02. Il processo software

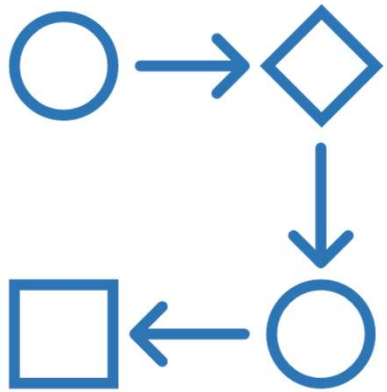
IS 2024-2025

Corso di Laurea in Informatica
Dipartimento di Informatica
Università of Pisa

DALLA PUNTATA PRECEDENTE ☺

Con **processo software** si intende il modo con cui produciamo il software

- Comincia quando iniziamo ad esplorare il problema
- Finisce quando il prodotto viene ritirato dal mercato
- Strutturato in **fasi**:
  - Analisi dei requisiti
  - Specifica
  - Progettazione
  - Implementazione
  - Integrazione
  - Mantenimento
  - Ritiro



- Riguarda tutti gli **strumenti**, le **tecniche** e i **professionisti** coinvolti nella varie fasi

PROCESSO SOFTWARE

Processo software = sequenza di attività necessarie a sviluppare un sistema software

Esistono diversi (modelli di) processi software, ma tutti includono

- **specifica**, aka. definizione di cosa deve essere fatto
- **progettazione** e **implementazione**
- **validazione**, aka. verifica che il sistema faccia quanto richiesto
- **evoluzione**, aka. modifica/aggiornamento del sistema

MODELLARE PROCESSI SOFTWARE

Il **modello** di un processo software fornisce una **rappresentazione astratta** del processo stesso

- **Suddivisione** in attività (cosa fare, quando, chi, e cosa si ottiene)
- **Organizzazione** delle attività (ordinamento, criteri per terminare un'attività e passare alla successiva)
- Esempio: Standard ISO 12207

Esempio (non software): Preparazione di un dolce da forno

- Modello di sviluppo **generico**:
  - ✓ fare la spesa
  - ✓ impastare e mettere nella teglia mentre il forno si scalda
  - ✓ infornare
  - ✓ (quando pronto) rimuovere dal forno

- NB: Non è la ricetta di un dolce, ma una sequenza di attività **valida** per un **qualunque dolce** da forno

MODELLI DI CICLO DI VITA DEL SOFTWARE

Build-and-Fix (un «non-modello»)

Modelli prescrittivi

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

Unified Process

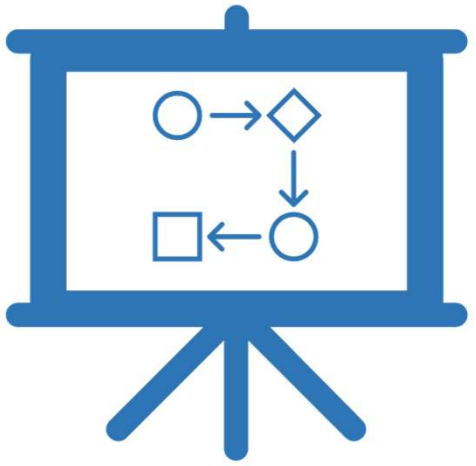
Modelli agili

- Extreme Programming
- Scrum
- Continuous Delivery

COSÌ TANTI?

Li vedremo tutti perché

- non solo perché sono i più noti, ma
- soprattutto perché ognuno di essi ha affrontato un **nuovo aspetto**



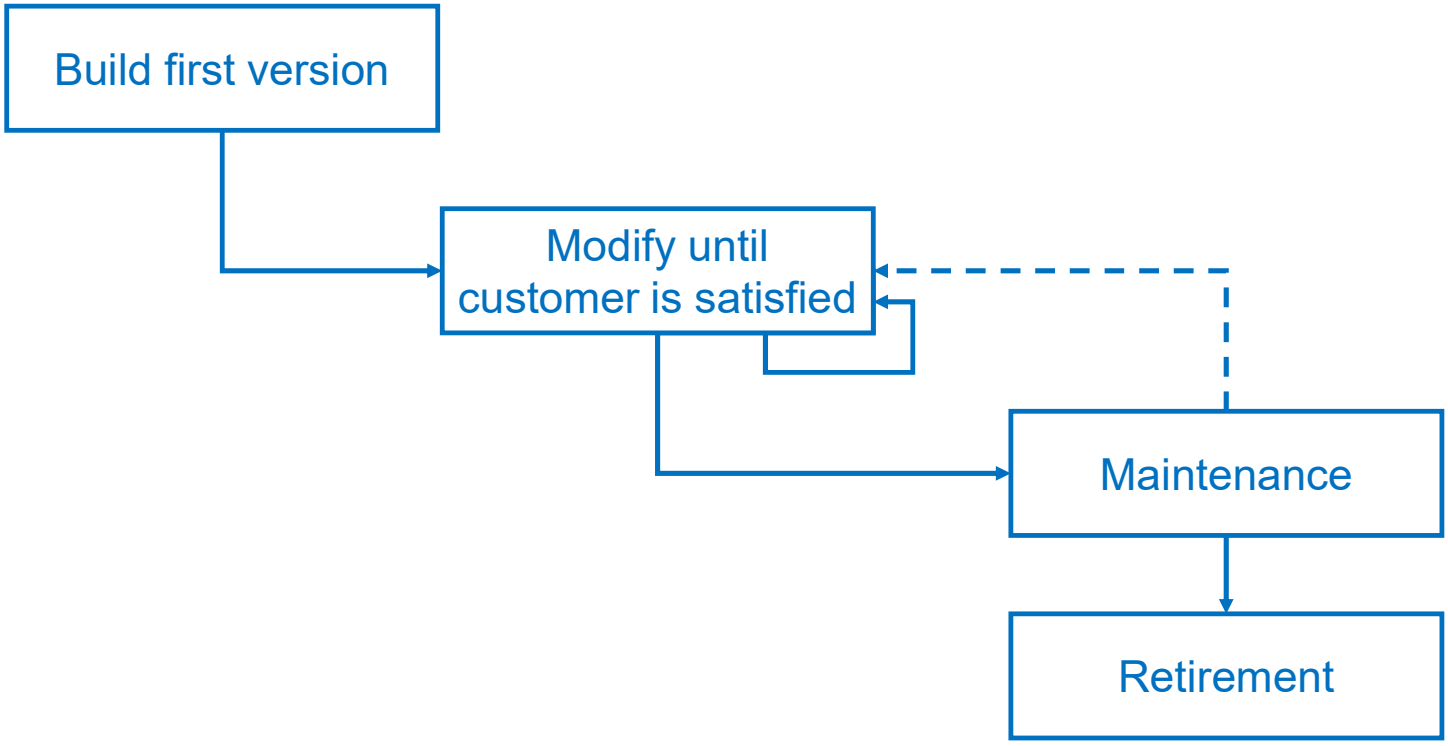
Gli aspetti peculiari sono quelli che ci interessano di più

- Guidano lo sviluppatore nella scelta di un ciclo di vita per ogni progetto
- Un ciclo di vita di un progetto reale può combinare idee di diversi modelli

BUILD-AND-FIX

Sistema software sviluppato **senza specifica/progettazione**

- lo sviluppatore scrive un programma
- che poi è modificato più volte finché non soddisfa il committente



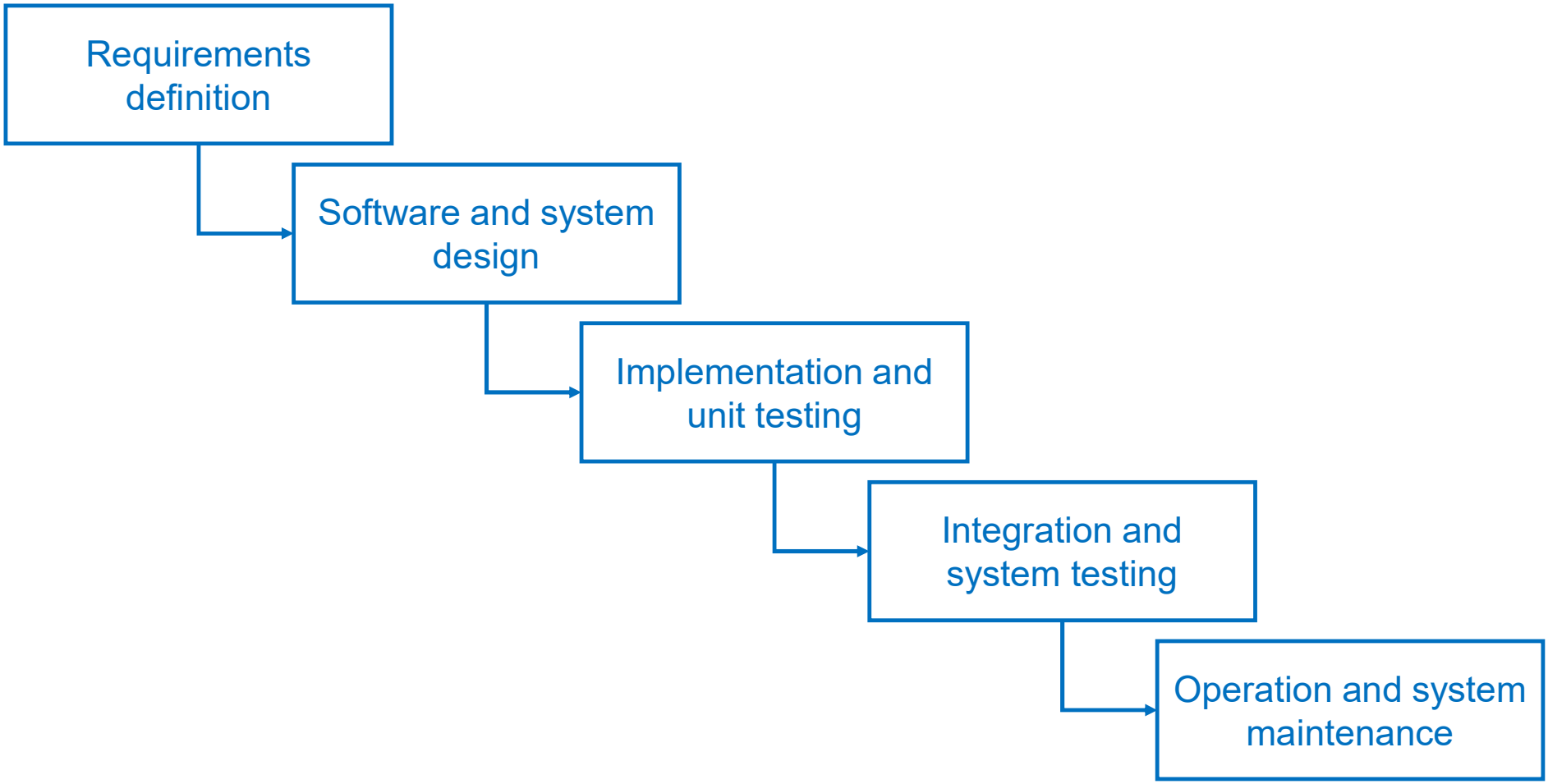
BUILD-AND-FIX (CONT.)

Quando usarlo?

