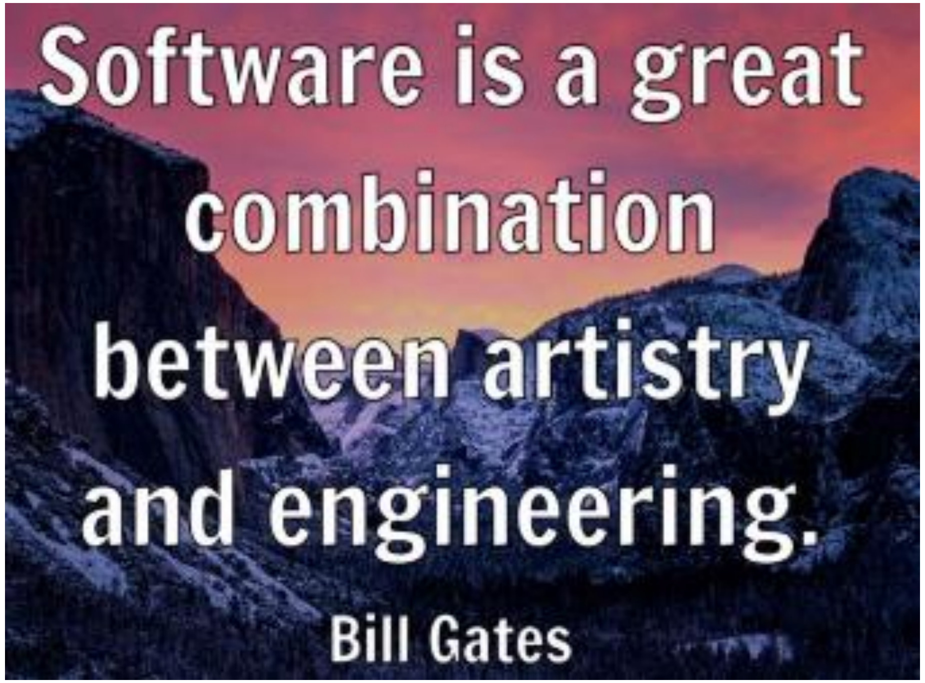
Software Engineering Course

Chapter 1

Introduction to Software Engineering

2020-2023

Dr. S. Kouah



Software is a great combination between artistry and engineering.

Bill Gates

Chapter 1: **Introduction to Software Engineering**

1. Definitions & objectives
2. Software Engineering Principles
3. Software Quality
4. Software life cycle
5. Software life cycle models

Software?

- Today, Information Technology is at the heart of all big companies. The information system of a company is composed of hardware and software.
- Information system is:
  - **80% Software**
  - **20% Hardware**

Software?

- Software can be defined as a set of: **programs**, **processes**, **rules**, and **documents** related to the functioning of a set of information processing.
- **Software** = set of **programs** cooperating together via **interfaces** and **procedures** dedicated to achieving a **set of requirements** for its **users**.
- **Software**: **<u>Not only computer code</u>** (code + data structures + documentation + test suites + configuration information, etc.)

Definitions & Objectives

- **Software development in the late of 1960s:**

- Hardware techniques have reached a relative degree of maturity:

  - Hardware is relatively reliable
  - The market is standardized

- The place of software in complex systems is constantly increasing, nevertheless software producers rarely respect the costs or development timelines, when they are not the requirements of their customers!

- Thus, Computer issues are mainly **software problems**.

↓

Software Crisis!!!

