Processo Software

E’ inteso come il modo in cui produciamo software e parte nell'esplorare il problema fino al suo ritiro dal mercato.

Le fasi sono:
- Analisi dei requisiti
- Specifica
- Progettazione
- Implementazione
- Integrazione
- Mantenimento
- Ritiro

Il processo software riguarda anche gli strumenti e le tecniche per lo sviluppo, il mantenimento e i professionisti coinvolti.

Software Engineering

Disciplina sia tecnologica che gestionale che ha lo scopo di produrre ***Fault-Free Software*** facile da modificare, rispettando i tempi, il budget e le necessità del cliente.

Nascita

Siamo negli anni ‘60 e la qualità del software era in generale inaccettabilmente bassa (***Crisi del software***) ma viene trovata una soluzione a questa crisi nell'***ingegneria del software*** (usare per la produzione di software tecniche e paradigmi delle discipline ingegneristiche).

Specificità del Software

Un Software a differenza di altri prodotti dell’ingegneria non è vincolato da materiali o leggi fisiche, non si “fabbrica” ma si “sviluppa”, non si “consuma” ma si “deteriora”, è più flessibile alle modifiche o alla manutenzione.

Inoltre un software è progettato per minimizzare l’effetto del fallimento (***Fault Tolerance***).

Manutenzione

La manutenzione include tutti i cambiamenti al prodotto software anche dopo la consegna al cliente.

Si divide in:

- ***manutenzione correttiva*** (20%), rimuovere gli errori lasciando invariata la specifi***a***
- ***manutenzione migliorativa,*** effettuare cambiamenti alla specifica e nella loro implementazione e si divide a sua volta in:
    - *Perfettiva* (60%), modificare per migliorare le qualità del software (introdurre nuove funzionalità, migliorare funzionalità esistenti)
    - *Adattiva* (20%), modificare dopo cambiamenti a livello legislativo, hardware, di sistema operativo, ecc…

Temi di ingegneria del software

Processo Software

- Organizzazione e gestione dei progetti
- Metodi di composizione dei gruppi di lavoro
- Strumenti di pianificazione, analisi, controllo
- Cicli di vita del software
- Definizione e correlazione delle attività
- Modelli ideali di processo di sviluppo

Realizzazione di sistemi sw

- Strategie di analisi e progettazione
    - Tecniche per la comprensione e la soluzione di un problema
    - Top-down, bottom-up, progettazione modulare, OO

- Linguaggi di specifica e progettazione
    - Strumenti per la definizione di sistemi software
    - Reti di Petri, Z, OMT, UML

- Ambienti di sviluppo
    - Strumenti per analisi, progettazione e realizzazione
    - Strumenti tradizionali, CASE, CAST

Qualità del sw

- Modelli di qualità
    - Definizione di caratteristiche della qualità

- Metriche software
    - Unità di misura, scale di riferimento, strumenti
    - Indicatori di qualità

- Metodi di verifica e controllo
    - Metodi di verifica, criteri di progettazione delle prove

    - Controllo della qualità, valutazione del processo di sviluppo

Stakeholders

- Fornitore: Chi sviluppa
- Committente: Chi richiede
- Utente: Chi lo utilizza

Processo Software - Modelli di ciclo di vita

Si indica il percorso da svolgere per sviluppare un prodotto o sistema software.
Inizia con l'esplorazione dell'idea e dopo varie fasi finisce con la dimissione del sw.

Il processo va ''modellato'', cioè strutturato (suddividerlo in attività).
Per ogni attività del processo dire cosa, quando e quali prodotti.

I Modelli di ciclo di vita si basano sull' organizzazione delle attività (ordinamento e criteri per terminare e passare alla successiva).

Esistono vari tipi di questi modelli e ognuno ha affrontato un nuovo aspetto:

- **Build-and-Fix**: un non-modello
    - Attività non identificate né organizzate
    - Progetti non gestiti

- Modelli prescrittivi
    - **Cascata**
    - **Modello a V**
    - **Rapid Prototyping**
    - **Incrementale**
    - **A spirale**
    - **Unified Process**

- Modelli agili
    - **Extreme Programming**
    - **Scrum**

Build-and-Fix

Il prodotto viene sviluppato senza specifica e senza progettazione, viene scritto un programma che poi viene modificato più volte.