- Forse adeguato a un progetto di 100 linee di codice
- Improponibile per prodotti di ragionevole grandezza
  - La manutenzione senza specifica o documentazione della progettazione è estremamente difficile (⇒ costosa)

Meglio scegliere un «vero» modello del ciclo di vita quando si inizia lo sviluppo di un progetto, per evitare sorprese

MODELLO A CASCATA [ROYCE, 1970]



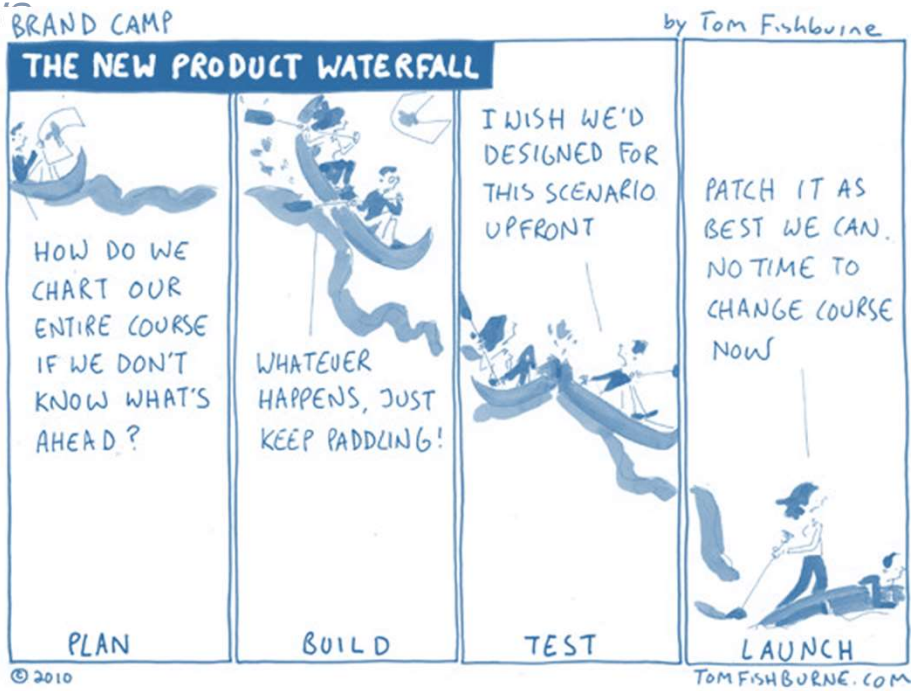
MODELLO A CASCATA: PRO

Primo modello a **distinguere e definire le fasi di un processo software**

- Importanza di **analisi e progettazione** prima della codifica
- Passaggio a nuova fase possibile solo dopo **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”

Contro: Eccessiva produzione di **documenti** e modello «**rigidamente in cascata**»

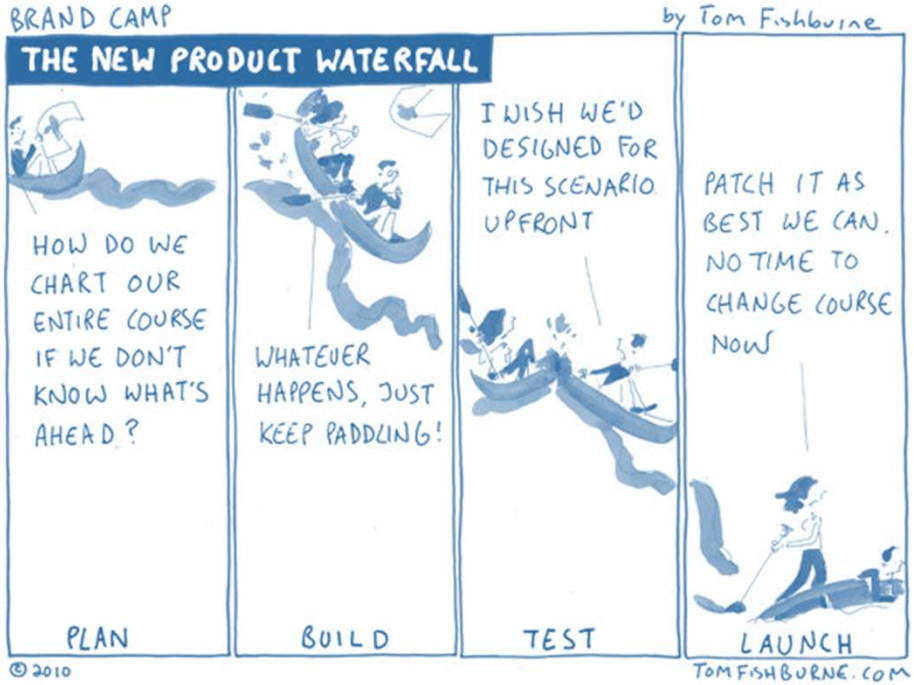
- Manca l’interazione col cliente (che vede solo il prodotto finito)
- Spesso c’è una sostanziale differenza tra come il cliente immagina un prodotto e come il prodotto viene realizzato
- A quel punto, se ci sono discrepanze (e ci sono!) tutto il processo deve essere ripetuto



MODELLO A CASCATA: CONTRO

Modello «**rigidamente in cascata**»

- Manca l'interazione col cliente (che vede solo il prodotto finito)
- 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



MODELLO A CASCATA: CONTRO

Modello «**rigidamente in cascata**»

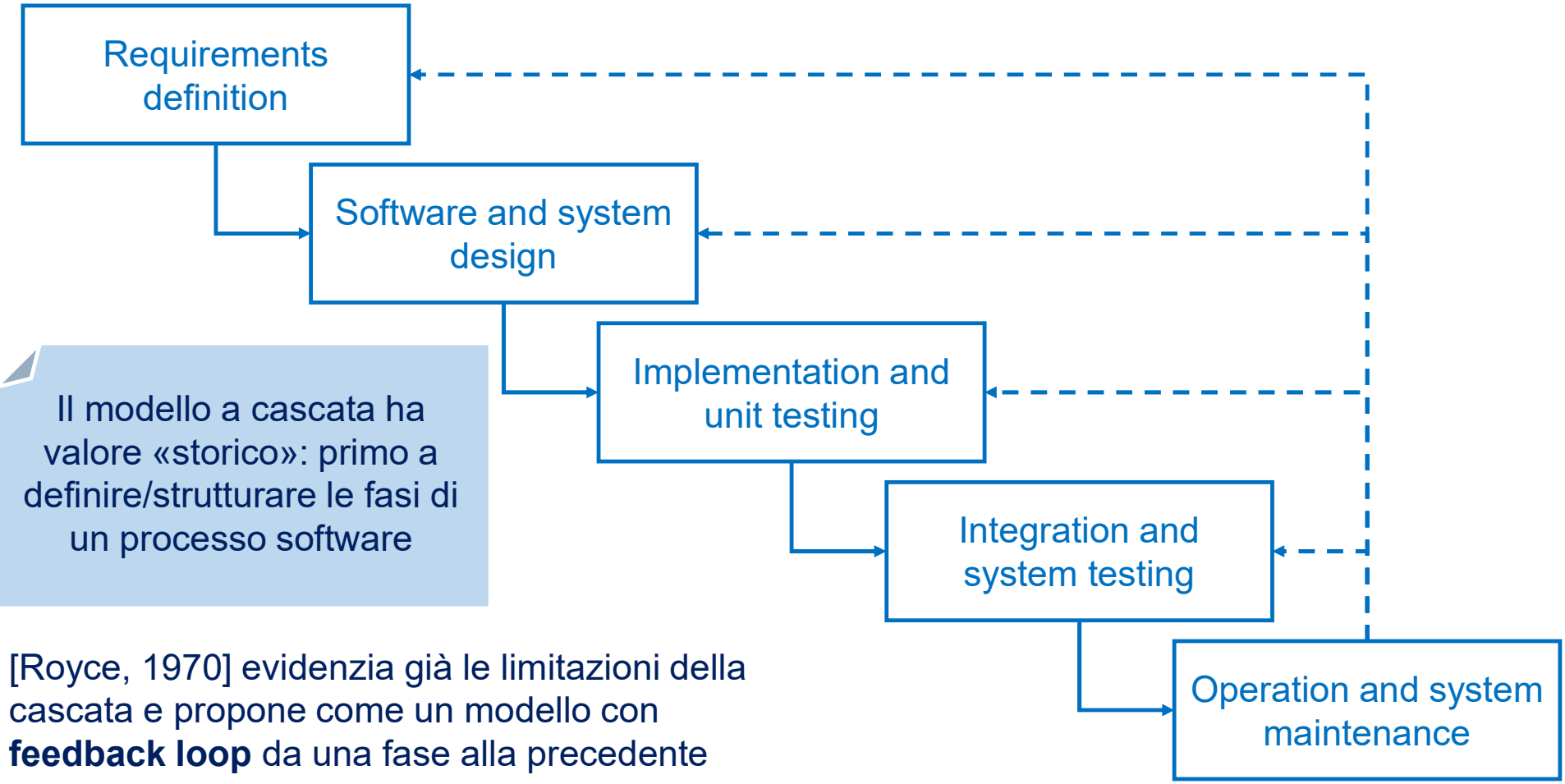
- Manca l'interazione col cliente (che vede solo il prodotto finito)
- Spesso c'è una sostanziale differenza tra come il cliente immagina un prodotto e come il prodotto viene realizzato
- A quel punto, se ci sono discrepanze (e ci sono!) tutto il processo deve essere ripetuto

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, ovvero **definire e strutturare le fasi di un processo software**

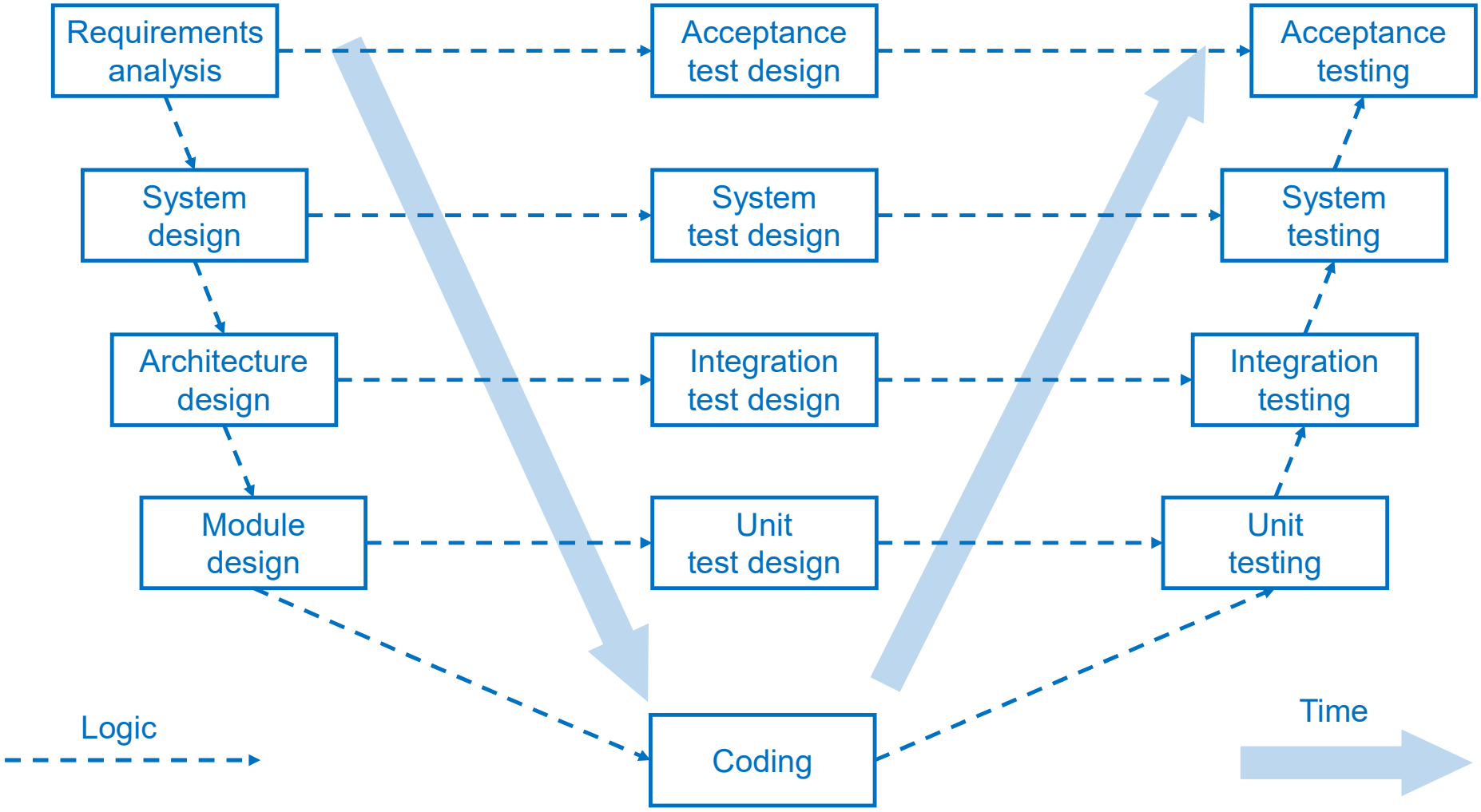
DIAMO A ROYCE QUEL CHE È DI ROYCE ☺



Il modello a cascata ha valore «storico»: primo a definire/strutturare le fasi di un processo software

[Royce, 1970] evidenzia già le limitazioni della cascata e propone come un modello con **feedback loop** da una fase alla precedente

MODELLO A V


Requirements analysis
System design
Architecture design
Module design
Coding
Unit test design
Integration test design
System test design
Acceptance test design
Unit testing
Integration testing
System testing
Acceptance testing
Logic
Time

MODELLO A V (CONT.)

Il modello a V prende la **metà inferiore** del modello a cascata e la **piega verso l'alto** a forma di V

- Le **attività di dx** verificano e **convalidano** i prodotti delle **attività di sx**
- **SX**: attività di **analisi** che scompongono le esigenze degli utenti in parti piccole e gestibili
- **DX**: attività di **sintesi** che aggregano (e testano) tali parti per verificare che il sistema soddisfi le esigenze degli utenti

E al **centro**?

