UNIVERSITÀ DEGLI STUDI
DI GENOVA

Università degli Studi di Genova

Fondamenti di Ingegneria del Software

Lorenzo Vaccarecci

2024-2025

Indice

Capitolo 1

Modelli di processo di sviluppo software

1.1 Introduzione

Processo: insieme strutturato e organizzato di attività che si svolgono per ottenere un risultato.

Perchè modellare il processo? Per dare ordine, controllo e ripetibilità con l'intenzione di migliorare la produttività e la qualità del prodotto.

1.1.1 Processo prescrittivo e adattivo

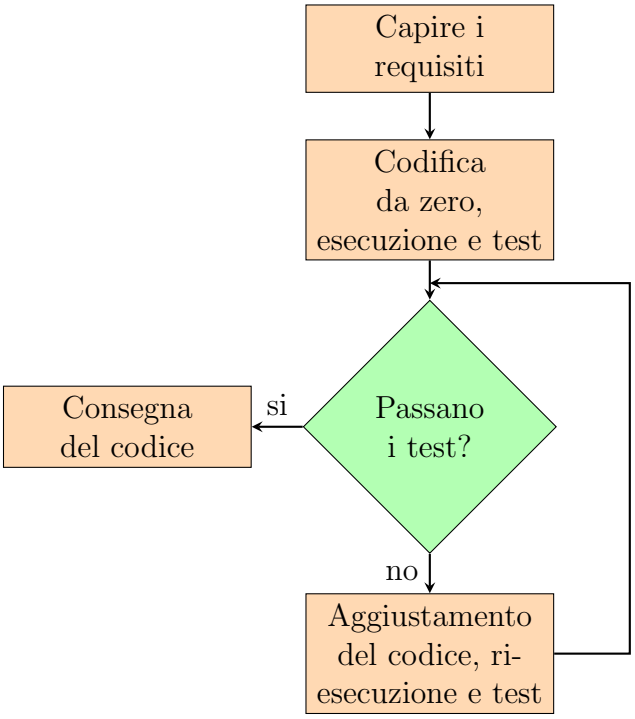
- **Processo prescrittivo**: un processo che segue un modello predefinito e rigido, con passaggi specifici e ben definiti.
- **Processo adattivo**: un processo che permette modifiche e adattamenti durante il suo svolgimento.

Perchè studiare i modelli di processo? Perchè uno dei compiti dei manager aziendali è quello di decidere il modello di processo da adottare considerando la tipologia del software da progettare e il personale disponibile.

1.2 Modelli di processo

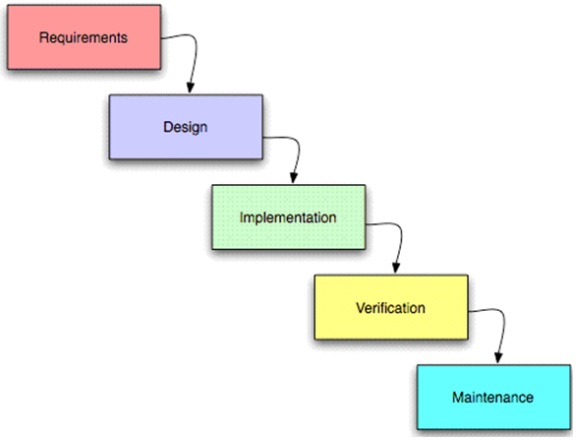
1.3 Code and Fix

- Si arriva al codice finale "per tentativi"
- Non adatto per progetti grandi con tanti sviluppatori
- Non è un modello di processo vero e proprio



1.4 Modello a cascata

- Storicamente il primo modello del processo di sviluppo software
- Ogni fase produce un prodotto che è l'input della fase successiva
- Con il modello waterfall abbiamo il passaggio dalla dimensione artigianale alla produzione industriale del software
- Molto rigido: non si può tornare indietro



