verifiability: ease of preparation of acceptance and certification procedures (testing, etc.).
Integrity: the ability of software to protect its various components from unauthorized access and modification.
Ease of use (Usability): the ease with which users of a software can learn how to use it, how to operate it, how to prepare the data, but also how to interpret the results and effects in the event of an error.

03 Expected quality of software

Internal criteria make it possible to achieve these external quality factors

Internal

Modularity: this is the breakdown of the software into easily understandable and relatively independent components.

Completeness: this is the degree to which the specifications are implemented. Software is complete if all of its external specifications are operational.

Consistency: This is the ability to go back in the development cycle. In particular, to report an error detected during maintenance at the level of implementation, design, or analysis.

Generality: Potential range of application of software components

Self-documentation and readability: possibility of extracting documentation from the software components.

04 Software Life cycle

The software life cycle models the sequence of different activities in the technical process of software development.

An activity includes: tasks, constraints, resources, a way of being carried out.

The main activities are:

- Requirements capture
- Global Specification
- Architectural and detailed design
- Programming
- Configuration and integration management
- Validation and verification
- Delivery and maintenance

04 Software Lifecycle

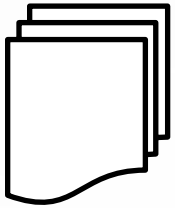
1.1. Requirements analysis

Goal: To avoid developing unsuitable software. The field of application as well as the curre nt and future state of the system environment will be studied in order to determine: Boundaries, Role, Available and Required Resources, Constraints of use and performance ... etc.

- Experts in the field of application
- Future users of the system

Requirements analysis

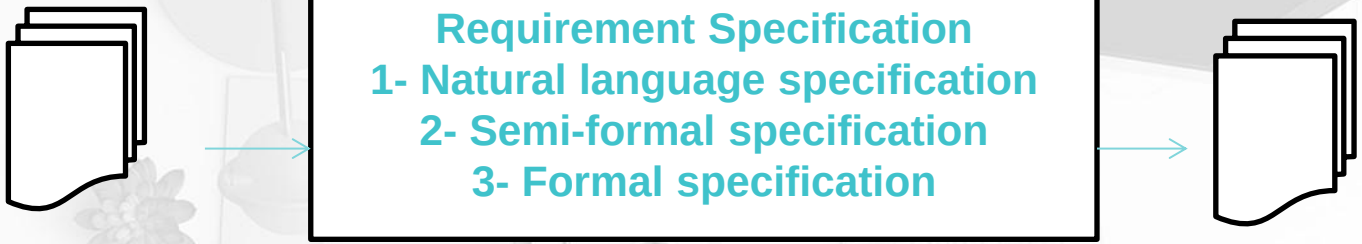
- Maintenance
- Questionnaire
- Observation of existence
- Similar situation study



- Specifications document
- Preliminary user Manual

1.2. Requirements Specification:

Aim: To establish an initial description of the future system. to produce a description of wha t the system should do but without specifying how it does it



- Specifications document
- Preliminary User Manual

- Model E/A
- Data flow diagram
- Transition state diagram
- Petri nets and grafcet
- Diagram of the O.O methods ...

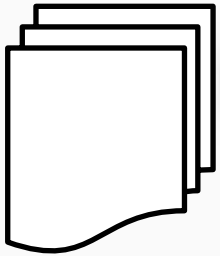
- Functional specifications
- Technical and IT feasibility

1.3. Design:

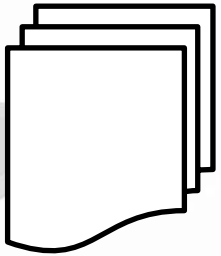
Goal: To enrich the description of the software with implementation details in order to arrive at a description very close to a program (describe the how).

- ✓ Architectural design (or overall design) aims to break down software into simpler co mponents, defined by their interfaces and functions (the services they provide).
- ✓ The detailed design provides for each component a description of how the functions or services are performed: algorithms, data representation.