- **Progettazione dei test** da svolgere nella parte dx
  ⇒ Evidenzia come sia possibile progettare i test **durante le fasi di sviluppo** (e prima della codifica)
  ⇒ Attività di **progettazione del test** come parte del **processo software**

Il modello V è uno degli **standard SQA** (assicurazione della qualità del software) e viene utilizzato per descrivere le attività di test nel processo software
(idea poi ripresa nel *test driven development*)

MODELLI SEQUENZIALI VS ITERATIVI

Modelli sequenziali
(fondamentalmente)

Modello a cascata

Modello a V

Modelli iterativi

Rapid prototyping

Modello incrementale

Modello a spirale

RAPID PROTOTYPING

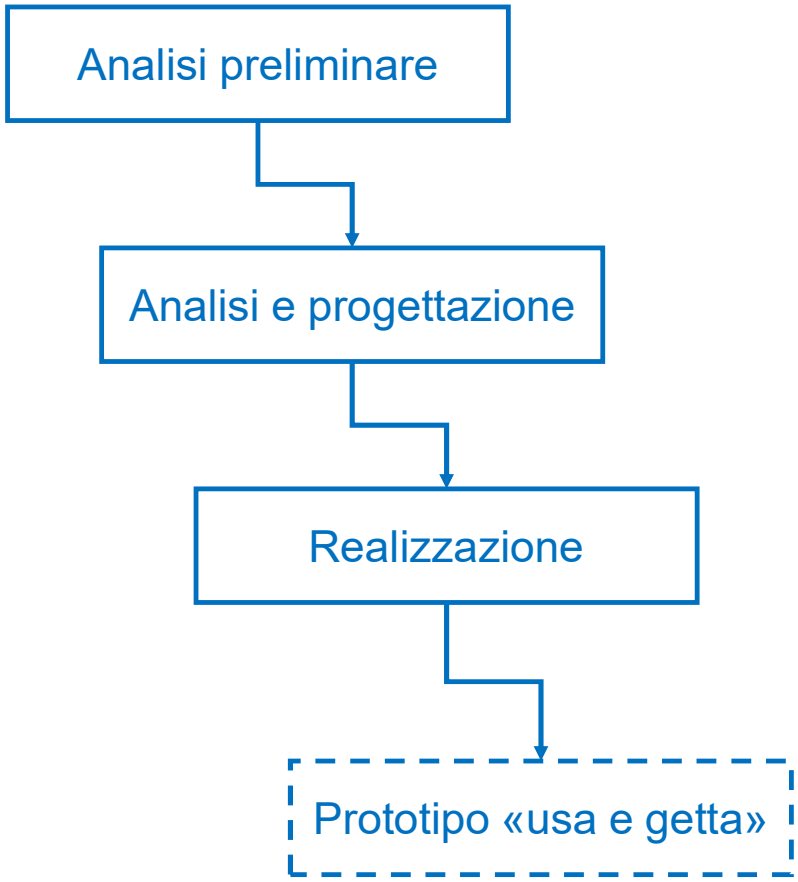
Costruire velocemente un **prototipo** per permettere al **committente** di **sperimentarlo**

- Aiuta il cliente a meglio descrivere i requisiti
- Si passa poi alla fase di specifica

Quando usarlo?

Utile se i **requisiti** inizialmente **non sono chiari**

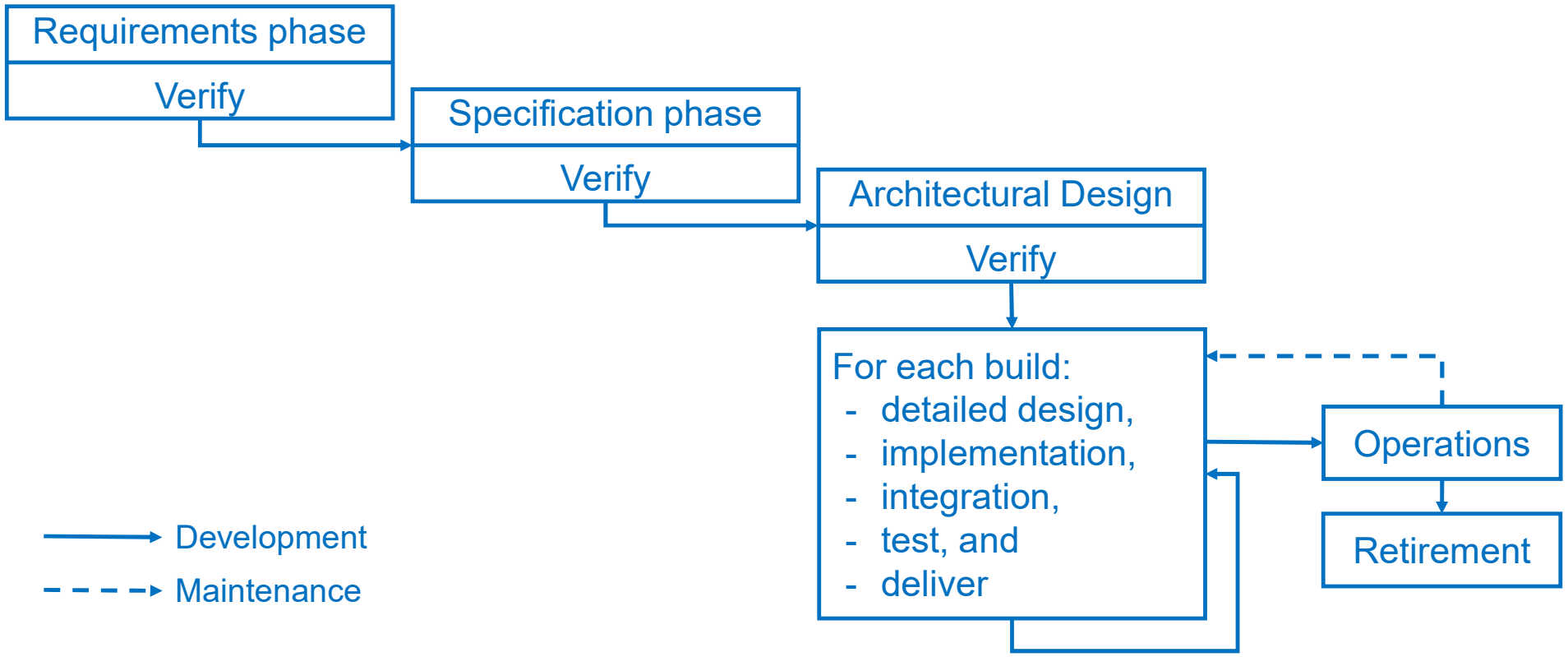
Anche noto come modello evolutivo



MODELLO INCREMENTALE

Il sistema è costruito **iterativamente** aggiungendo nuove funzionalità

- I **requisiti** del progetto sono definiti **inizialmente**
- Il sistema è implementato, integrato e testato con **passaggi incrementali**



Fino a qui martedì 17/9

MODELLO INCREMENTALE: PRO E CONTRO

Pro (anche rispetto al modello a cascata)

- Consente di **ritardare** la realizzazione di **componenti** che dipendono criticamente da fattori esterni (e.g., teconologie o hardware sperimentale)
- Si riesce ad «**uscire**» **rapidamente** con prime versioni del sistema
  - Sistema utilizzabile in tempi brevi (seppure con funzionalità ridotte)
  - Permette di ottenere **feedback** dai clienti sul lavoro di sviluppo svolto
  - Riduce il **costo** per soddisfare **cambiamenti** di esigenze/requisiti

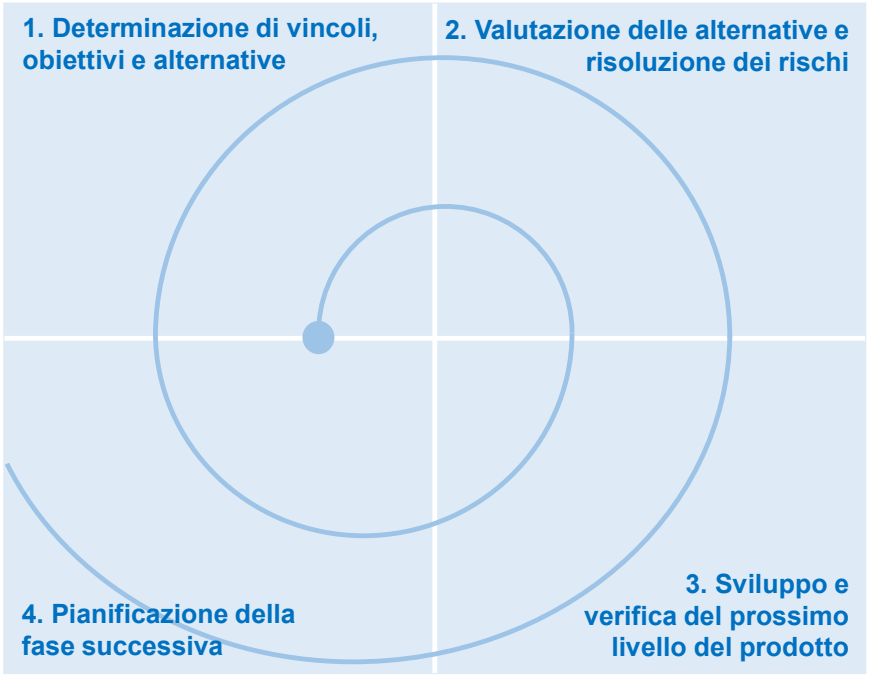
Contro

- Processo software **poco visibile**
- Se non implementato bene, diventa «**build-and-fix**»
  - La **struttura** del sistema tende a **degradarsi** con i nuovi incrementi (e diventa sempre più difficile e costoso introdurre ulteriori modifiche)
  - Necessario spendere tempo e risorse in **refactoring**

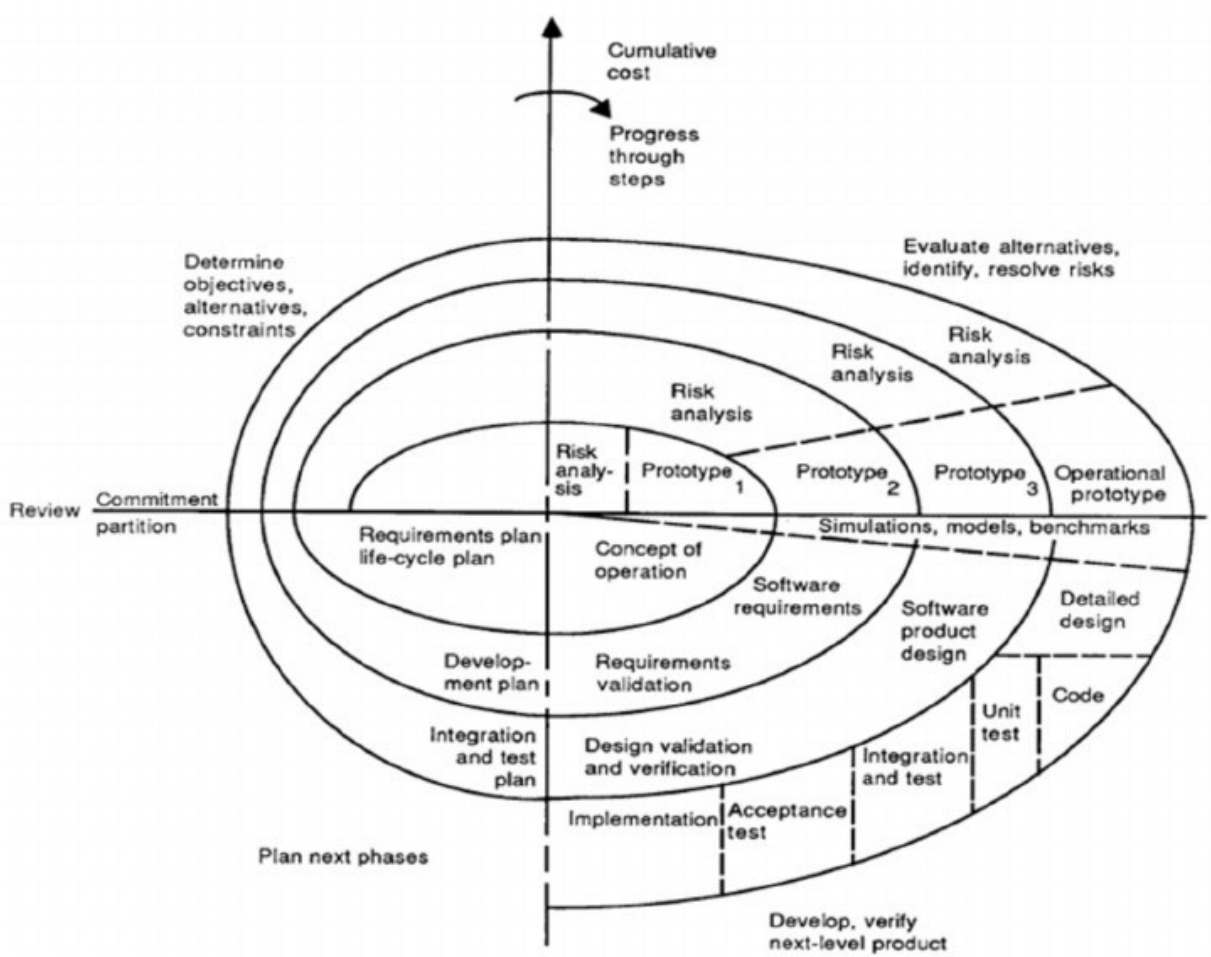
MODELLO A SPIRALE [BOHEM, 1988]

