# Ingegneria dei software

dmotrio.xyz

7 aprile 2022

## Indice



## Elenco delle figure



## Elenco delle tabelle

# 1 T1 - Software Development Process

L'Ingegneria del Software non è solo scrivere codice è piuttosto un concetto di risoluzione dei problemi del mondo reale sfruttando il software.
I requisiti sono sempre più stringenti: tempi brevi, sistemi complessi, molte features.
Un buon software deve avere ottime **maintenability, dependability, efficiency, acceptability**.
I problemi e le soluzioni sono sempre complesse ma il software permette massima flessibilità. È un sistema discreto.
Le sfide principali sono rappresentate da **eterogeneità, delivery, trust**.
La fase di problem solving si suddivide in analisi e sitesi.

## 1.1 Cos'è l'ingegneria del software?

L'Ingegneria del software è un **insieme di tecniche, metodologie, strumenti** che aiutano nella produzione di software di alta qualità dati un budget, una scadenza, e delle modifiche continue.
La sfida principale è quelli di aver a che fare con complessità elevate e ad un aumento delle responsabilità, dato che un ingegnere del software non deve solo scrivere codice, ma piuttosto lavorare con competenza e confidenzialità, attenendosi ad un'etica.

### 1.1.1 Processo del software

Dopo una rappresentazione astratta, si procede con un set di attività strutturate: specifiche dei **requirements, design, implementazione, validazione, evoluzione.**

## 1.2 Modelli di sviluppo del software

Distinguiamo tra **plan-driven** e **agile** development. Nel primo si pianificano i requisiti e solo in seguito si sviluppa il software. Nel secondo si sviluppa il software un pezzo alla volta, a stretto contatto con il cliente per dei feedback.

### 1.2.1 Modello a cascata

Modello plan-driven, le specifiche e lo sviluppo sono separati.
I pro:

- ottima documentazione
- manutenzione semplice

I cons:

- specifiche congelate dopo la pianificazione iniziale
- cliente poco coninvolto
- tempi lunghi

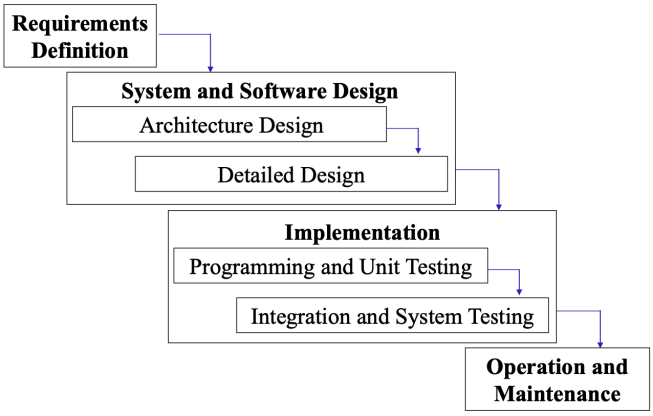


Figura 1: Modello a cascata

### 1.2.2 Modello a spirale

Diverse fasi che susseguono a spirale, la gestione del rischio viene gestita tramite prototipazione che permette di testare i prodotti.
I pro:

- prevenzione dei rischi
- completezza della documentazione
- flessibilità
- elevata usabilità
- buon design
- facilità di manutenzione
- ridotto costo di sviluppo

I cons:

- modello costoso
- riservato ad esperti e a progetti costosi

Il prototipo è un'implementazione limitata del sistema, rappresentanti solo alcuni aspetti alla volta.

