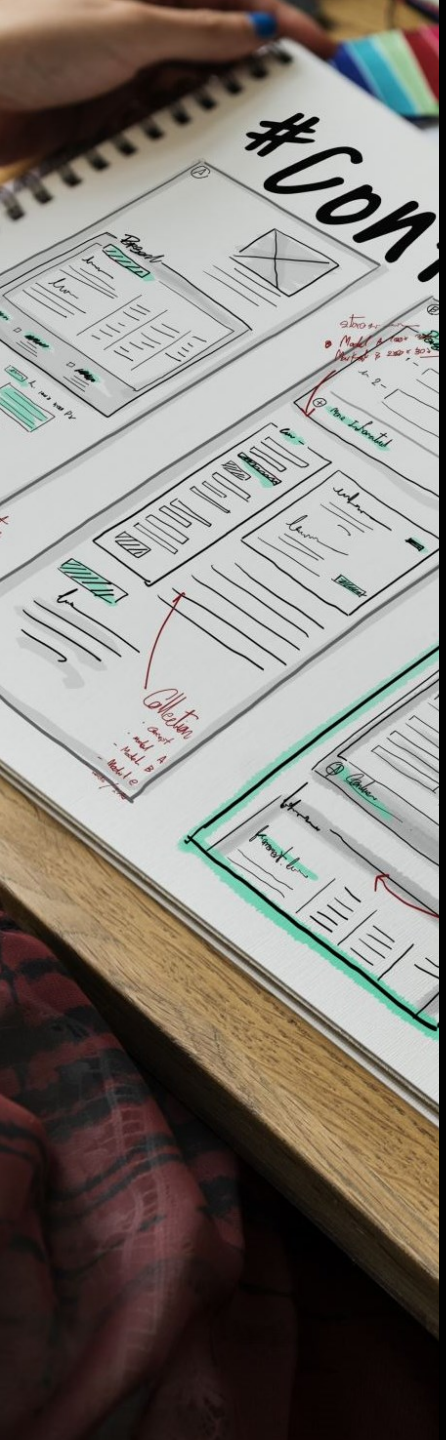


Sviluppo del Software Sicuro

Vincenzo Calabrò

Fasi di sviluppo del SW
e modelli di ciclo di vita del SW

Definizioni

Processo di produzione

- La sequenza di attività svolte per progettare, realizzare, consegnare e modificare un prodotto SW

Ciclo di vita di un software

- L'insieme di stadi in cui il sistema viene a trovarsi

Modelli di processo

- Determinano l'organizzazione delle diverse fasi di sviluppo

Fasi di sviluppo

Le attività del processo sono organizzate in

- Studio di fattibilità
- Specifica
- Progettazione (o design)
- Implementazione e test dei moduli
- Integrazione e Test del sistema
- Consegna
- Manutenzione

Studio di fattibilità

Scopo di questa fase è la produzione di un documento detto *Feasibility Study Document* (FSD) che valuti i costi e i benefici della applicazione proposta

L' FSD deve contenere

- La definizione del problema
- Le possibili soluzioni e le relative motivazioni
- Per ognuna delle soluzioni proposte, la stima dei benefici, dei costi, delle risorse richieste e dei tempi di consegna

Specifica (o analisi dei requisiti) (1)

Ha lo scopo di determinare le funzionalità e le proprietà del SW in termini di performance, facilità d'uso, portabilità, facilità di manutenzione, ecc.

E' una fase cruciale del processo di sviluppo

La fase di specifica produce il RASD

- *Requirements Analysis and Specification Document*

Esempio di requisiti: il sistema LIBSYS

- Consideriamo un ipotetico sistema che fornisca una interfaccia unica ad un insieme di database di articoli e documenti.

- Gli utenti possono effettuare ricerche, acquistare, scaricare e stampare gli articoli per uso e studio personale.

(*da* Ian Sommerville, Software Enginnering. 7th ed., Addison Wesley. 2004, ISBN 0-321-21026-3)

Esempi di requisiti di LIBSYS

- L'utente deve essere in grado di effettuare ricerche su tutti i database disponibili o su un sottoinsieme di essi.
- Il sistema deve fornire all'utente dei *visualizzatori* appropriati per leggere i documenti disponibili.
- Per ogni ordine di acquisto deve esistere un identificatore unico (ORDER_ID) che l'utente deve poter salvare in modo permanente.

Esempio di requisiti di LIBSYS ambigui

Considerare i termini «*visualizzatori* appropriati»

- Nelle intenzioni degli utenti: *programmi special-purpose* in grado di leggere tipi specifici di documenti
- Interpretazione degli sviluppatori: un *unico visualizzatore di testo* che mostri il contenuto di qualsiasi tipo di documento

Specifica (o analisi dei requisiti) (2)

Cosa occorre

- Capire l'interfaccia tra l'applicazione e l'ambiente esterno
- Capire il dominio di applicazione
- Identificare i principali stakeholder e comprendere le loro aspettative
  - Diversi stakeholder possono avere punti di vista differenti
  - La specifica deve integrare e riconciliare i diversi punti di vista
- Stabilire chiaramente le qualità che devono essere raggiunte

Specifica (o analisi dei requisiti) (3)

La fase di specifica è un processo incrementale che richiede continua interazione con il cliente

5 W: una linea guida per una buona specifica

Dominio

- Who will use the system
  - Chi sono gli utenti interessati
- Why should it be developed + why will the users use it
  - Quali sono gli obiettivi e le aspettative degli utenti
  - Quali sono le entità del dominio e che influsso ha l'applicazione su di esse

Specifica (o analisi dei requisiti) (4)

Requisiti funzionali

- What (vs How) will it provide
  - Cosa deve fare l'applicazione
  - Descrivere ***cosa*** deve fare e non ***come*** farlo

Requisiti non funzionali

- Where will it be used, on which architecture
  - Descrivono affidabilità (disponibilità, integrità, sicurezza, ecc), prestazioni, interfaccia tra sistema ed utenti, limiti operativi, limiti fisici, ecc.

Requisiti del processo e manutenzione

- When and how long will it be used
  - Procedure per il controllo della qualità, priorità di sviluppo delle funzionalità, possibili cambiamenti, ecc.

Specifica (o analisi dei requisiti) (5)

RASD

- Deve permette al cliente di verificare le caratteristiche specificate
- Deve consentire al progettista di procedere allo sviluppo dell'architettura del SW
- Proprietà richieste:
  - facilmente comprensibile
  - preciso, completo, coerente, non ambiguo, tracciabile
  - modificabile
- Dovrebbe includere:
  - *user manual* preliminare
  - *system test plan*
    - Definizione delle modalità di test del sistema

Progettazione

Scopo di questa fase è la produzione di un documento contenente la descrizione dell' architettura del SW sia a livello *globale* che a livello dei *singoli* moduli

La fase di design produce il *Design Document*

- Definizione delle componenti (o moduli)
- Relazioni tra le componenti
- Interazione tra le componenti

Obbiettivi della progettazione

- Sviluppo concorrente e separazione delle responsabilità

Implementazione e test dei moduli

È la fase in cui i programmi vengono realizzati dai programmatori i quali si occupano di effettuare i test sulle funzionalità inerenti i singoli moduli

Per ogni componente si fornisce

- Codifica
- Documentazione
- Specifica dei test effettuati

Integrazione e test del sistema

Questa fase ha lo scopo di assemblare il codice prodotto dai diversi gruppi di programmatori e verificarne l'effettiva compatibilità risolvendo eventuali errori derivanti dall'interazione di due o più moduli

- Non sempre questa fase viene considerata concettualmente distinta dalla fase di implementazione

Consegna

In questa fase il sistema viene distribuito agli utenti che ne verificano il funzionamento, individuando eventuali malfunzionamenti o dissimilarità rispetto alle specifiche di progetto

La consegna avviene in due fasi:

- ***Beta testing***: il SW viene distribuito ad un insieme selezionato di utenti allo scopo di effettuare test in casi reali. Gli errori riscontrati vengono corretti prima della distribuzione effettiva
- ***Distribution***: il SW viene definitivamente rilasciato agli utenti. Gli errori che vengono riscontrati vengono solitamente corretti nelle versioni successive o tramite l'utilizzo di appositi software correttivi (*patch*)

Manutenzione

È la fase che include tutti i cambiamenti (correzione + evoluzione) che seguono la distribuzione del SW

Spesso comporta più del 50% del costo totale del SW

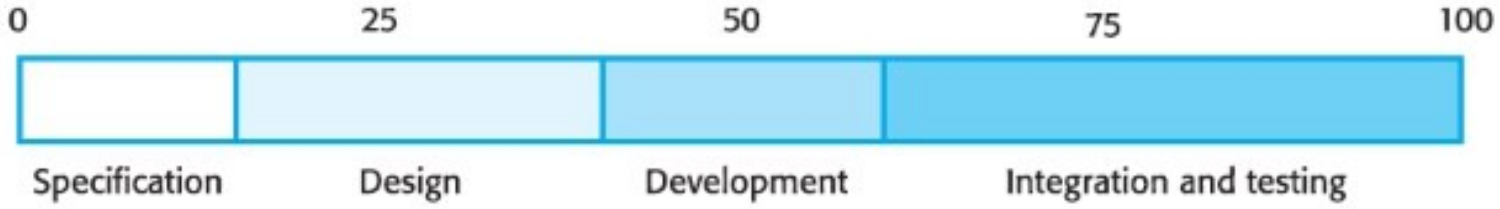
- Stima dei contributi della EU alle compagnie SW
  - 80% del budget impegnato per progetti IT speso per il mantenimento

Classificazione:

- manutenzione **correttiva** ≈ 20%
- manutenzione **adattativa** ≈ 20%
- manutenzione **perfettiva** ≈ 50%

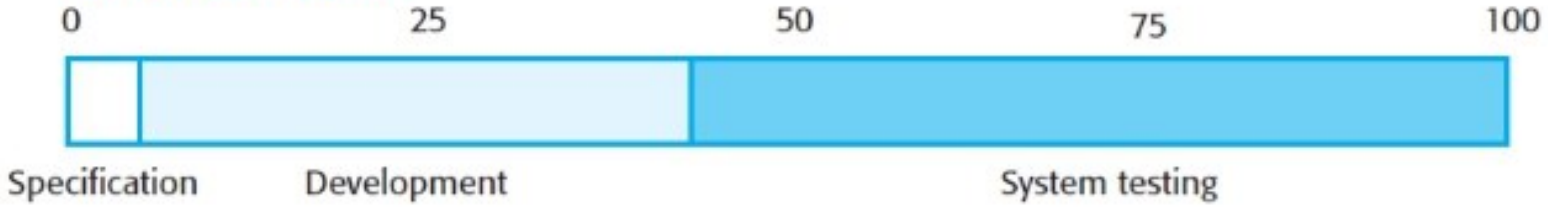
Costi di sviluppo

Waterfall model



Iterative development

Mediamente....



Distribuzione del carico di lavoro

- Analisi requisiti e specifica: meno del 5%
- Design e codifica: circa 30%
- Testing: oltre il 50% !!!!

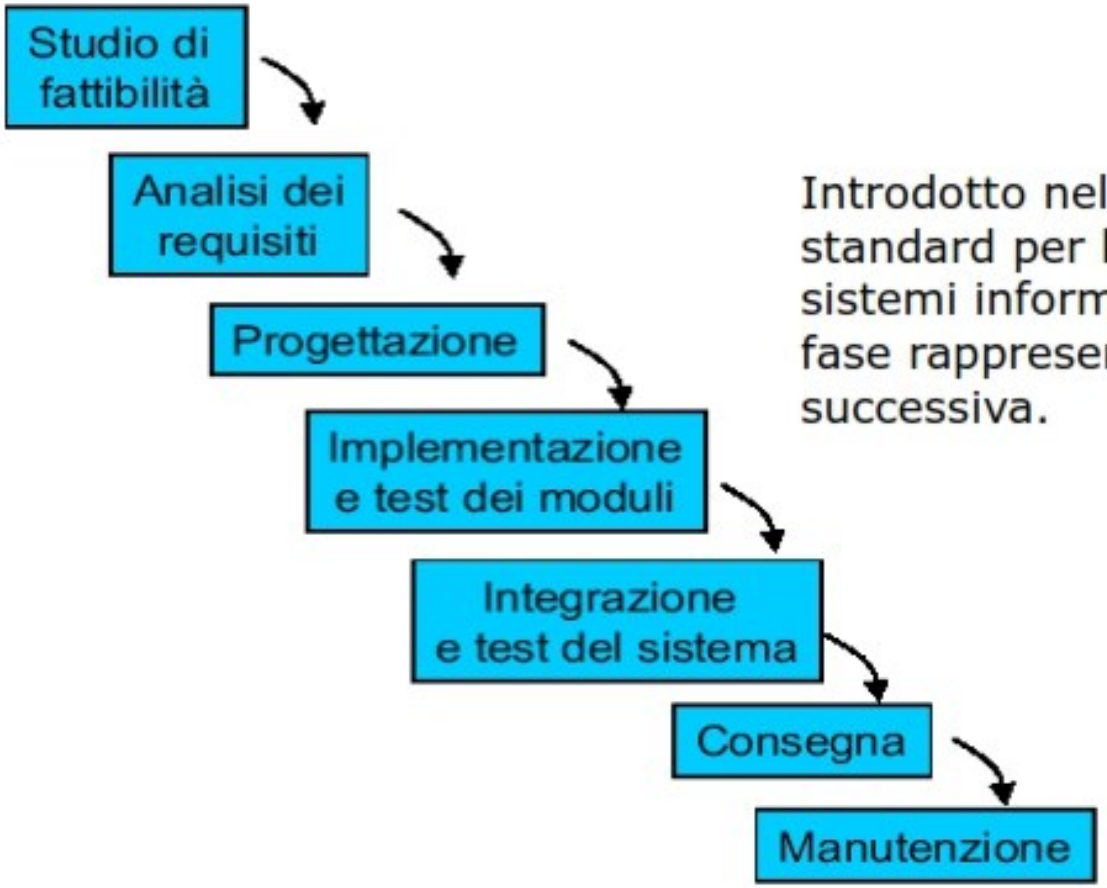
Modelli di processo

I cicli di vita differiscono principalmente nel modo e nell'ordine con cui le attività sono realizzate

I modelli di ciclo di vita sono classificati in

- Modelli a cascata
- Modelli evolutivi
- Modelli a spirale/V
- Modelli agili

Modello a cascata (waterfall)