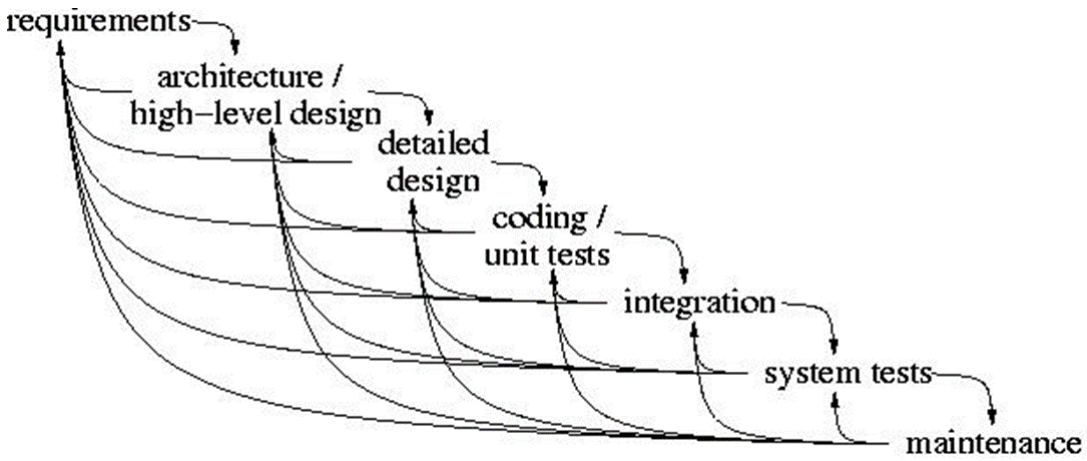
Vantaggi	Svantaggi
Enfasi su aspetti come l'analisi dei requisiti e il progetto di sistema trascurati nell'approccio code & fix	Lineare, rigido, monolitico: no feedback tra fasi, no parallelismo, unica data di consegna
Pospone l'implementazione dopo avere capito i bisogni del cliente	La consegna avviene dopo anni, intanto i requisiti cambiano o si chiariscono: così viene consegnato software obsoleto
Introduce disciplina e pianificazione	Viene prodotta troppa documentazione poco chiara: l'utente spesso non conosce tutti i requisiti all'inizio dello sviluppo
E' applicabile se i requisiti sono chiari e stabili	Alcuni difetti superati da modello waterfall con feedback e iterazioni

1.4.1 Studio di fattibilità

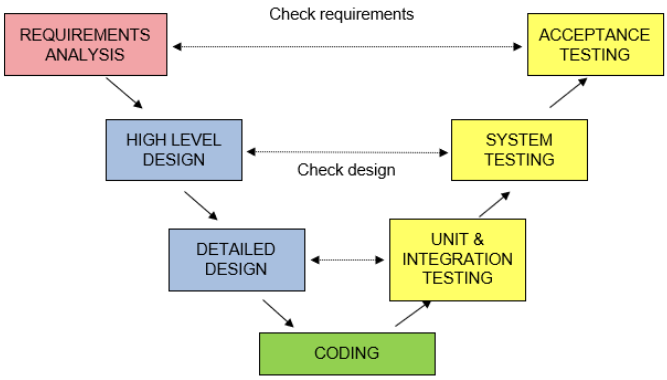
- Fase che precede lo sviluppo vero e proprio
- Viene analizzata la fattiblità e convenienza del progetto
- Stima dei costi
- Si valuta il Return Of Investment (ROI)

1.4.2 Varianti del modello a cascata

- Cascata con prototipazione: prima di iniziare lo sviluppo si costruisce un prototipo "usa e getta" con il solo scopo di fornire agli utenti una base concreta per meglio definire i requisiti.
- Cascata con feedback e iterazioni: posso tornare a una fase precedente.



- V-Model:
  - Enfasi sulle fasi di testing
  - Evidenzia come le attività di testing (parte destra della V) sono collegate a quelle di analisi e progettazione (parte sinistra della V)
  - Ogni controllo fatto a destra che non dia buon esito porta a un rifacimento/modifica di quanto fatto a sinistra
  - **Parallelismo**: creazione dei test e una volta che ho il codice li eseguo
  - Problemi (anche per Waterfall):
    * Versione funzionante solo alla fine!
    * Errore in fase iniziale può avere conseguenze disastrose



1.5 Modelli evolutivi

Idea: sviluppare un implementazione iniziale, esporla agli utenti e raffinarla attraverso successivi rilasci del SW (release)

