Ciclo di vita del software

ebenatti@fermimn.edu.it

Cosa si studia per il ciclo di vita

In questo corso oltre a scrivere software dobbiamo porci come piccoli progettisti e chiederci:
- Come avviene la produzione industriale del software?
- Come funziona una software house?
- Quali sono le figure professionali presenti (analisti, programmatori, tester ecc.)
- Come viene documentato il software?
- Come scrivere programmi di qualità e vendibili?

Per rispondere a queste domande e a molte altre si crea una vera e propria disciplina chiamata **ingegneria del sofware.**

Ingegneria del software

L' **ingegneria del software** (software engineering in inglese) è quella disciplina informatica che si occupa dei processi produttivi e delle metodologie di sviluppo finalizzate alla realizzazione di sistemi software.
E’ una disciplina, di studio, che nel corso degli anni ha avuto diverse evoluzioni date dalla della complessità dei computer, si è manifestata una sempre crescente difficoltà, da parte delle aziende di software, nel portare a termini i propri progetti rispettando il budget, i tempi, o i requisiti previsti
Questa situazione di difficoltà è nota come "**crisi del software**"

Crisi software

La crisi del software (anni 60') si manifesta in diversi modi:
- mancato rispetto dei tempi di consegna dei progetti,
- mancato soddisfacimento delle specifiche dei progetti,
- produzione di software non efficiente,
- difficoltà di manutenzione del software, perchè non facile da capire e complesso
- assenza di documentazione sui software esistenti

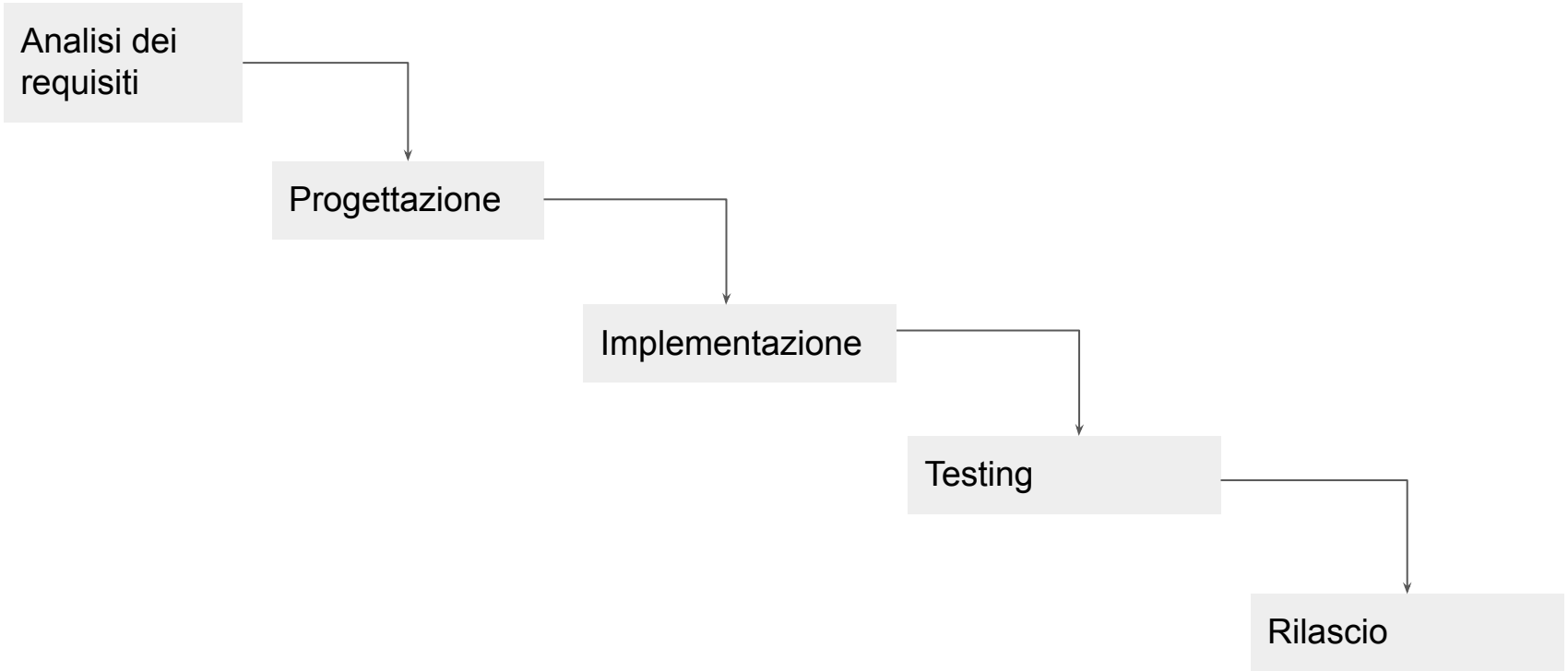
Per superare la "crisi del software" si è capito che per fare progetti complessi che non è più sufficiente scrivere software e basta

Era necessario **strutturare il lavoro** in modo da poter essere **replicabile**, cioè avere dei tempi e dei costi replicabili per progetti simili. Vengono quindi teorizzati i cicli di vita del software.

