# INGEGNERIA DEL SOFTWARE-intro 4 marzo

*I software sono costruiti da più persone:*

* *Problemi di comunicazione: incomprensioni, gap di linguaggio*
* *Problemi di coordinamento*

*Sono presenti più versioni:*

* *Problemi per mantenere negli anni*

## SOFTWARE:

insieme di programmi, procedure, regole, documentazione e dati

* Documento requisiti
* Pianificazione progetto
* Definizione test
* Definizione test case
* Creazione script (build + deployment)
* Manuale utente

#### Tipi di SW:

* Software embedded in prodotti non SW (auto, lavatrice)
* Stand alone (office, social network)
* Embedded in azienda (Information system, portale didattica)

#### Criticità SW:

* **Safety**: sicurezza della persona 🡪 ci sono dei SW che possono gravare sulla vita (guida autonoma)
* **Mission**: SW che sbagliando potrebbe gravare sull’azienda (banca, finanza, retail)

*NOTA:*

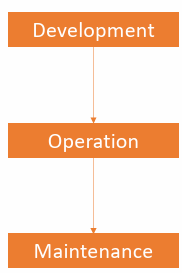
* *SW non è gratis*
* *SW cambiano rapidamente*
* *SW non è perfetto*
* *SW è complesso*

### PROCESSO PER SOFTWARE

* **Prodotto software**: proprietà **funzionali**
  + Cosa fa, casi d’uso
  + Dati input 🡪 determinati output
  + Casi d’uso
* Proprietà **NON funzionali**: (invalidano un SW)
  + **Usabilità**
    - Ore per imparare ad usarlo
    - Quanto gli utenti apprezzano il sw
    - Funzioni utili all’utente
  + **Efficienza**
    - Response time
    - Memoria, cpu, banda, energia usata
  + **Affidabilità/Disponibilità**:
    - Difetti visibili all’utente per un periodo di tempo
    - Percentuale di tempo in cui il prodotto non è disponibile per l’utente finale
  + **Mantenibilità**:
    - Quanto effort per add, modify, cancel di funzioni sw
    - Effort per fixare un difetto
    - Effort per deploy su diverse piattaforme
  + **Sicurezza:** autorizzazioni, condivisione, protezione
  + **Safety:** assenza di situazioni rischiose per persone
  + ***Dependability****= Safety + security + reliability*

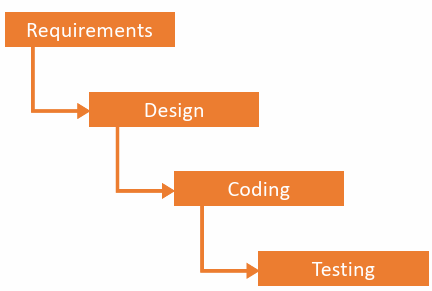
Caratteristiche delle proprietà NON funzionali:

* + - Difficili da ingegnerizzare
    - Spesso dimenticate
    - Fare la differenza con prodotti competitors

****

#### PROCESSO SW:

**Si divide in 3 macrofasi**

* Sviluppo
  + Requisiti
  + Design
  + Coding 🡪 scrivo codice, implemento algoritmi
  + Testing 🡪 verifica delle funzionalità del software
    - Fornire input ad una funzione e aspettarsi degli output
    - Unit testing: test delle singole unità
    - End-to-end testing: test complete del funzionamento
* Operazione 🡪 messa in esercizio
* Mantenimento
  + Fix bug
  + Cambiamento condizioni
  + Cambiamento leggi
  + Novità

## TOOL:

* Git, subversion, Polarion, Clearcase, Team Foundation Server

#### TOOL per requisiti:

* Context diagram 🡪 cosa sta dentro e cosa sta fuori al nostro sistema
* Stakeholders
* User stories/personas 🡪 identificano gli utenti che useranno il sistema
  + Storiella della persona che usa il sistema
  + Cosa si aspetta
  + Per cosa lo usa
* Casi d’uso: cosa l’utente deve fare (alto livello)
* Scenario
* Lista dei Requisiti funzionali
* Lista dei Requisiti non funzionali
* Glossario: attraverso un diagramma UML

#### Tool software:

* Design
  + Component diagram
  + Package diagram
  + Class diagram
  + Iteraction diagram
* Development:
  + Visual studio code
* Test:
  + Unit test con white box *🡪 definisco un test con una chiara percentuale di righe che sono coperte da quel test (coverage)*
  + Unit test con black box

## Proprietà del processo:

* Costo
* Effort 🡪 person hours 🡪 quante ore servirebbero se a farlo sarebbe solo una persona
* Puntualità: data di consegna promessa vs data di consegna effettiva
* Conformità:

LEGGI:

* **Leggi generali**
* **Leggi dell’ingegneria del software**
  + Deficienza nei **requisiti** sono le prime cause di failure
  + **Design** errato e requisiti sono le prime cause di difetti
    - Difetti dai requisiti e design sono molto costosi da fixare
  + **Modularità** e strutture **gerarchiche** aiutano la gestione della complessità
    - Posso aggiungere tranquillamente nuovi moduli
  + **Riuso** garantisce alta qualità e bassi costi
  + Per avere un buon design, serve che ci sia qualcuno che abbia una profonda **conoscenza nel dominio** per cui sto producendo sw
  + Test possono solo provare la presenza di errori, non la loro assenza
  + Sistemi cambiano
    - Devo aggiornare sw man mano 🡪 nel caso in cui si sono troppe modifiche, magari devo fare da 0: architecure erosion 🡪 refactoring
  + Produttività non è mai costante
  + Aggiungere risorse alla fine del progetto può essere letale
  + Processo di scrittura del sw va adattato al progetto
* **Leggi dei sistemi informativi**
  + Legge di Conway: la struttura del sistema prodotto da un’organizzazione rispecchia la struttura di comunicazione dell’organizzazione
    - Modo in cui implemento i processi, riflette la struttura del team
* **KISS**
  + Keep It Simple Stupid 🡪 non creare complessità dove non serve
  + Dividere il problema più grande in problemi più piccoli
  + Usare moduli indipendenti tra loro