Ogni **iterazione** è divisa in **quattro fasi**:

1. Definizione degli obiettivi
2. Analisi dei rischi
3. Sviluppo e validazione
4. Pianificazione del nuovo ciclo



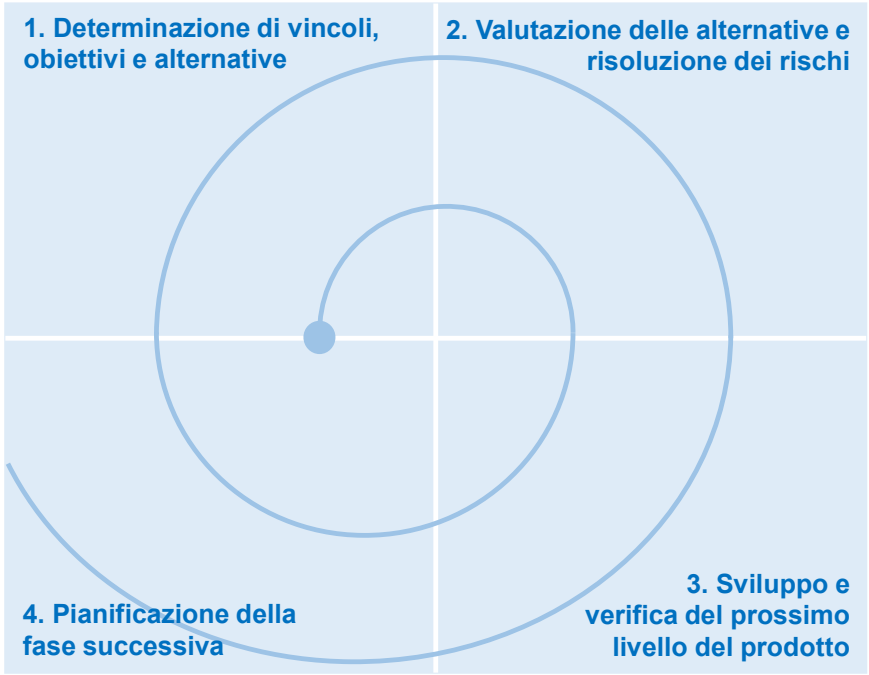
RAPPRESENTAZIONE ORIGINALE DI BOHEM



MODELLO A SPIRALE [BOHEM, 1988]

Modello **astratto** da istanziare

- cosa fare in ogni iterazione e in ogni fase
- applicabile ai cicli tradizionali



Evidenzia gli aspetti **gestionali**:

- Pianificazione delle fasi
- Modello «**risk-driven**» (ovvero guidato dall'analisi dei rischi – come, ad esempio, 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**

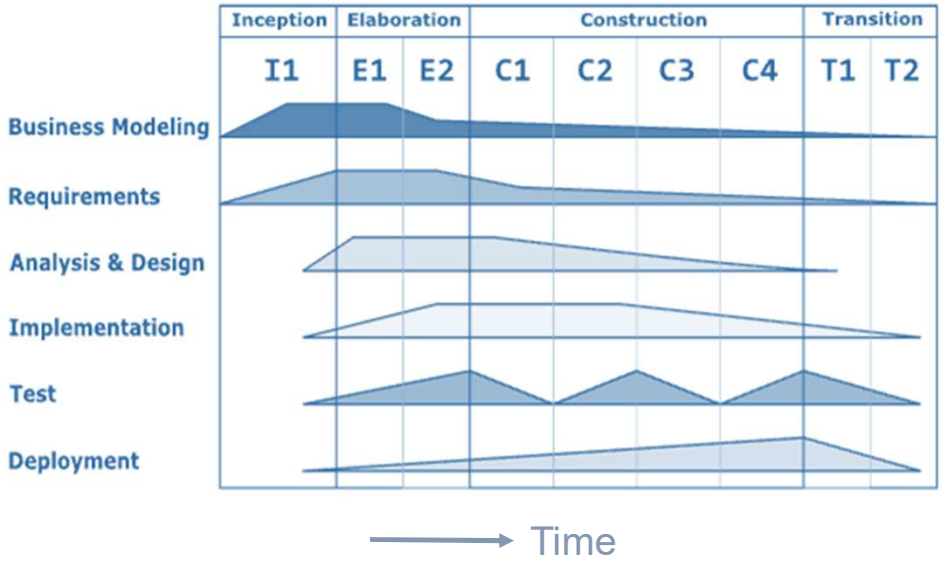
UNIFIED PROCESS [BOOCH ET AL, 1999]

Guidato da **casi d'uso** e **analisi dei rischi**

- Sin dalla fase di raccolta e analisi dei requisiti

Modello **iterativo incrementale** incentrato sull'**architettura**

- Nelle prime fasi, definizione dell'architettura di massima
- Dettagli lasciati alle fasi successive
- Consente di avere subito una visione generale del sistema, facilmente adattabile al cambiamento dei requisiti



RICAPITOLANDO

Un processo software organizza attività (tecniche, collaborative e gestionali) al fine di specificare, progettare, sviluppare e validare un sistema software

Le attività fondamentali di un processo software sono:

- Specifica
- Progettazione
- Sviluppo
- Validazione
- Evoluzione

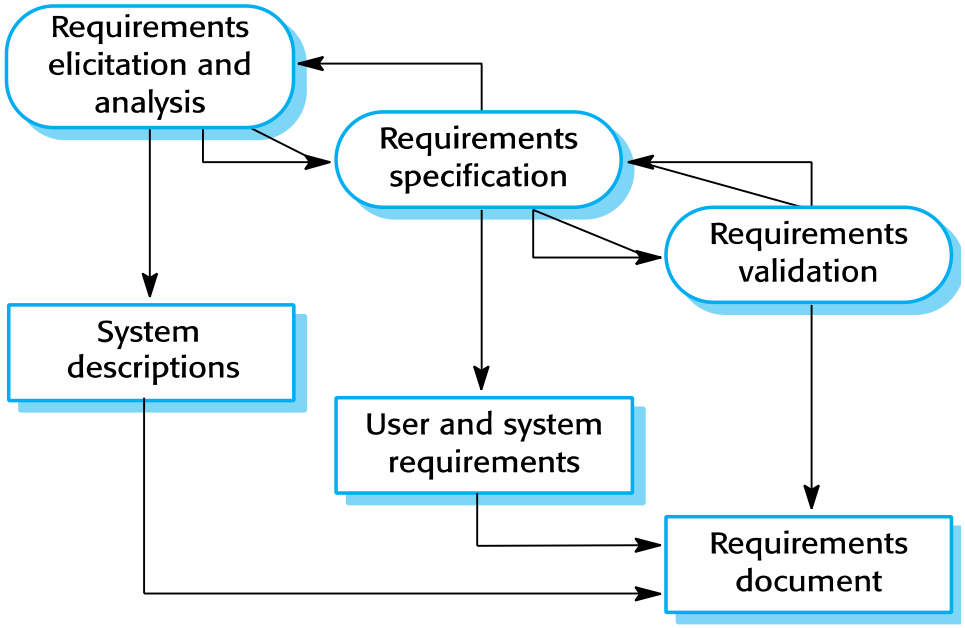
(organizzate in modo diverso nei diversi modelli di processo esistenti – ad esempio, in sequenza nel modello a cascata, iterativamente nel modello incrementale)

LA FASE DI SPECIFICA

Stabilisce quali **servizi** sono richiesti e i **vincoli** sul funzionamento del sistema

Processo di **ingegneria dei requisiti**:

- Estrazione e analisi dei requisiti
  - Cosa richiedono o si aspettano gli stakeholder dal sistema?
- Specifica dei requisiti
  - Definire i requisiti in dettaglio
- Convalida dei requisiti
  - Verifica della validità dei requisiti



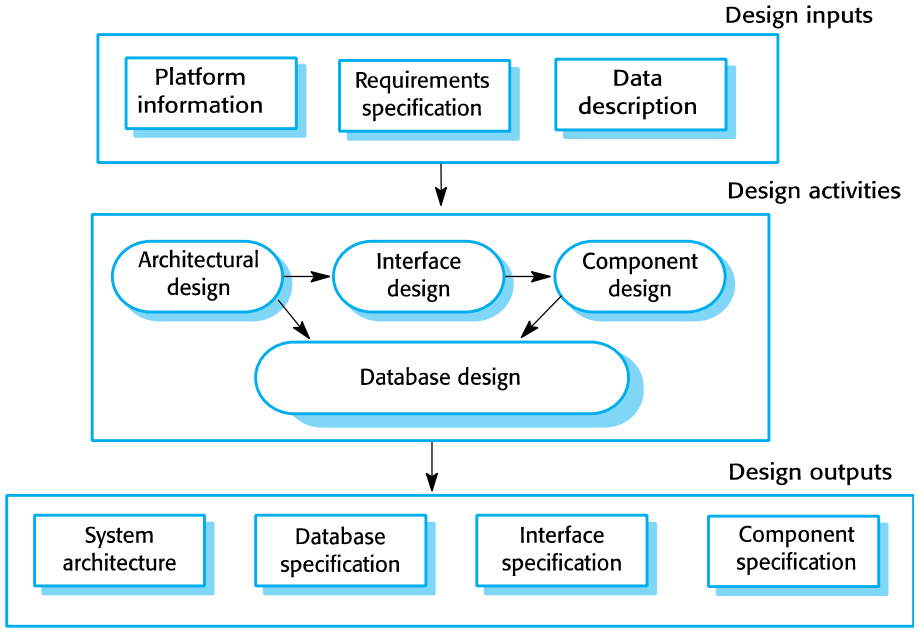
Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

LA FASE DI PROGETTAZIONE

Definizione di una **struttura** software che realizzi la specifica

- ***Architectural design***: identificazione della struttura del sistema (in termini di componenti, relazioni e loro distribuzione)
- ***Database design***: definizione delle strutture dati e della loro rappresentazione in database
- ***Interface design***: Definizione delle interfacce tra i componenti del sistema
- ***Component design***: Definizione (di dettaglio) dei componenti del sistema, eventualmente identificando quelli realizzabili mediante riuso di componenti esistenti

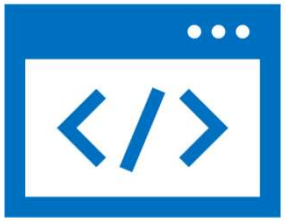
NB: Inizia la conversione della specifica in un sistema eseguibile



Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

LA FASE DI SVILUPPO

La struttura software progettata nella fase di design è realizzata **implementando** uno o più programmi e/o configurando sistemi applicativi esistenti



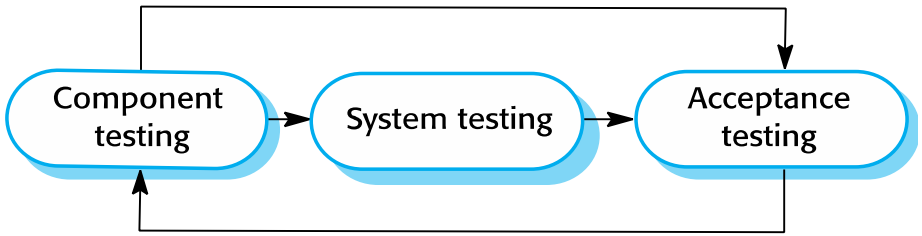
Attenzione:

- Le fasi di **progettazione** e **sviluppo** sono spesso eseguite in **interleaving** nella produzione di sistemi software
- La fase di sviluppo include:
  - Programmazione: attività individuale (eventualmente svolta in cooperazione) senza un processo standard
  - Debugging: attività volta ad identificare e correggere eventuali errori/bug introdotti durante la programmazione

LA FASE DI VALIDAZIONE

Verifica e **validazione** (V&V) hanno lo scopo di dimostrare che un sistema è conforme alle sue specifiche e soddisfa i requisiti del cliente