Sottocategorie:

- Prototyping
- Modelli incrementali
- Modelli iterativi

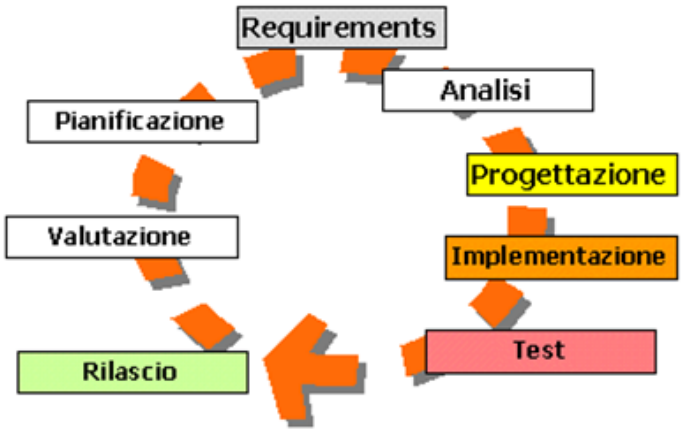
1.5.1 Modelli a Prototyping

- Realizzazione di un prototipo funzionante del sistema, su cui validare i requisiti (o l'architettura)
- Il prototipo ha meno funzionalità ed è meno efficiente

Vantaggi	Svantaggi
Permette di raffinare requisiti definiti in termini di obiettivi generali e troppo vaghi	Il prototipo è un meccanismo per identificare i requisiti, spesso da "buttare": problema economico e psicologico, il rischio è di non farlo e così scelte non ideali diventano parte integrante del sistema
Rilevazione precoce di errori di interpretazione	

1.5.2 Modelli Iterativi-Incrementali

- Sviluppo di varie release, di cui solo l'ultima è completa
- Dopo la prima release, si procede in parallelo
- Le fasi di sviluppo vengono percorse più volte



Modelli Incrementali

- Ogni release aggiunge nuove funzionalità
- Nella fase di pianificazione si decide il requisito/funzionalità da includere nella release successiva.
- Si trattano per prime le funzionalità ad alto rischio
- Si cerca di massimizzare il valore per gli utenti

Modelli Iterativi

- Da subito sono presenti tutte (o buona parte) delle funzionalità che sono via via raffinate, migliorate