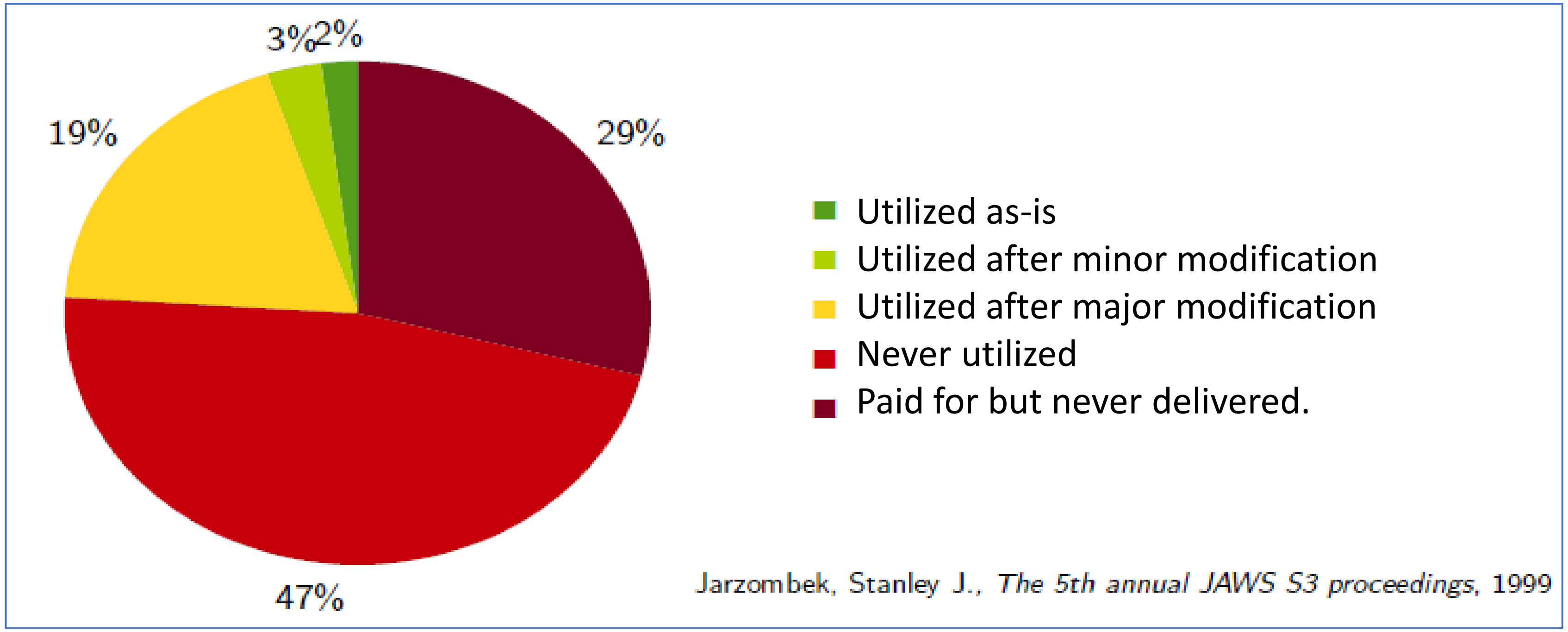
Definitions & Objectives

Software Crisis?

- ✓ Delivery **times** not respected
- ✓ Expected **cost** not respected
- ✓ Does not meet the requirements of the user or the customer
- ✓ Hard to **use**, **maintain**, and **scale**

According to a study by the U.S. Department of Defense (DoD), 70% of initiated projects are abandoned before they are completed.

DoD study in 1995 on software produced as part of nine major military projects.



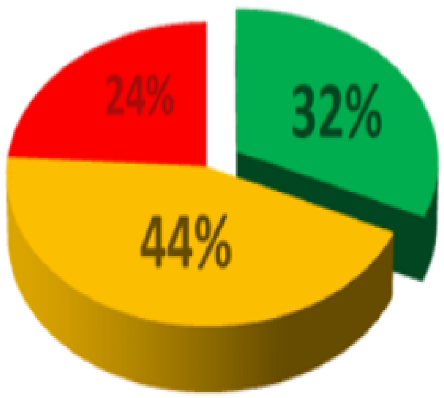
Jarzombek, Stanley J., *The 5th annual JAWS S3 proceedings*, 1999

Definitions & Objectives

Software Crisis?

Source: The Standish Group

The Chaos Report 2009

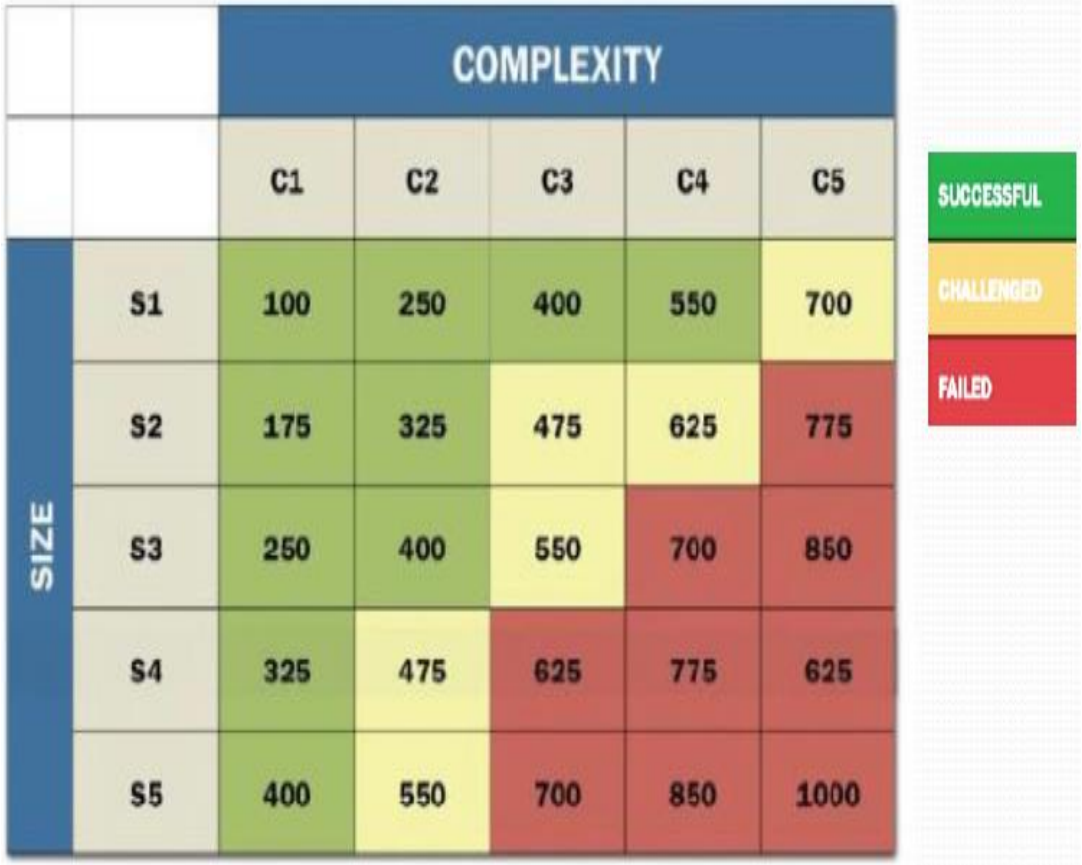


Successful: on-time, on-budget, and with all features and functions as defined in the initial scope;