- Spesso fatta mediante **test** del sistema
- Il test del sistema prevede l'esecuzione del sistema con casi di test derivati dalla specifica dei dati reali che devono essere elaborati dal sistema
  - **Component testing**: i componenti sviluppati sono testati individualmente e in modo indipendente
  - **System testing**: il sistema è testato nella sua interezza (con particolare attenzione verso le cosiddette *emergent properties)*
  - **Customer testing**: il sistema è testato utilizzando dati forniti dal cliente per verificare che rispetti le necessità del cliente stesso



Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

LA FASE DI EVOLUZIONE

Cambiamenti ed **evoluzioni** sono inevitabili nei sistemi moderni, ad esempio:

- I requisiti possono cambiare nel tempo (e.g., nuove esigenze di business)
- Nuove tecnologie possono consentire di migliorare un sistema esistente

I cambiamenti comportano ***rework*** (costoso in termini di tempo e risorse)

- Ripetizione (di una parte) dell'analisi dei requisiti e loro aggiornamento
- Progettazione e sviluppo di nuove funzionalità
- Sviluppo di adattamenti di funzionalità esistenti

Necessità di **anticipare i cambiamenti** per ridurre i costi del rework

- **Change tolerance**: il processo software può organizzare le attività in modo che i cambiamenti siano implementabili a costi contenuti)
- Più facile con un modello incrementale, con i cambiamenti implementati
  - In nuovi incrementi (se riguardano funzionalità ancora da implementare)
  - Su sistemi parziali/solo sulle parti interessate (se riguardano parti già implementate)

EVOLUZIONE: ALTRA DIMENSIONE DA CONSIDERARE

La **rapidità di sviluppo e rilascio** è considerata sempre più fondamentale nei sistemi software

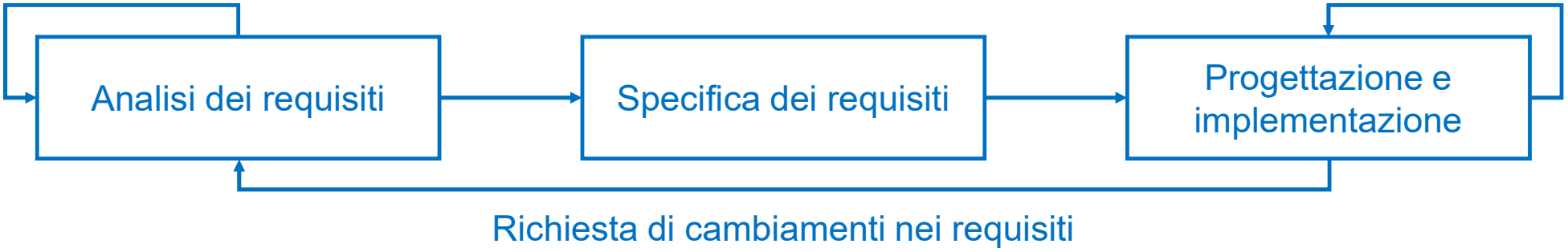
- **Fast-changing** business requirement
- Praticamente impossibile fissare un insieme stabile di requisiti software
- Necessità di soddisfare rapidamente i cambiamenti nei requisiti di un sistema software (aka. **evoluzione rapida**)

Quale modello di ciclo di vita usare?

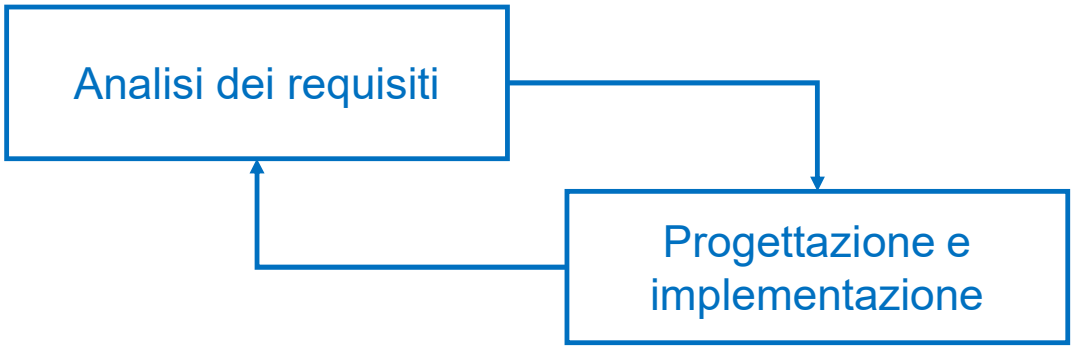
- Quelli visti finora sono essenziali per alcuni tipi di sistemi, ma non permettono di soddisfare quanto elencato sopra
- Il modello **agile** (proposto alla fine degli anni 90) mira proprio a ridurre il tempo di rilascio dei sistemi software
  - Le fasi di specifica, progettazione e sviluppo sono eseguite in interleaving
  - Il sistema è sviluppato come una serie di version incrementale con gli stakeholder coinvolti nella specifica e nella valutazione di tali version
  - Rilascio frequente per valutazione nuove funzionalità
  - Molti strumenti a supporto dello sviluppo (e.g., *automated* testing)

PLAN-DRIVEN VS AGILE

Plan-driven



Agile



AGILE MANIFESTO[1]

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

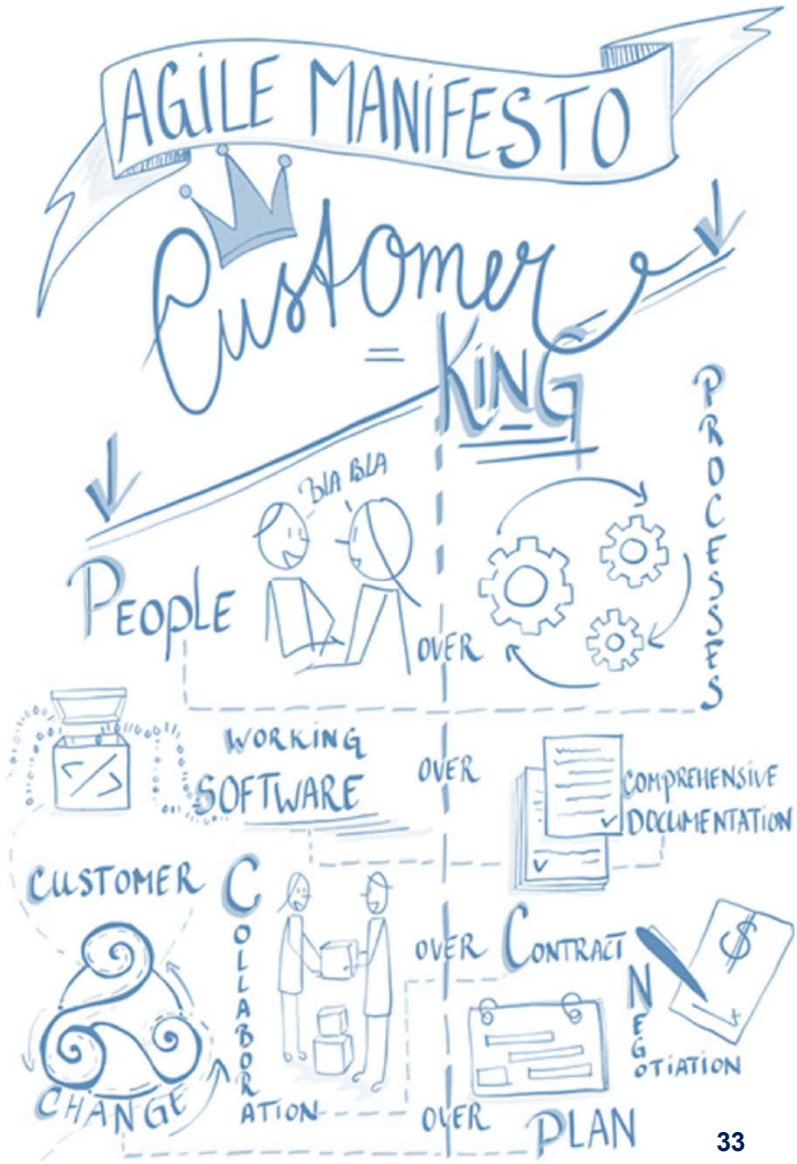
Individuals and interactions over *processes and tools*

Working software over *comprehensive documentation*

Customer collaboration over *contract negotiation*

Responding to change over *following a plan*

That is, while there is value in the items on the right, we value the items on the left more



1. https://agilemanifesto.org

AGILE: PRINCIPI FONDAMENTALI

Customer involvement: I clienti devono essere coinvolti durante il processo di sviluppo (per fornire e prioritizzare nuovi requisiti, e valutare le iterazioni del sistema)

Incremental delivery: Il sistema software è sviluppato in versioni incrementali, con il cliente che specifica i requisiti da soddisfare in ciascuna versione

People not process: Le abilità del team di sviluppo devono essere riconosciute e sfruttate, lasciando i team di sviluppo liberi di sviluppare «a modo loro» (senza forzarli a seguire un processo)

Embrace change: Accettare che i requisiti del sistema cambieranno nel tempo e progettare il sistema in modo da favorire tali cambiamenti

Maintain simplicity: Focalizzarsi sulla semplicità sia nel software sviluppato sia nel processo di sviluppo (elimando la complessità dal sistema ogni volta che questo è possibile)

AGILE: INTERPRETAZIONE DEI PRINCIPI

Comunicazione // customer involvement + people not process

- persone e **interazioni** 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. *backend developer* con il cliente

- collaborare **direttamente** con i clienti
  - «oltre il contratto»
  - migliori risultati dei rapporti contrattuali

AGILE: INTERPRETAZIONE DEI PRINCIPI

Semplicità // maintain simplicity

- mantenere una descrizione formale del sistema più semplice e chiara possibile
- avere software funzionante è più importante di una sua documentazione estensiva
- bisogna mantenere il codice semplice e avanzato tecnicamente (a discapito di documentazione, mantenuta al minimo indispensabile)

AGILE: INTERPRETAZIONE DEI PRINCIPI

Feedback // customer involvement + incremental delivery

- 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 (invece di pianificare tutto all'inizio e aderire a tale piano)

AGILE: APPLICABILITÀ

Quando **conviene** essere **agile**?

- Sviluppo di prodotti di piccola/media dimensione (fino a ~50 sviluppatori)
- Sviluppo di sistemi custom all'interno di un'organizzazione
  - Disponibilità da parte dei clienti ad essere coinvolti nel processo di sviluppo
  - Meno regole/regolamenti esterni a cui il software deve aderire

Quando **viene** (realmente) **usato** agile?

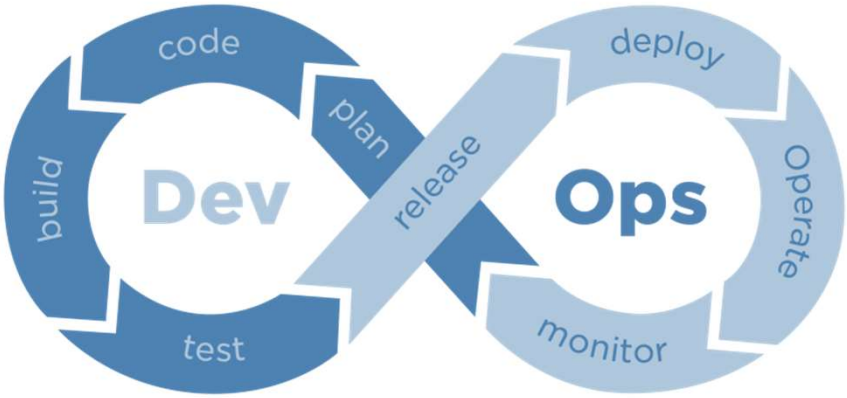
- Praticamente, quasi tutti i prodotti/app moderni sono sviluppati con un approaccio agile

AGILE & CI/CD

La diffusione di **agile** ha portato alla diffusione di pratiche di **Continuous Integration** e **Continuous Deployment** (**CI/CD**)

- **CI**: integrazione continua di tutte le modifiche/aggiunte fatte da un team di sviluppo all'interno di un «*main branch*»
  - Validata attraverso strumenti di *automated build/testing*
  - Produce continuamente nuove versioni di un software

- **CD**: dispiegamento continuo (e automatico) delle nuove versioni di un software ottenute tramite *continuous integration*