1.3. Design:



Conception
1- Architectural design
2- Detailed Design



- Functional specifications
- Technical and IT feasibility

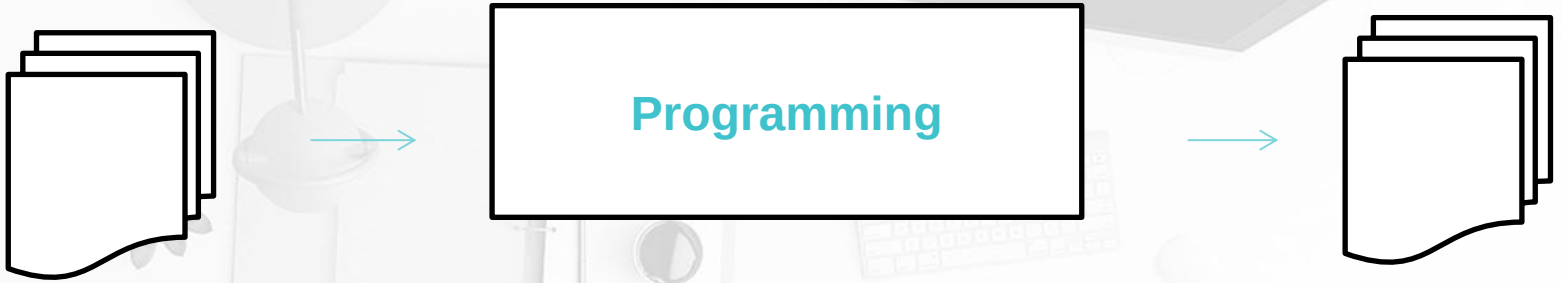
- Functional design and object-oriented design
- For each component, write the algorithms

- Design File:
- (Architecture and Algorithm)

1.4. Programming:

Implementation of the designed solution

✓ Choice of development environment, programming language(s), development standar ds, etc.



- Design File:
- (Architecture and Algorithm)

- Source code
- Final User Manual
- Documentation

1.5. Configuration Management and Integration:

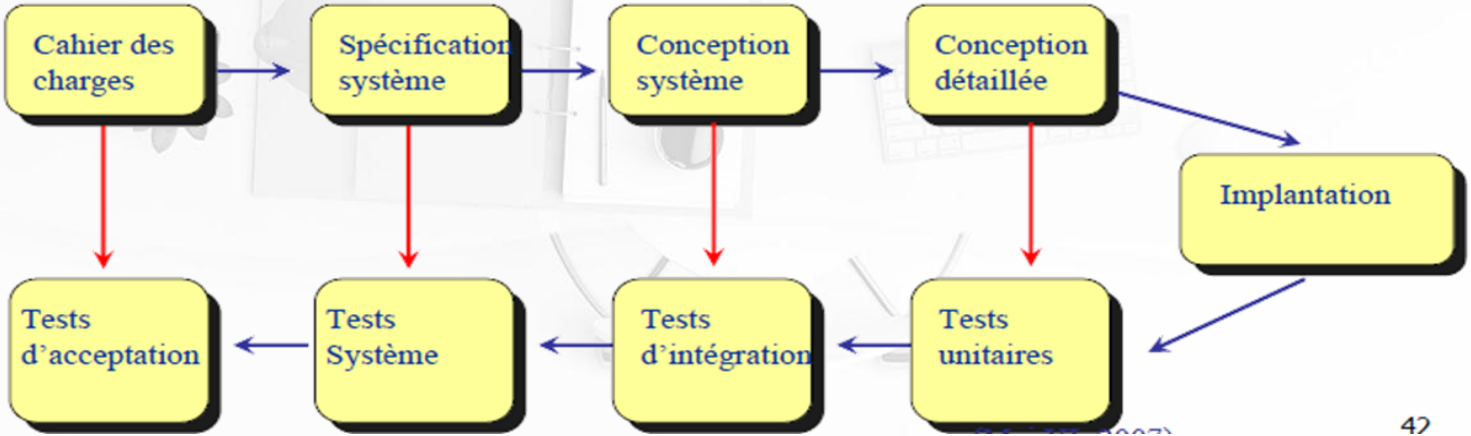
Configuration management aims to control the evolution and updating of component s throughout the development process.