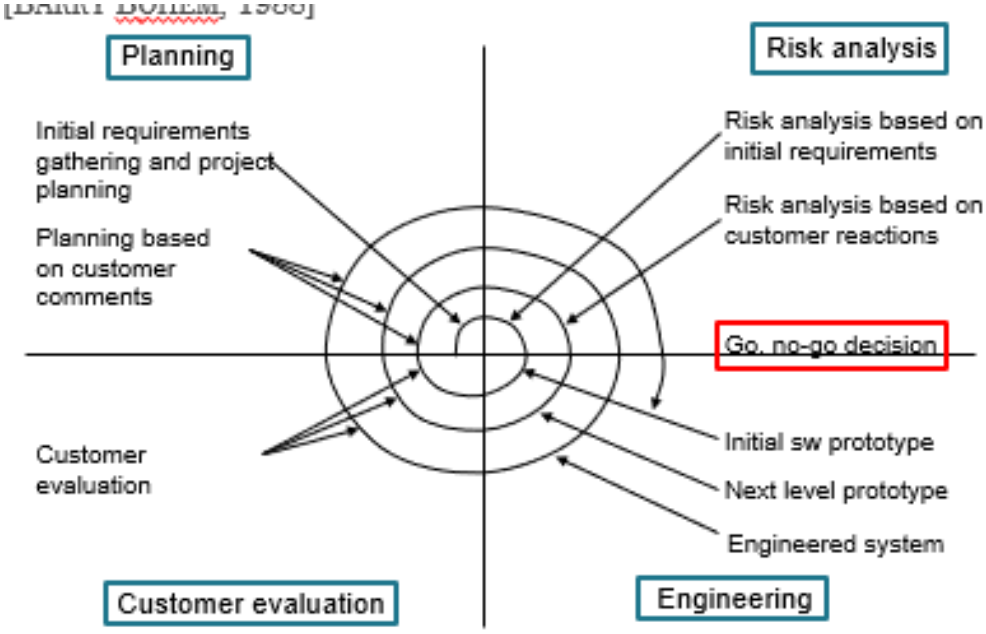
1.6 Modello a spirale

- Sistemi di grandi dimensioni
- Approccio "evolutivo" con interazioni continue fra cliente e developer
- Modello "risk-driver": tutte le scelte sono basate sui risultati dell'analisi dei rischi
- 'Meta-modello': dà un'idea generale ma quando si inizia a lavorare bisogna scegliere un modello esistente
  - Requisiti chiari e stabili → modello a cascata
  - Requisiti confusi → prototipo

Rischio: circostanza potenzialmente avversa in grado di pregiudicare lo sviluppo e la qualità del software

Ogni scelta/decisione ha un rischio associato, due caratteristiche importanti nella valutazione di un rischio sono:

- Gravità delle conseguenze
- Probabilità che si verifichi la circostanza



- **Planning**: determinazione di obbiettivi, alternative, vincoli
- **Risk Analysis**: analisi delle alternative e identificazione/risoluzione dei rischi
- **Engineering**: sviluppo del prodotto di successivo livello
- **Customer Evaluation**: valutazione dei risultati dell'engineering dal punto di vista del cliente

Vantaggi	Svantaggi
Adatto allo sviluppo di sistemi complessi	Non è un rimedio universale (panacea)
Primo approccio che considera il rischio (risk-driver)	Necessita competenze di alto livello per la stima dei rischi
	Richiede un'opportuna personalizzazione ed esperienza di utilizzo
	Se un rischio rilevante non viene scoperto o tenuto a bada si inizia da zero

1.7 Unified Process

- Specifico per sistemi ad oggetti, con uso di notazione UML per tutto il processo
- Guidato dagli **Use Case**
- Incorpora molte delle idee 'buone' dal modello a spirale
- Meta-modello
- Supportato da tool(visuali) in ogni fase
- Processo prescrittivo per eccellenza

1.7.1 Le iterazioni

- Possibili diverse iterazioni che terminano con il rilascio del prodotto
- Ogni iterazione consiste di quattro fasi (anche ripetute più volte) che terminano con una milestone (= rilascio di artefatti soggetti a controllo)
- Ogni fase è costituita da diverse attività:
  - Requisiti (R)
  - Analisi (A)
  - Design (D)
  - Codifica (C)
  - Testing (T)

1.7.2 Le fasi

- Inception: studio di fattibilità, requisiti essenziali del sistema, risk analysis
- Elaboration: sviluppa la comprensione del dominio e del problema, gli Use Case della release da rilasciare, l'architettura del sistema
- Construction: Design (in UML), codifica e testing del Sistema
- Transition: Messa in esercizio della release nel suo ambiente (deploy), training e testing da parte di utenti fidati

1.8 Sviluppo basato sui componenti

Modello che va nella direzione del **riutilizzo del software**

Vantaggi	Svantaggi
Riduce la quantità di software da scrivere	Sono necessari dei compromessi: requisiti iniziali potrebbero differire da quelli che si possono soddisfare con le componenti disponibili
Riduce i costi totali di sviluppo e i rischi	Integrazione non sempre facile
Consegne più veloci	Spesso i componenti usati sono fatti evolvere dalla ditta produttrice senza controllo di chi li usa

1.9 Metodi Plan-Driven e Agili

Plan-Driven	Agile
Seguono un approccio classico dell'ingegneria dei sistemi fondato su processi ben definiti e ocn passi standard	Rispondere ai cambiamenti dei requisiti in modo veloce
	Filosofia del programmare come "arte" piuttosto che processo industriale
	Cosa più importante soddisfare il cliente e non seguire un piano (contratto)