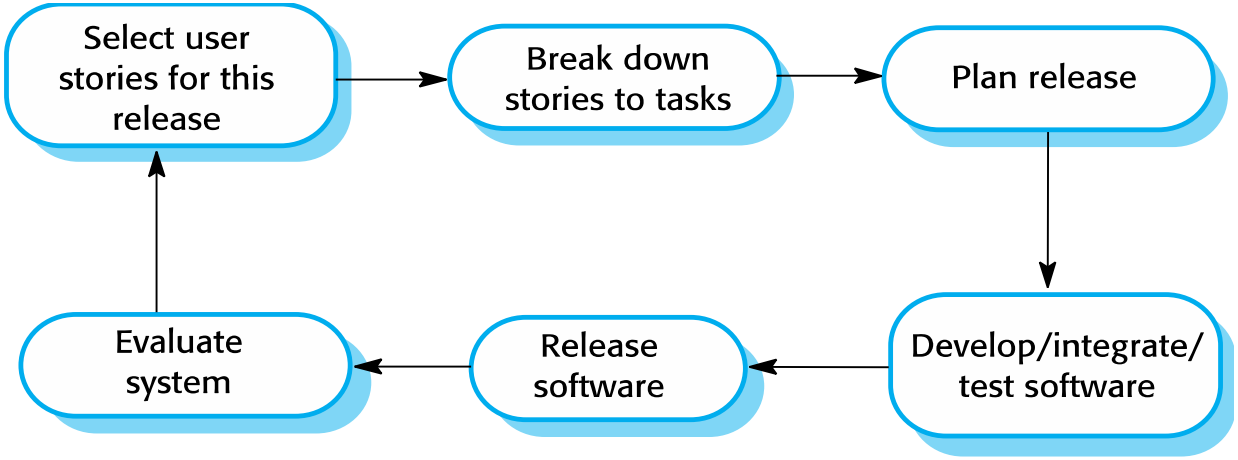
DevOps (contrazione di «development» e «operations»): modello incentrato su team in cui comunicano, collaborano e si integrano sia sviluppatori sia addetti alle operations

ESSERE AGILE: EXTREME PROGRAMMING (XP)

XP è un approccio «estremo» allo **sviluppo** iterativo e agile del software

- Nuove versioni di un sistema software sviluppate più volte in un giorno
- Versioni incrementali rilasciate al cliente ogni 2 settimane
- Tutti i test sono eseguiti per ogni build e una build è accettata solo se tutti i test sono passati con successo

XP adotta il seguente **ciclo di rilascio**



Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

XP: PRATICHE COMUNI

Planning incrementale // per **incremental delivery**

- I requisiti sono raccolti sotto forma di **user stories**
- Le user stories da includere in una release sono determinate sulla base del tempo disponibile e della loro relativa priorità
- Gli sviluppatori dividono le user stories in **task**

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

XP: PRATICHE COMUNI (CONT.)

Release piccole // per **incremental delivery** e **embrace change**

- Prima versione (minimale) che fornisca le funzionalità di base del sistema
- Release frequenti e incrementali di nuove funzionalità

Design semplice // per **maintain simplicity**

- Progettazione volta a soddisfare solamente i requisiti correnti
- Comprensibile a tutti

Test-first development // per **incremental delivery** e **embrace change**

- Casi di test prima del codice
- Spesso generati con framework per unit test automatici

Refactoring // per **people not process** e **maintain simplicity**

- Refactoring continuo da fare non appena ci si accorge di possibili miglioramenti
- Codice semplice, facilmente manutenibile e auto-esplicativo (motto: «se un metodo richiede un commento, riscrivilo!»)

XP: PRATICHE COMUNI (CONT.)

Pair programming // per **people not process**

- Gli sviluppatori lavorano in coppia
- Verifica continua e supporto vicendevole nel processo di sviluppo

Collective ownership // per **people not process**

- Le coppie di sviluppatori lavorano su tutte le aree del sistema software
- Tutti hanno responabilità su tutto il codice (tutti possono cambiare tutto)

Sustainable pace // per **people not process** e **maintain simplicity**

- No al (troppo) lavoro straordinario
- Assunzione: lavoro straordinario ⇒ calo in qualità del codice e produttività

On-site customer // per **customer involvement**

- Un rappresentante del cliente deve essere disponibile/parte del team XP
- Responsabile di fornire/prioritizzare nuovi requisiti del sistema

DIFFUSIONE E UTILIZZO DI XP

XP si concentra su aspetti (troppo) tecnici e per questo non è facilmente integrabile con le pratiche di gestione utilizzate nella maggior parte delle organizzazioni

Di conseguenza, sebbene gli approcci agili utilizzino pratiche proposte da XP, il metodo (così come è stato orginariamente proposto) non è molto diffuso/utilizzato nelle organizzazioni moderne

Alcune pratiche chiave che vengono però utilizzate sono

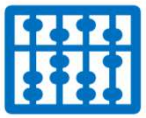
- User stories for specification
- Refactoring
- Test-first development
- Pair programming
- On-site customer (limitato, però, a interazioni periodiche)

ESSERE AGILE: SCRUM

Scrum è un metodo agile per lo **sviluppo** iterativo e incrementale di un sistema software

- Il nome deriva dalla terminologia usato per «mischia» nel gioco del rugby
- L'idea è quella di ottenere un processo in cui un insieme di persone si muove all'unisono per raggiungere un obiettivo che garantisce la soddisfazione di
  - ambizioni di squadra e
  - ambizioni personali

SCRUM: TRE FIGURE COINVOLTE



Product owner: Individuo (o piccolo gruppo) con il compito di identificiare le caratteristiche del prodotto software da sviluppare

- Elenca e prioritizza i requisiti
- Revisiona il **product backlog** (insieme dei requisiti conosciuti) per garantire che il progetto continui a rispettare quanto necessario al business

Scrum master: figura responsabile di assicurare che il processo **Scrum** sia seguito e portato avanti in modo efficace

- Garantisce le **condizioni ambientali** e le **motivazioni** necessarie ad eseguire al meglio il lavoro commissionato
- Si interfaccia con il resto della compagnia per assicurarsi che il team non soffra di interferenze esterne
- Non viene percepito come «project manager» (e **non ha autorità** sul team)

Development team: gruppo autogestito di sviluppatori software

- Non dovrebbe superare le 7 unità ⇐ principio del ***two-pizzas team***[1]
- **Responsabili** dello **sviluppo** del software e di altri documenti essenziali

1. https://martinfowler.com/bliki/TwoPizzaTeam.html

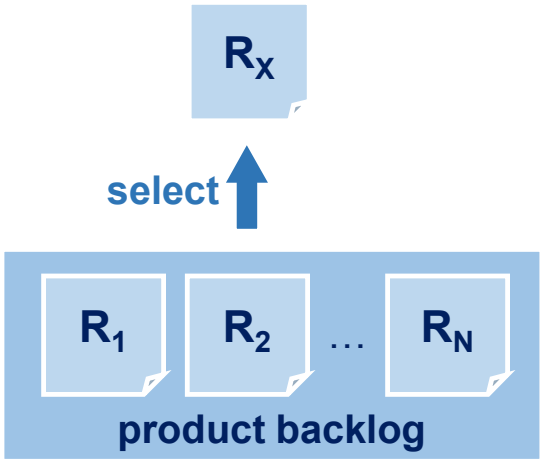
SCRUM: FASI

1) **Pre-game phase** (pianificazione «di massima»)
   - **Planning sub-phase**: Definizione del sistema che deve essere sviluppato, in termini di **product backlog** (contiene tutti i requisiti attualmente conosciuti)
   - **Architecture sub-phase**: Design di alto livello del sistema, inclusa l'**architettura**, in base agli elementi contenuti nel product backlog
2) **Game phase** (sviluppo)
   - 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

SPRINT

Gli sprint hanno **durata prefissata** (~2-4 settimane)

Si parte dal **product backlog**, che contiene la lista dei requisiti **TBD** (to be done), ovvero di ciò che ancora deve essere aggiunto al progetto



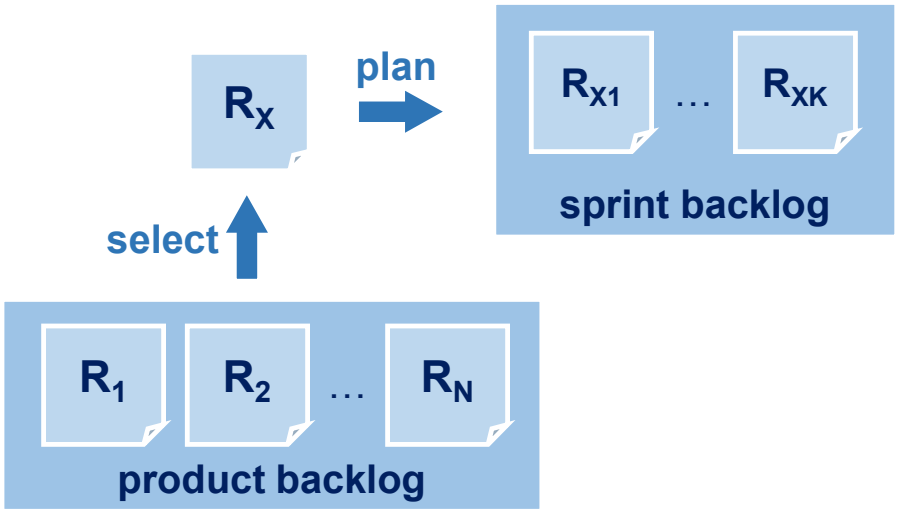
La fase di **selezione** coinvolge tutto il team che lavora al progetto e il cliente

→ Selezione di **cosa sviluppare** durante lo sprint

SPRINT (CONT.)

Si prosegue con la **pianificazione** dello sprint

- Gestito dal **product owner**
- Creazione dello **sprint backlog**

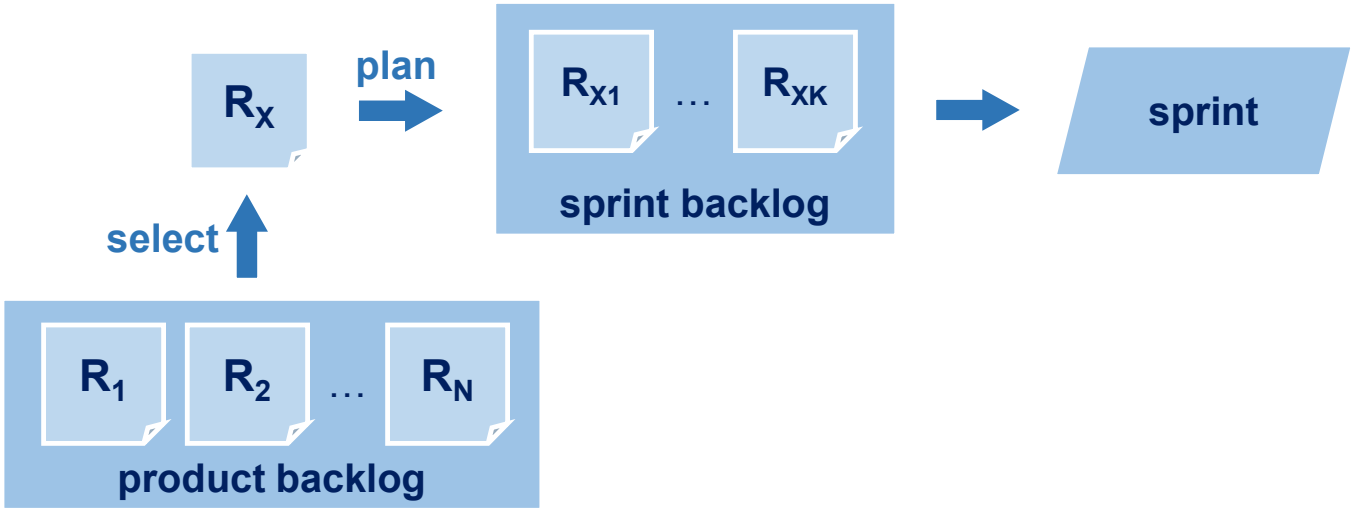


Da questo momento in poi i team lavorano in **isolamento**: lo **scrum master** li **«protegge»** da distrazioni esterne (incluse le comunicazioni con cliente e organizzazione)

SPRINT (CONT.)