The purpose of integration is to create one or more executable systems from the com ponents (Combine components).

1.6. Validation and verification:

The purpose of validation is to answer the delicate question: has the right system been described, the one that meets the users' expectations?

The verification answers the question: Is the development correct in relation to the over all specification? This consists of ensuring that the successive descriptions and the soft ware itself satisfy the specification.

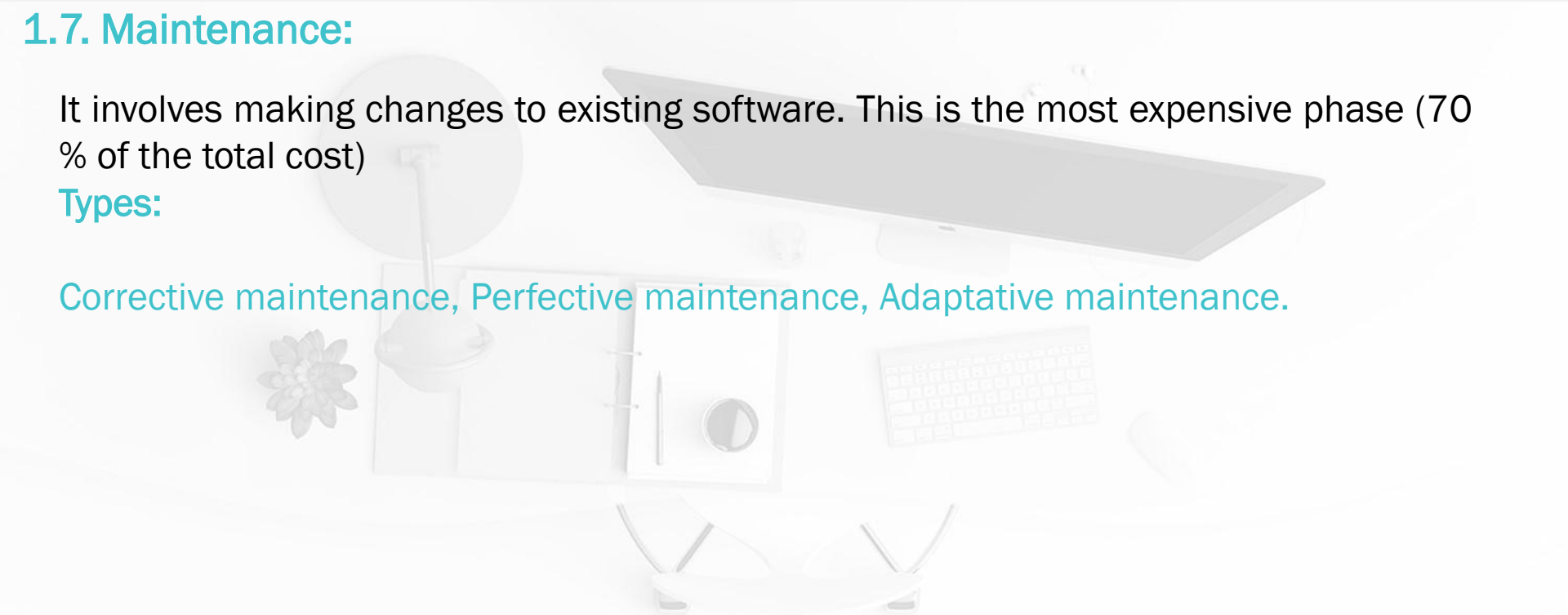


1.7. Maintenance:

It involves making changes to existing software. This is the most expensive phase (70 % of the total cost)

Types:

Corrective maintenance, Perfective maintenance, Adaptative maintenance.



