

---

# Il processo software

## Modelli di ciclo di vita

---

Laura Semini, Dipartimento di Informatica



# Il processo software

---

Con **processo software** si indica il percorso da svolgere per sviluppare un prodotto o sistema software.

- Inizia con l'esplorazione dell'idea e finisce con la dismissione del software
- Il processo software include anche
  - **gli strumenti e le tecniche** per lo sviluppo
  - e i **professionisti** coinvolti

# Processo sw

---

- Modellare il processo significa strutturarlo:
  - Suddividerlo in *attività*
  - *di ogni attività, dire:*
    - Cosa
    - Quali prodotti
    - Quando
- Un esempio di standard di processo sw: ISO 12207

# Modello di ciclo di vita

---

- Organizzazione delle attività
  - Ordinamento delle attività
  - Criteri per terminare e passare alla successiva
- Esempio: preparazione di un dolce
  - Possiamo dare un modello di sviluppo generico:
    - Fare la spesa, mentre il forno si scalda impastare e mettere nella teglia, infine infornare.
  - Non è la ricetta: è una sequenza di passi valida per tutti i dolci da forno

# Evoluzione dei modelli di ciclo di vita

---

## Build-and-Fix: un non-modello

- Attività non identificate né organizzate
- Progetti non gestiti

## Modelli prescrittivi

- Cascata
- Modello a V
- Rapid Prototyping
- Modello incrementale
- Modello a spirale

## Unified Process

## Modelli agili

- Extreme Programming
- Scrum
- Continuous delivery
- Devops

# Così tanti ???

---

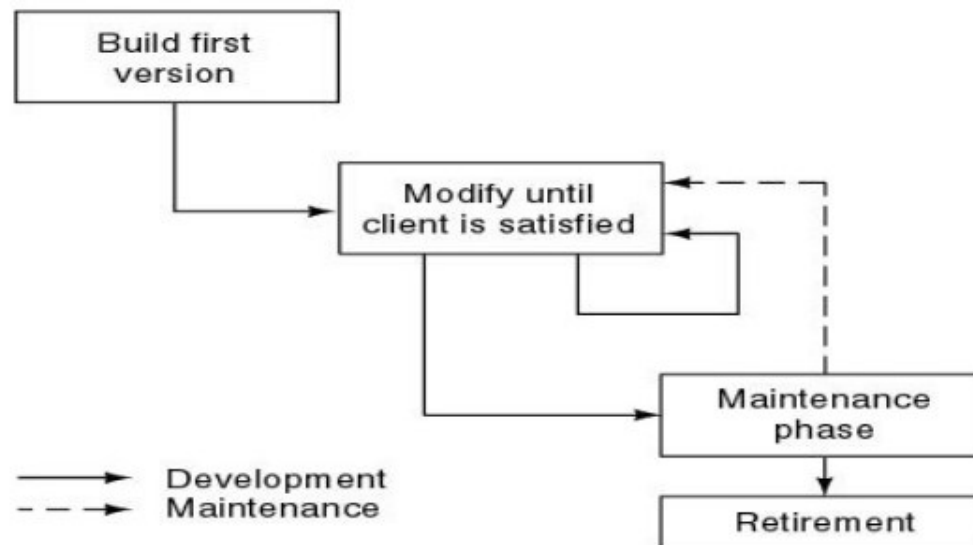
Li vedremo tutti perché:

- Sono i più noti
- Ma soprattutto perché ognuno di essi ha affrontato un nuovo aspetto
  - Sono queste caratteristiche peculiari che ci interessano e che guidano lo sviluppatore nella scelta di un ciclo di vita per ogni progetto
  - Un ciclo di vita di un progetto reale può mescolare idee di diversi modelli

# Build-and-Fix Model

---

- Il prodotto è sviluppato senza specifica e senza un tentativo di progettazione
  - lo sviluppatore scrive un programma
  - che poi è modificato più volte finché non soddisfa il committente



# Build-and-fix Model

---

- Forse adeguato a un progetto di 100 linee di codice
- Diventa totalmente improponibile per prodotti di ragionevole grandezza
- Inoltre la manutenzione di un prodotto senza specifica o documentazione che ne spieghi la progettazione è estremamente difficile
- **Prima di iniziare lo sviluppo di un progetto dobbiamo scegliere un "vero" modello del ciclo di vita**



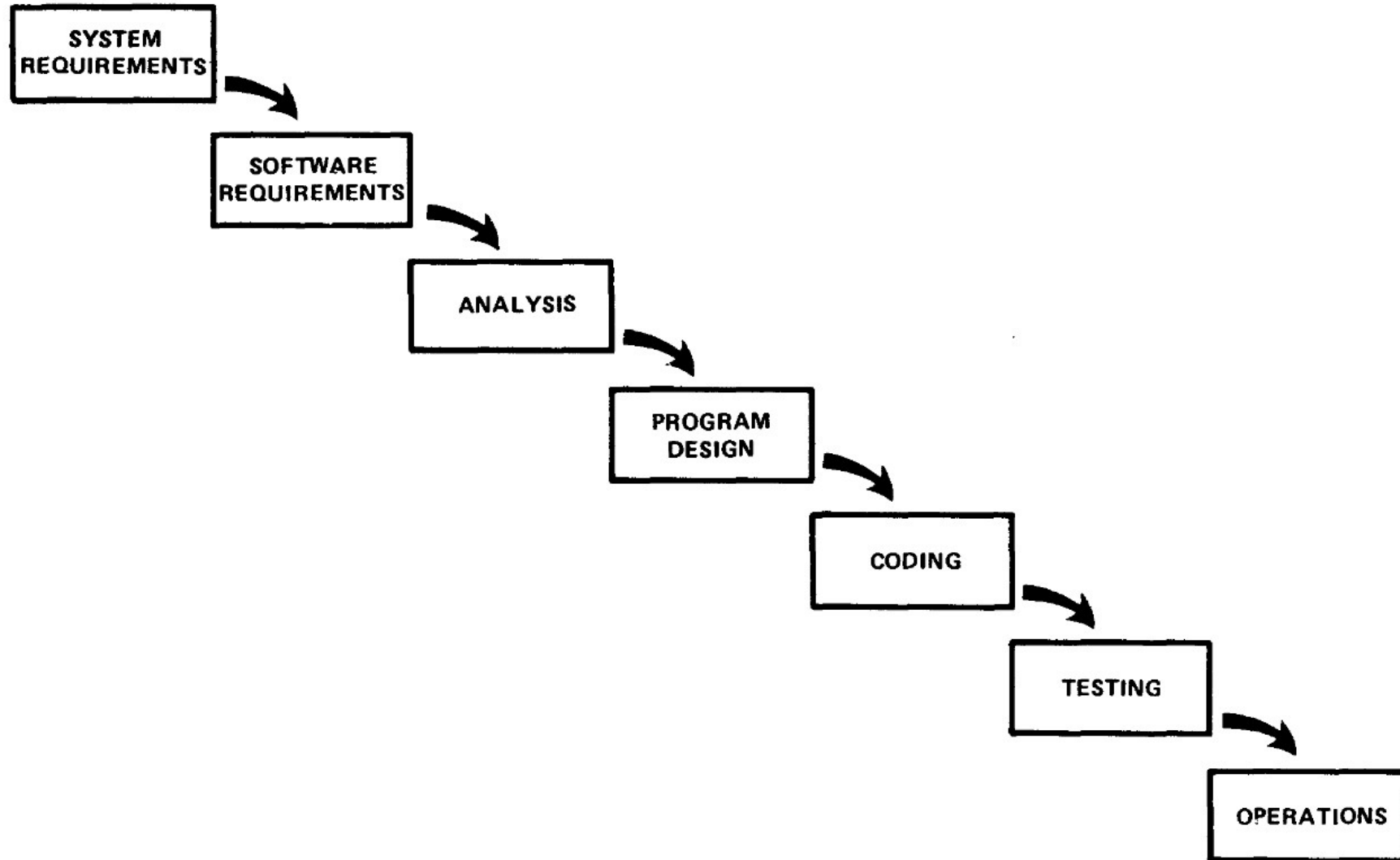
# Occorre pianificare, bene, il lavoro

---



# Il modello a cascata [Royce,1970]

---



# Il modello a cascata [Royce,1970]

---

- Il valore di questo modello è stato quello **di distinguere e definire le fasi di un processo software**.
  - Evidenziando l'importanza delle fasi di **analisi e di progettazione** prima di passare alla codifica.
- Inoltre il modello richiede che il passaggio a una nuova fase sia possibile solo dopo il **completamento della precedente**:
  - Ogni fase produce un documento che deve essere approvato da un gruppo di valutatori prima di passare alla fase successiva
  - Si parla anche di modello “**document driven**”