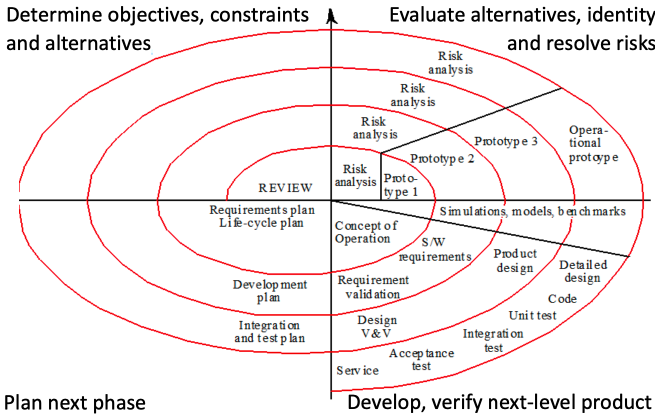


Figura 2: Modello a spirale

### 1.2.3 Sviluppo incrementale

Modello che si suddivide in fasi:

1. raccolta dei requisiti
2. versione iniziale
3. fase di design
4. fase di implementazione
5. produzione di versione finale

I pro:

- naturale presenza di prototipi ad ogni aggiunta di features
- basso rischio di fallimento
- qualità di testing in base alla priorità

I cons:

- scarsa visibilità d'insieme
- sistemi mal strutturati
- skill speciali necessarie

Adatto a progetti piccoli o parti di progetti grandi.

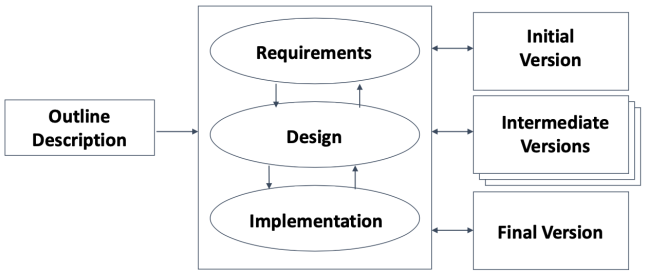


Figura 3: Modello a incrementale

### 1.2.4 Sviluppo guidato dai test

Vengono prima scritti i test e poi l'implementazione mettendo le difficolta in primo piano.
Rende il debug più semplice.
Si aggiunte il nuovo test e poi la features cosi da veriicare che tutti i test precedenti siano validi.

### 1.2.5 Sviluppo agile

Questo modello di sviluppo è basato sul concetto di delivery del prodotto continuo avendo delle features in continua evoluzione.
Il cliente è al centro dello sviluppo, il team si autoorganizza e le features vengono aggiunte man mano.
Essendoci la mancanza di planning il team deve essere esperto per non perdersi.
Spesso si creano documentazioni sbagliate o incomplete.

### 1.2.6 Extrem programming

L' XP viene scelte quado i requisiti cambiano velocemente, i team sono ridotti e affiatati(pair programming per esempio).
Tipo di programmazione agile, basato sul design semplice, release minori, refactoring continuo, alta semplicità.

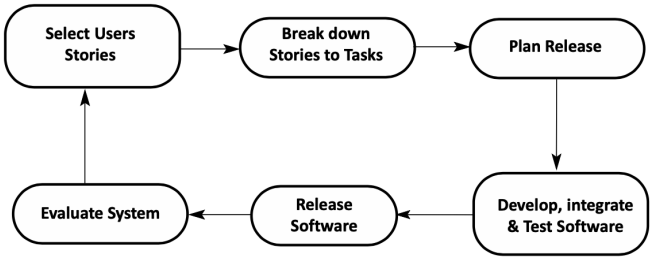


Figura 4: Modello a extrem programming

## 1.3 Software riusabile

È comodo lavorare per **microservizi atomici**, di modo da usarli in seguito se si affronta un problema simile, un'esempio sono le API, è sempre meglio riusare qualcosa che riscriverlo da zero.
Questo "*mindset*" riduce i temi e i costi di sviluppo nel luno periodo.
Il contro è il fatto che i microservizi non sono fatti espressamente per una task specifica e si va a sacrificare qualche requisito.

# 2 T2 - Coding, Debugging, Testing

Lo stile di coding è importante perchè un programma viene scritto una volta e letto spesso da altri!
Stare a ttenti ai layout, nomi, commenti, ecc...