04 Software Lifecycle

Documents:

End User Manual	Implementation
Architectural Design Brief	Design
Source Code	Implementation
Specifications Needs	analysis
Preliminary User Manual	Needs Analysis
Detailed Design File	Design
Test Report	Tests
Implementation	Documentation
Functional Specification	Specification

05 Software Lifecycle Models

Linear models

- waterfall model
- V-shaped model

...

Nonlinear models

- Prototyping
- Spiral model
- Incremental Modeling
- Unified Models
- ...

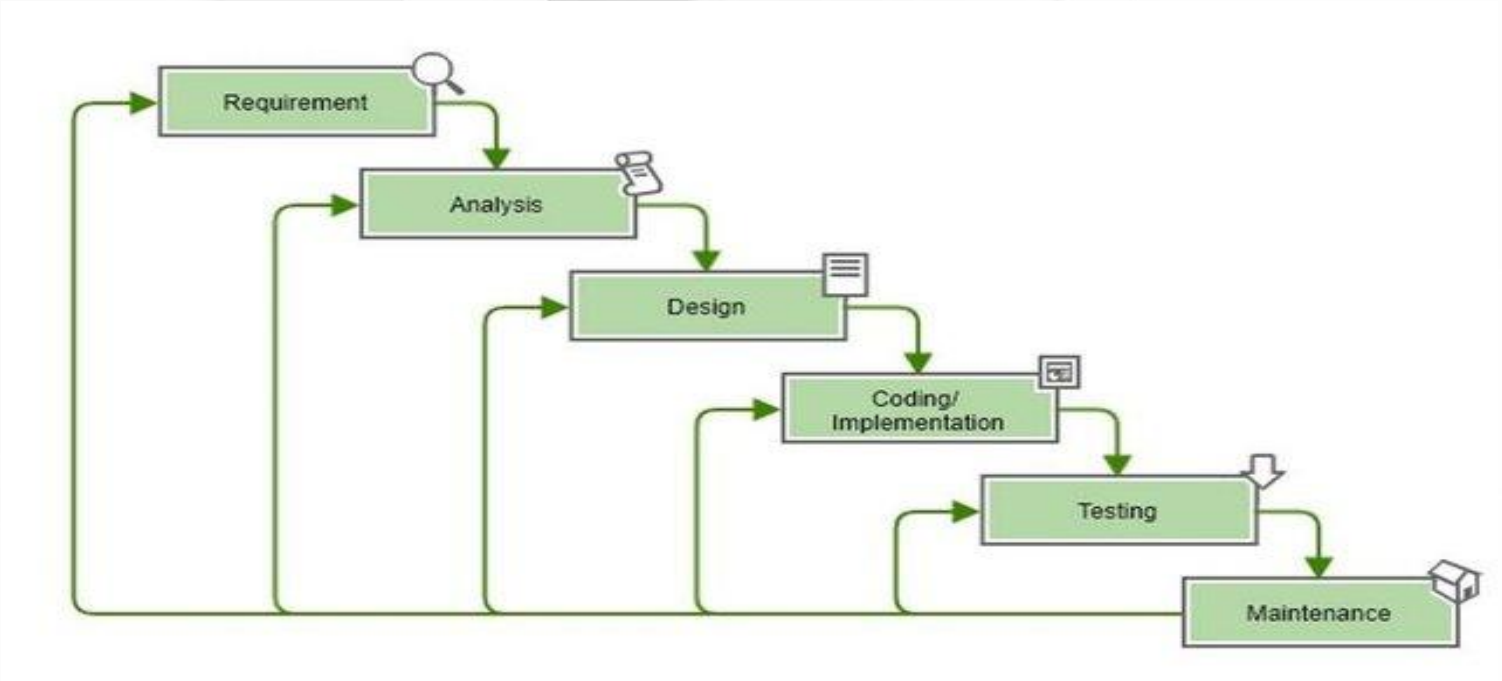
05 Software Lifecycle Models

1. waterfall Model

- Dates from the 70s but remains relevant
- Linear model with sequential phases
- Checking each phase before moving on to the next
- We cannot move on to the next stage as the previous one is not over
- Production of documents at the end of each phase
- Changing a stage has a big impact on future stages
- ...