# Critiche al modello a cascata ;-)





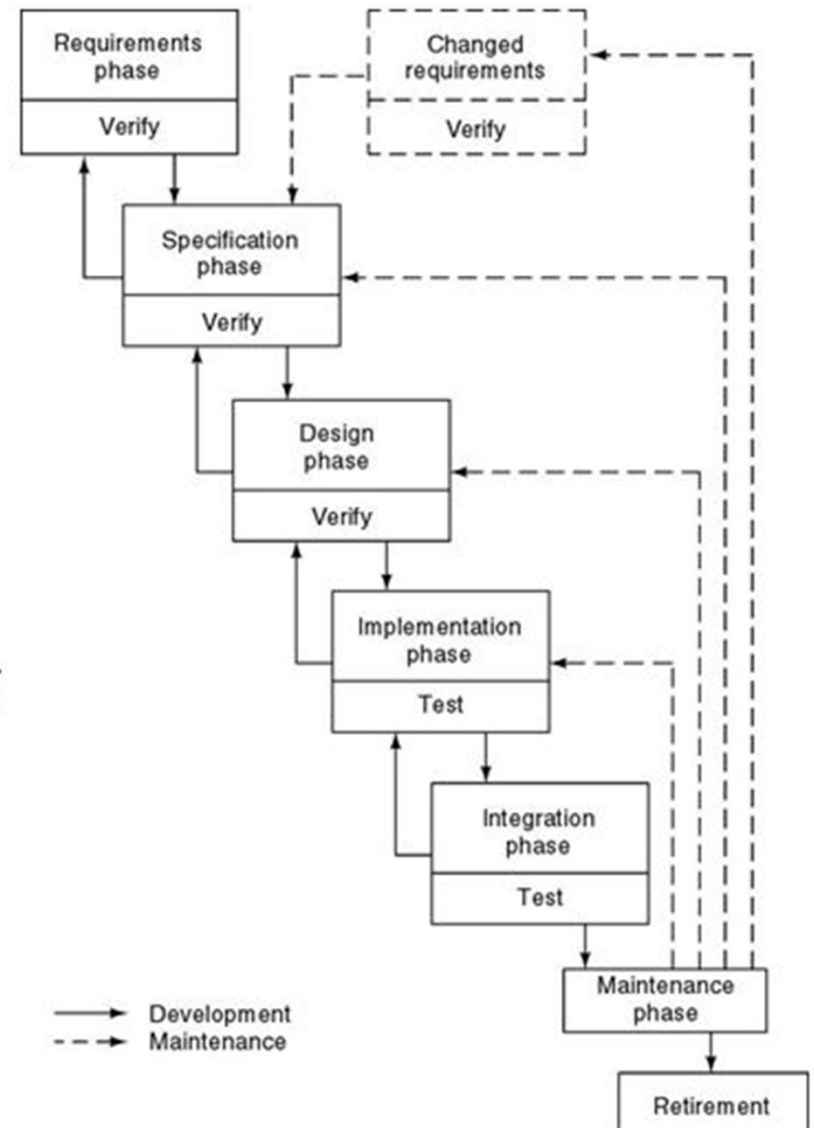
# Analisi del modello a cascata

---

- Il vero punto debole è che **manca l'interazione col cliente** che vede solo il prodotto finito, alla fine del processo
  - Spesso c'è una sostanziale **differenza tra come il cliente immagina un prodotto e come il prodotto viene realizzato**
  - A quel punto, se ci sono discrepanze (e ci sono!) **tutto il processo deve essere ripetuto**
- Un ulteriore punto critico è **l'eccessiva produzione di documenti**:
  - Nessuna fase è completa finché anche il documento per quella fase non è stato terminato e approvato dal gruppo di valutatori (Software Quality Assurance)
- Ricordiamo quindi il modello a cascata per il **valore che ha avuto storicamente**:
  - **Definire e strutturare le fasi di un processo sw**

# Diamo a Royce quel che è di Royce

- Il modello a cascata è a tutti noto come quello visto, **rigidamente "in caduta"** (tutta la letteratura riporta quel modello)
- Royce in realtà già nell'articolo del 1970, evidenzia le limitazioni della cascata e propone come un **modello con feedback loops** da una fase alla precedente.





# Modello a V

---

- Evidenzia come sia possibile progettare i test durante le fasi di sviluppo (prima della codifica)
- Attività di (progettazione del) test come parte del processo di sviluppo del software
- Il modello a V prende la metà inferiore del modello a cascata e la piega verso l'alto a forma di V
  - le attività a destra verificano o convalidano i prodotti dell'attività a sinistra
  - il lato sinistro della V rappresenta le attività di analisi che scompongono le esigenze degli utenti in pezzi piccoli e gestibili
  - mentre il lato destro della V mostra le corrispondenti attività di sintesi che aggregano (e testano) questi pezzi in un sistema che soddisfa le esigenze degli utenti



# Modello a V

---

- Noto anche come **modello di validazione o di verifica**,
  - introdotto da Hughes Aircraft negli anni '80 come parte dello sforzo di pre-proposta per il **programma Advanced Automation System** della **Federal Aviation Administration**
- Oggi il modello V **è uno degli standard SQA** (assicurazione della qualità del software)
  - viene utilizzato per descrivere le **attività di test nell'ambito del processo di sviluppo del software**
  - Idea ripresa nel **Test Driven Development**

# Modelli iterativi

---

Modello a Cascata

Modello a V

Rapid Prototyping

Incrementale

Modello a spirale

Unified Process

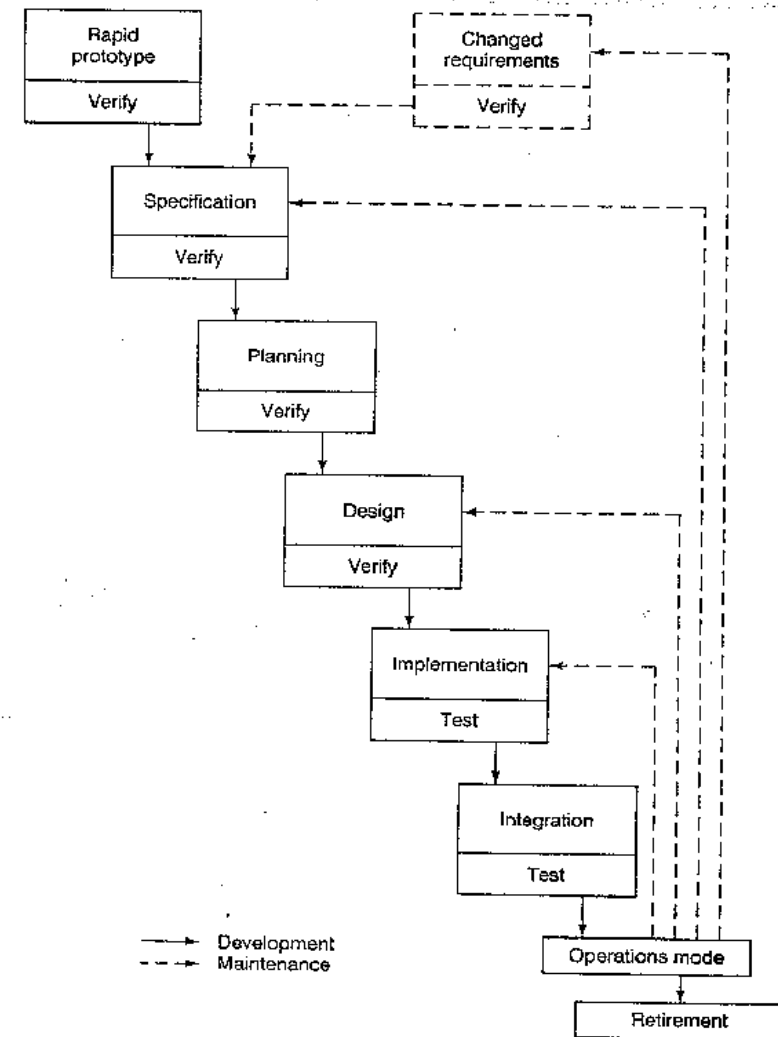
Modelli fondamentalmente  
sequenziali

Modelli iterativi



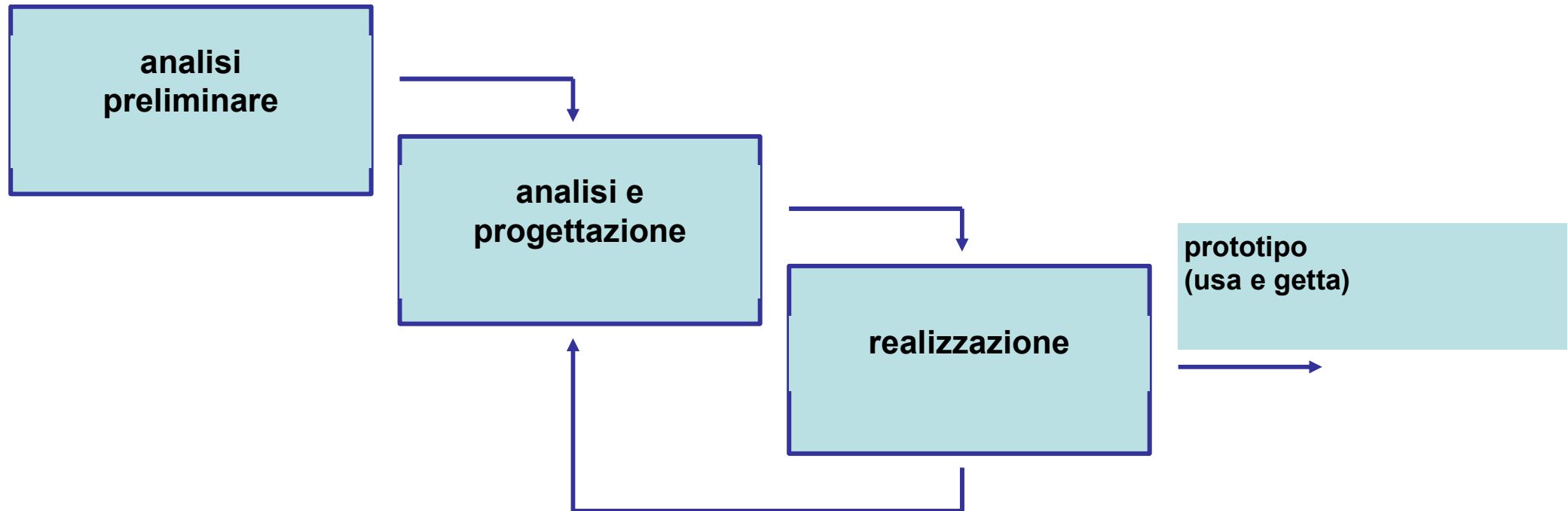
# Rapid Prototyping

- Lo scopo è quello di costruire velocemente un **prototipo** per permettere al committente di sperimentarlo
- Il **prototipo** aiuta il cliente a meglio **descrivere i requisiti**
  - si passa a questo punto alle fase di specifica
- **Utile quando i requisiti non sono chiari**
- Anche noto come **modello evolutivo**