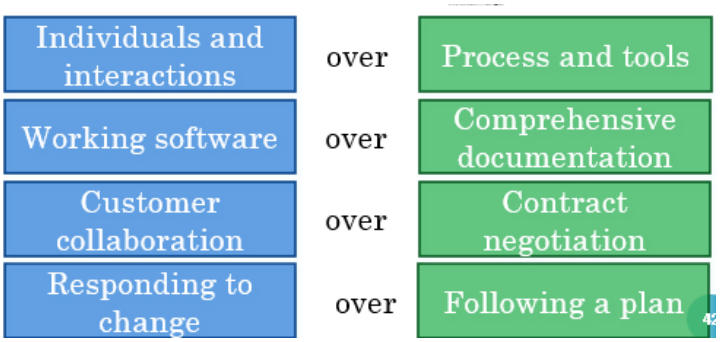


Figura 1.1: The Agile Manifesto

1.9.1 Come scegliere?

Metodi plan-driven:

- Sistemi grandi e comploessi, safety-critical o con forti richieste di affidabilità
- Requisiti stabili e ambiente predicibile

Metodi agili:

- Sistemi e team piccoli, clienti e utenti disponibili, ambiente e requisiti volatili
- Team con molta esperienza
- Tempi di consegna rapidi

1.10 DevOps

Metodo di sviluppo evolutivo

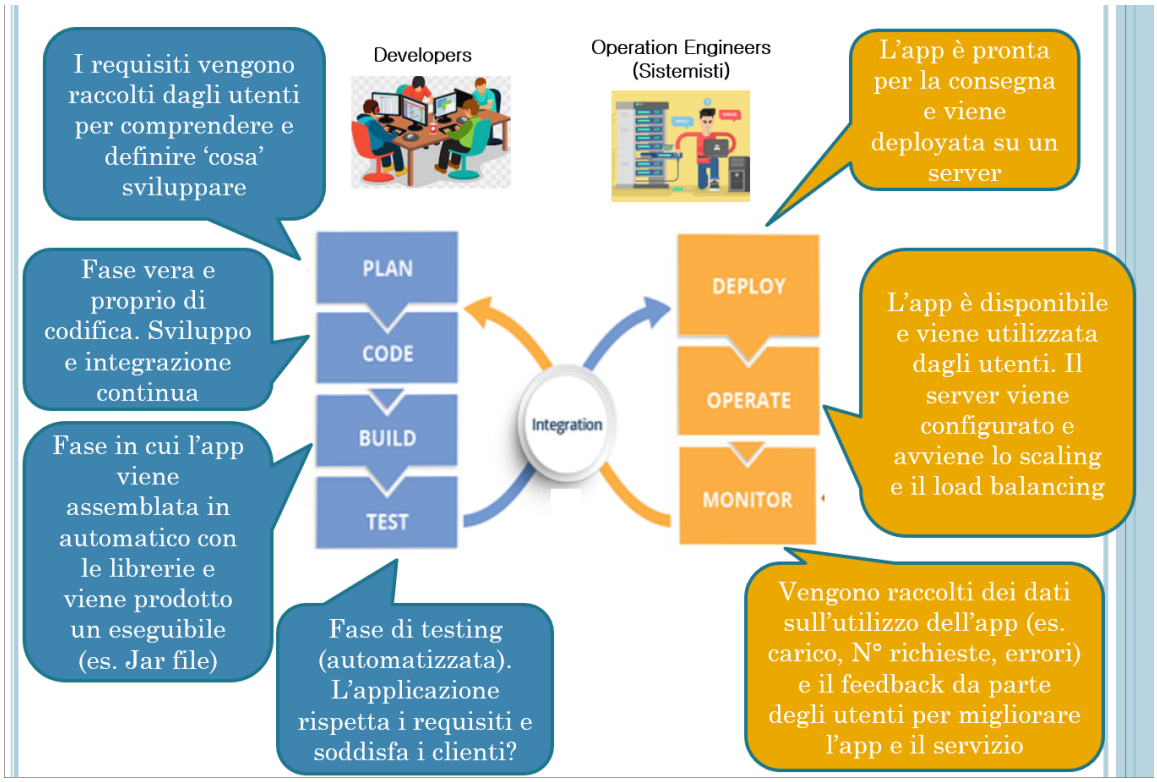


Figura 1.2: DevOps

1.10.1 Continuous Integration

La Continuos Integration (CI), o Integrazione Continua, è una pratica di sviluppo software in cui i programmatori integrano frequentemente il proprio lavoro (codice) nel repository condiviso del progetto, in genere diverse volte al giorno.