05 Software Lifecycle Models

1. Waterfall Model



Software Lifecycle Models

Benefits

- The schedule is established in advance and the project manager knows exa ctly what will be delivered to him and when he will be able to take delivery

Disadvantages

- Model that is too sequential, i.e. lasts too long
- Validation too late (costly questioning of previous phases)
- Sensitivity to the arrival of new requirements (redo all the steps)
- **Well suited when needs are clearly identified and stable**

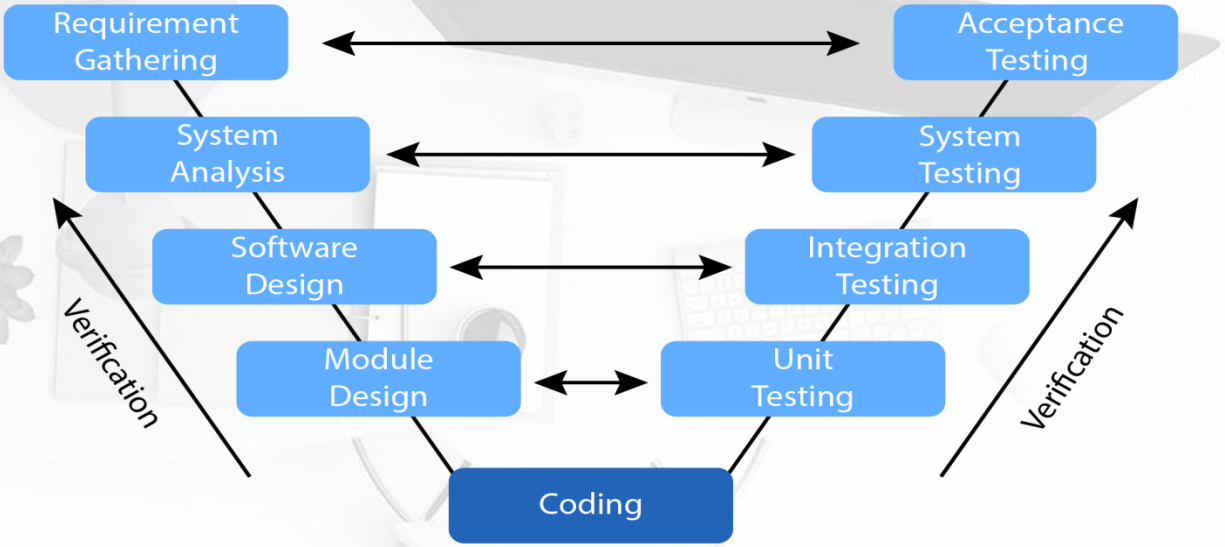
2. V-shaped model

To date, the V-cycle remains the most widely used life cycle. It is a Test-Oriented Li fe Cycle:

- Each creative activity "specification, design, and coding" corresponds to a verif ication activity "validation, integration, unit tests"
- Each phase prepares the corresponding phase of verification "verification is ta ken into account at the very moment of creation"

05 Software Lifecycle Models

2. V-shaped model



05 Software Lifecycle Models

Benefits

- The preparation of the last phases (validation and verification) by the first ones (soft ware construction), makes it possible to avoid stating a property that is impossible to verify after completion
- Each deliverable must be testable

