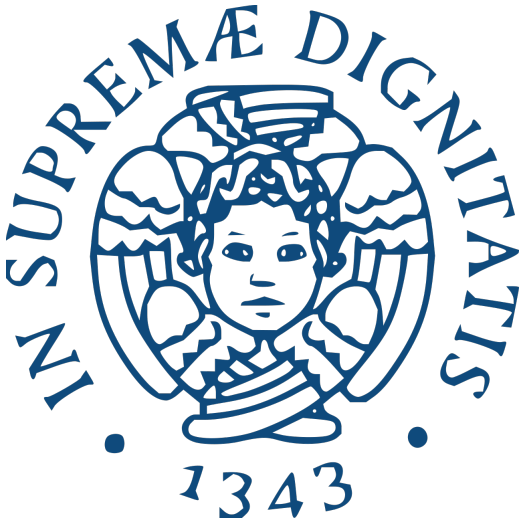
Ingegneria del Software

Angelo Passarelli

September 22, 2023



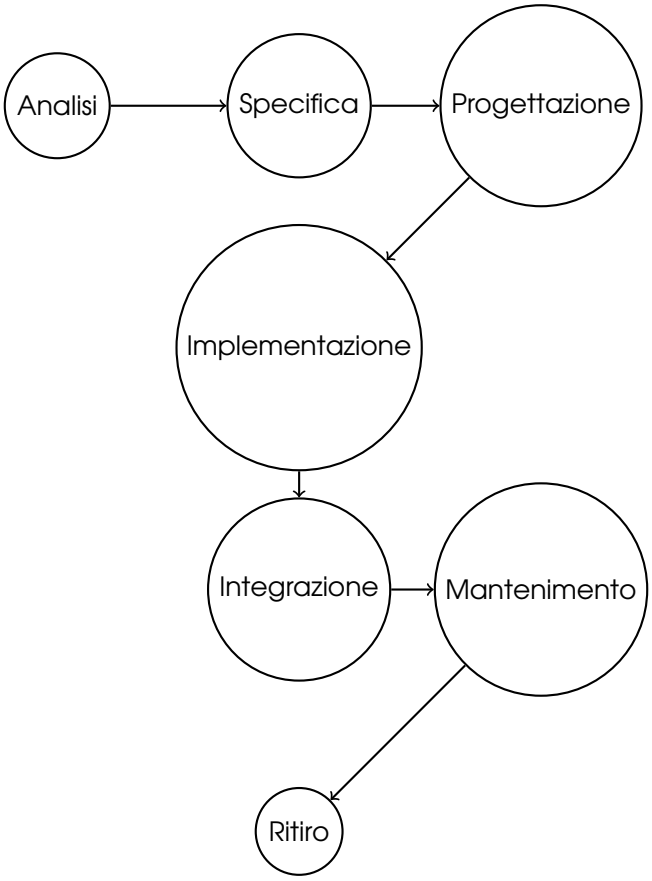
Appunti basati sulle lezioni e dispense della professoressa Laura Semini [1]

[1] http://didawiki.cli.di.unipi.it/doku.php/informatica/is-a/start

Contenuti

0 Introduzione

Fasi del progetto



Specificità del Software

1. **Fault toulerance**: capacità del software di essere tollerante ai guasti.
2. **Difetto latente**: difetto nascosto che si trova difficilmente in fase di testing; e anche nel caso comparisse è quasi impossibile da ritrovare.
3. **Robustezza**: capacità di funzionare anche con input non previsti e/o non testati.
4. Il software non presenta costi materiali e nemmeno costi marginali, ovvero il costo di un'unità del prodotto.

5. Infine il software non si consuma nel tempo, ma potrebbe diventare obsoleto.

La Manutenzione

Costi La fase di manutenzione è quella che richiede costi più alti. Per evitare uno spreco durante questa fase è necessario studiare bene l'analisi dei requisiti, in quanto un errore in questa fase si propagherà in modo esponenziale, in termini di costi, nelle fasi successive.