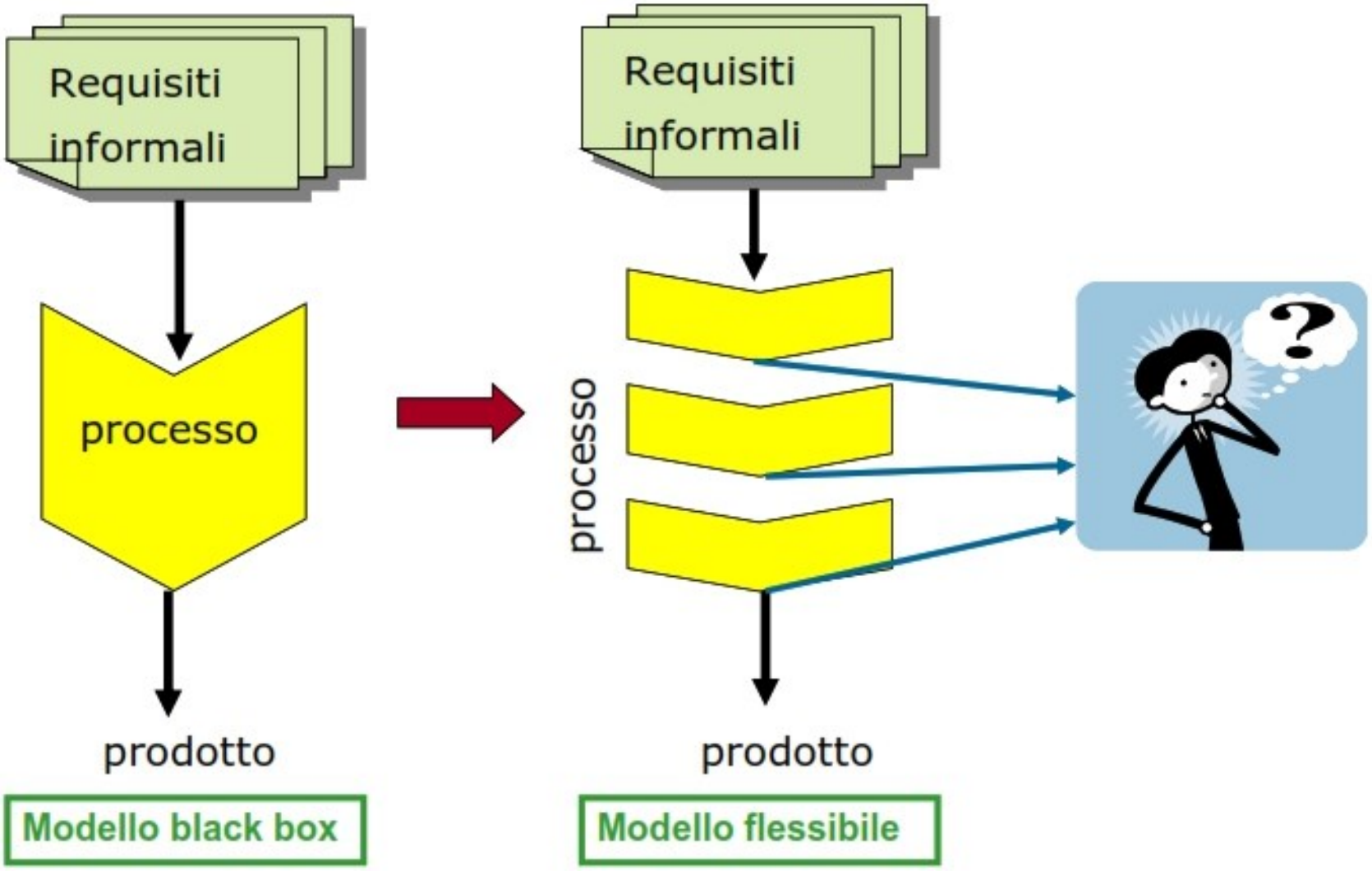


Introdotto nel 1970 rappresenta lo standard per la produzione di sistemi informatici. L'output di ogni fase rappresenta l'input della successiva.

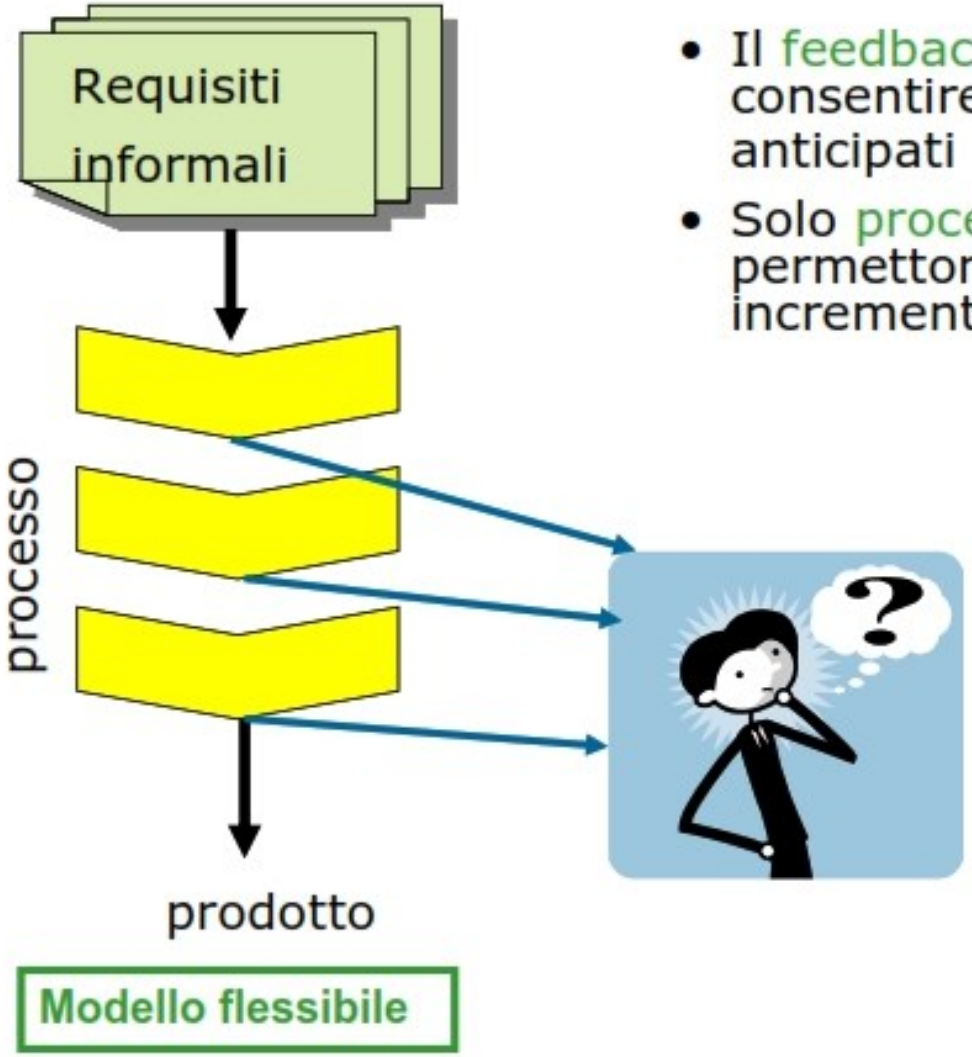
Modello a cascata: caratteristiche e limiti

- Ne esistono molte varianti e ciascuna organizzazione tende a definire il "proprio"
- Il modello implica il completamento di una fase prima di passare alla fase successiva
- È un processo black box e non consente prototipazione
- Non tiene in considerazione il fatto che i requisiti cambiano
  - Cambia il contesto
    - Hardware, dominio di applicazione, algoritmi
  - Specifiche sbagliate
    - Requisiti non catturati correttamente o incompleti
    - Scarsa conoscenza del dominio

Come affrontare l'evoluzione (1)