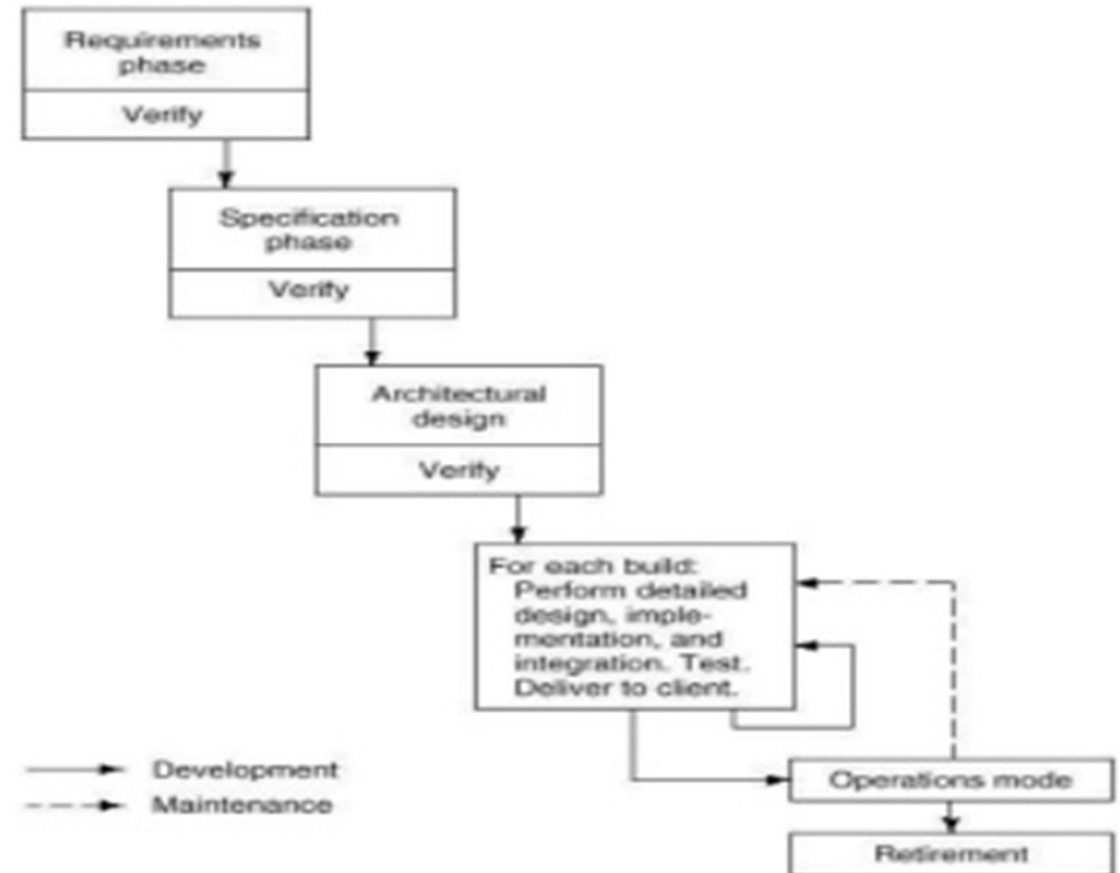
# Rapid Prototyping: rappresentazione semplificata

---



# Modello incrementale

- Il sistema è costruito iterativamente aggiungendo nuove funzionalità
  - I requisiti e il progetto sono definiti inizialmente
  - Il sistema è implementato, integrato e testato con una serie di passaggi incrementali



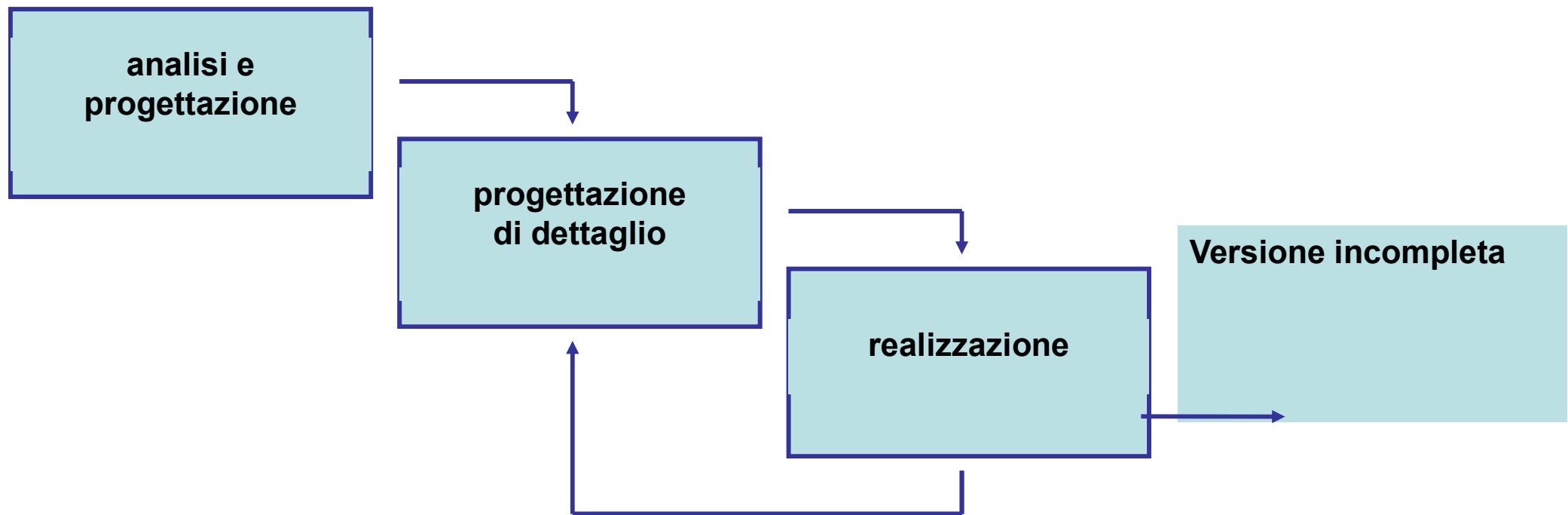
# Modello incrementale

---

- Si applica in caso di requisiti stabili, serve a
  - Ritardare la realizzazione delle componenti che dipendono criticamente da fattori esterni (tecnologie, hardware sperimentale, ecc)
  - “uscire” velocemente con qualcosa
- Se non progettato bene diventa un Build-and-Fix

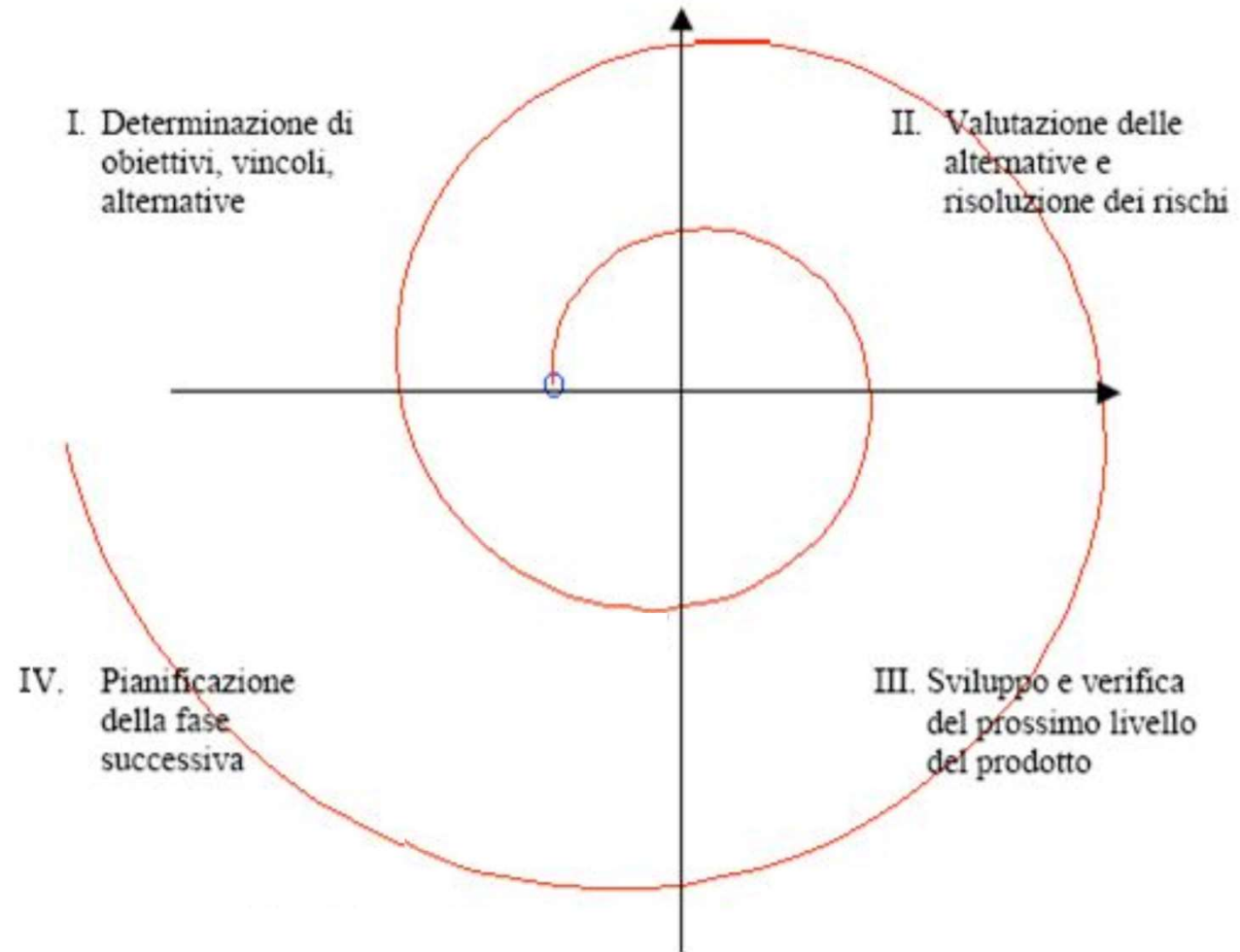
# Modelo incrementale: rappresentazione semplificata