Challenged: late, over budget, and/or with less and functions than defined in the initial scope;

Failed: cancelled prior to completion, or delivered but never used.

The greater the **complexity** & **size**, the greater the **risk of failure.**



		COMPLEXITY				
		C1	C2	C3	C4	C5
SIZE	S1	🟩 100	🟩 250	🟩 400	🟩 550	🟨 700
	S2	🟩 175	🟩 325	🟨 475	🟨 625	🟥 775
	S3	🟩 250	🟩 400	🟨 550	🟥 700	🟥 850
	S4	🟩 325	🟨 475	🟥 625	🟥 775	🟥 625
	S5	🟩 400	🟨 550	🟥 700	🟥 850	🟥 1000

🟩 SUCCESSFUL

🟨 CHALLENGED

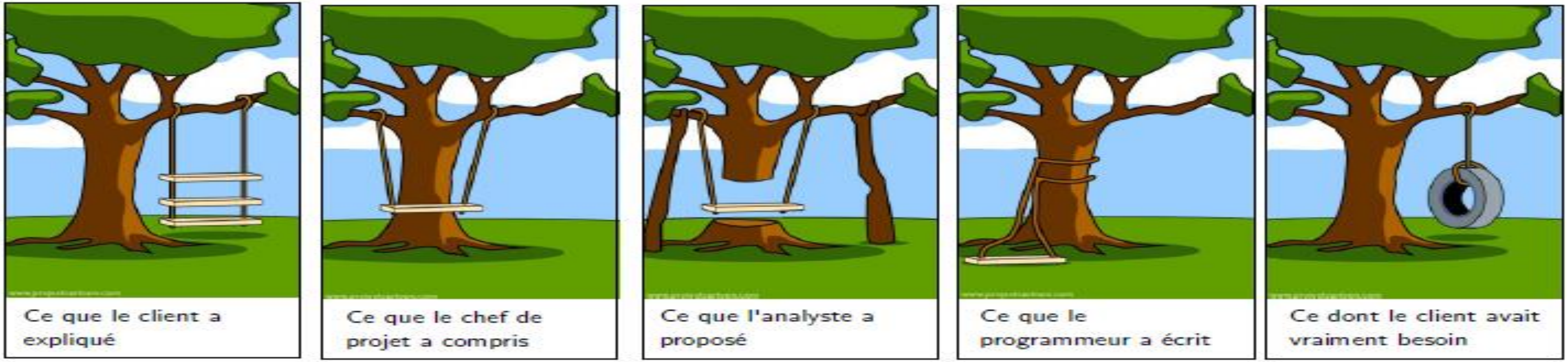
🟥 FAILED

The **software crisis** of the 1960s gave rise to **Software Engineering**, which was formally recognized in 1968 (First conference on the topic).

Definitions and objectives

Some Causes of the Software Crisis:

- Size and complexity of the software.
- Diversification of application areas.
- Lack of methodological framework, development methods and tools.
- Poor requirements understanding .
- Lack of metrics for software process and product quality.
- Etc.


Ce que le client a expliqué
Ce que le chef de projet a compris
Ce que l'analyste a proposé
Ce que le programmeur a écrit
Ce dont le client avait vraiment besoin

There was a necessity to move from an ad hoc approach to software engineering.

What is a Software Engineering ?

- Engineering : Engineering science like aeronautical, maritime, civil engineering, etc.
- Software : complex manufactured object with its own production techniques.

What is a Software Engineering ?

- Software Engineering → Engineering (the art of specifying, realizing, developing) + Software
- Set of methods, techniques and tools dedicated to the design, development and maintenance of computer systems.
- Is a field of engineering sciences whose purpose is the design, manufacture, and maintenance of secure and quality complex computer systems.

Definitions and objectives

What is a Software Engineering ?

The art of specifying, designing, implementing, and evolving, with reasonable means and within reasonable timeframes, quality programs, documentation and procedures in order to utilize a computer to solve various problems. (Gaudel, 1996)

Objective of Software Engineering (SE)?

SE is concerned with software manufacturing processes by ensuring that the following 4 criteria are fulfilled:

- ✓ **Cost** initially planned (Cost)
- ✓ Expected **quality** (Quality)
- ✓ Responds to user **requirements** (Functionality)
- ✓ Initially planned **deadlines** (Delay).

CQFD Rule (Cost + Quality + Functionality + Delay)

Definitions and objectives

Software Engineering ?