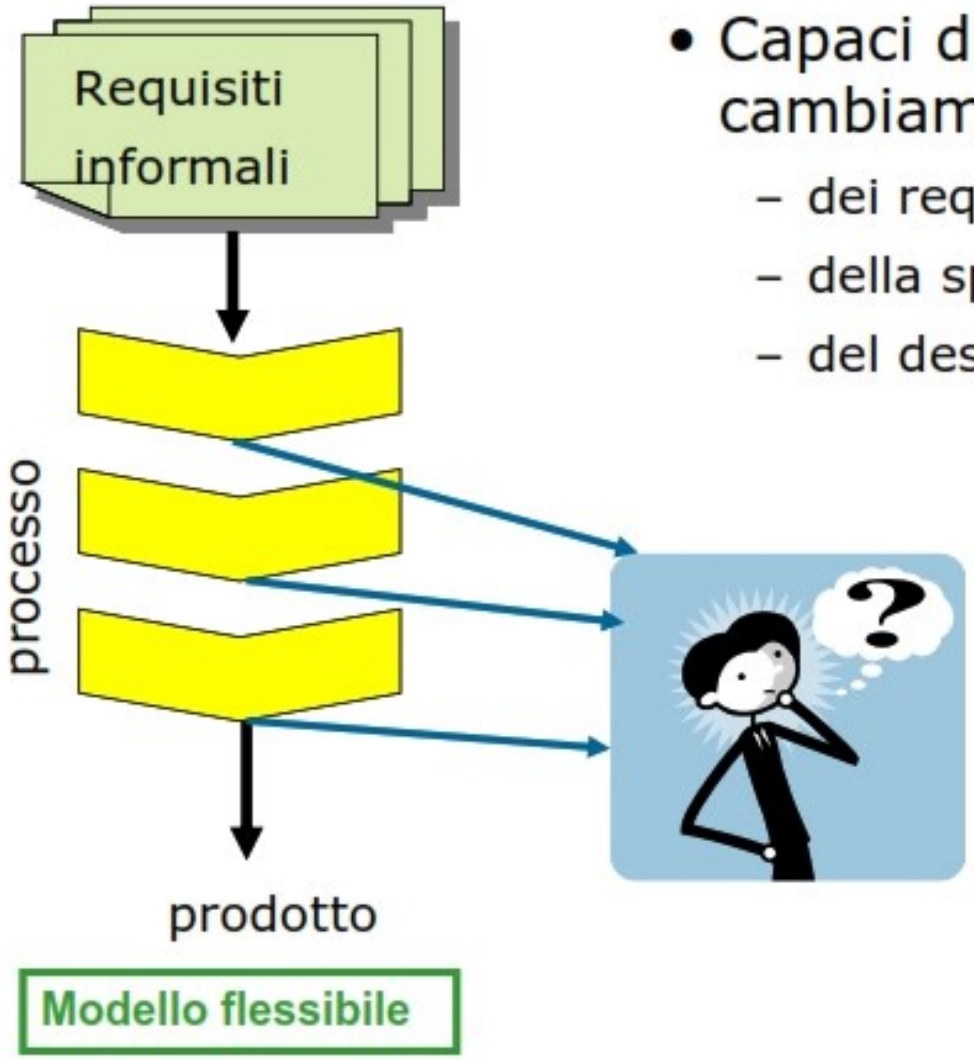
Come affrontare l'evoluzione (2)



- Il feedback è fondamentale per consentire controlli e cambiamenti anticipati
- Solo processi incrementali permettono un *feedback* sugli incrementi successivi

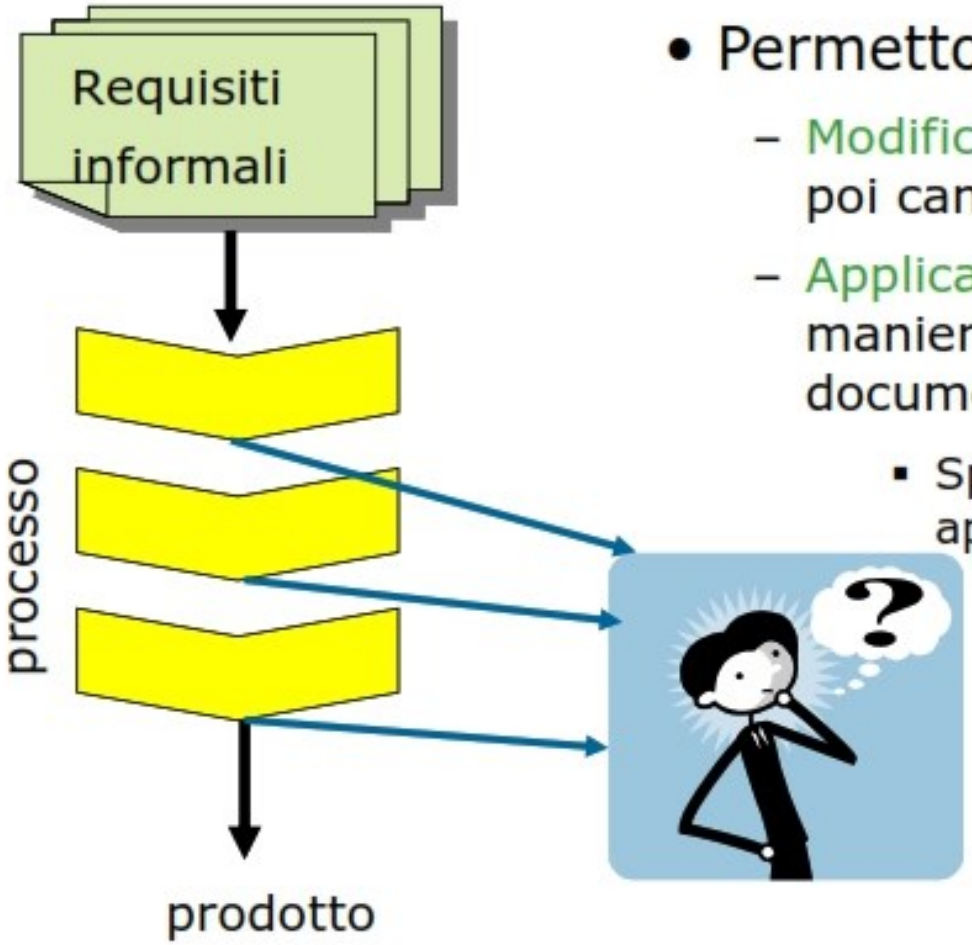
Processi incrementali (1)

- Capaci di adattarsi ai cambiamenti:
  - dei requisiti
  - della specifica
  - del design



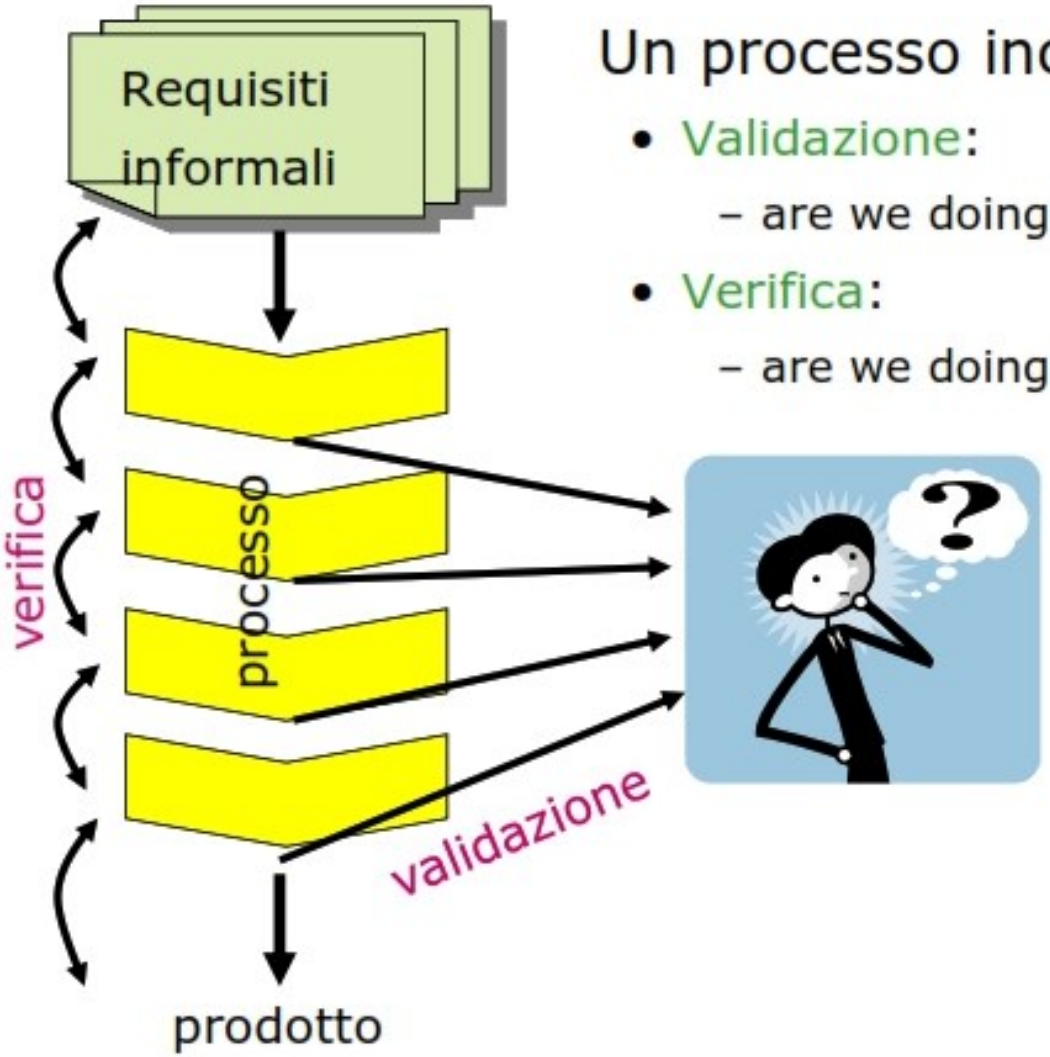
Processi incrementali (2)