La manutenzione si divide in:

- **Manutenzione Correttiva**: rimuove gli errori, lasciando invariata la specifica.
- **Manutenzione Migliorativa**: consiste nel cambiare quella che è la specifica, e a sua volta può dividersi in:
  - Perfettiva: modifiche per migliorare e/o introdurre nuove funzionalità.
  - Adattiva: modifiche indotta da cambiamenti esterni, come leggi o modifiche all'hardware o al sistema operativo.

Stakeholders

- Fornitore: colui che sviluppa il software.
- Committente: chi lo richiede e paga.
- Utente: chi lo usa.

1 Modelli di Ciclo di Vita

Definizione (Processo Software)**.** Con processo software si indica il percorso da seguire per sviluppare un prodotto o più nello specifico un software. Fanno parte del processo sia gli strumenti e le tecniche per lo sviluppo che i professionisti coinvolti.

1.1 Modelli Sequenziali

1.1.1 Build-and-Fix

Il prodotto è sviluppato senza alcuna fase di progettazione preliminare, lo sviluppatore scrive il software e poi lo modifica ogni volta che non soffisfa il committente.

Contro Diventa improponibile per progetti grandi e la manutenzione diventa difficile senza documentazione nè specifica.

1.1.2 Modello a Cascata

Questo modello è stato il primo a distingure il processo software in più fasi, evidenziando l'importanza della progettazione e dell'analisi.

Viene chiamato anche modello *document driven* dato che ogni fase produce un documento, e per passare alla successiva occorre aver approvato il documento della fase precedente.

Contro Troppo pesante da seguire, inoltre non si può tornare indietro, e mancando l'interazione con il cliente, se non è soddisfatto, và tutto ripetuto dall'inizio.

1.1.3 Modello a V

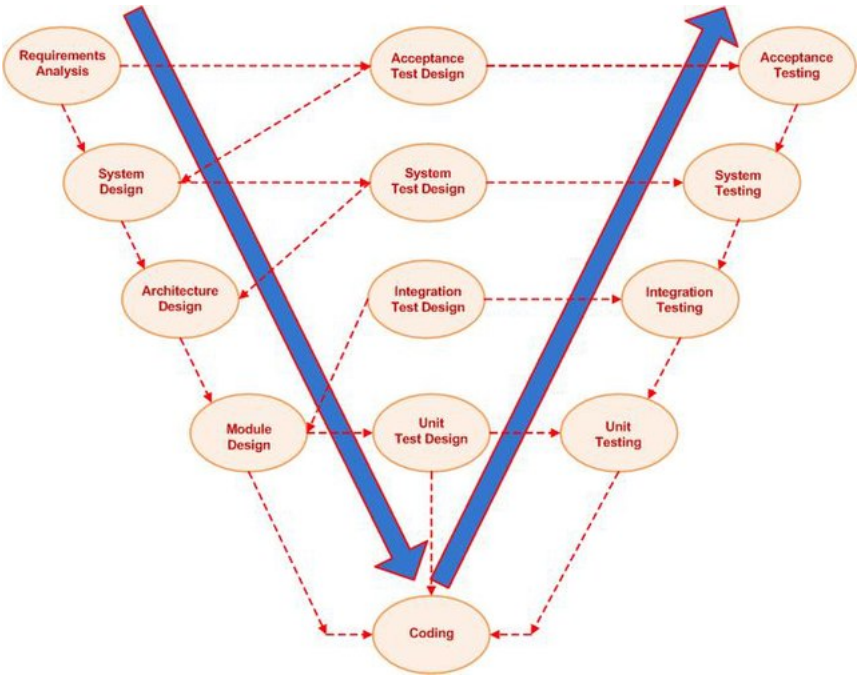


Figure 1: Le frecce blu rappresentano il *tempo*, mentre quelle tratteggiate le *dipendenze*

Questo modello evidenzia come sia possibile progettare i test durante le fasi di sviluppo (quelle a sinistra, prima della fase di *coding*). Mentre sulla destra sono presenti i test veri e propri che devono verificare e convalidare l'attività in corrispondenza sulla sinistra.