---



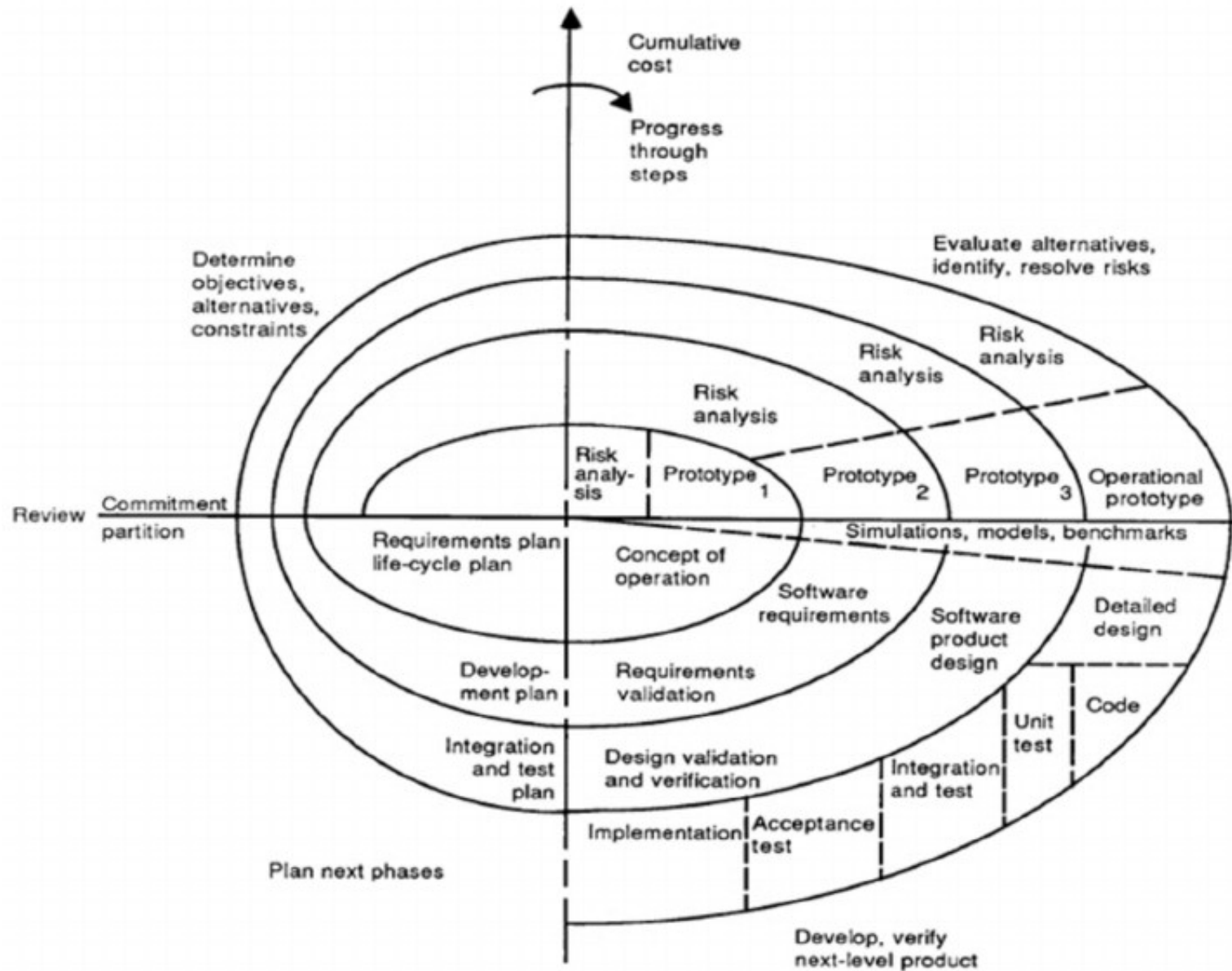
# Il modello a spirale

- Proposto da Boehm nel 1988
- Iterativo, ogni iterazione è organizzata in 4 fasi:
  - Definizione degli obiettivi
  - Analisi dei rischi
  - Sviluppo e validazione
  - Pianificazione del nuovo ciclo





# Il modello a spirale: rappresentazione originale di Bohem



# Il modello a spirale

---

- È un modello astratto
  - Va specializzato per dire cosa fare in concreto in ogni iterazione e in ogni sua fase
  - Applicabile ai cicli tradizionali
- Evidenzia gli aspetti gestionali
  - Pianificazione delle fasi
  - **Centrato sull'analisi dei rischi** (modello “risk driven”)
    - Tipici rischi:
      - Dominio poco noto, Linguaggi o strumenti nuovi, Personale non addestrato
  - Ispirato dal **plan-do-check-act cycle** [William E. Deming 1950]
  - Prevede maggior comunicazione e confronto con il committente

# Unified Process

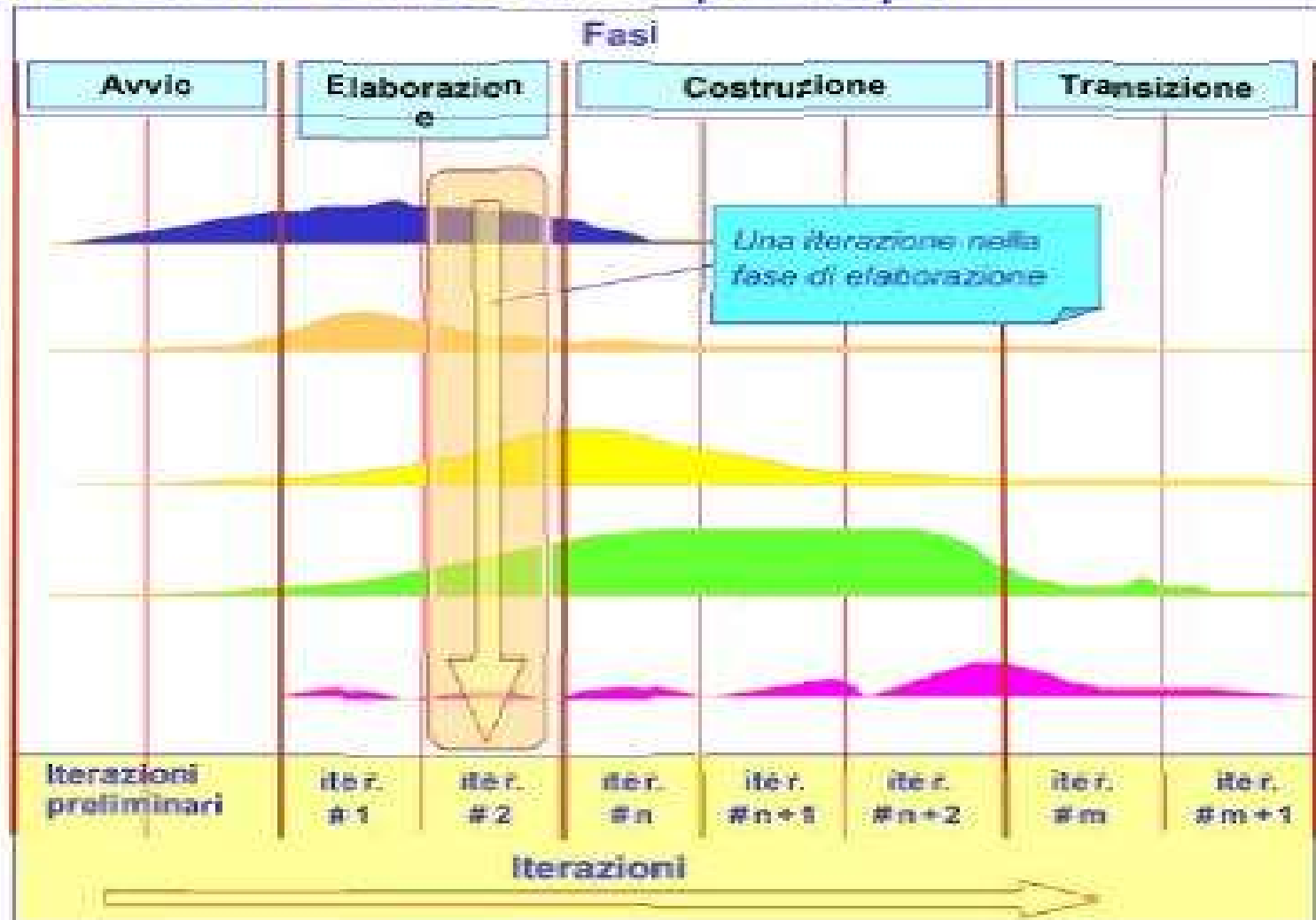
---

- Proposto nel 1999 da
  - Grady Booch, Ivar Jacobson, James Roumbaugh
- Caratteristiche
  - Guidato dai casi d'uso e dall'analisi dei rischi
    - Raccolta dei requisiti e passi successivi guidati dallo studio degli use case
  - Incentrato sull'architettura
    - il processo assegna alla descrizione dell'architettura del sistema un ruolo molto importante. L'approccio è infatti quello di concentrarsi, soprattutto nelle prime fasi, sull'architettura di massima, lasciando i dettagli alle fasi successive. In tal modo è possibile avere da subito una visione generale del sistema facilmente adattabile al cambiamento dei requisiti
- Iterativo incrementale

# Schema di un ciclo

## Fasi, iterazioni e workflow principali

Workflow
Requisiti
Analisi
Progettazione
Implementazione
Test



# Processi agili

---

Per **metodologia agile (o leggera)** o **metodo agile** si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente.

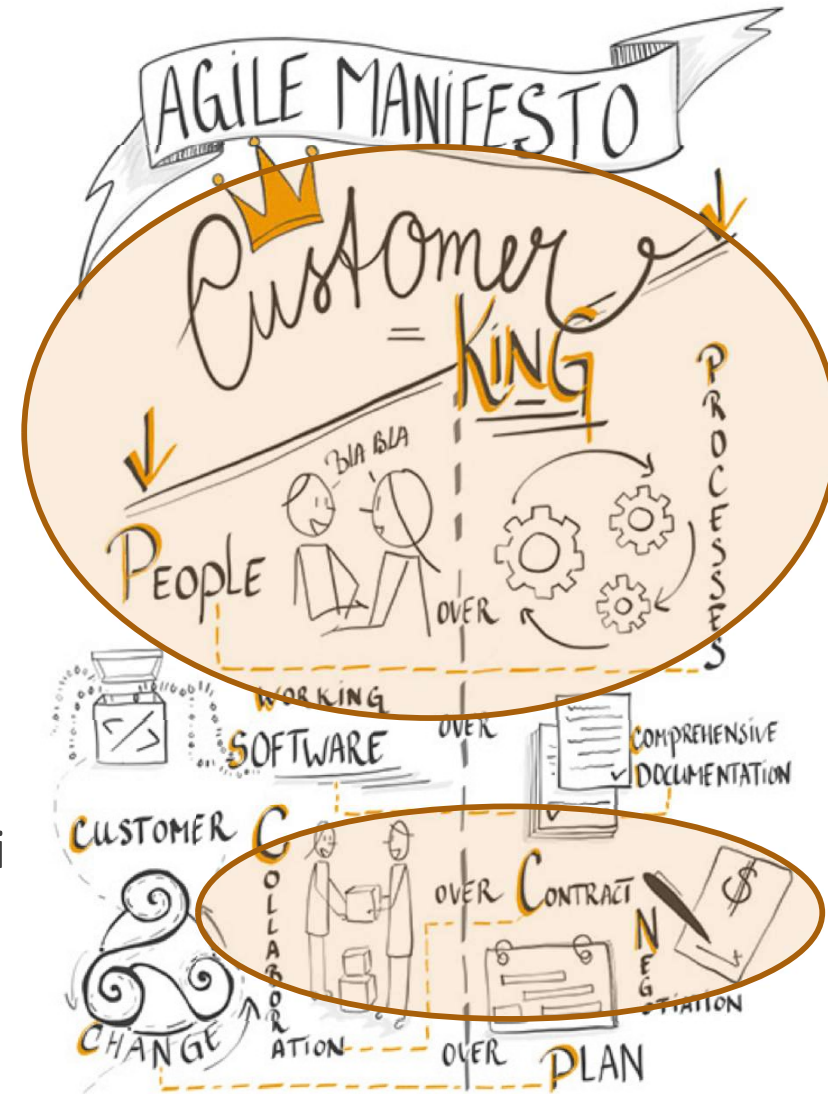
Adatti a progetti con meno di 50 sviluppatori