“Pro”: forse adeguato con piccoli progetti di al massimo un paio di centinaia di righe di codice ma inutile in altri casi.
Contro: niente documentazione e quindi manutenzione complessa, improponibile con prodotti di una discreta dimensione.

Modello a Cascata (Royce, 1970)

Il prodotto attraversa varie fasi in un ordine specifico e può passare di fase in fase solo dopo il suo completamento (inclusa la documentazione).

Pro:

- distingue e definisce le fasi di un processo sw evidenziando l’importanza delle fasi di analisi dei requisiti e di progettazione prima di passare alla codifica;
- è un modello “document driven”, quindi da una certa fase si può passare alla successiva solo dopo il completamento della precedente e l’approvazione del documento prodotto di un gruppo di valutatori;

Contro:

- manca l’interazione col cliente che vede il prodotto solo alla fine del processo
- eccessiva produzione di documenti (una fase non può finire senza che sia stato completato e approvato il relativo documento)

Si ricorda il modello a cascata quindi per il suo valore storico.
Già nel 1970 Royce propose di migliorare il modello con dei “feedback loops” da una fase precedente alla successiva.

Modello a V (Hughes Aircraft, 1982)

Anche detto sequenziale, il modello evidenzia come sia possibile progettare i test durante le fasi di sviluppo (prima della codifica), che è un concetto ripreso nel *test driven development.*

Da Wikipedia:

Il modello, derivante da quello a cascata, invece di discendere lungo una linea retta, dopo la fase di programmazione risale con una tipica forma a V.
Il modello dimostra la relazione tra ogni fase del ciclo di vita dello sviluppo del software e la sua fase di testing.
Il V-model realizza un metodo ben strutturato, in cui ogni fase è implementabile dalla documentazione dettagliata della fase precedente. Le attività di testing, come il testing della fase di progetto, iniziano già all'inizio del progetto prima della codifica e ciò consente di risparmiare un ampio tempo di progetto.

Rapid Prototyping (o modello evolutivo)

Lo scopo è quello di costruire il prima possibile un prototipo che il cliente possa utilizzare e provare.

Pro:
- utile quando i requisiti non sono chiari
- aiuta il cliente a descrivere i requisiti

Modello Incrementale

E' un modello in cui il sistema è costruito aggiungendo qualcosa a quello che era già stato fatto. I requisiti sono definiti all'inizio e il sistema è implementato, integrato, testato con passaggi incrementali.
Si applica in caso di requisiti stabili.

Pro:
- ritarda la realizzazione delle componenti che dipendono da fattori esterni (tecnologie, hw, ...)
- permette la produzione veloce di qualcosa

Contro:
- se non viene progettato bene diventa un build-and-fix

Modello a Spirale (Boehm, 1988)

E' un tipo di modello di tipo astratto iterativo e ogni iterazione è organizzata in 4 fasi:

- definizione degli obiettivi
- analisi dei rischi
- sviluppo e validazione
- pianificazione del nuovo ciclo

Va specializzato per dire cosa fare in concreto in ogni sua fase e iterazione (di tipo astratto). Inoltre è un modello che prevede una maggior comunicazione e confronto con il committente.

Pro:
- evidenzia gli aspetti gestionali
- pianificazione delle fasi
- centrato sull'analisi dei rischi ("risk driven")

Contro (o per meglio dire in questo caso, tipici rischi in cui ci si può imbattere):

- dominio poco noto
- linguaggi o strumenti nuovi
- personale non addestrato

Unified Process (Booch-Jacobson-Roumbaugh, 1999)

E' un modello iterativo incrementale guidato dai casi d'uso e dall'analisi dei rischi e molto importante ne è la descrizione dell'architettura del sistema.
Infatti l'approccio è quello di concentrarsi nelle prime fasi sull'architettura di massima lasciando i dettagli alle fasi successive così da avere una visione generale del sistema fin da subito.
Consiste in una serie di cicli che hanno tutti la precedente forma e che si concludono con una release del sistema.
Ogni ciclo ha diverse iterazioni che sono raggruppate in 4 fasi e ognuna delle fasi finisce con la decisione del manager di continuare o terminare il progetto.