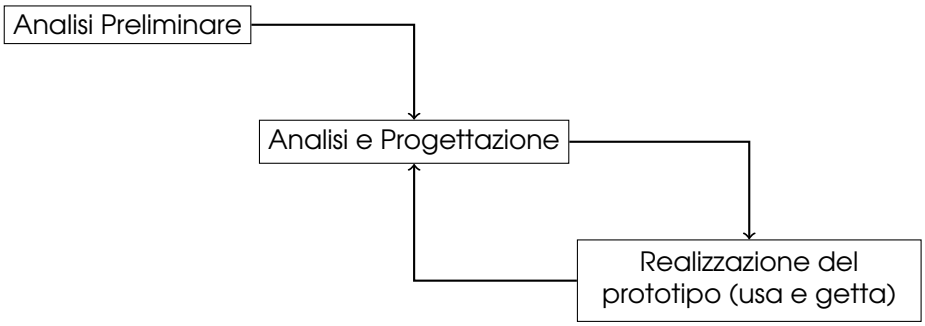
Standard SQA Questo modello è uno degli standard *SQA* (Software Quality Assurance), usato per descrivere le attività di test durante il processo di sviluppo.

1.2 Modelli Iterativi

1.2.1 Rapid Prototyping

L'obbiettivo è quello di costruire rapidamente un prototipo del software per permettere al committente di sperimentarlo.

Questo modello diventa utile quando i requisiti non sono chiari, quindi ogni prototipo aiuterà il cliente a descriverli meglio.

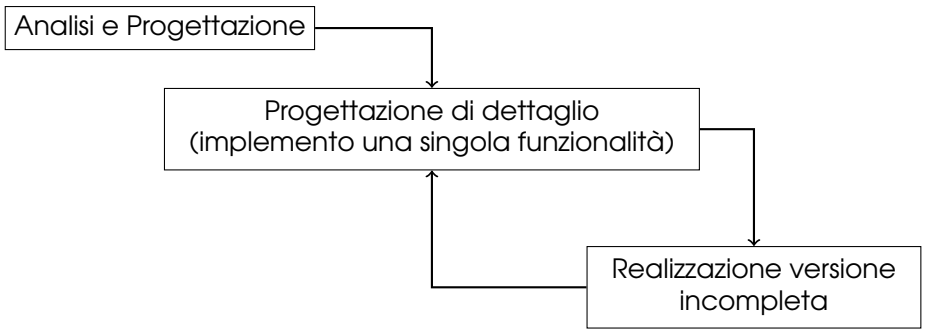


1.2.2 Modello Incrementale

Il software viene costruito in modo iterativo, aggiungendo di volta in volta nuove funzionalità.

I requisiti e la progettazione vengono definiti inizialmente, per questo è possibile applicarlo solo in caso di requisiti stabili.

Contro Se non viene realizzata una buona progettazione, questo modello sfocia in un *Build-and-Fix*.