Disadvantages

- The software is used very late, you have to wait a long time to know if you have built t he right software
- Does not contain risk analysis activities
- **Ideal when the needs are well known, "When the analysis and design are clear"**

05 Software Lifecycle Models

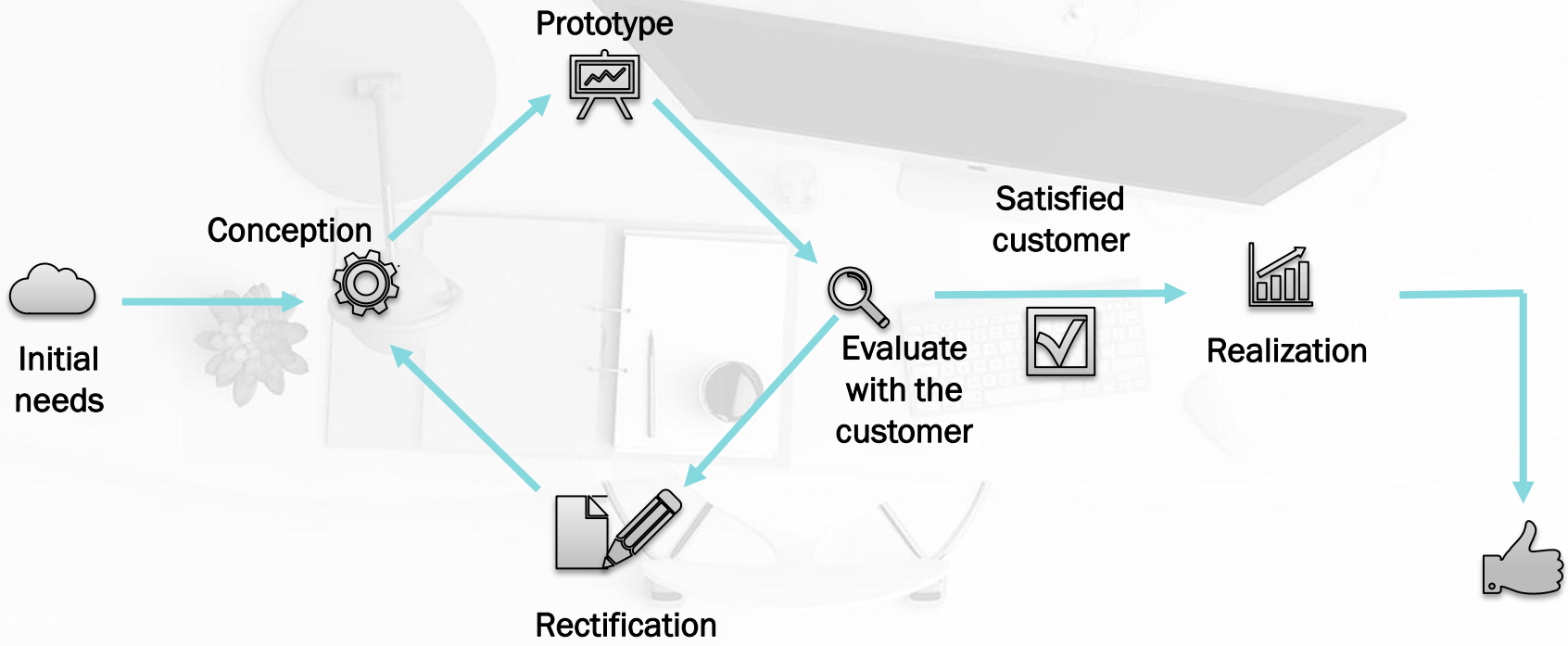
3.a. Disposable Prototyping

- Skeleton of the software that is created only for a particular purpose and in a particu lar phase of development, if this prototype is kept; then it's called evolutionary prototyping