Una metodologia agile si basa sui **principi del Manifesto di Snowbird, feb 2001**

# Agile manifesto (aka di Snowbird)

## Comunicazione:

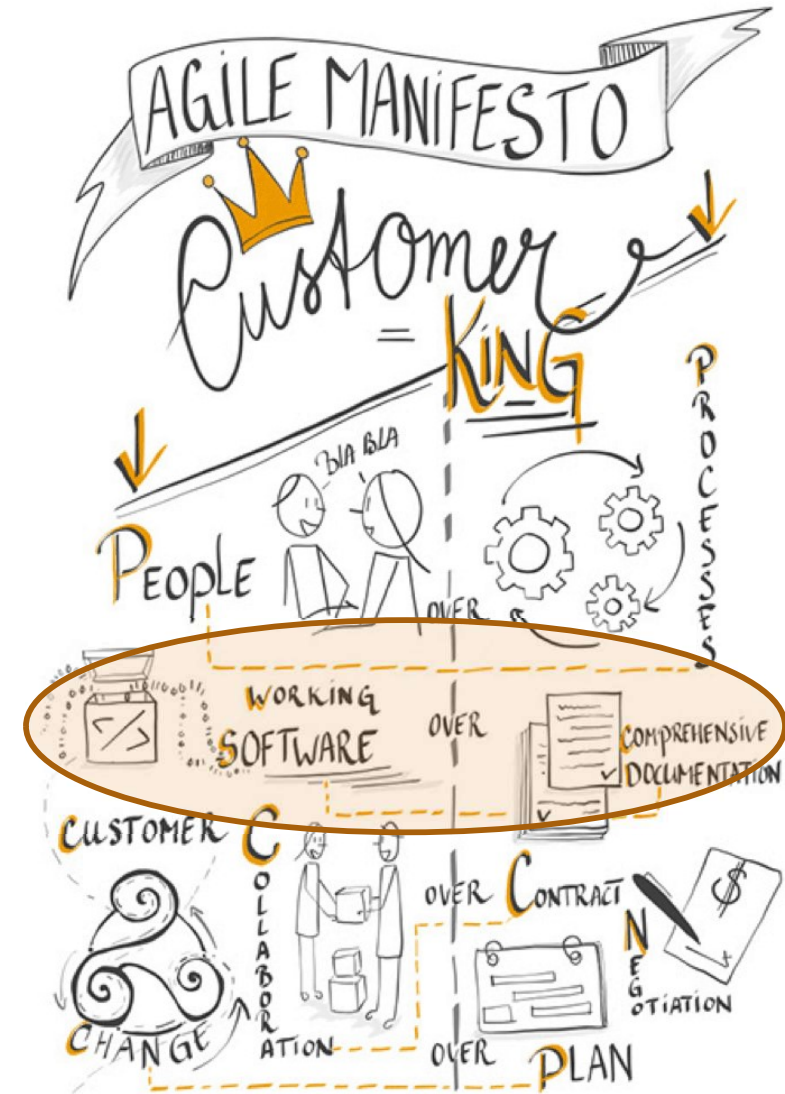
- le persone e le **interazioni** sono più importanti di processi e strumenti
  - la comunicazione tra gli attori di un progetto sw è la miglior risorsa del progetto;
  - tutti possono parlare con tutti, e.g. l'ultimo dei programmatori con il cliente;
- **collaborare con i clienti** al di là del contratto
  - la collaborazione diretta offre risultati migliori dei rapporti contrattuali;



# Agile manifesto (aka di Snowbird)

## Semplicità:

- analisti mantengano la descrizione formale il più semplice e chiara possibile
- è più importante avere **software funzionante** che **documentazione**
- bisogna mantenere il codice semplice e avanzato tecnicamente, riducendo la documentazione al minimo indispensabile;





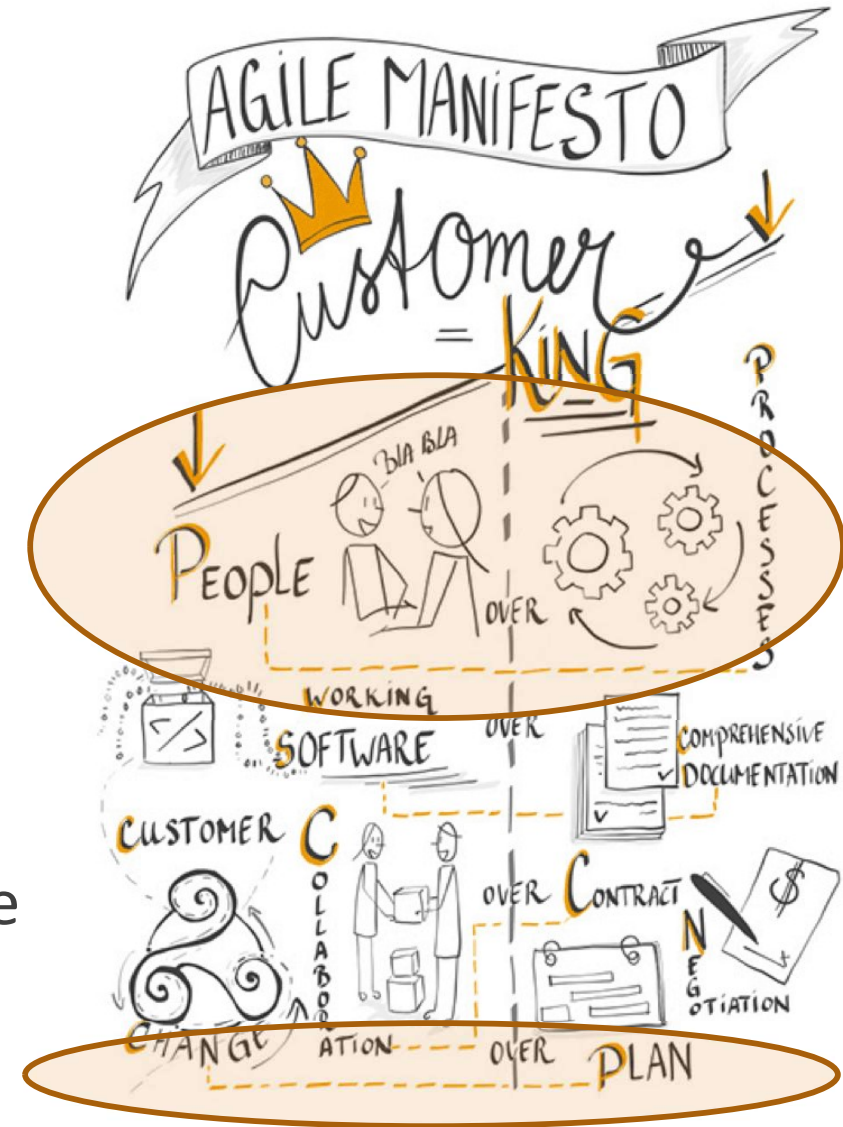
# Agile manifesto (aka di Snowbird)

## Feedback

- rilasciare nuove versioni del software ad **intervalli frequenti**
- sin dal primo giorno si testa il codice

## Coraggio

- dare in uso il sistema il prima possibile e implementare i cambiamenti richiesti man mano
- **rispondere ai cambiamenti** più che aderire al progetto

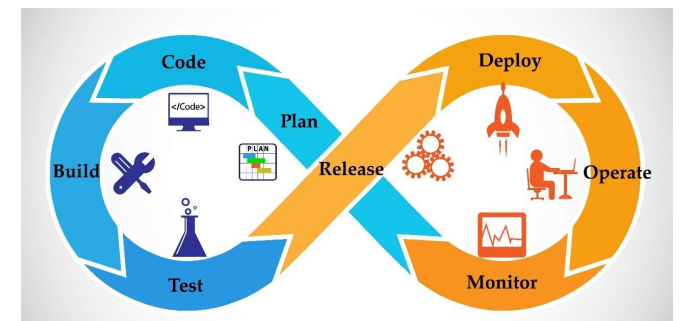




# Continuous Integration, Continuous delivery, DevOps

---

- Concetti chiave che nascono nei processi Agile sono:
- **Continuous Integration**: rendere il più possibile automatico il processo di consegna e integrazione dei moduli
  - Costruzione
  - Unit Test
  - Integration Test
- **Continuous Delivery**: Rilascio frequente e supportato da strumenti di nuove versioni del sistema sw
- **DevOps**: “Development” e “Operations”
  - Maggior collaborazione tra
    - Sviluppatori
    - Responsabile della manutenzione, della sicurezza, della gestione dell'infrastruttura tecnologica di un'organizzazione



# eXtreme Programming

---

- eXtreme Programming è un esempio di processo agile
- Si basa su un insieme di prassi:
  - **Pianificazione flessibile**
    - basata su scenari proposti dagli utenti
    - coinvolge i programmatori
  - **Rilasci frequenti**
    - due-quattro settimane
    - inizio di una nuova pianificazione

# eXtreme Programming

---

## Prassi di XP (continua)

- **Progetti semplici**
  - comprensibili a tutti
- **Verifica (testing)**
  - di unità e di sistema (basati sugli scenari)
  - supporto automatico
- **Test Driven Development**
  - casi di test prima del codice
- **Cliente sempre a disposizione (circa ogni settimana)**

# eXtreme Programming

---

## Prassi di XP (continua)

- **Programmazione a coppie**
  - un solo terminale, il *driver* scrive il codice mentre il *navigatore* controlla il lavoro del suo compagno in maniera attiva.
- **No lavoro straordinario**
- **Collettivizzazione del codice**
  - accesso libero
  - integrazione continua
  - standard di codifica

# eXtreme Programming

---

## Prassi di XP (continua)

- **Code Refactoring**
  - modifying it without changing its behavior,
  - Uno dei motti del XP è “se un metodo necessita di un commento, riscrivilo!” (codice *auto-esplicativo*).
- **Daily Stand Up Meeting**

# SCRUM

---

SCRUM è un processo Agile il cui nome deriva dalla terminologia del gioco del Rugby (mischia).

SCRUM: un processo in cui un insieme di persone si muove all'unisono per raggiungere un obiettivo predeterminato, tale obiettivo garantisce la soddisfazione delle ambizioni di squadra e delle ambizioni personali.