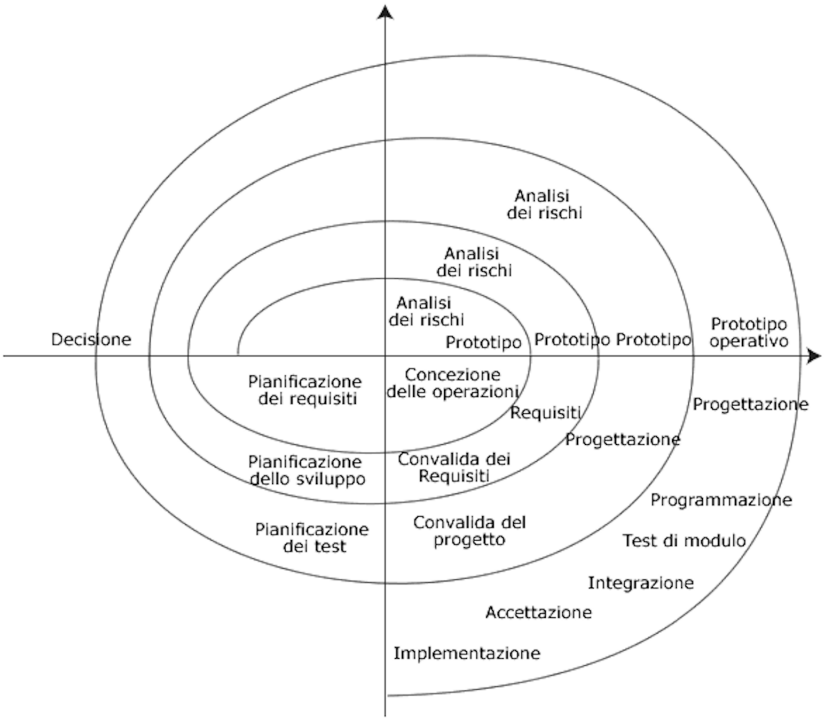

Analisi e Progettazione
Progettazione di dettaglio
(implemento una singola funzionalità)
Realizzazione versione
incompleta

1.2.3 Modello a Spirale

In questo caso ogni iterazione è formata da 4 fasi che corrispondono ai quadranti del piano:

1. *Quadrante in alto a sinistra*: definizione degli obiettivi e dei vincoli.
2. *Quadrante in alto a destra*: analisi e risoluzione dei rischi.
3. *Quadrante in basso a destra*: sviluppo e verifica del prossimo livello.
4. *Quadrante in basso a sinistra*: pianificazione della fase successiva.

Questo modello viene anche chiamato *risk driven* in quanto è incentrato principalmente sull'analisi dei rischi. Inoltre si ispira profondamente al metodo iterazivo *plan-do-check-act cycle* [2]



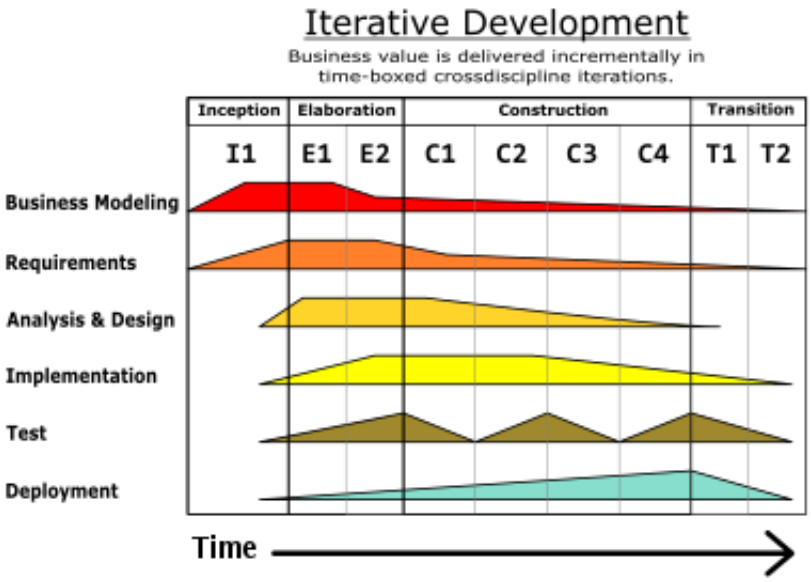
[2] https://it.wikipedia.org/wiki/Ciclo_di_Deming

1.3 Unified Process

In questo modello vengono distinte quattro fasi chiamate *Inception*, *Elaboration*, *Construction* e *Transition*. Ogni fase può presentare un numero variabile di iterazioni anche in base alla dimensione del progetto.

Questo modello viene definito iterativo incrementale, *incrementale* perchè alla fine di ogni iterazione si ottiene un rilascio del sistema con funzionalità in più o migliorate rispetto al rilascio precedente.