- In other words:

✓ Have a well-defined **methodology** that takes into account the **software life cycle**

✓ Have a **set of elements** that **document** each stage of the **life cycle** and show the traces from one stage to the next.

✓ A set of **milestones** that can be reviewed at regular intervals in the **software life cycle**.

Principle of Software Engineering

The Software Engineering is based on several fundamental principles (of common sense). As examples, we cite the following principles:

- ✓Rigor
- ✓Generalization
- ✓Structuring
- ✓Abstraction
- ✓Modularity
- ✓Documentation
- ✓Verification
- ✓Anticipation of change
- ✓Incremental construction (refinement)

Principle of Software Engineering

- ✓Rigor: The software **should provide a precise solution** -> Serious, careful, non-approximate development.

- ✓Generalization: Grouping a **set of similar functionalities** into a **parameterizable functionality** (generic, inheritance) -> Solution can be reused.

- ✓Structuring: The **way of decomposing** software (using a **bottom-up** or **top-down** approach).

- ✓Abstraction: Mechanism that **allows presenting a context** by **expressing relevant elements** and **omitting those that are not** (considering important aspects of a system).

- ✓Separation of concerns: Tackles with the complexity of large systems, by separating different issues or concerns to concentrate on one at a time.

- ✓Modularity: A principle that involves **decomposing** a system into **simpler subsystems** in order to manage complexity.

Principle of Software Engineering

- ✓Documentation: Management of documents including their identification, acquisition, production, storage, and distribution.
- ✓Verification: Determination of the respect of the specifications established on the basis of the identified requirements in the previous development phase.
- ✓Change anticipation: Designing a system with sufficient richness to allow incremental modification.
- ✓Incremental construction (refinement): Consists of building a solution step by step by getting closer and closer to the goal. (Ex.: perform the essential functions of a system then add the secondary aspects).
- ✓...etc.

Software Quality

In SE, numerous works have led to the definition of **software quality** in terms of **factors** that depend on various features such as application domain and the utilized tools . Among these factors we can mention:

✓**Validity**: Exactly performing its functions, as defined by the specifications and specification.

✓**Reliability** or **robustness**: operating under abnormal conditions.

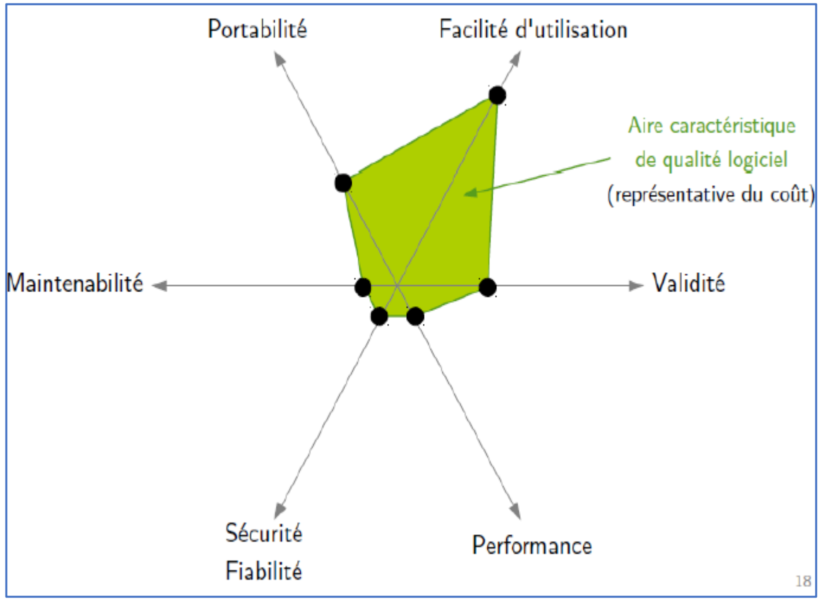
✓ **Ease of Use**: Easy to learn, utilize, prepare data, interpret errors, as well as catching up on users' errors.

Software Quality

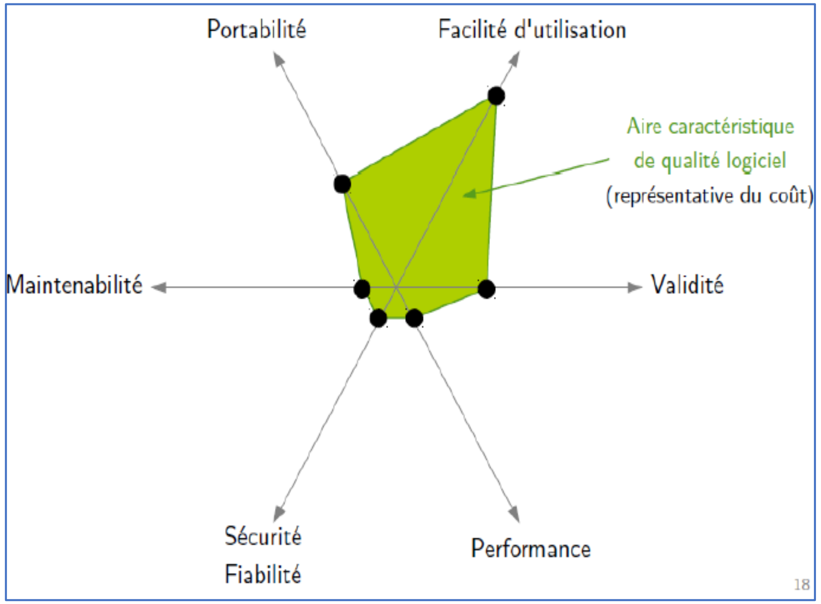
- ✓**Compatibility**: The facility of which the software can be integrated with other software.
- ✓**Efficiency**: Optimal utilization of resources (material).
- ✓**Portability**: The facility of which the software can be transferred across diverse software and hardware environments.
- ✓**Extensibility** : The facility of which the software prepares for maintenance, i.e. modification or extension of the required functions.
- ✓**Reusability**: Can be reused in whole or in part in new applications.
- ✓**Verifiability**: facility to prepare test procedures.
- ✓**Integrity**: Protecting its code and data against unauthorized access.