Inizia lo **sprint**

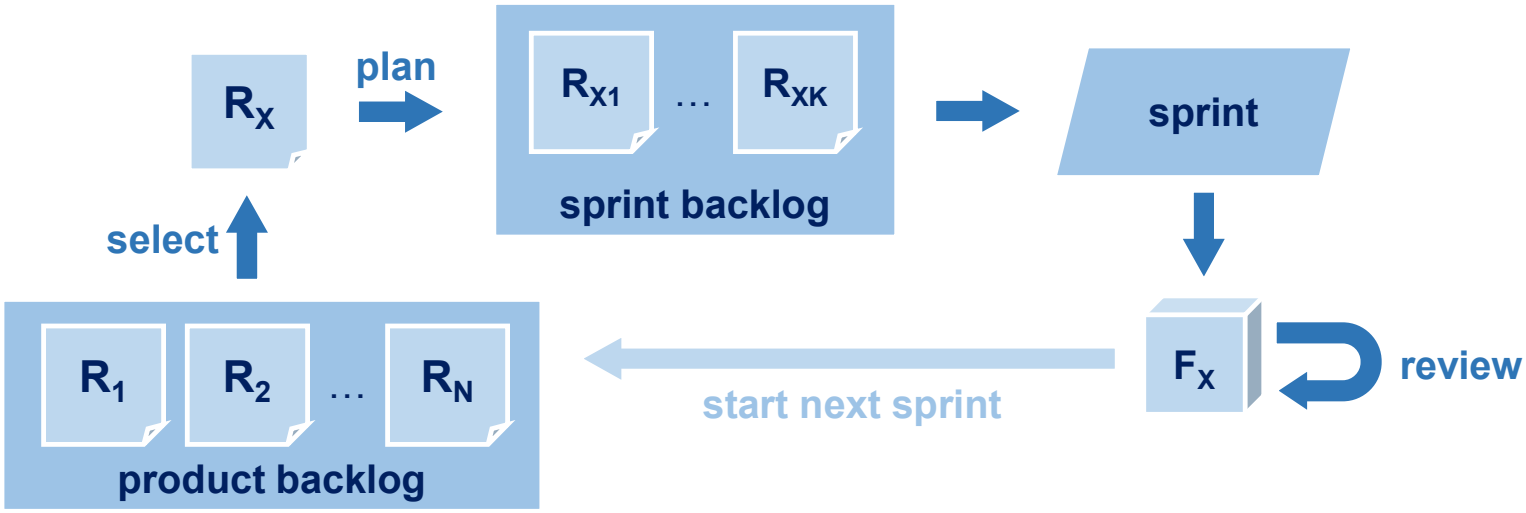
- Lo **scrum master** organizza brevi **meeting** giornalieri (chiamati **scrum**) con condivisione di informazioni: *Cosa ho fatto ieri? Cosa farò oggi? Quali problemi ho riscontrato e sto affrontando?*
- Il tempo rimanente è dedicato allo **sviluppo**



SPRINT (CONT.)

Al termine dello sprint, l'incremento prodotto viene **revisionato**

- Incontro di quattro ore, in collaborazione tra team e clienti/utenti
- **Feedback**: *È questo il prodotto che vogliamo costruire? Cosa penserebbero gli utenti finali del prodotto? È ancora il prodotto che ci è stato richiesto? Ci sono cambiamenti o nuove idee?*

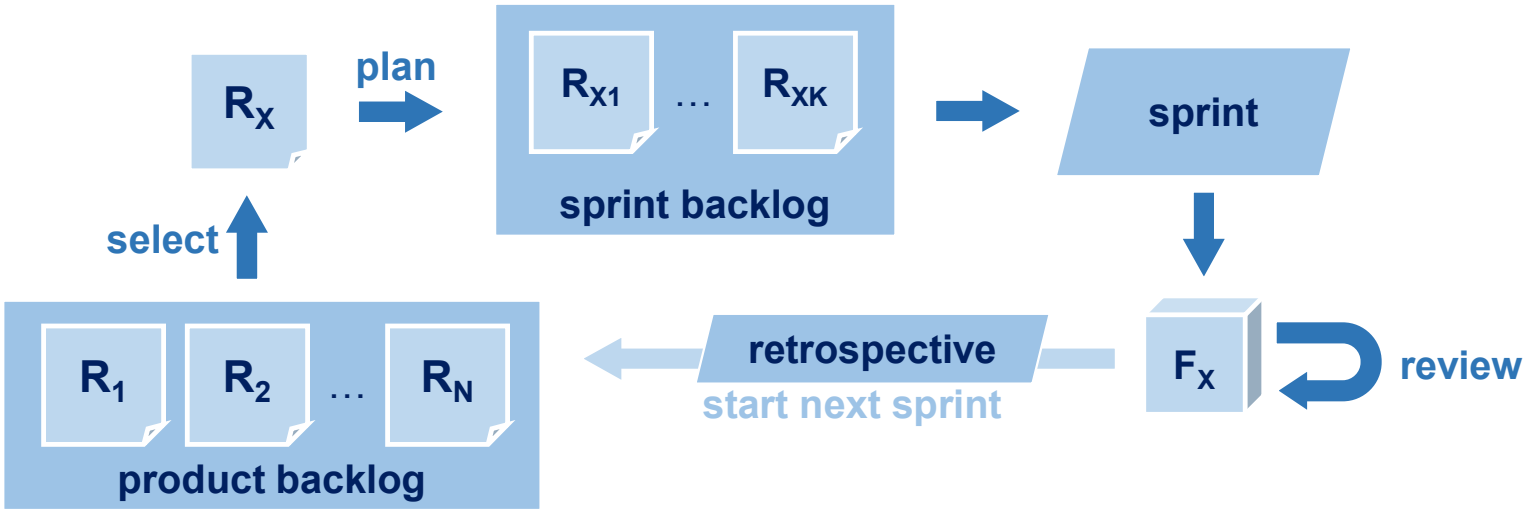


Può poi iniziare lo **sprint successivo**

SPRINT (CONT.)

Tra uno sprint è il successivo, viene organizzato un evento di **retrospettiva** (~3 ore)

- L'intero team scrum riflette, impara e si adatta per lo sprint successivo
- Obiettivo: contribuire al miglioramento continuo e alla crescita continua del team scrum



SCRUM: FASI (CONT.)

1) **Pre-game phase** (pianificazione «di massima»)
   - **Planning sub-phase**: Definizione del sistema che deve essere sviluppato, in termini di **product backlog list** (contiene tutti i requisiti attualmente conosciuti)
   - **Architecture sub-phase**: Design di alto livello del sistema, inclusa l'**architettura**, in base agli elementi contenuti nel product backlog
2) **Game phase** (sviluppo)
   - 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

3) **Post-game phase** (chiusura definitiva della release)
   - Conclude il processo di sviluppo e il prodotto viene preparato per il rilascio
   - Include integrazione, test, documentazione per l'utente, formazione e preparazione del materiale di marketing.

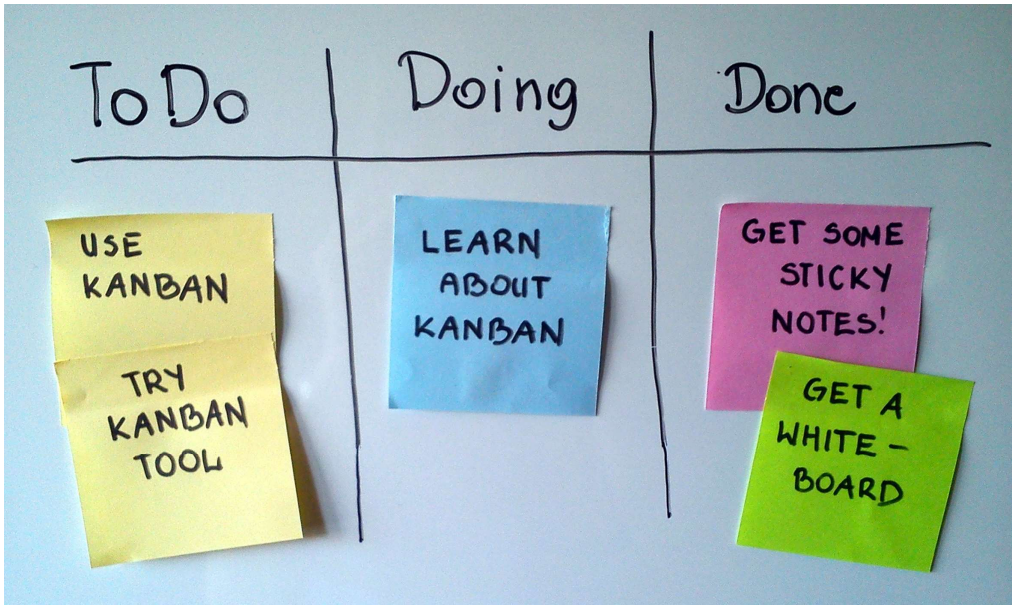
VANTAGGI DI SCRUM

- Il prodotto è partizionato in un insieme di **«pezzetti» gestibili** e facilmente comprensibili
- I requisiti non ancora stabili **non impegnano**/richiedono relativi progressi
- L'intero team ha visibilità di tutto e, di conseguenza, la comunicazione migliora
- I clienti ottengono **increment periodici**, avendo quindi la possibilità di vedere/provare il prodotto e fornire feedback
- Si stabilisce naturalmente una forma di **fiducia** tra clienti e sviluppatori: tutti collaborano e si aspettano che il progetto abbia successo

KANBAN

Organizzazione di un progetto // obiettivo: ridurre il tempo richiesto dall'inizio alla fine

- Visualizzazione, tramite **board** Kanban

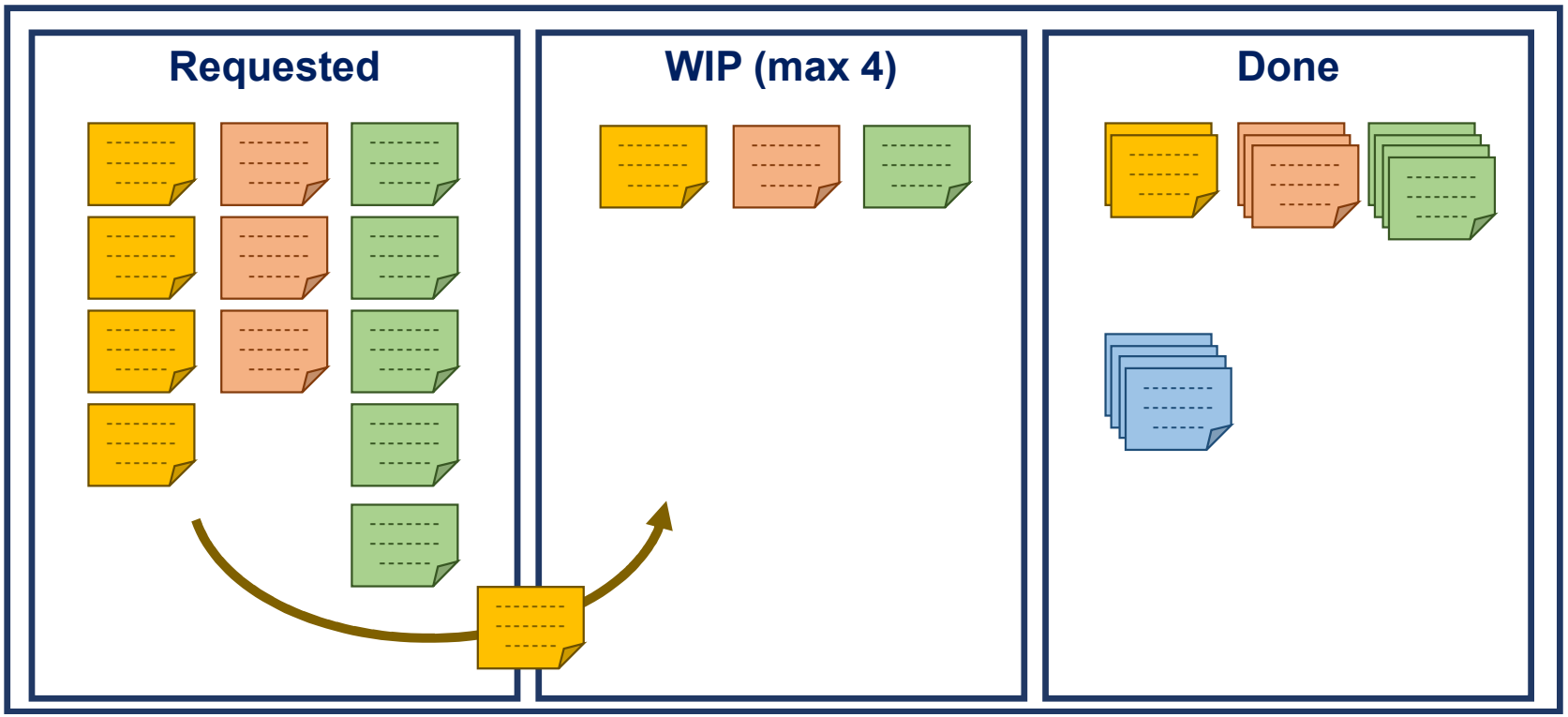


J. Iasovski: *A Scrum board suggesting to use Kanban*, CC-BY 3.0, 2011.

KANBAN (CONT.)