Inoltre viene data molta importanza all'architettura del sistema, infatti già dalle prime fasi ci si concentra soprattutto sull'architettura anche se a livello molto superficiale, lasciando i dettagli alle fasi successive. In questo modo è molto facile avere una visione generale del sistema che sarà facilmente modellabile sulla variazione dei requisiti. Per questo, piuttosto che dai requisiti, ci si fà guidare principalmente dai *casi d'uso* e dall'*analisi dei rischi*.



1.4 Processi Agili

Definizione (Metodo Agile)**.** Con metodo agile si intende un metodo per lo sviluppo del software che si basa principalmente sul coinvolgimento del committente. Questa metodologia si riferisce ai principi del *Manifesto di Snowbird* del 2001.

I concetti chiave di questi processi sono:

- Continuous Integration: rendere il più automatico possibile la consegna e l'integrazione dei singoli moduli.
- Continuous Delivery: rilascio frequente e supportato delle nuovi versioni del software.
- DevOps: *Development* e *Operations*, ovvero maggiore collaborazione tra sviluppatori e responsabili della manutenzione, della sicurezza e dell'infrastruttura dell'azienda.

1.4.1 Il Manifesto di Snowbird

Il *Manifesto di Snowbird* si fonda su quattro punti fondamentali:

1. Comunicazione: la comunicazione fra tutti gli attori del progetto è centrale, soprattutto le interazioni e la collaborazione con i clienti.
2. Semplicità: si mantiene il codice sorgente il più semplice possibile, ma comunque avanzato tecnicamente, in questo modo si riduce la documentazione al minimo indispensabile.
3. Feedback: sin dal primo giorno di sviluppo il codice viene testato, in modo da poter rilasciare versioni ad intervalli molto frequenti.
4. Coraggio: dare in uso il sistema il prima possibile ed implementare i cambiamenti richiesti man mano.

Di seguito sono riportati due modelli che si basano sui *processi agili*.

1.4.2 eXtreme Programming

Si basa su un insieme di consuetudini:

- *Pianificazione flessibile*: è basata su un insieme di scenari proposti dagli utenti e i programmatori vengono coinvolti direttamente.
- *Rilasci frequenti*: più o meno ogni 2-4 settimane, e alla fine si ricomincia con una nuova pianificazione.
- *Progetti semplici*: comprensibili a tutti.
- *Testing*: test basati sui singoli scenari e con supporto automatico.
- *Test Driven Development*: i casi di test vengono definiti prima della scrittura del codice.
- *Cliente sempre a disposizione*
- *Programmazione a coppie*: viene usato un solo terminale, una persona svolge il ruolo di *driver* che scrive il codice, mentre un'altra fà il *navigatore*, ovvero controlla il lavoro del *driver* attivamente.

- *No al lavoro straordinario*
- *Collettivizzazione del codice*: accesso libero e continua integrazione.
- *Code Refactoring*: modificare il codice senza cambiare il suo comportamento e commentarlo il più possibile.
- *Daily Stand Up Meeting*

1.4.3 SCRUM

Definizione (SCRUM)**.** Con *SCRUM* si intende un processo *iterativo* ed *incrementale*, dove alla fine di ogni iterazione vengono rilasciate un insieme di funzionalità potenzialmente rilasciabili.