Le fasi sono:

- **Avvio**
    - Fattibilità; Analisi dei rischi; Requisiti essenziali per definire il contesto del sistema; Eventuale prototipo

- **Elaborazione**
    - Analisi dei requisiti; Analisi dei rischi; Sviluppo di un'architettura base; Piano per la fase di costruzione

- **Costruzione**
    - Analisi, Disegno, Implementazione, Testing

- **Transizione**
    - Beta testing, Aggiustamento delle prestazioni, Creazione di documentazione aggiuntiva, Attività di formazione, Guide utenti, Creazione di un kit per la vendita

Processi Agili (basato sui principi del Manifesto di Snowbird, feb 2011)

Per metodologia agile (o leggera) si intende un metodo particolare per lo sviluppo di sw che coinvolge il più possibile il committente.

I punti principali del manifesto sono:

- **Comunicazione:**
    - è una risorsa fondamentale del progetto, più importante di processi e strumenti

    - tutti possono parlare con tutti e la collaborazione diretta anche con il cliente offre risultati migliori

- **Semplicità:**
    - gli analisti devono mantenere la descrizione formale il più semplice e chiara possibile
    - è più importante avere un software funzionante con un codice semplice e avanzato che una buona documentazione (cercando invece di ridurla al minimo)

- **Feedback:**
    - sin dal primo giorno testare il codice e rilasciare nuove versioni del sw ad intervalli frequenti

- **Coraggio:**
    - rispondere ai cambiamenti e implementarli man mano cercando di dare in uso il sistema il prima possibile

eXtreme Programming (Esempio di Agile)

Si basa su

- pianificazione flessibile (scenari proposti dagli utenti e coinvolge i programmatori)
- rilasci frequenti (2-4 settimane o all'inizio di una nuova pianificazione)
- progetti semplici comprensibili a tutti
- verifica (testing) di unità e di sistema (basati su scenari) e supporto automatico
- test driven development
- cliente sempre a disposizione (circa ogni settimana)
- programmazione a coppie con un solo terminale in cui il *driver* scrive il codice e il *navigatore* controlla in maniera attiva
- nessun lavoro straordinario
- collettivizzazione del codice (accesso libero, integrazione continua, standard di codifica)
- code refactoring (il codice deve essere autoesplicativo)
- daily stand up meeting

Scrum (''mischia'' dal rugby)

Processo agile in cui un insieme di persone si muove all'unisono per raggiungere un obiettivo che garantisce la soddisfazione personale e della squadra.
E' un processo iterativo incrementale che fornisce alla fine di ogni iterazione un set di funzionalità potenzialmente rilasciabili.

Scrum ha 3 fasi principali:

- **Pre-game phase** divisa a sua volta in:
    - Planning sub-phase
        - Include la definizione del sistema che deve essere sviluppato
        - Viene creata una Product Backlog List, che contiene tutti i requisiti attualmente conosciuti
    - Architecture sub-phase
        - Viene pianificato un design di alto livello del sistema, inclusa l'architettura, in base agli elementi contenuti nel Product Backlog

- **Development (Game) phase** (parte "agile" dell'approccio Scrum)
    - Nella Development Phase, il sistema viene sviluppato attraverso una serie di Sprint
        - Cicli iterativi nei quali vengono sviluppate o migliorate una serie di funzionalità
        - Ciascuno Sprint include le tradizionali fasi di sviluppo del software
        - L'architettura del sistema evolve durante lo sviluppo negli Sprint
        - Uno Sprint si svolge in un intervallo di tempo che va da una settimana ad un mese

- **Post-game phase** (contiene la chiusura definitiva della release)

Oltre alle 3 fasi sono anche presenti 3 ruoli principali:

- **product owner**, quella persona a cui fanno riferimento tutti gli interessati al progetto (compreso il cliente) in grado di effettuare stime e gestire il procedimento secondo la pianificazione. Ha il potere di accettare o rifiutare i risultati di un lavoro e di terminare uno sprint se necessario.