Ciclo di vita del software

Il ciclo di vita di un software è l’insieme delle attività e delle azioni da intraprendere per sviluppare un software.

Modello a cascata



Analisi

Ruolo professionale: analista

La fase ha lo scopo di identificare i requisiti dell'applicazione da realizzare.

Nel mondo reale di solito le applicazioni hanno un **committente,** cioè colui che commissiona il software (e lo finanzia).

In questa fase si fanno e documentano:
- interviste con il committente, raccolta dei requisiti, spesso ambigui
- interviste con i potenziali utilizzatori del software
- studio di fattibilità tecnico ed economico (vedi Gestione Progetto)
- SRS, Software Requirement Specification

I documenti di analisi sono in linguaggio naturale, facile, comprensibile da tutti.

Definizione di requisito

Funzionali: sono i requisiti che descrivono le funzionalità che il sistema renderà disponibili all'utente, sottolineando l'interazione tra utente e sistema,

Non funzionali: questo tipo di requisiti non descrive una funzionalità ma una caratteristica del sistema, ad esempio una caratteristica prestazionale (tempo di risposta, operazioni al minuto) ecc.

Tecnologici: questi requisiti sono relativi agli aspetti tecnologici e infrastrutturali (es. linguaggio di programmazione, sistema operativo, applicazioni software compatibili ecc).

Analisi dei requisiti

Viene creato documento un SRS = Software Requirement Specification
E' un documento di base di "accordo" tra la softwarehouse e il committente,

Il bravo analista deve valutare che i requisiti del cliente siano fattibili e cercare di portare i requisiti più critici (quelli meno fattibili) ad un livello di fattibilità tale che nelle successive fasi il **progetto non fallisca** (vedi gestione progetto), pensando con il cliente le possibili soluzioni o migliorie possibili.

Il bravo analista si relaziona con i cliente (che spesso di programmazione sa poco…) e porta il cliente a capire le sue vere necessità, che possono essere facilmente capite portando il cliente nella direzione che a noi permetterà di riuscire a realizzare il progetto.

Analisi dei requisiti

E infine, far contemplare il **costo** della realizzazione di un software e delle fasi successive, con il **prezzo** che il cliente intende spendere come budget.

Spesso di fatto, l'analisi dei requisiti è una gara tra domanda e offerta di prestazioni informatiche, che contrappongono il committente da una parte e la software house dall'altra.

Vedi Gestione progetto

Priorità dei requisiti

Nel documento SRS di solito si riportano come

- Must
- Should
- May.

La priorità potrebbe essere un elemento discriminatorio per l’evoluzione del progetto.

Es. il cliente desidera la visualizzazione dei prodotti supportando più immagini in alta qualità.

Conseguenza: Il software dovrà adattarsi ed essere visualizzato su monitor ad alta qualità.

Progettazione

Figura professionale: Progettista

Dai requisiti software scritti spesso in linguaggio naturale, 'obiettivo della fase di progettazione è identificare l'architettura del software (quali saranno le classi principali, quali saranno i loro principali attributi e metodi, e come dovranno interagire tra loro). Il progettista si concentra sulla progettazione delle interfacce delle varie classi (l'implementazione dei metodi è lasciata ai programmatori).

Attualmente UML rappresenta il linguaggio standard utilizzato per la stesura degli schemi di progetto di un software.

Implementazione

Figura professionale: programmatore

Realizza ciò che è stato progettato con le tecnologie (linguaggi di programmazione) scelti in fase di progettazione.

Test

Figura professionale coinvolta: tester

Obiettivo: individuare errori del software (bug) prima del rilascio.

Spesso sono programmatori a svolgere questa attività che è meglio non sia svolta dallo stesso programmatore che ha scritto il suo codice.

Si redirige il cosiddetto un piano di test.
La modalità di test prendono il nome dalle lettere greche:
- alpha test: test interno di pre rilascio, di un prototipo non completo al fine di allineare il lavoro secondo le specifiche previste prima del rilascio
- beta test: rilascio del codice ad un ristretto gruppo di utenti al fine di raccogliere i feedback e individuare ancora più errori.
- release candidate: prima versione definitiva, in teoria senza errori, in teoria..

Rilascio

Il software è pronto per la commercializzazione e l'uso reale e non simulato agli utilizzatori (release del software).
Nei software reali spesso si fa il deployment che è quell'insieme di operazioni che tipicamente si fa **presso la sede del cliente** mediante cui si configura l'ambiente, si installa il software, si collauda e si verifica l'effettivo funzionamento operativo.