Software Quality

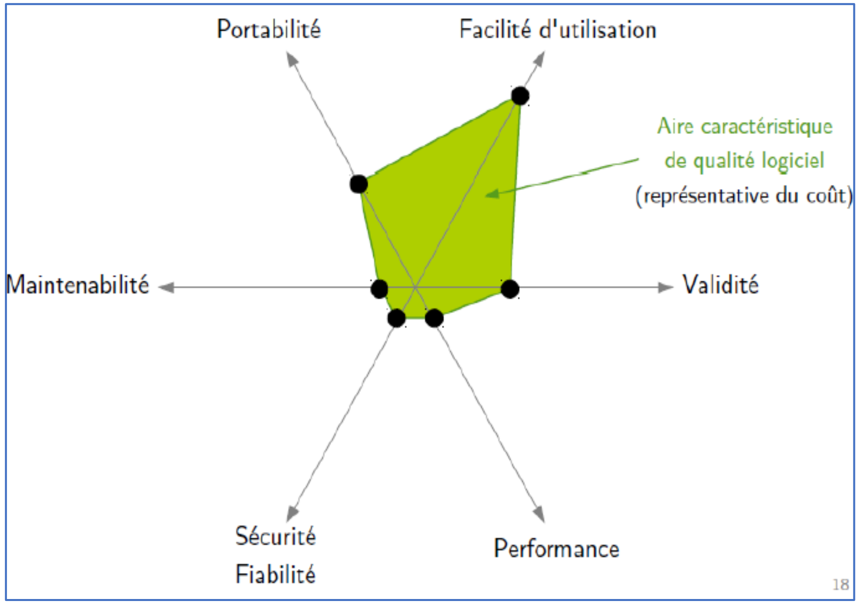
Remote control



Client Mail



Video Game



The question is Which quality?

- Some qualities at the expense of others.
- The Choice depends on several criteria, such as the nature of the application and the user requirements.

Software Life Cycle

- The **development process** impacts the **software quality**. Therefore, to obtain quality software, it is necessary to master its development process.

- A **software development process** is a structured set of activities leading to the production of software.

- The **life of software** is made up of different stages, **from** its **birth** **to** its **death** or **withdrawal** from service.

- So a **Software Life Cycle** =
  - **Birth** of the system (arrival of a customer request).
  - **Development** (Development Cycle)
  - **Use** and **Maintenance**.
  - **Death** (Disassembly: software no longer desired or no longer meets evolving needs)

Software Life Cycle

- The **software life cycle** is the time period from the beginning to the end of the software process.
- It **Starts** with the **proposal or decision to develop software** and **ends** with **its decommissioning**.

Software Life Cycle

There are many life cycle models, the most common include the following activities:

- ✓ Requirement Analysis
- ✓ Global Specification
- ✓ Design
- ✓ Programming (Coding)
- ✓ Integration
- ✓ Facility
- ✓ Maintenance
- ✓ Verification and Validation
- ✓ Test

Each activity use and production documents.

Software Life Cycle

Requirements Analysis: Understand the customer's requirements (Determine the functionalities of the system and its utilization constraints)

- **Data (Input)**: Provided by application domain experts and future users.

- Result (Output) :<u>Requirements Document (Specifications)</u>

- Specifications: set of documents describing the relevant aspects of the future system environment, its role, its future use, in a language understandable by users and developers.

Software Life Cycle

Global Specification: Establish a clear description of what the system must do (functional and non-functional requirements, etc.). **Describe the WHAT**.

Data (Input): Specifications + technical considerations and system feasibility.

Result (Output): Technical requirements specification document

Technical requirements specification document: covers all situations, more understandable than the specifications.

Note: Generally, it is sometimes difficult or even impossible to separate the activities of requirements analysis and specification. They appear grouped together.

Software life cycle

Design: Develop a concrete solution carrying out the specification (Description very close to a program), generally broken down into two successive phases:

a) General design, also called: overall design, preliminary design or architectural design:

b) Detailed design

Result: detailed design document.