Organizzazione di un progetto // obiettivo: ridurre il tempo richiesto dall'inizio alla fine

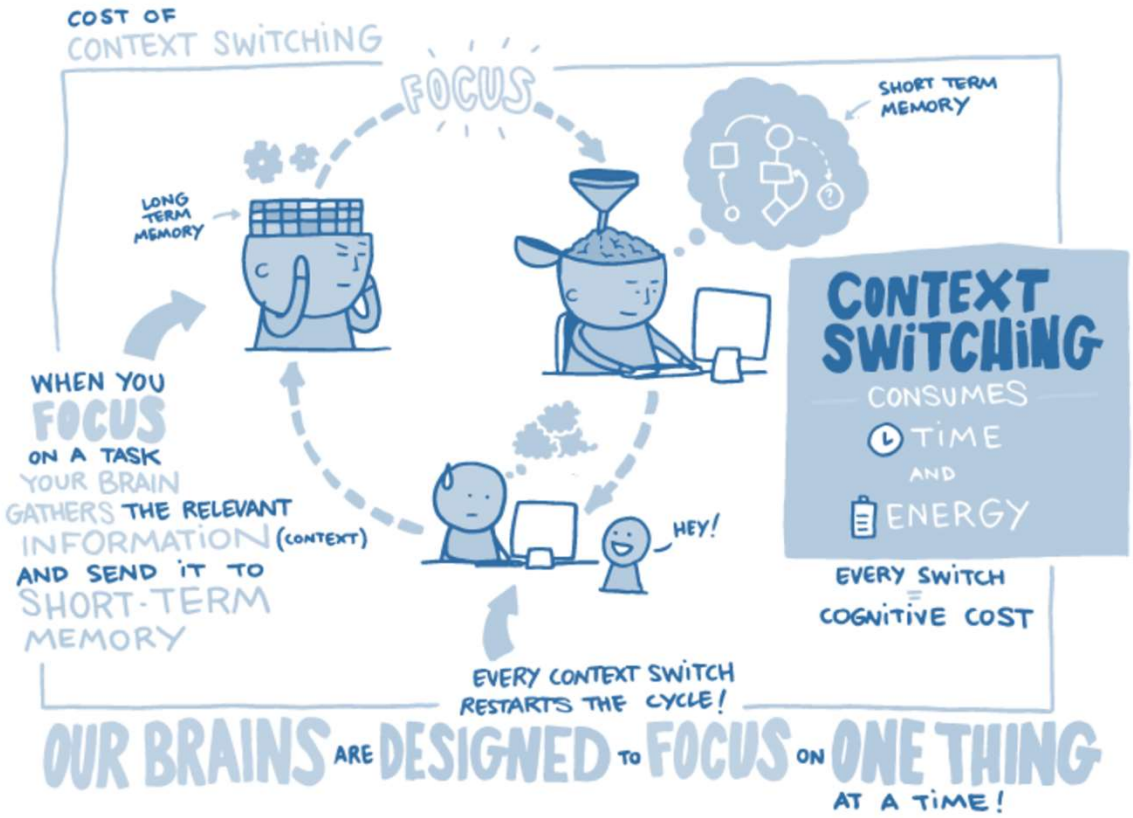
- Visualizzazione, tramite **board** Kanban
- Limitazione del lavoro in corso (**WIP** – *work-in-progress*)
- Ottimizzazione dell'**efficienza**



KANBAN (CONT.)

I **limiti al WIP** consentono di completare più velocemente i singoli task:

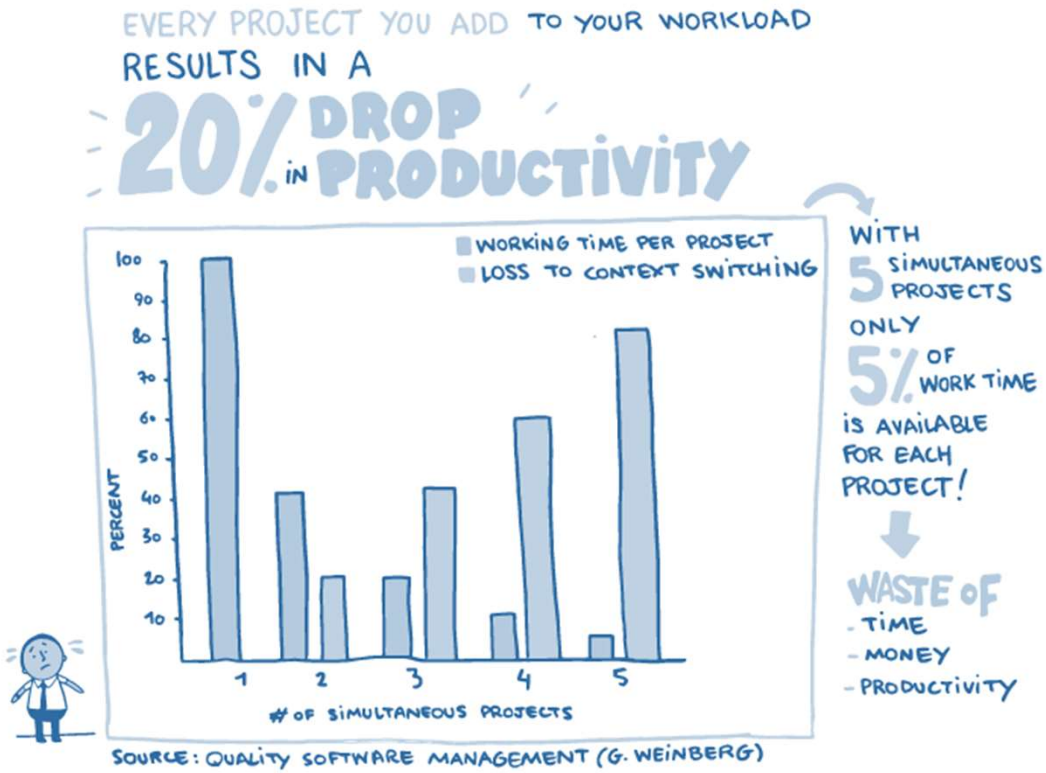
- Riduzione del **task switching**
  - Completare un task richiede molto più tempo se l'autore riceve contemporaneamente notifiche (e-mail o avvisi sul desktop)
  - Ogni notifica interrompe la concentrazione e i processi di pensiero in corso
- Più facile individuare i **colli di bottiglia** nei processi di lavoro prima che diventino un blocco



Da: A. Liard. **Why you should limit work in progress and stop multitasking**. Medium, 2019.

⇒ Si fornisce **valore** ai clienti più rapidamente

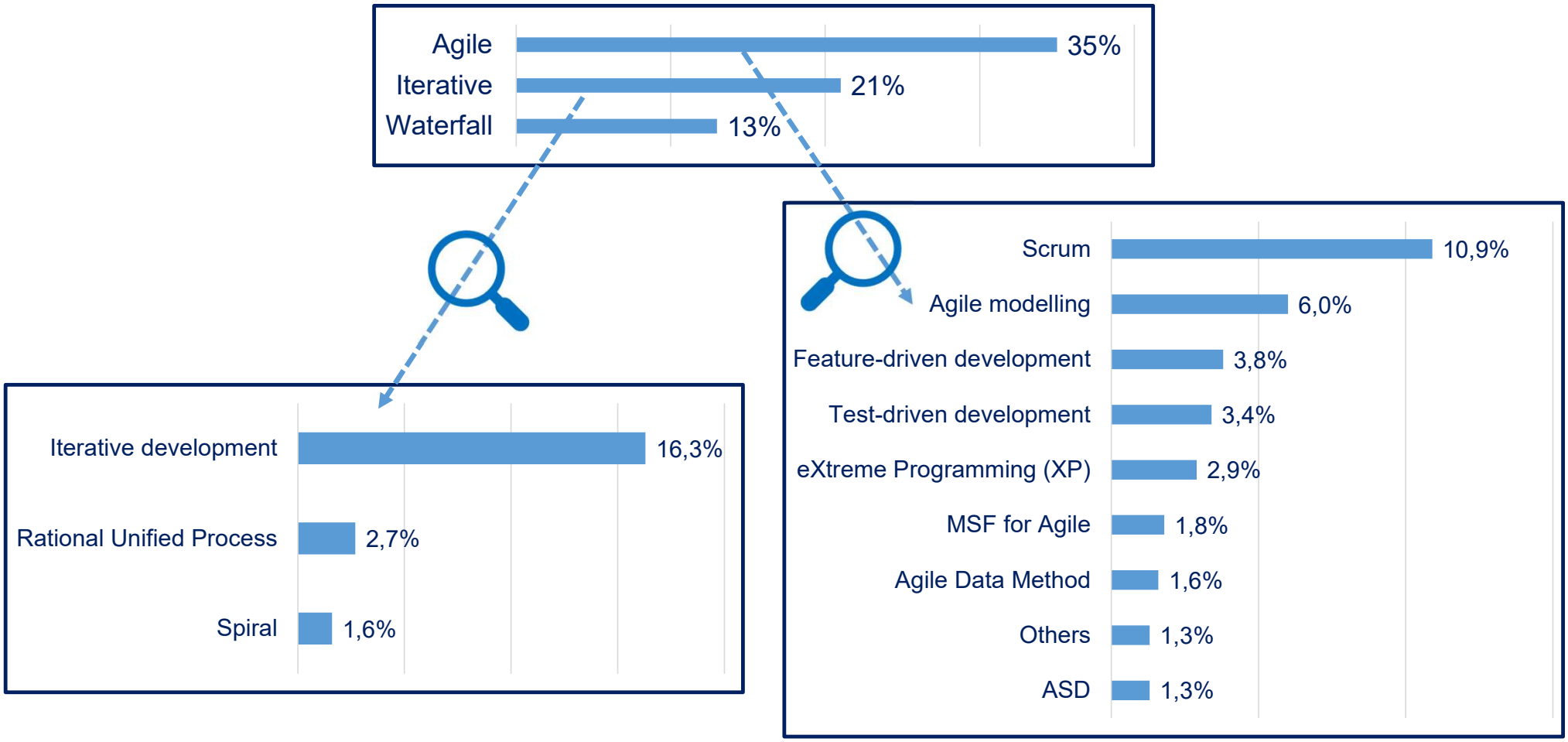
KANBAN (CONT.)



Da: A. Liard. **Why you should limit work in progress and stop multitasking**. Medium, 2019.

La limitazione del WIP consente di fornire **valore** ai clienti più rapidamente

QUALI MODELLI SONO UTILIZZATI?[1]



1. Sorgente: Forrester/Dr. Dobb's Global Developer Technographics Survey, Q3 2009 (1298 IT professionals)

RICAPITOLANDO

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 + prodotto rilasciato «a puntate»
- Modello a spirale → analisi dei rischi

Modelli agili → «ribellione alle prescrizioni» + molte nuove idee

- Extreme Programming (XP)
- Scrum


Q
A

RIFERIMENTI

Contenuti

- **Capitolo 2** di "Software Engineering" (G. C. Kung, 2023)
- **Capitolo 3** di "Software Engineering" (G. C. Kung, 2023)

Approfondimenti

- K. Schwaber, J. Sutherland. **La Guida Definitiva a Scrum: Le Regole del Gioco**, Scrum.Org and ScrumInc, 2014. Link: https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ITA.pdf

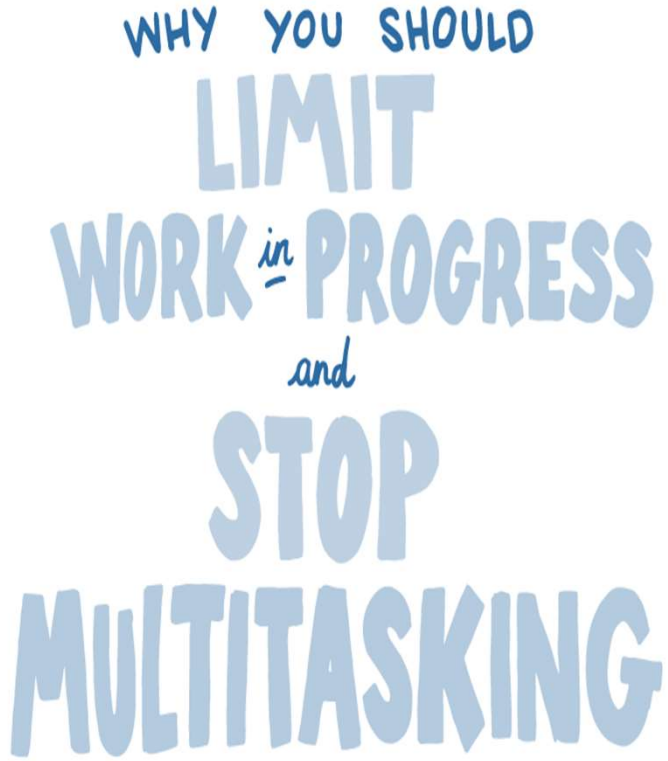
HOMEWORK

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 una strategia/app per limitare il multitasking



Da: A. Liard. **Why you should limit work in progress and stop multitasking**. Medium, 2019.