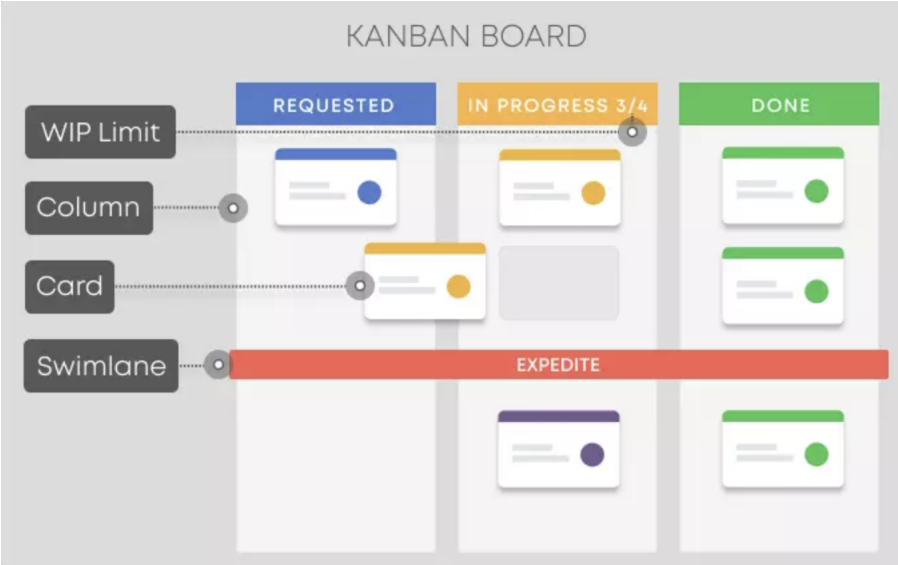
Il processo è diviso in tre fasi:

1. **Pre-game phase**:
   (a) Planning sub-phase: viene creata una *Product Backlog List* che contiene tutti i requisiti conosciuti.
   (b) Architecture sub-phase: viene già pianificato il design di alto livello e l'architettura del sistema.
2. **Development phase**: in questa fase il sistema viene sviluppato attraverso una serie di *Sprint*, ovvero cicli iterativi nei quali vengono sviluppate o migliorate una serie di funzionalità, e ogni sprint può durare circa 1-4 settimane. Lo *Sprint* ovviamente include le classiche fasi di sviluppo del software.
3. **Post-game phase**: il prodotto viene preparato per il rilascio, ovvero si prepara l'*integrazione*, i *test*, la *documentazione* per l'utente e la preparazione del materiale di *marketing*.

I ruoli principali durante l'esecuzione di un processo *SCRUM* sono tre:

- **Product Owner**: ci si riferisce a quella persona responsabile di accettare o rifiutare i risultati di un lavoro e di poter terminare uno *Sprint*, inoltre fà da raccordo fra tutti soggetti interessati nel progetto.
- **Membri del Team**: i membri decidono cosa fare in ogni *Sprint*, ogni team è indipendente e i membri non fanno capo ad alcun project manager. Ogni membro ha diverse specializzazioni (*cross-functional*), in modo tale da non avere persone con troppo carico di lavoro e ognuno si occupa di un singolo lavoro alla volta.
- **Scrum Master**: non ha alcuna autorità sul team, ma si occupa di supportarlo e motivarlo, garantendo anche le condizioni ambientali per lavorare al meglio.

Kanban Board Questa lavagna permette di gestire al meglio il flusso del lavoro. Come mostrato in figura è presente un *Work In Progress Limit* che definisce un limite alla quantità di post-it che possono essere presenti in ogni colonna. Questo limite permette di completare più velocemente i singoli lavori, in modo tale di dare qualcosa al cliente il prima possibile e di individuare facilmente i colli di bottiglia che possono rallentare gli altri lavori. Inoltre permette di ridurre il *task switching*, ovvero il lavoro su più task contemporaneamente.



Gli eventi che fanno parte di uno *Sprint* sono i seguenti:

1. Sprint planning: il *product owner* gestisce l'evento di pianificazione dello *Sprint*.
2. Daily meeting: i membri del team e gli SCRUM master si ritrovano davanti la *kanban* e discutono delle difficultà che hanno riscontrato.
3. Review: alla fine di di una modifica concreta al software, questo viene ispezionato in collaborazione con gli utenti per ottenere un feedback e per discutere su cambiamenti o nuove idee.
4. Retrospettiva: questa fase permette di riflettere, studiare e adattarsi per lo *Sprint* successivo.