Esempio: il software di un sistema di sicurezza di un treno, deve essere installato, configurato, provato e collaudato sul campo.

Software "da scaffale" o da "download" non hanno questo passaggio ma semplicemente rendono disponibile l'eseguibile di installazione.

Criticità del modello a cascata

Questo modello funziona bene in teoria, ma cosa succede se:
- mi accorgo di un bug a software già rilasciato, o il software non soddisfa il committente
- alcuni requisiti con il committente non erano chiari e lo diventano solo a codifica iniziata,
- emergono problemi di realizzazione del codice a causa di una progettazione che ha tralasciato alcuni aspetti o ne ha progettati male altri?

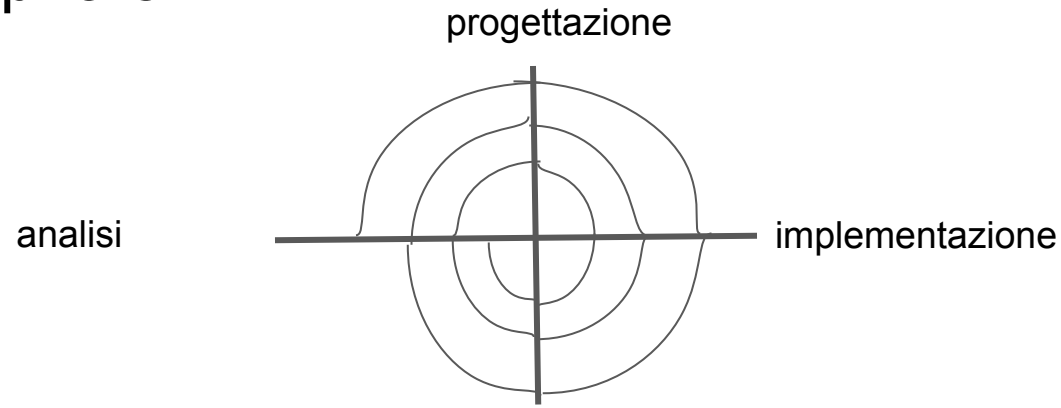
Modelli iterativi

I cicli di vita moderni iterano più volte le varie attività di analisi, progettazione, sviluppo e rilascio in modo che ad ogni sequenza (ciclo) di attività ci sia un feedback migliorativo che via via migliori il software creato.

L'obiettivo è giungere velocemente (al termine della prima iterazione) a un prototipo funzionante ma non completo, che possa ricevere il feedback del committente,

Ogni nuovo prototipo creerà un miglioramento del prototipo con sempre più funzioni per avvicinarsi sempre di più al prodotto finito.

Modello a spirale



Modello incrementale e interattivo

La ripetizione delle attività avviene fino a quando:
- il budget finisce,
- l’eventuale evoluzione del modulo software rischia di provocare un peggioramento del sw invece di un miglioramento,
- i requisiti sono rispettati in un modo sufficientemente maturo

Metodologie agili

Si sviluppa nei primi anni 2000 un manifesto agile e nuove tipologie di ciclo di vita così dette, perchè si basano molto sullo sviluppo rapido in produzione e sintetizzano in modo diverso le altre attività.

- Due di queste sono Scrum (mischia) e code XP

Cardini di queste metodologie agili:

- considerazione delle iterazioni tra persone di sviluppo e strumenti,
- collaborazione col cliente,
- rispondere al cambiamento tecnologico,
- forte uso prototipi in produzione.

Manutenzione del software

E’ l’insieme di attività di modifica post installazione di un software, tre tipologie:

- **Correttiva,**
- **Adattattva,**
- **Evolutiva.**

Il rilascio di nuove versioni è sempre possibile anche su un progetto software complesso per risolvere uno di questi casi.
Spesso la manutenzione può essere **critica** in determinati contesti per software dove la sicurezza è fattore critico. (esempio. software aeronautico)

Il software deve essere progettato per essere facilmente mantenibile (=modificabile a distanza di tempo). L’assistenza e la manutenzione del software può essere a distanza di tempo fonte di grande lavoro e ricavi per la nostra software house.

Documentazione

Ogni progetto software **deve** avere della documentazione che certifichi il lavoro svolto,

- La documentazione deve rispettare delle regole e di solito è numerata, datata e aggiornata nel tempo. Ad esempio ogni richiesta di modifica di un cliente potrebbe essere registrata e quando implementata documenta.
- Per il software ad esempi di documentazione sono UML, manuali utente, manuali di manutenzione, javadoc e commenti nei sorgenti dei moduli software.
- La documentazione è una **attività trasversale** a tutto il ciclo di vita!

Di altri documenti tecnici (disegni, prototipi ecc) rimando a gestione progetto.