- **membro del team**, quella persona che ha la responsabilità di costruire il prodotto e decidere cosa fare in ciascuno sprint. Ognuno realizza una cosa alla volta senza bisogno di avere un project manager in team indipendenti cross-functional (non specializzati in una singola cosa).

- **scrum master**, non ha autorità sul team ma è colui che supporta il team garantendo condizioni ottimali necessarie ad eseguire al meglio il lavoro.

Analisi dei Requisiti

Studio di affidabilità

Fase preliminare che si basa su una descrizione delle caratteristiche del sw da realizzare e su delle analisi tecniche (e non) per capire se il prodotto è realizzabile e ha senso di essere realizzato.

Si basa quindi anche su valutazioni di costi e benefici, fattibilità tecnologica e aspetti economici e di mercato.

Attività di analisi dei requisiti

Slogan: COSA NON COME.

Ha lo scopo di studiare e definire il problema da risolvere per capire e documentare cosa deve essere fatto e per effettuare negoziazioni committente/fornitore.

La fase di analisi dei requisiti è molto importante per i costi della realizzazione del prodotto. Ad esempio il costo per correggere un errore nella fase dei requisiti è molto basso ma se non viene trovato durante l'analisi o nelle prime fasi questo costo continua a salire fino all'ultima fase.

Questa fase può produrre:

- Documento e/o modello analitico di descrizione del dominio e descrizione dei requisiti
- manuale utente e casi di test (opzionali)

Dominio

Il dominio è il campo di applicazione del prodotto e questo deve essere acquisito dal gruppo di analisti ancora prima dell'incontro con i committenti (per poter porre loro le giuste domande e capire ogni sottile differenza fra termini all'apparenza sinonimi).

Per definire il dominio si costruisce un ***glossario*** che si arricchisce man mano e che come il dominio quindi si evolve dopo ogni incontro.

Glossario: Definizione di termini chiave del dominio utile per comprendere e documentare il dominio e validare i requisiti.

Acquisire conoscenza sul dominio e sui requisiti

Per poter acquisire conoscenza sul dominio e sui requisiti ci sono varie tecniche.
Una di queste tecniche sono le interviste.
Gli analisti incontrano i committenti e si procede con delle vere e proprie interviste (strutturate o non strutturate). Uno dei difetti di questa tecnica è che è difficile condurre una buona intervista.

Le altre tecniche possono essere: questionari a risposte multiple, costruzione di prototipi, osservazione di futuri utenti al lavoro, studio di documenti e i casi d'uso (per descrivere requisiti funzionali).

Il caso d'uso è un modo con cui l'utente può usare il prodotto e commentare l'elaborato.
I casi d'uso inoltre non devono contenere solo la sequenza di eventi corretta ma anche le eccezioni.

Requisiti

Requisito è una proprietà che deve essere garantita dal sistema per soddisfare una necessità di un utente.

I requisiti possono essere funzionali e non funzionali.

Quelli funzionali sono i requisiti di maggior valore di solito e descrivono le funzionalità che il sistema deve realizzare (azioni che deve compiere, reazioni a specifici input, comportamenti in situazioni particolari).

Quelli non funzionali (o di qualità) descrivono invece le proprietà del sistema software in relazione a determinati servizi o funzioni e possono anche essere relativi al processo (caratteristiche di qualità come efficienza e affidabilità, caratteristiche del processo di sviluppo come linguaggi di progr. e standard di processo, caratteristiche esterne come i vincoli legislativi, requisiti fisici e cioè l'hw).

Documento dei requisiti

Il documento dei requisiti è un contratto tra sviluppatore e committente nel quale si specifica cosa il prodotto deve fare e quali sono i vincoli da soddisfare (e di solito anche una deadline per la consegna).

Il documento è composto in 5 passi:

- Acquisizione (fase già vista con l'acquisizione dei requisiti)
- Elaborazione
- Negoziazione
- Convalida
- Gestione

Il documento dei requisiti è parte integrante del contratto, per questo viene completato prima della firma.