- Permettono il *riciclo*
  - Modificare prima il progetto, e poi cambiare il codice
  - Applicare i cambiamenti in maniera consistente in tutti i documenti
    - Spesso i cambiamenti sono applicati solo al codice



Modello flessibile

Validazione e Verifica



Un processo incrementale consente

- Validazione:
  - are we doing the right product?
- Verifica:
  - are we doing the product right?

Modelli di processi incrementali

Esistono in molte forme

- Prototipazione
- Modello a fasi di release (o *incremental delivery*)
- Modello a spirale
  - Ottimale per lo sviluppo di sistemi sicuri

Prototipazione (1)

Basato sul principio "*Do it twice*"

Un prototipo è un *modello approssimato* dell'applicazione che ha lo scopo di fornire il feedback necessario a individuare con esattezza tutte le caratteristiche del sistema e tutti gli errori di progettazioni effettuati

"Cosa" *prototipare* dipende da cosa è critico accertare

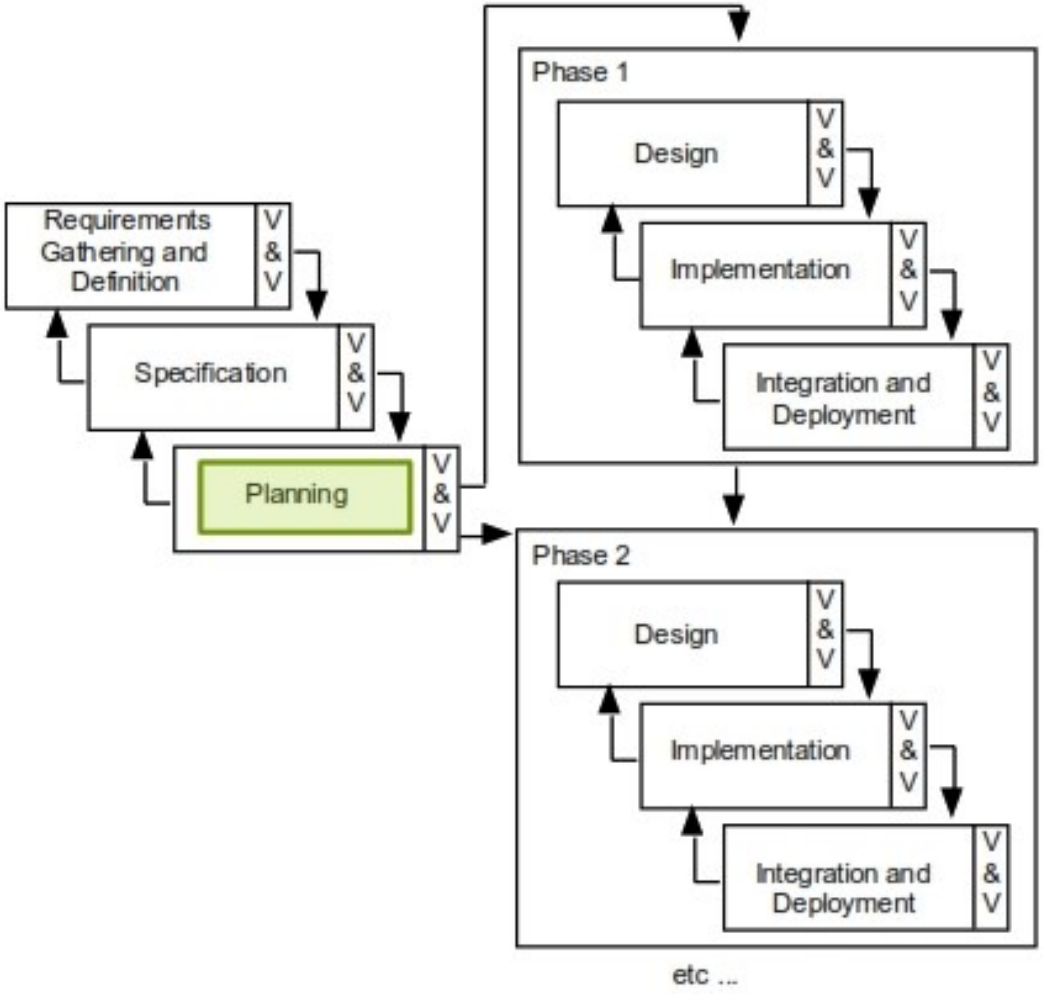
Prototipazione (2)

Per evitare che l'adozione di questo modello comporti costi troppo elevati è necessario che i progettisti siano coscienti della funzione del prototipo e che tutte le fasi del processo siano basate su un criterio di massima riusabilità **delle componenti sviluppate**

- Massimizzare la modularizzazione
- Ingegnerizzare le sole componenti critiche nel prototipo

Prototipo "throw-away" vs prototipo evolutivo

Modello a fasi di release



Modello a fasi di release

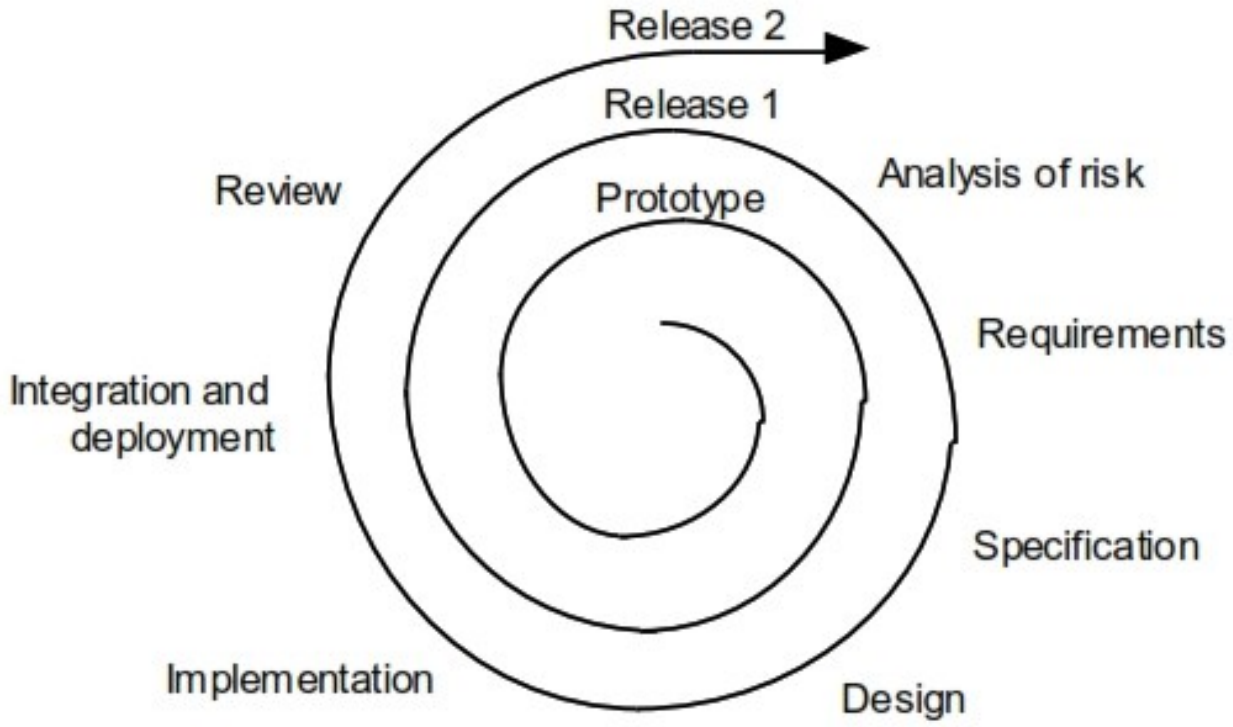
Basato sul principio "early subset, early delivery, ... early feedback"