Software life cycle

General design or architectural design: Architectural description in terms of components (components, their functionalities, interfaces, interactions between components, choice of software architecture). The result of this process is a general design document.

Data (Input): Technical Specification Document

Result (Output): General design document

Software life cycle

Detailed Design: Refines the **architectural** design. It starts by breaking down the components discovered during the general design into more basic components. This decomposition should be continued to the level where the components are easy to implement and test (elementary software components).

- Highly dependent on programming language. Each software component must then be described in detail: its interface, the algorithms used, errors handling , its performance, etc.

Software Life Cycle

Programming: During this phase, the detailed design is translated into a programming language (Choice of Programming Languages, Development Environment, Standards, etc**.**)

Integration: Assembling all or parts of the components to create an executable system. This activity uses configuration management to assemble consistent versions of each component.

Installation: After integrating the software, we can install it in its operating environment, or in an environment that simulates this operating environment, and test it to ensure that it behaves as required in the developed specification .

Software Life Cycle

Maintenance: After the installation follows the phase of **operating and maintenance**. The software is now employed in its operational environment, its behavior is monitored and, if necessary, it is modified. This last activity is called software maintenance.

It may be necessary to modify the software to correct defects, improve its performance or other characteristics, adapt the software to a new environment or meet new or modified requirements. We can therefore distinguish between corrective maintenance, perfective maintenance and adaptive maintenance.

Software Life Cycle

Validation and Verification:

Validation: Has the right system been described? (Building the right Product?)? Good means here that meets the expectations of users and the constraints of the environment.

Verification: Is the development correct in accordance with its specification? (Building the product right?).

Test: (several test activities)

- **Unit testing**: Testing isolated components.
- **Integration testing**: Testing a set of assembled components.
- **System Test**: test the system on its future operating site.
- **Non regression test**: Test the system following a modification.

Software Life Cycle

There are different software process models that organize these activities differently, such as:

✓**Waterfall model**

✓**V-pattern**

✓**Spiral**

✓**By prototyping**

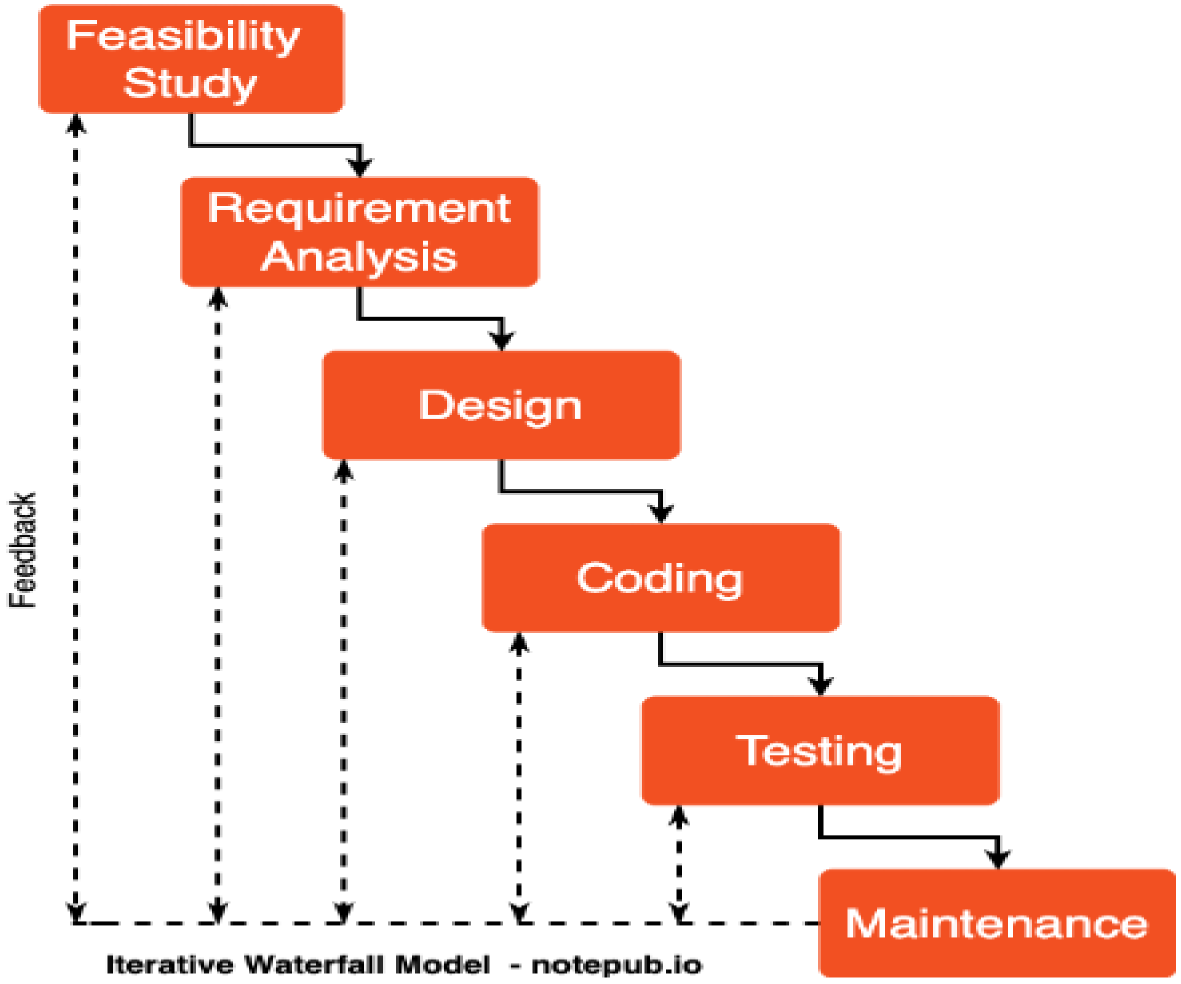
✓**By increment**

✓**Y-shaped**

✓**In W**

✓**…Etc**

"Cascade" Life Cycle (Waterfall Model)



SDLC - Prototyping Model - Notepub

"Cascade" Life Cycle (Waterfall Model)

The “**cascade**” life cycle dates from 1970, it is the work of Royce.

- **Linear life cycle** without any evaluation between the start of the project and the validation.

- The project is **divided into successive phases** over time and each phase corresponds to a specific main activity producing a certain number of deliverables.

- We ca only move on to the next step if the results of the previous step are considered satisfactory.

- The activity of a step is carried out with the results provided by the previous step; thus, each step serves as a check on the work performed in the previous step.

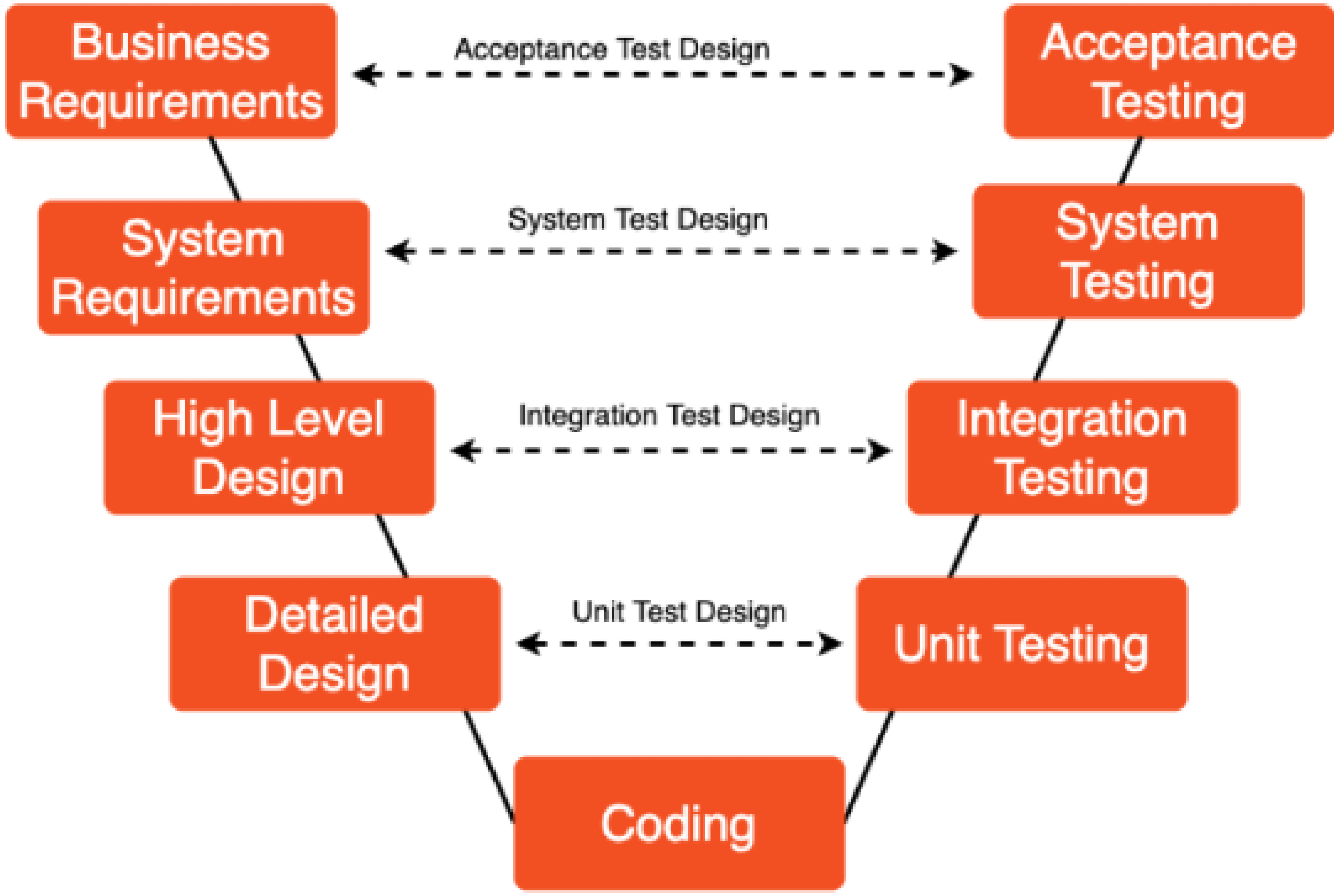
- Each phase can only call into question the previous phase which, in practice, is insufficient.