Elaborazione

La struttura del documento è:

- Introduzione: perché il sistema è desiderabile e come si inquadra negli obiettivi del cliente
- Glossario: con i termini e i concetti tecnici usati
- Definizione dei requisiti funzionali
- Definizione dei requisiti non funzionali
- Architettura: la struttura in sottoinsiemi cui riferire i requisiti
- Specifica di System and Software requirements: specifica dettagliata dei requisiti funzionali
- Modelli astratti del sistema: modelli (formali o semi-formali) che illustrano un solo punto di vita ciascuno (controllo, dati, funzioni)
- Evoluzione del sistema: previsioni di successivi cambiamenti
- Appendici: (descrizione piattaforma hw, requisiti di db, manuale utente e piani di test)
- Indici: lemmario

Lemmario: Lista di termini con puntatori ai requisiti che li usano.

N.B.: Requisiti ben posti sono nella forma: *il <sistema> deve <funzionalità>/<proprietà>.*

Convalida

E' il passo della validazione di un documento già strutturato.
E' composta da:

- Deskcheck che è suddiviso in:
    - Walkthrough: lettura sequenziale dei documenti
    - Ispezione: lettura strutturata dei documenti (tecnica del lemmario, ricerca di rimozioni o distorsioni o generalizzazioni)
- Uso di strumenti di analisi del linguaggio naturale
- Prototipi

Difetti da evitare nel documento dei requisiti sono:
- le omissioni (mancanza di un requisito) che generano incompletezza
- le inconsistenza (contraddizione fra requisiti o con l'ambiente operativo)
- le ambiguità (requisiti con significati multipli)
- sinonimi e omonimi (termini diversi con stesso significato e viceversa)
- presenza di dettagli tecnici
- ridondanza

Tecnica del Lemmario: utilizza i termini del glossario con i puntatori ai requisiti che li nominano e facilita quindi la ricerca di inconsistenze, sinonimi, omonimi e ridondanze.

Tecnica del Metamodello in PNL (Noam Chomsky):

I filtri linguistici sono i processi attraverso i quali costruiamo la nostra mappa del mondo e sono:

- **Generalizzazione**
  - Essa è il processo attraverso il quale le persone partendo da una esperienza specifica la decontestualizzano traendone un significato generale.
  - Di solito si utilizzano quantificatori universali (sempre, tutto, ...) e operatori modali (devo, posso, voglio)
  - Per trovarle chiediamoci “sul serio tutti?” ”davvero mai?”

- **Cancellazione**
  - Essa è un processo di selezione dell'esperienza. Le persone, infatti, prestano attenzione solo ad alcuni pezzi del proprio vissuto escludendone altri.
  - Può essere cancellazione dell’indice referenziale (non è espresso l’attore dell’azione) o comparativo mancante (quando non c’è il termine di paragone)
  - Per trovarle chiediamoci “chi?” ”quando?”

- **Deformazione**
  - Essa è una percezione "distorta" della realtà. A volte utilizziamo un singolo episodio per distorcere totalmente una situazione che, invece, può essere interpretata e vissuta in modi diversi.
  - Per trovarle chiediamoci “davvero...?”

Negoziazione

E’ una fase di modifica alla lista dei requisiti e loro divisione in classi.

Classi come:

- Must Have (Requisiti obbligatori - indispensabili)
- Should Have (Requisiti desiderabili - utili)
- Could Have (Requisiti opzionali - se c’è tempo)
- Want to Have (Requisiti postponibili - per successive versioni del prodotto)

Gestione

DIvisa in 3 passi:

- **Identificazione** dei requisiti

- **Attributi** dei requisiti (esempio di attributi)
  - Stato (proposto, approvato, rifiutato, incorporato)

    - Vantaggi (importanza)
    - Sforzo in gg/uomo
    - Rischio (valutazione fattibilità)
    - Stabilità
    - Versione destinazione

- **Tracciabilità**
    - Mettere a confronto i requisiti con componenti del sistema, moduli e test