Si parte da sottoinsiemi critici sui quali richiedere il feedback del cliente

L' incremental delivery è ancor oggi attuato

- Versioni *Beta* per il testing

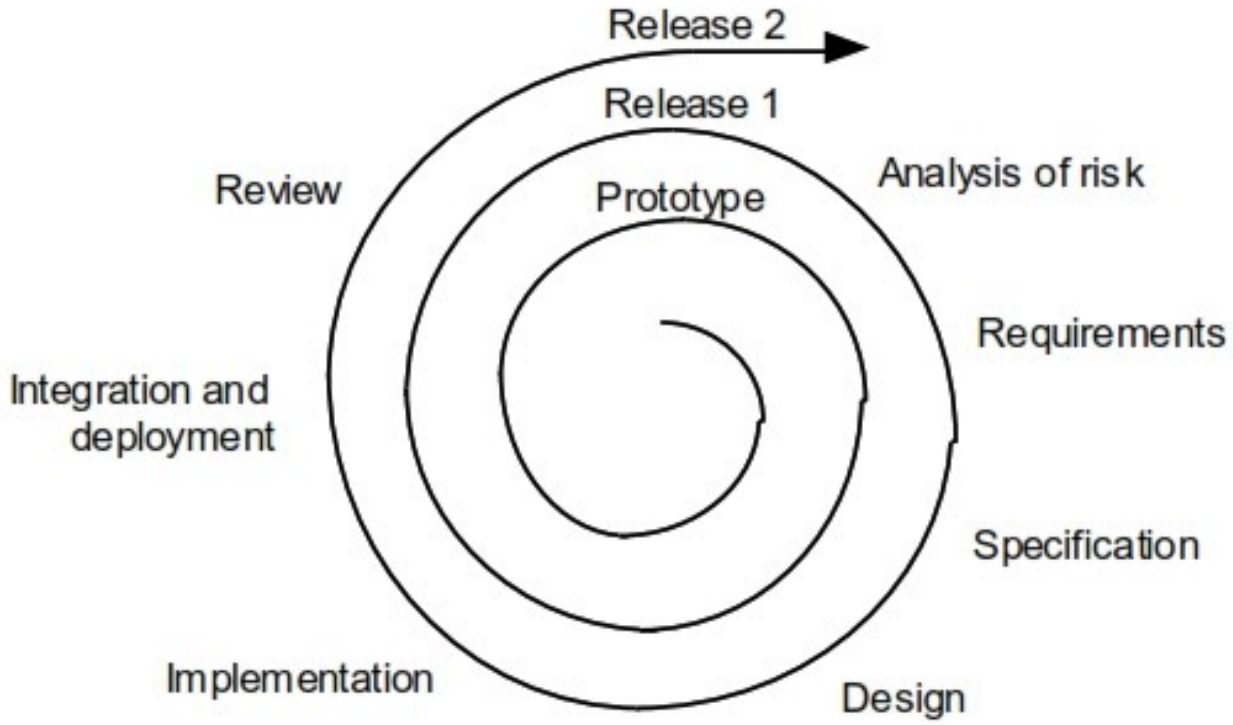
Modello a spirale (1)



E' un processo ciclico tra le seguenti attività:

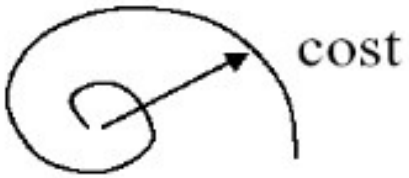
- analisi dei rischi, sviluppo (specifica, design, codifica), testing e revisione della release

Modello a spirale (2)