"Cascade" Life Cycle (Waterfall Model)

- Specification errors are generally detected during testing, or even when the software is delivered to the user. Correcting them then requires going through all the phases of the process.

- This model is best suited for **small projects or those with well-known and fixed specifications.**

"V" Life Cycle



SDLC - V Model - notepub.io

V Model

[SDLC - V Model - Notepub]

"V" Life Cycle

– Derived from the waterfall model.

– the beginning of the development process determines its final stages.

– Suitable for **medium-sized and complex projects**.

– The first branch corresponds to a classical cascade model. Any component description is accompanied by test definitions.

- With the test cases provided in the first branch, the steps of the second branch can be well organized and planned.

"V" Life Cycle

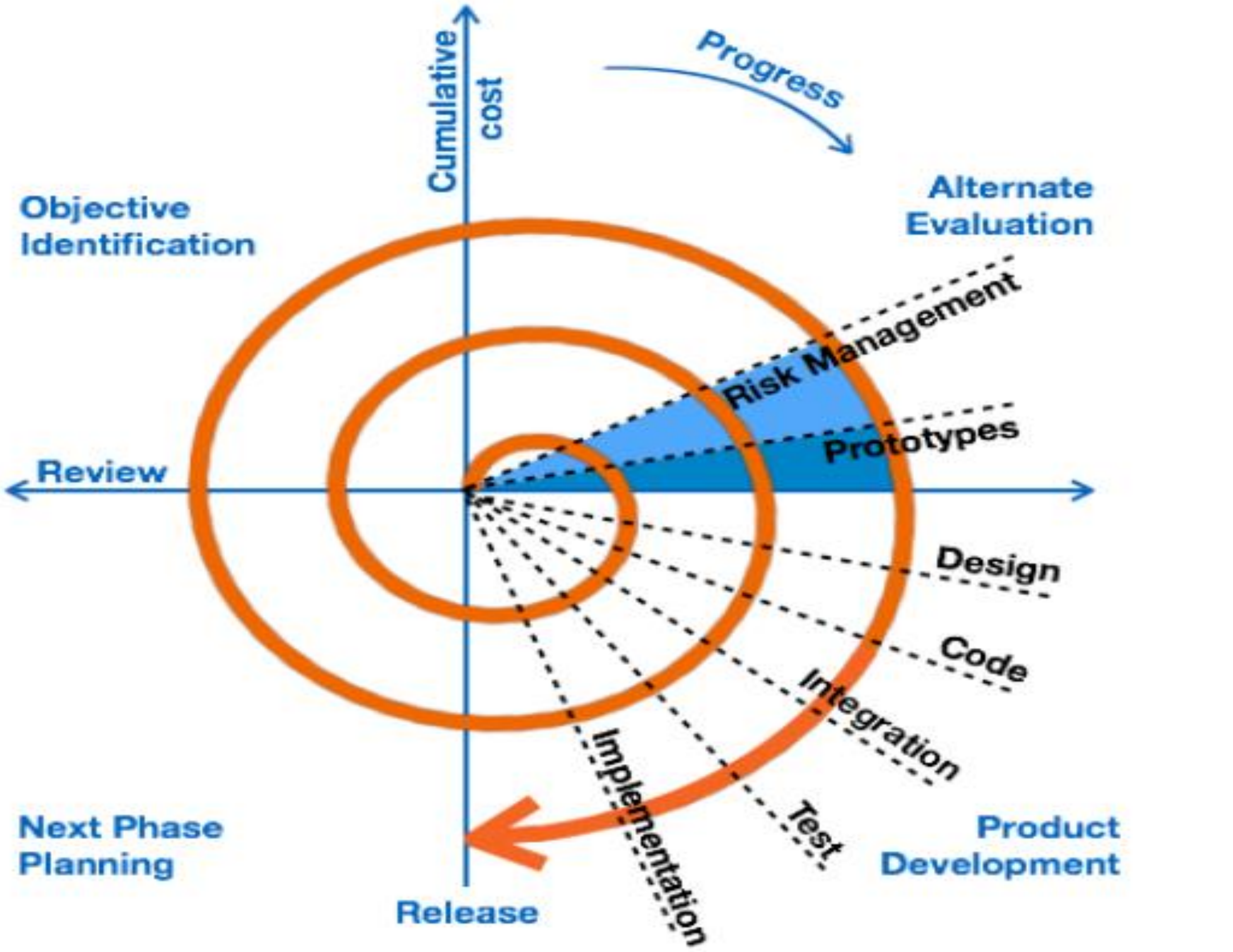
– The second branch corresponds to actual tests carried out on components produced.

– The integration is then carried out until the final software system is obtained.

– The advantage of such a model is to avoid stating a property that is impossible to objectively verify once the software has been produced.

– the V cycle is standardized, it is widely used, especially in industrial computing and telecommunications

Spirale Life Cycle



SDLC - Spiral Model (tutorialspoint.com)