E' un processo

- Che può essere adottato per gestire e controllare lo sviluppo del software
- E' iterativo, incrementale, per lo sviluppo e gestione di ogni tipologia di prodotto
- Fornisce alla fine di ogni iterazione un set di funzionalità potenzialmente rilasciabili



# SCRUM: fase 1

---

## Pre-game phase

- **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

# SCRUM: fase 2

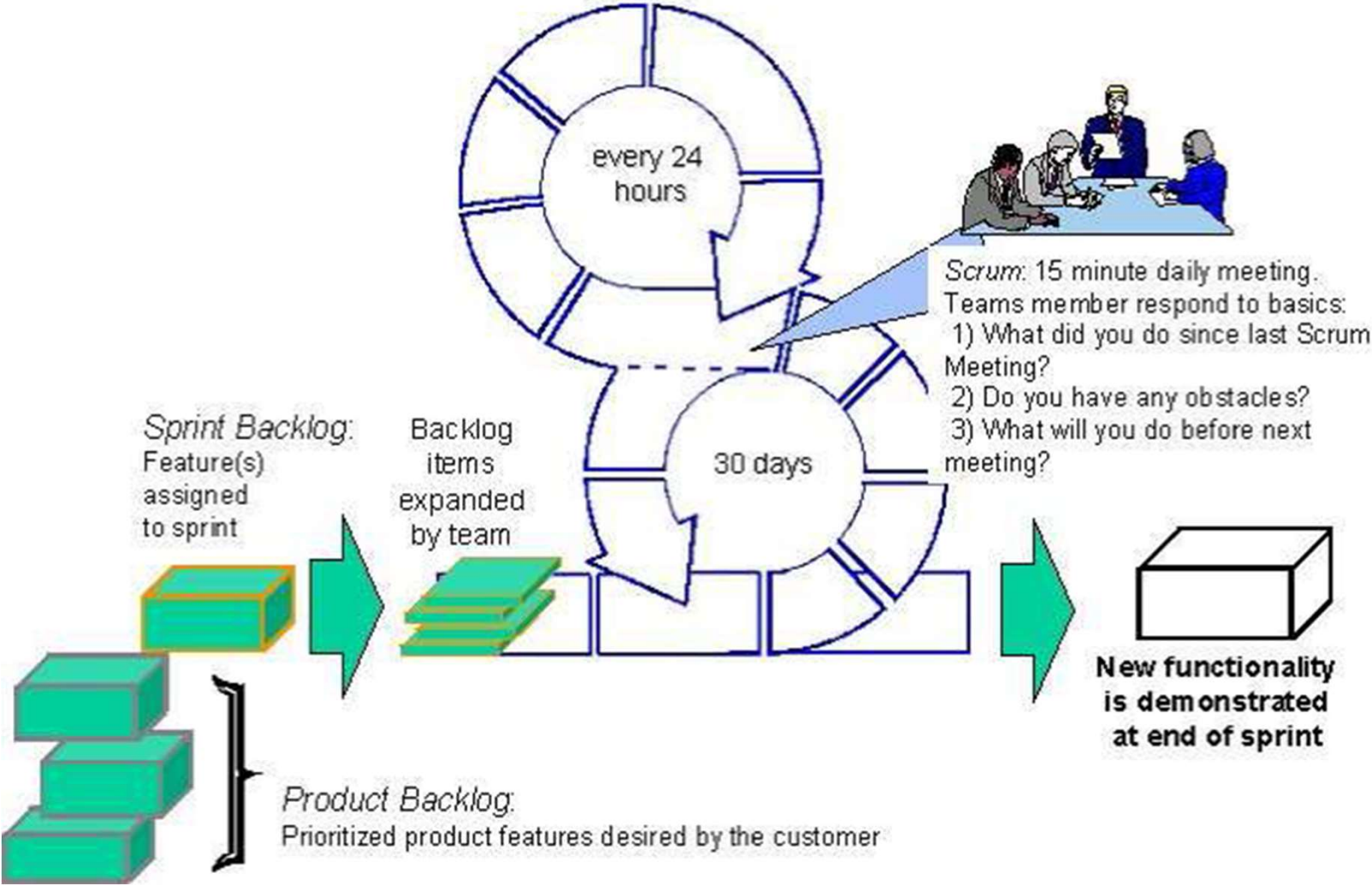
---

## Development (Game) phase

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



# SCRUM: sprint e schedule quotidiano



# SCRUM: fase 3

---

- Post-game phase (contiene la chiusura definitiva della release)
  - La gestione conclude il processo di sviluppo e il prodotto viene preparato per il rilascio. Questo include:
    - integrazione,
    - test,
    - documentazione per l'utente,
    - formazione e
    - preparazione del materiale di marketing.

# Tre ruoli: Product Owner

---

Tale ruolo è occupato da quella persona a cui fanno riferimento tutti i soggetti interessati al progetto, compreso il cliente finale.

Figura di raccordo in grado di effettuare stime, aggiustare i processi che presentano difetti e di gestire l'intero procedimento secondo la pianificazione inizialmente fatta.

Poteri :

- Accettare o rigettare i risultati di un lavoro
- Terminare uno sprint se necessario

# Tre ruoli: Membro del team

## Responsabilità:

- Costruiscono il prodotto
- Decidono cosa fare in ciascuno Sprint

## Caratteristiche:

- Diverse specializzazioni, ma anche
- Cross-functional (l'eccessiva specializzazione rischia di avere persone cariche di lavoro e altre che aspettano)
- Team organizzati indipendentemente
- Senza project (or team) manager
- Ognuno realizza una cosa alla volta (no multi-tasking)

## Team:

- 7 + 2 persone
- Se è possibile nella stessa sede/ufficio

## Cross-functional teams (1)



# Tre ruoli: Scrum Master

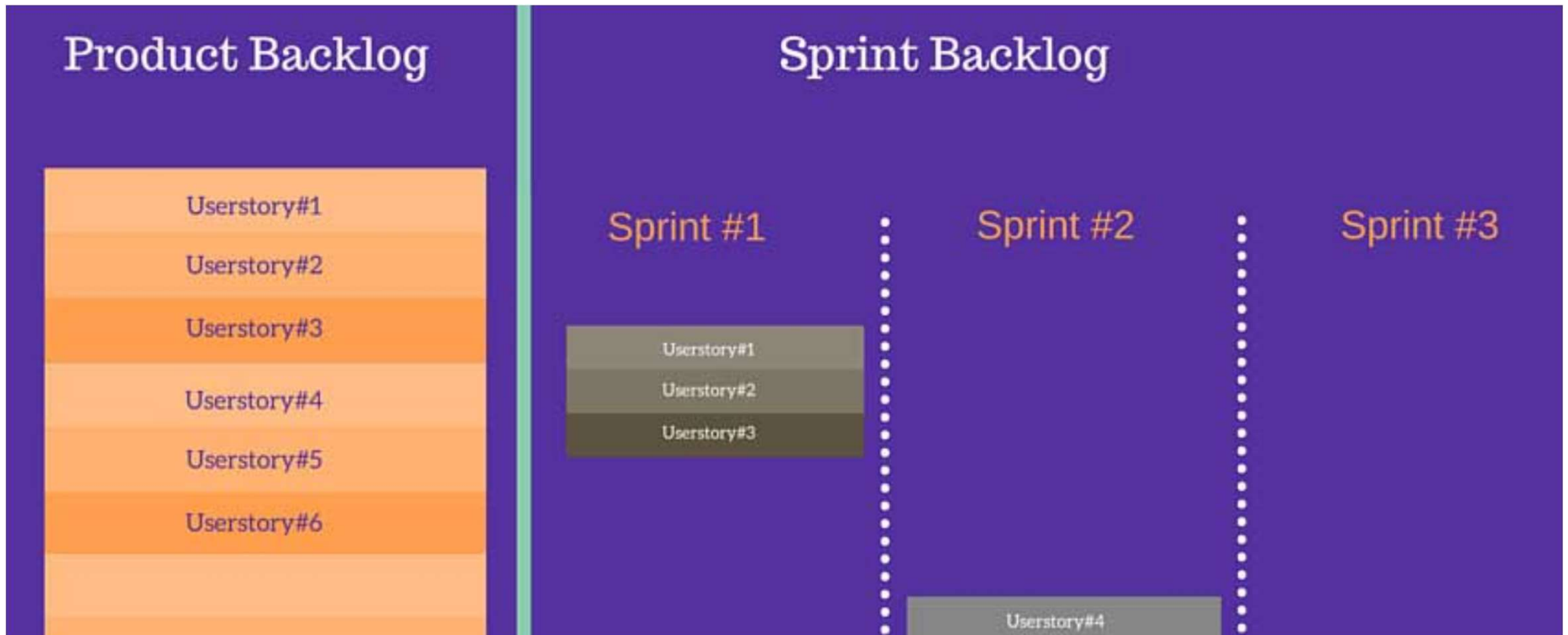
---

- Figura che si occupa di supportare il team
  - garantendo le condizioni ambientali e le motivazioni necessarie ad eseguire al meglio il lavoro commissionato
- Non ha autorità sul team



# SCRUM: backlog

---



# SCRUM: kanban





# SCRUM: kanban





# SCRUM: kanban con WIP limit



WIP sta per Work In Progress

Il WIP limit, indicato con un numero in cima a una colonna, limita la quantità di Post-Its che possono essere presenti nella colonna

# SCRUM: kanban con WIP limits

---

Gli WIP limits consentono di completare più velocemente i singoli elementi di lavoro:

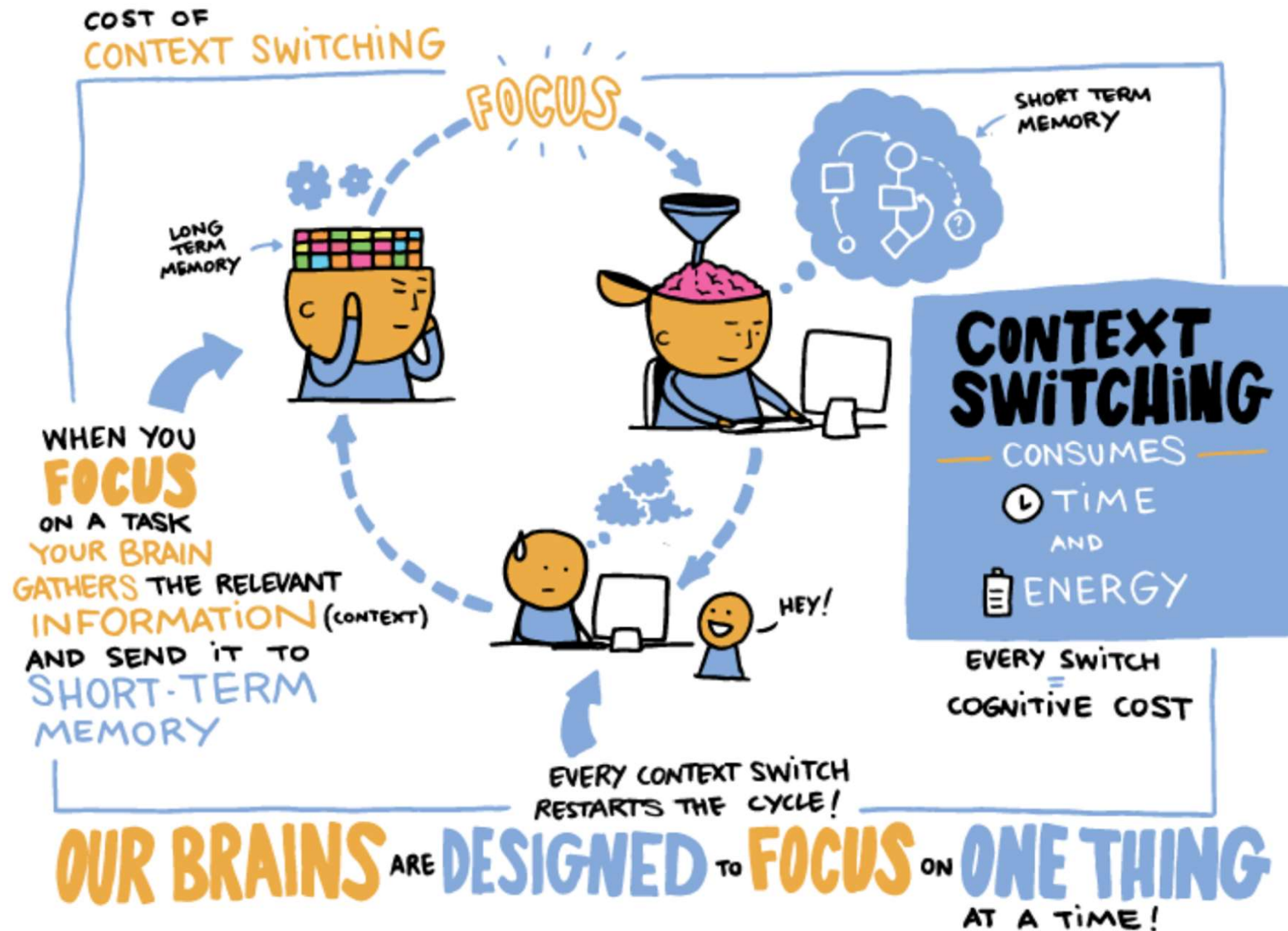
Un importante prerequisito per fornire **valore ai clienti il più rapidamente possibile**.

Il team ha la possibilità di individuare **i colli di bottiglia** nei processi di lavoro prima che diventino un blocco.

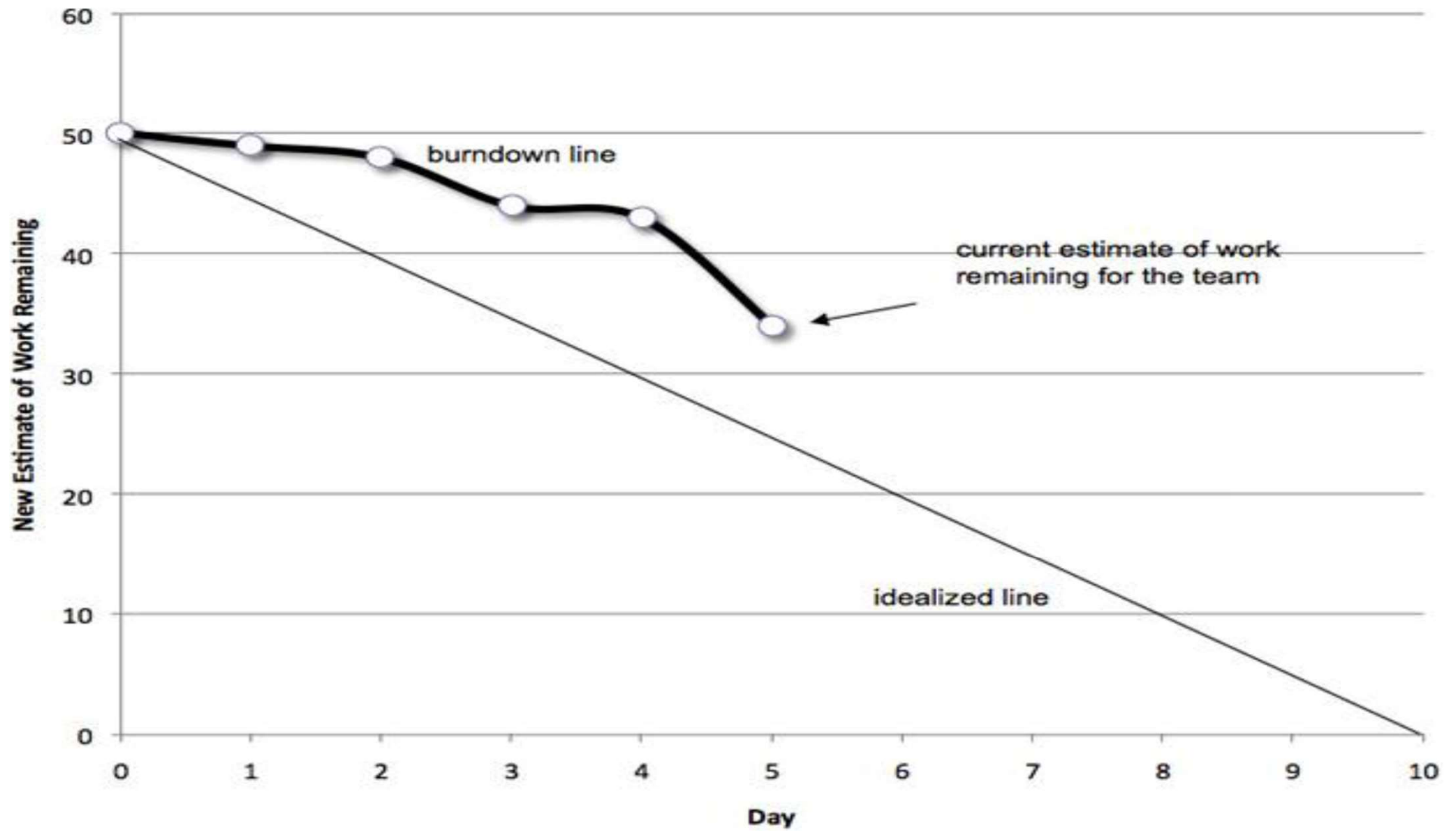
## **Ridurre il task switching**

Completare un task richiede molto più tempo se l'autore riceve contemporaneamente notifiche e-mail con un avviso sul desktop o simili. Ogni notifica interrompe la concentrazione e i processi di pensiero in corso

# Pericoli del context switching



# SCRUM: backlog vs burndown chart



# Eventi di uno sprint



# Eventi di uno sprint: planning

---



## Planning:

- Il product owner gestisce l'evento di pianificazione dello Sprint
- Con il Team si definisce lo sprint backlog



# Eventi di uno sprint: Daily meeting

---



I membri del team si posizionano in semicerchio davanti alla kanban, gli Scrum master si posizionano nelle vicinanze. Si risponde alle domande:

- "Cosa ho fatto ieri?",
- "Cosa farò oggi?"
- "Quali difficoltà sto affrontando?"

# Eventi di uno sprint: Review

---



Incontro di 4 ore, una volta consegnato un incremento di prodotto

Il team in collaborazione con gli utenti ispeziona il software per ottenere un feedback. Si discutono domande come:

- "È questo il prodotto che vogliamo costruire?".
- "Cosa penserebbero gli utenti finali del prodotto?"
- "Qual è il feedback degli utenti finali?"
- "È ancora il prodotto che ci è stato richiesto?".
- "Ci sono cambiamenti o nuove idee?".