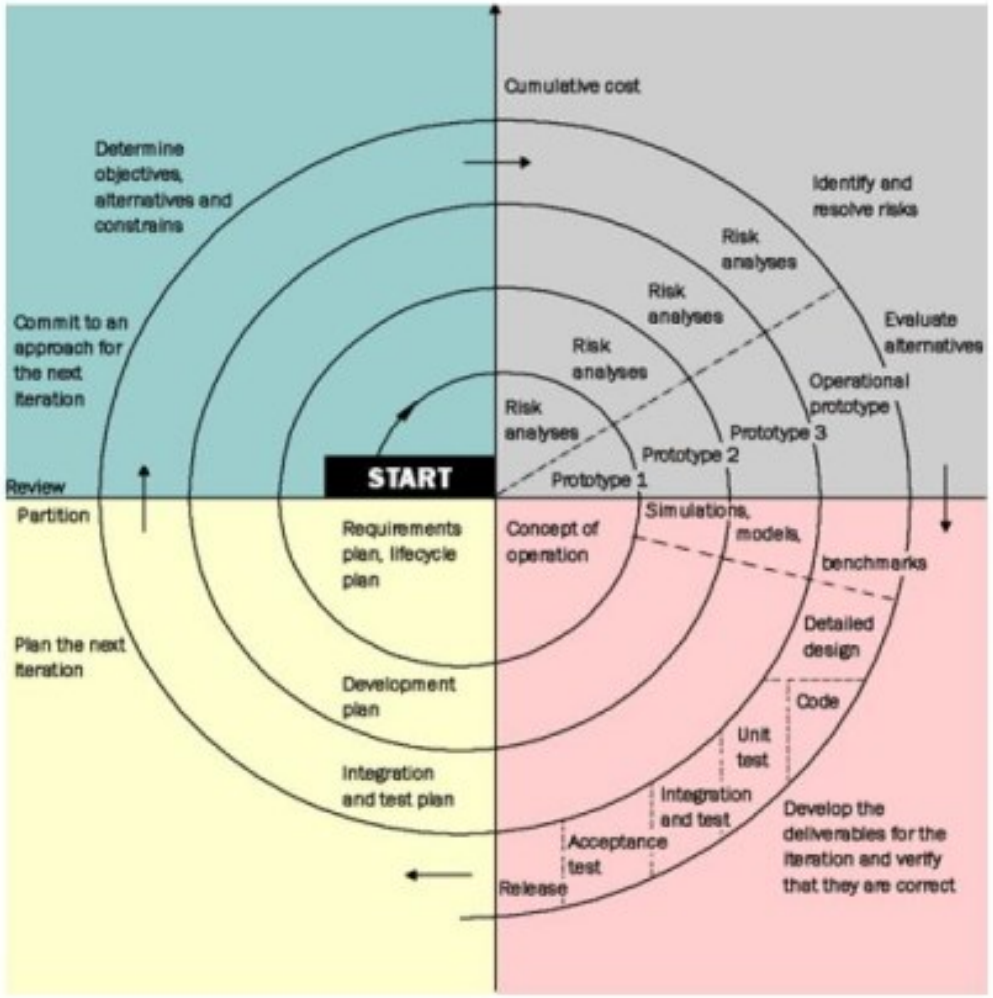
Ad ogni ciclo il costo aumenta

come in una spirale

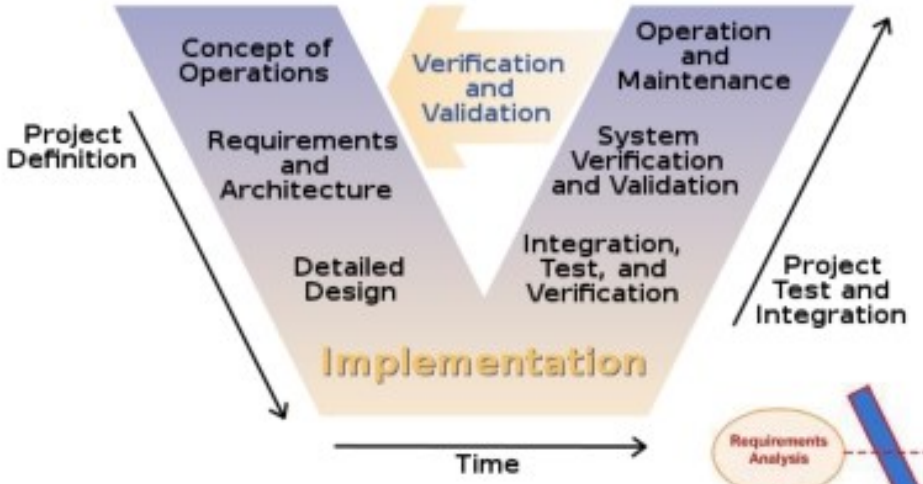


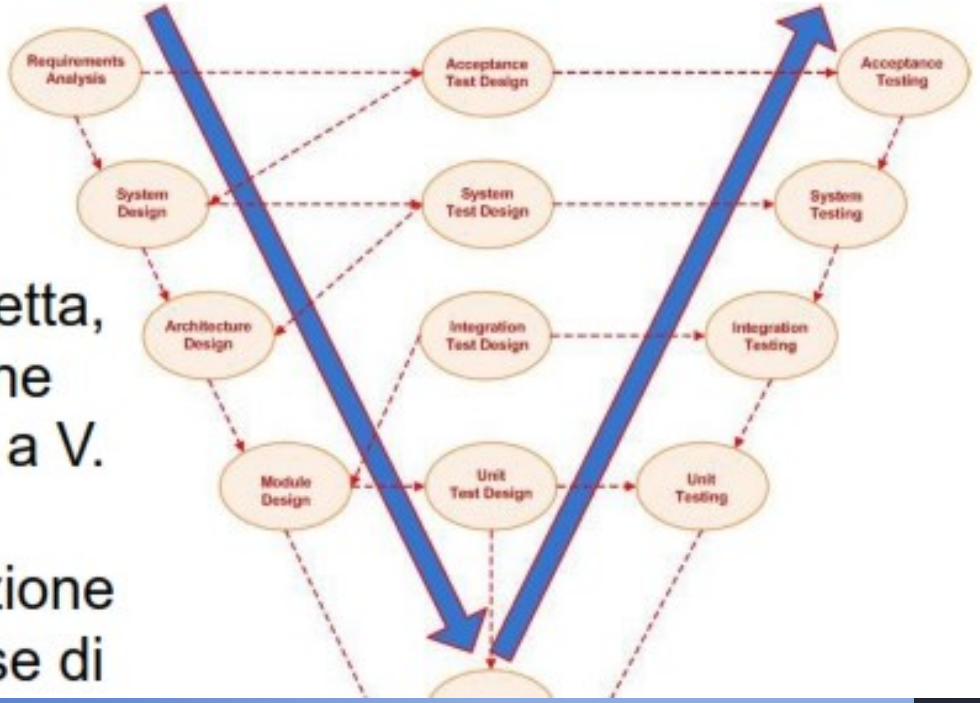
Modello a spirale (3)

- Ingloba *prototipazione* ed approccio *iterativo*
- Può essere visto come un *metamodello* di sviluppo
- Il mantenimento è una forma di sviluppo continuo



Modello a V





Questo modello estende il Modello a cascata: non discende lungo una linea retta, ma dopo la programmazione risale con una tipica forma a V.

Il modello dimostra la relazione tra fase di ciclo di vita e fase di

Nuova tendenza

- **Metodologie pesanti** per i vecchi metodi come il *Modello a cascata*
- **Metodologie iterative** per i metodi come il *Modello a spirale*
- **Metodologie agili** (o *leggere*) che coinvolgono quanto più possibile il committente, ottenendo in tal modo una elevata reattività alle sue richieste

Perchè *agile methods* e *continuous integration*

1. Spesso è praticamente impossibile derivare un set completo di requisiti software stabili (varie cause)
2. Processi di sviluppo software che prevedono di specificare completamente i requisiti e passare poi a progettazione, implementazione e test del sistema non sono idonei allo sviluppo rapido del software
3. Per i safety-critical control systems, in cui è essenziale un'analisi completa del sistema, un approccio pianificato è quello giusto
4. Per i processi di sviluppo aziendale che si concentrano sullo sviluppo e la consegna rapidi del software sono essenziali
5. La crisi sw degli anni 90 su un processo di sviluppo più *plan-driven* che *sw-driven* ha portato alla nascita di approcci **agili** ed al concetto di **continuous integration**
   - *Agile alliance*, una organizzazione no-profit creata allo scopo di diffondere le metodologie agili

Manifesto per lo Sviluppo Agile di Software

agilemanifesto.org

Il manifesto stabilisce:

- Gli **individui** e le **interazioni** più che i processi e gli strumenti
- Il **software funzionante** più che la documentazione esaustiva
- La **collaborazione col cliente** più che la negoziazione dei contratti
- **Rispondere al cambiamento** più che seguire un piano di sviluppo (no *design for change*)

Ovvero, fermo restando il valore delle voci a destra, consideriamo più importanti le voci a sinistra.

Metodi agili

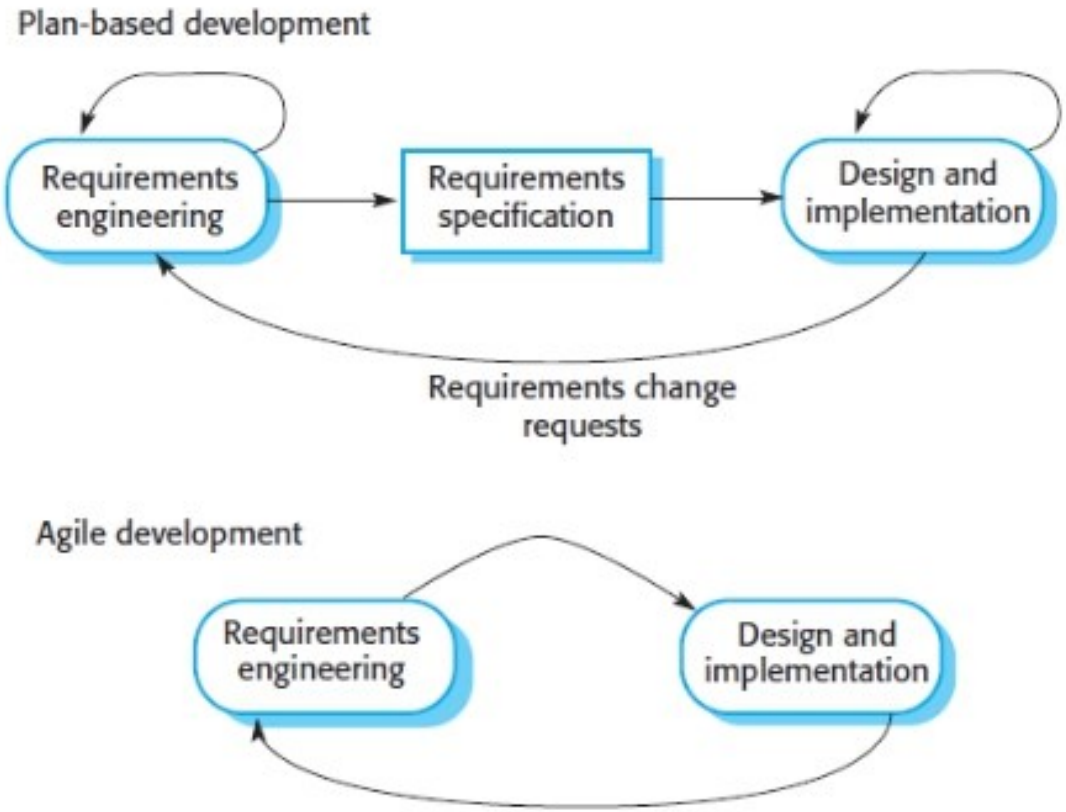
I metodi agili sono una famiglia di metodi di sviluppo che hanno in comune:

- **Rilasci frequenti** del prodotto sviluppato
- **Collaborazione** continua del team di progetto col **cliente**
- **Documentazione** di sviluppo **ridotta**
- **Valutazione** sistematica e continua **di** valori e rischi **dei cambiamenti**

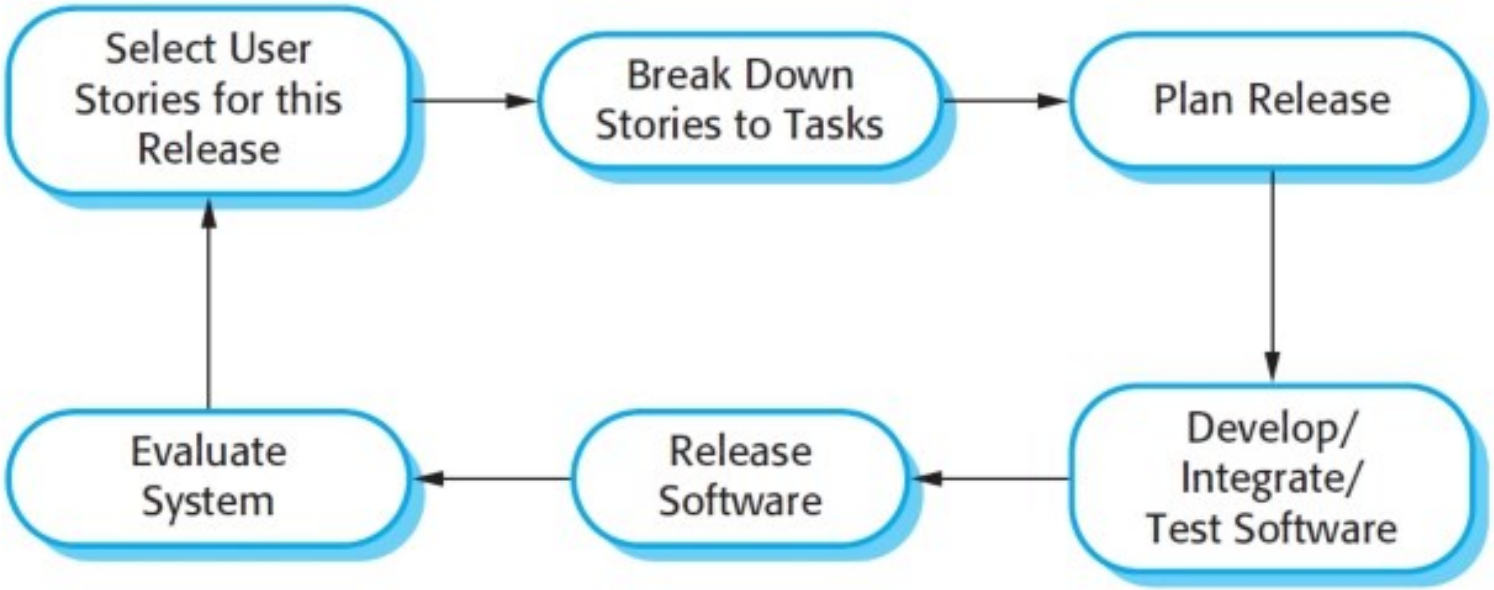
Esistono un certo numero di tali metodologie: Extreme Programming, SCRUM , ecc..

- Model-Driven Development come compromesso

Plan-driven vs SW-driven

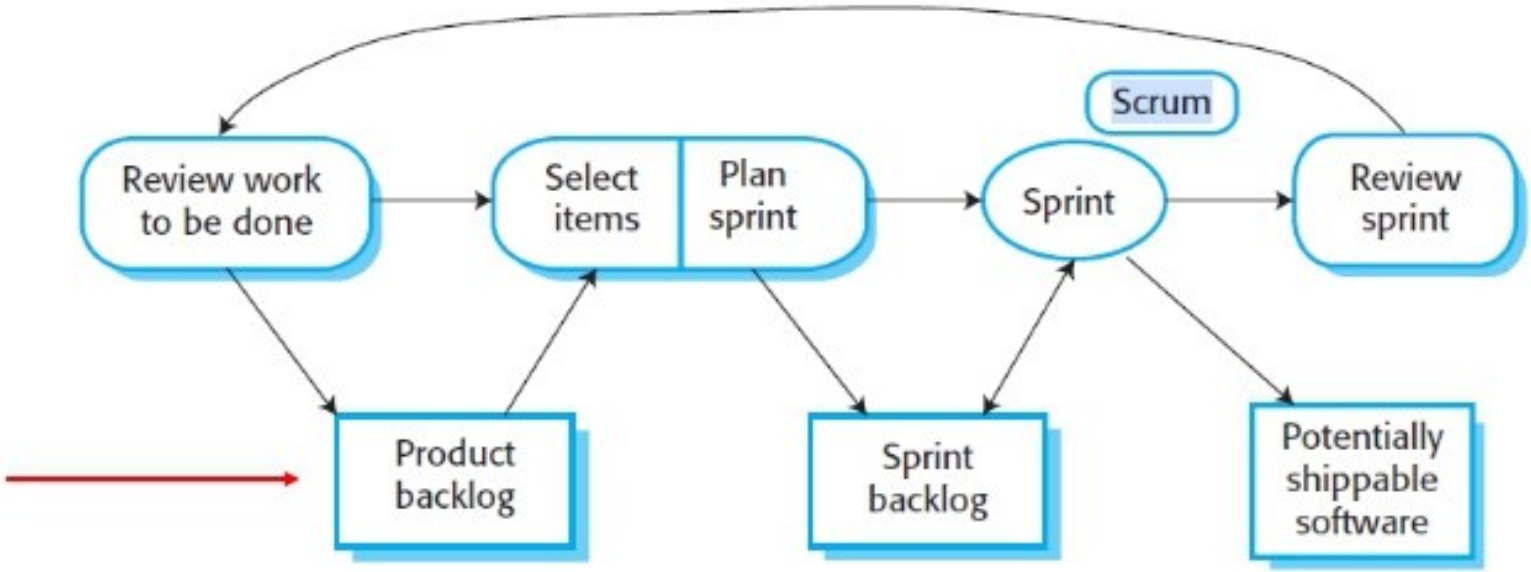


Processo XP



Test-first development
Pair programming
Refactoring
Continuous integration

Processo Scrum



- backlog del prodotto: l'elenco di elementi (caratteristiche del prodotto, requisiti, miglioramenti tecnici) su cui il team Scrum deve lavorare
- Alla fine di uno sprint cycle (o scrum), la funzionalità completata viene consegnata agli stakeholder
- continuous integration
- short daily meetings to review progress and to re-prioritize work
- whole team has visibility of everything
- customers see on-time delivery of increments and gain feedback

Limiti dei metodi agili

Non adeguato per software in sistemi embedded, e sistemi di grandi dimensioni/ complessi

Numero di problemi:

1. L'informalità vs legalità aziendale
2. nuovo sviluppo vs manutenzione
   - Mancanza di documentazione
   - Mantenere il cliente coinvolto
   - Continuità del team di sviluppo
3. piccoli team co-localizzati vs team distribuiti in tutto il mondo

Fondamentale per la scalabilità del metodi agili è la loro integrazione in approcci plan-driven