3.b. Scalable Prototyping

- The project is carried out over several iterations
- Developers build a prototype according to the customer's expectations
- The prototype is evaluated by the customer
- The customer gives feedback
- Developers adapt the prototype based on feedback and new customer requirement
- When the prototype satisfies the customer, the code is standardized according to sta ndards and best practices

3.b. Scalable Prototyping



05 Software Lifecycle Models

Benefits

- Active customer involvement, developer learns directly from the customer
- Adapt quickly to changing needs

Disadvantages

- Difficult to establish a schedule
- The process may never stop
- **Ideal when needs are unstable and/or require clarification, recommended for very s mall projects involving very few people**

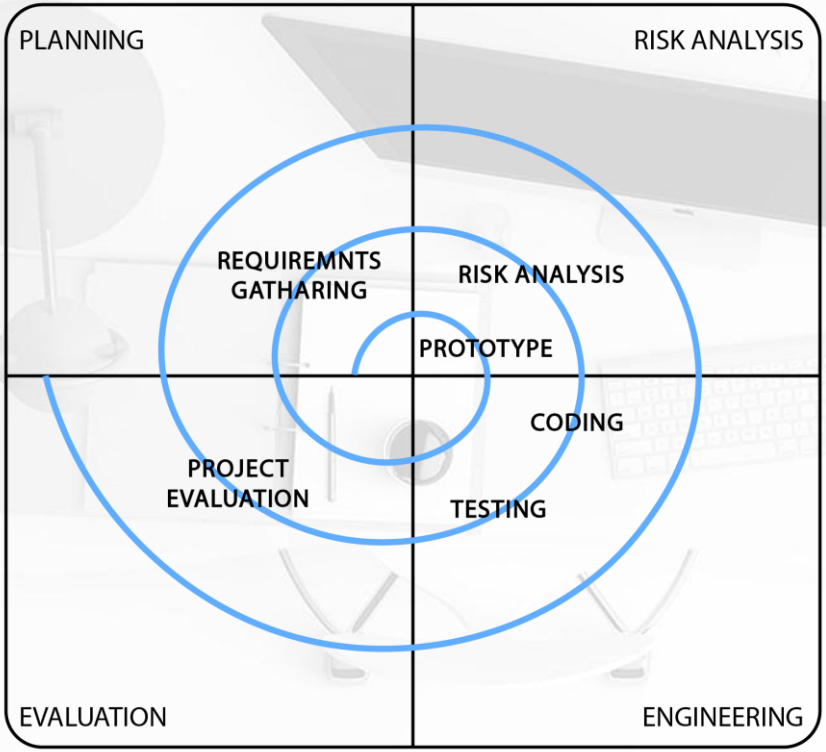
4. Spiral model

Each cycle of the spiral unfolds in four phases:

- ❑ Determination of cycle objectives, alternatives to be achieved and constraints based on the results of previous cycles, or preliminary needs analysis
- ❑ Risk analysis, evaluation of alternatives and possibly mock-up
- ❑ Development and verification of the chosen solution, a classic model can be used he re "Cascade, V, ... »
- ❑ Review of results and verification of the next cycle

05 Software Lifecycle Models

4. Spiral model



Software Lifecycle Models

Benefits

- Helps establish the most appropriate development model for risk
- All other development models are a variant of the spiral

Disadvantages

- Needs skills in thoroughly assessing and mitigating project uncertainties and risks
- Assessing the risks involved in the project can take up to the cost and it can be higher than the cost of building the system.

Bibliographies

- **Introduction to Software Engineering**, Ronald J. Leach, CRC Press, Taylor & Francis Group, 2016
  https://1library.net/document/rz3r50mz-introduction-to-software-engineering-pdf.html

- **Génie logiciel, principes, méthodes et techniques,** Presses Polytechniques et Universitaires Romande, 1996,
  https://www.leslibraires.fr/livre/596107-genie-logiciel-principes-methodes-et-techniques-alfred-strohmeier-didier-buchs--presses-polytechniques-et-universitaires-romandes