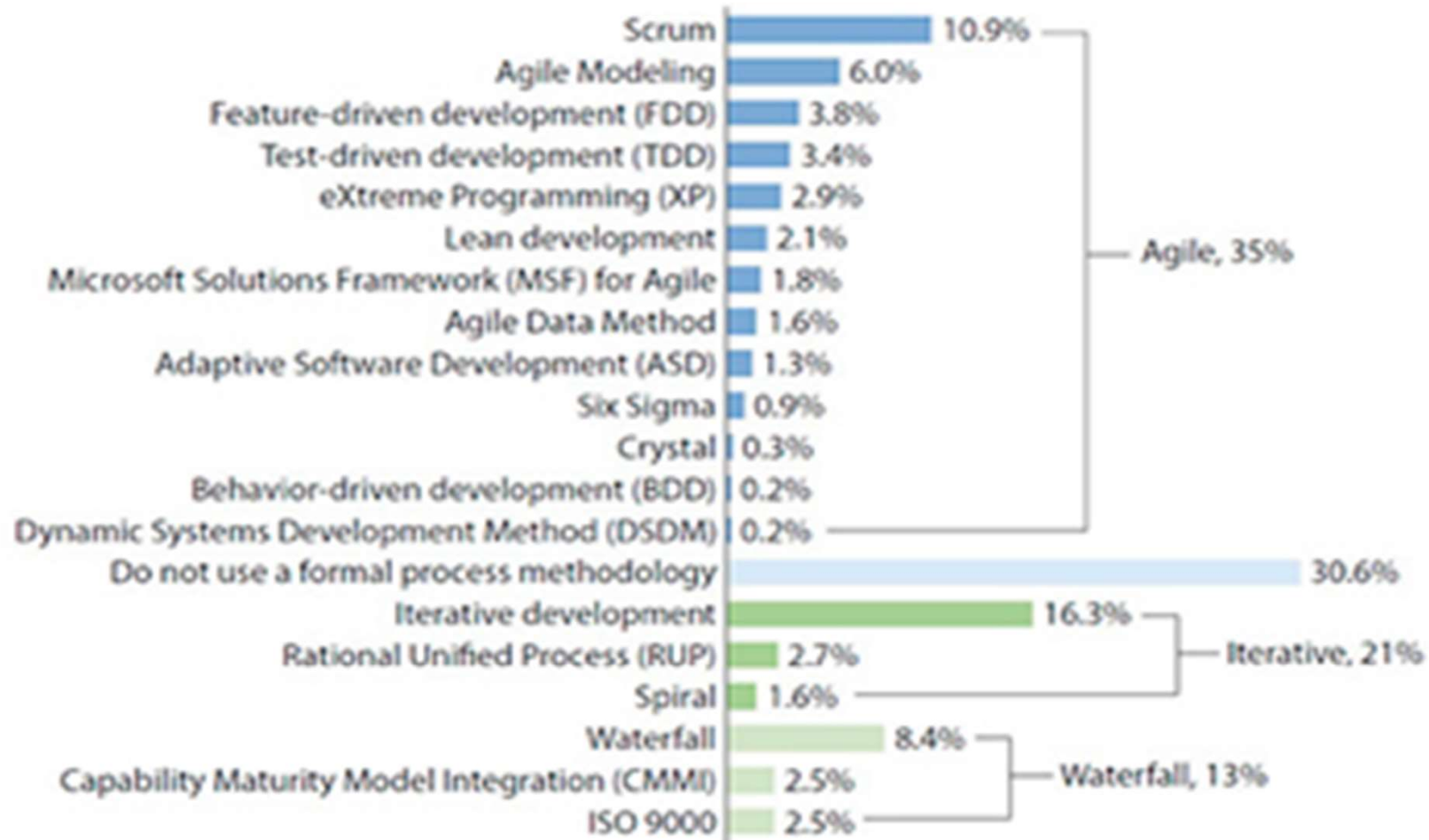
# Eventi di uno sprint: retrospettiva

---



Dopo ogni Sprint, il team Scrum organizza anche un evento di retrospettiva dello Sprint di 3 ore, in modo che l'intero team Scrum possa dedicare un po' di tempo a riflettere, imparare e riadattarsi per lo Sprint successivo. L'intero scopo dell'evento di retrospettiva dello Sprint è quello di contribuire al miglioramento continuo o alla crescita continua del team Scrum.

# Una rassegna



Base: 1,298 IT professionals

Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2009

Source: Forrester Research, Inc.

# Riassunto: nuove idee introdotte dai modelli di ciclo di vita

---

Build-and-Fix: un non-modello → forget it

## Modelli prescrittivi

- Cascata → definizione delle fasi, documentazione
- Modello a V → prima idea di test driven development
- Rapid Prototyping → iterativo con requisiti poco chiari
- Modello incrementale → iterativo con requisiti chiari ma prodotto rilasciato "a puntate".
- Modello a spirale → analisi dei rischi

## Unified Process

→ fasi vs iterazioni

## Modelli agili

→ "ribellione alle prescrizioni" + molte nuove idee

- Extreme Programming
- Scrum

# Bibliografia

---

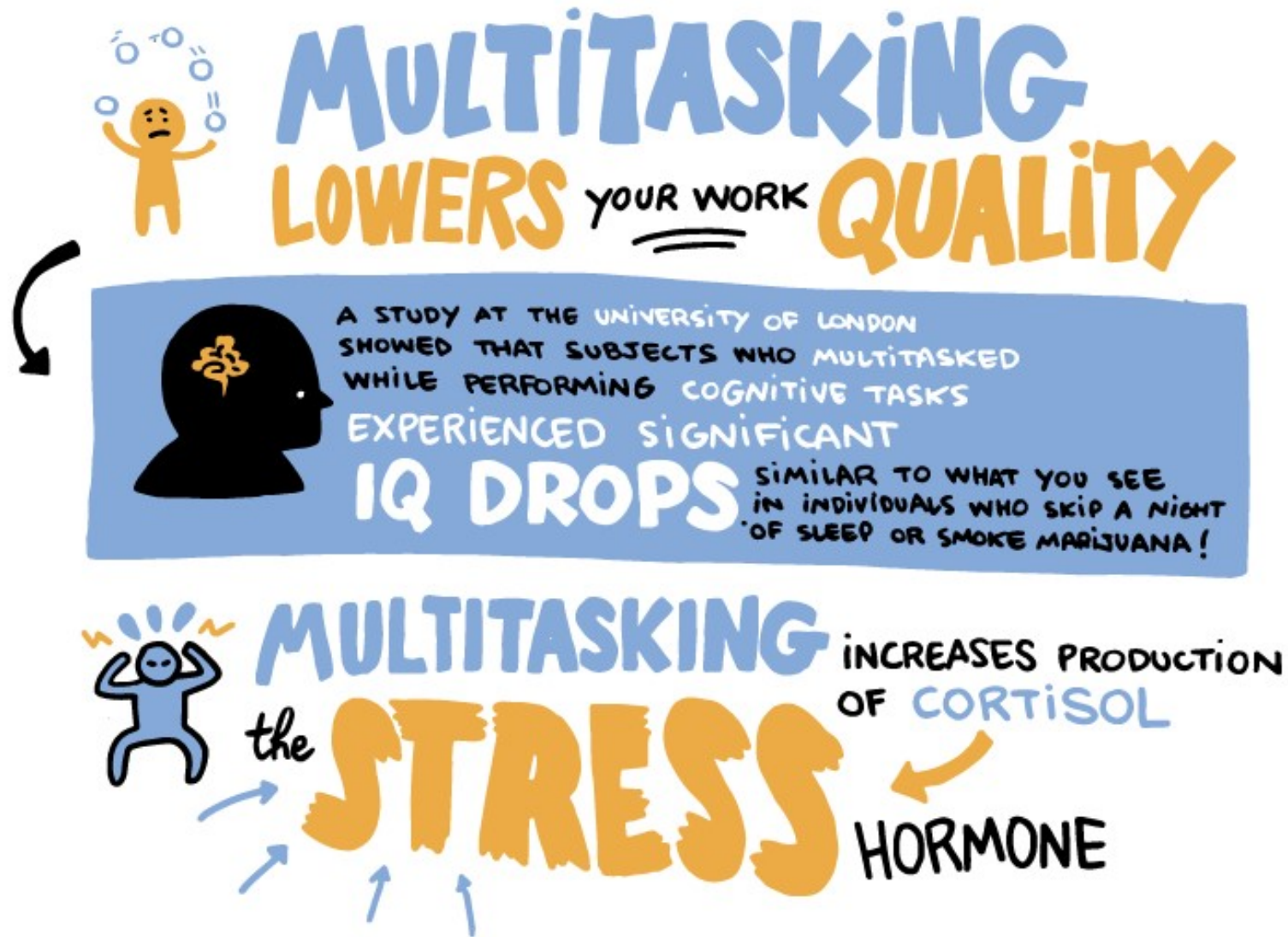
**Object Oriented and Classical Software Engineering** ,  
Stephen R.Schach, Fifth edition Cap 1,3 e 10

**Object-Oriented Software Engineering**,  
David C. Kung Cap 2

Altre letture che potrebbero interessarvi

- Il manifesto Agile di Snowbird
- Ken Schwaber e Jeff Sutherland, [\*La Guida a Scrum. La Guida Definitiva a Scrum: Le Regole del Gioco\*](#) ([PDF](#)), Scrum.Org and ScrumInc, 2014.

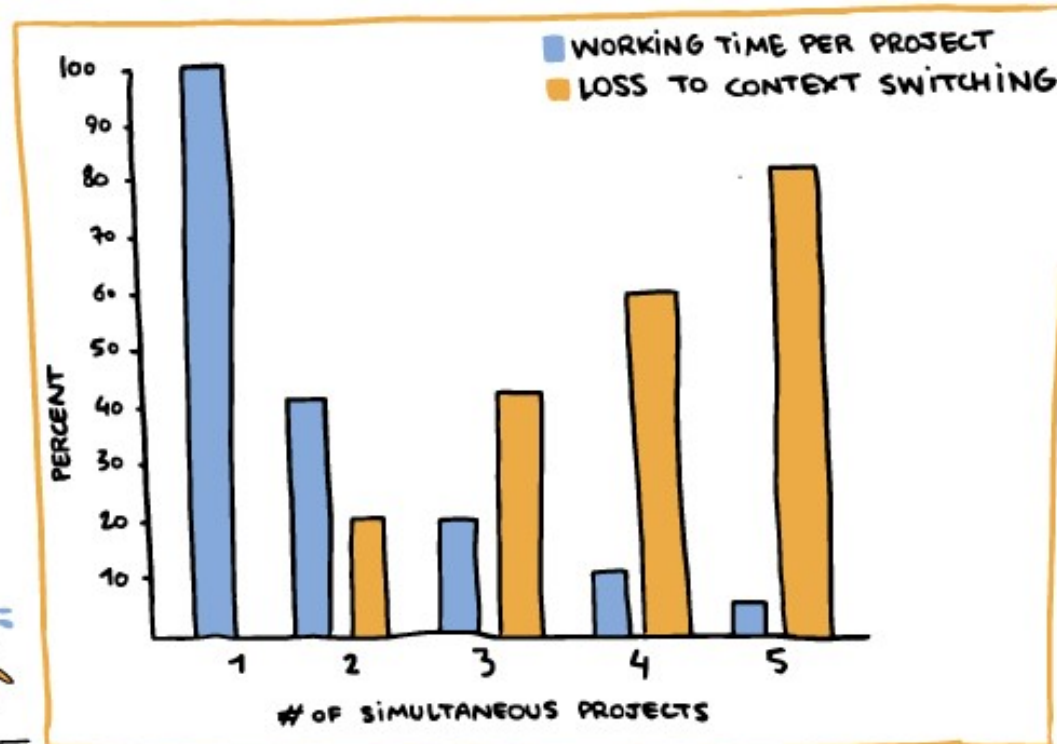
# No al multitasking



# Limitare WIP

EVERY PROJECT YOU ADD TO YOUR WORKLOAD RESULTS IN A

**20% DROP** IN **PRODUCTIVITY**



WITH 5 SIMULTANEOUS PROJECTS ONLY 5% OF WORK TIME IS AVAILABLE FOR EACH PROJECT!

↓  
**WASTE OF**  
- TIME  
- MONEY  
- PRODUCTIVITY

SOURCE: QUALITY SOFTWARE MANAGEMENT (G. WEINBERG)



# Homework

---

**STOP STARTING**  
**START**  
**FINISHING**

# Homework

---

Una delle sfide principali nell'adozione della Kanban è la limitazione del work-in-progress (WIP).

Questa sera definite i task di domani (backlog)

Fissate un WIP limit e cercate di non sgarrare

Suggerite su teams una strategia/app per limitare il multitasking

WHY YOU SHOULD  
**LIMIT**  
WORK *in* - PROGRESS  
*and*  
**STOP**  
MULTITASKING