## 2.1 Legge di Ambler per gli standard

Più uno standard è adottato e più sarà facile farlo usare al proprio team.
Non perdere tempo adietro a standard scemi.
Se hai dei dubbi usa gli standaard di google che sicuramente ne sanno più di me e te. **standard di google**

Solitamente si hanno degli standard aziendali e questi standard vanno a chiarire aspetti come:

- nomenclatura
- formattazione
- formato
- contenuto

## 2.2 Coding practices

- indentazione
- whitespaces
- naming, commenting

Fondaentale è spiegare i compiti delle classi, funzioni e procecssi complessi.
Utilizzare uno standard permette ai collechi di non dover riscrivere il codice. Rendendo più semplice le iterazioni del codice successive.

## 2.3 Gestione degli errori

Si fa un distinguo fra il prima, durante e dopo:

- prrevenzione
- rilevamento
- recupero

Per debuggare, bisogna riconoscere l'errore, isolare la fonte, identificarne la causa, trovare un fix, applicarlo e testarlo.

## 2.4 Testing

Permette di trovare errori ma non la loro assenza, **il tester non dovrebbe essere il programmatore**.

### 2.4.1 Unit testing

Test di singoli componenti.

### 2.4.2 Integration testing

L'intero sistema è visto come insieme di sottositemi, l'obbiettivo è quello di testare tutte le interfacce e le interazioni fra i sottositemi.

### 2.4.3 System testing

test dell'intero sistema per vedere se rispetta i requisiti funzionali.

### 2.4.4 Functional testing

Verifica delle funzionalità del sistema, test basati sui requisiti del progetto.

### 2.4.5 Performance testing

Test in situazioni estreme(stress testing, volume testing, recovery testing, penetration testing).

### 2.4.6 Acceptance testing

Il sistema è pronto per la produzione??
Test scelti ed effettuati dal cliente.(alpha e beta test)

# 3 T3 - System Modelling and UML

Rappresentazione astratta del sistema e dei problemi, si dedvono introdurre i componenti essenziali mediante una noazione consistente.
Il system modelling deve essere **predictive**(prima del development), **extracted**(da un sistema esistente) e **prescriptive**(definire regole per l'evoluzione del software).

## 3.1 UML

UML è semplice, espressivo, utile, consistent, estensibile.

# 4 T4 - Requirements engineering

Gli scopi dell'ingegnerizzazione dei requisiti è **identificare** i servizi necessari e i constrait, **definire** offerta e contratto, **ottenere** tutte le informazioni necessarie al design.
Si cerca di ottenere requirements:

- validi(reali necessità)
- non ambigui
- completi
- comprensibili
- consistenti
- prioritizzati
- verificabili
- modificabili
- tracciabili

## 4.1 Classificazione dei requirements

### 4.1.1 Requisiti funzionali(features di sistema)

Descrivono funzionalità di sistema o di servizio, come input di dati, output, operazioni svolte, workflow, autorizzazioni.

### 4.1.2 Requisiti non funzionali(features di sistema)

Descrivono i limitti di parti del sistema e del suo sviluppo.
Specificano criteri per giudicare l'operato del sistema.

### 4.1.3 Requisiti del dominio(features di sistema)

Derivato dal campo di utilizzo del software.

### 4.1.4 Requisiti volatili(natura statica/dinamica)

- mutable requirements = cambiano nel tempo(tasse, nomative, ecc)
- emergent requirements = cambiano quando il cliente capisce di più il sistema
- consequential requirements = emergono con l'informatizzazione del sistema che non lo era
- compatibility requirements = emergono dal dover interfacciare il sistema con altri sistemi

## 4.2 Rischi

I rischi della stesura dei requisiti possono essere:

- imprecisioni
- conflitti tra requisiti

### 4.3 Documento di specifica dei requirements

Questo documento specifica i requisiti del sistema, includendo una definizione e una specifica.
Prende il nome di system specification se include direttive su harware e software.
Software Requirements Specification (**SRS**) se include solo specifiche software.
Un **SRS** deve includere:

- introduzione
- descrizione generale
- features
- requirements

Il linguaggio naturale usato per stilare questo documento implica ambiguità, per questo si ricorre a una struttura ben definita per evitare ambiguità.

## 5 T5 - Requirements ecgineering and UML