Universita’ degli Studi di Perugia
Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Ingegneria del Software

Prof. Alfredo Milani

Modelli di produzione e ciclo di vita del software

Materiale note, grazie al contributo di: Alfredo Milani, Fabrizio Montecchiani, Carlo Ghezzi, Alessandro Riccardi et al.

Docente

- Alfredo Milani
  - Email: alfredo.milani@unipg.it
  - Studio: DMI 6° piano +39 075 585.5049
- **Ricevimento**
  - su appuntamento per email
  - Mercoledì 14:00-15:00
  - online

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Processi software

- Un **processo software** è un insieme di attività che porta alla creazione di un prodotto software.
- Un **modello** di processo software è una rappresentazione **semplificata** di un processo software.
  - Il modello espone le strutture principali del processo, ma non i dettagli delle singole attività.

Processi software

- Esistono molti tipi diversi di software e non esiste un unico modello che vada bene per tutti.
- Il modello giusto dipende dal tipo di software, dalle richieste del cliente, dalle capacità delle persone coinvolte.
  - Occorre assicurarsi che l'organizzazione di sviluppo e poi di produzione sia compatibile con l'architettura del prodotto

Legge di Conway:
Le organizzazioni che progettano sistemi ne progettano la struttura riproducendo le proprie strutture comunicative (es. l'organigramma)

Code and fix

- Agli inizi dell'informatica, lo sviluppo del software era principalmente un lavoro individuale.
- Il modello usato in quei giorni, e che molto spesso i programmatori adottano per sviluppare semplici programmi, è chiamato **code and fix**.
- Il modello code and fix consiste nell'iterazione di due passi:
  1. Scrittura del codice.
  2. Aggiustamento del codice per correggere errori, migliorare la funzionalità, aggiungere nuove caratteristiche.

Oltre code and fix

- Oggi lo sviluppo del software è una attività professionale che richiede una **pianificazione**.
- Esistono processi di produzione che sono guidati da piani, detti **plan-driven:**
  - tutte le attività sono pianificate in anticipo
  - l'avanzamento è misurato rispetto a quanto previsto dal piano

Oltre code and fix

- Esistono anche processi di produzione **agili**, in cui la **pianificazione** è **incrementale e continua** al fine di modificare il processo agevolmente in caso di cambiamenti.
- I grandi sistemi richiedono dei compromessi tra i due approcci.

Attività principali della produzione software

- La produzione di software può essere scomposta in **attività** specifiche, la cui organizzazione dipende dal modello di processo adottato.
- Possiamo raggruppare queste attività in quattro macro attività.

Attività principali della produzione software

- **Specifica**: vengono definite le funzionalità del software e fissati i vincoli operativi.
- **Sviluppo**: viene sviluppato il software che realizza le specifiche di cui al punto precedente.
- **Convalida**: il software sviluppato viene convalidato per garantire ciò che il committente richiede.
- **Evoluzione**: il software si evolve per soddisfare le necessità e i cambiamenti dei requisiti del committente.

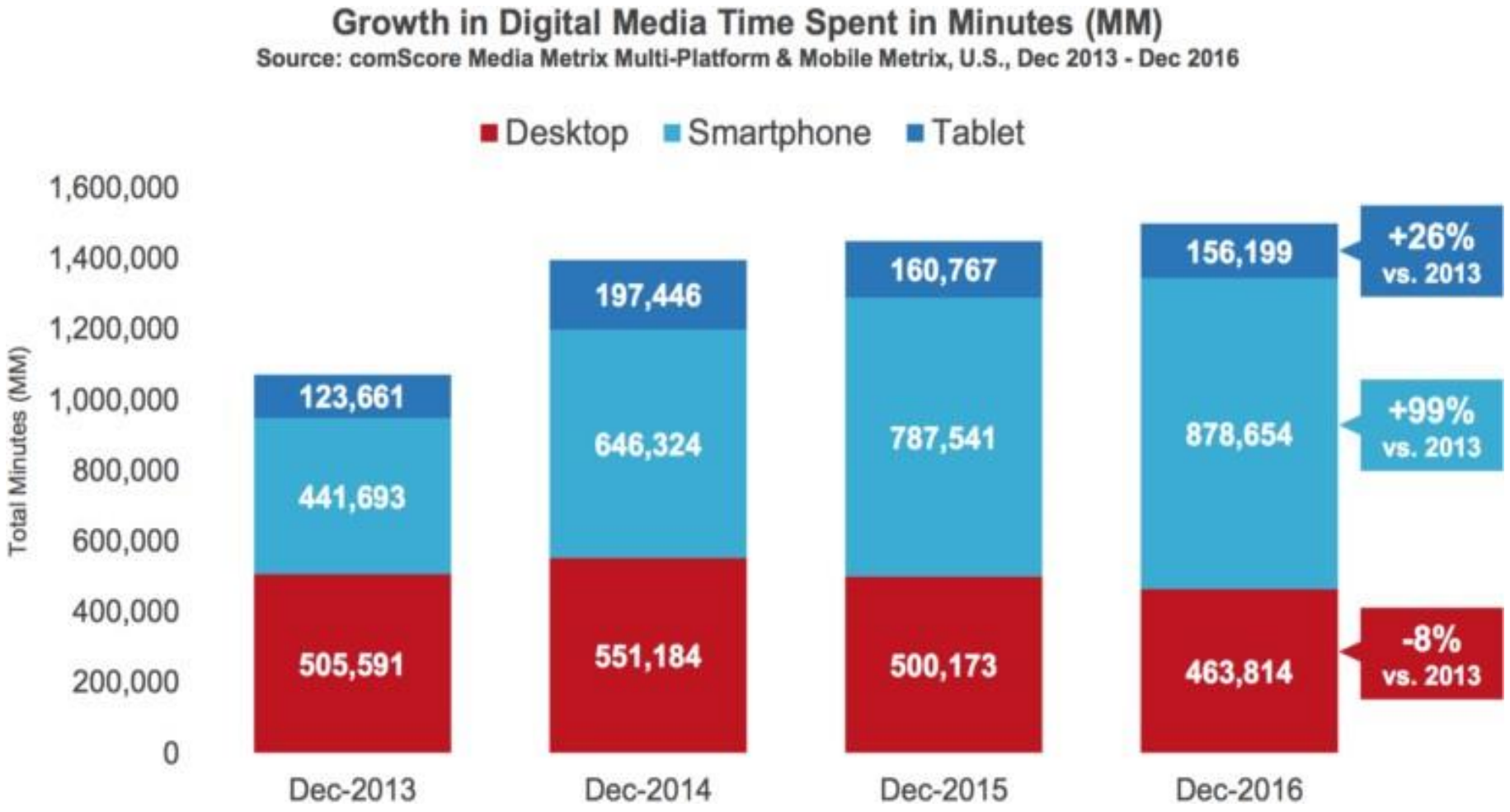
Studio di fattibilità

- Lo **studio di fattibilità** è una attività preliminare che viene eseguita prima che inizi il processo di produzione vero e proprio.
- Serve a trovare possibili soluzioni alternative, insieme a una discussione dei compromessi in termini di costi previsti e benefici.
- Tipicamente si decide se sviluppare un software ex novo, comprarlo da terzi, o abbandonare il progetto perché poco realistico.

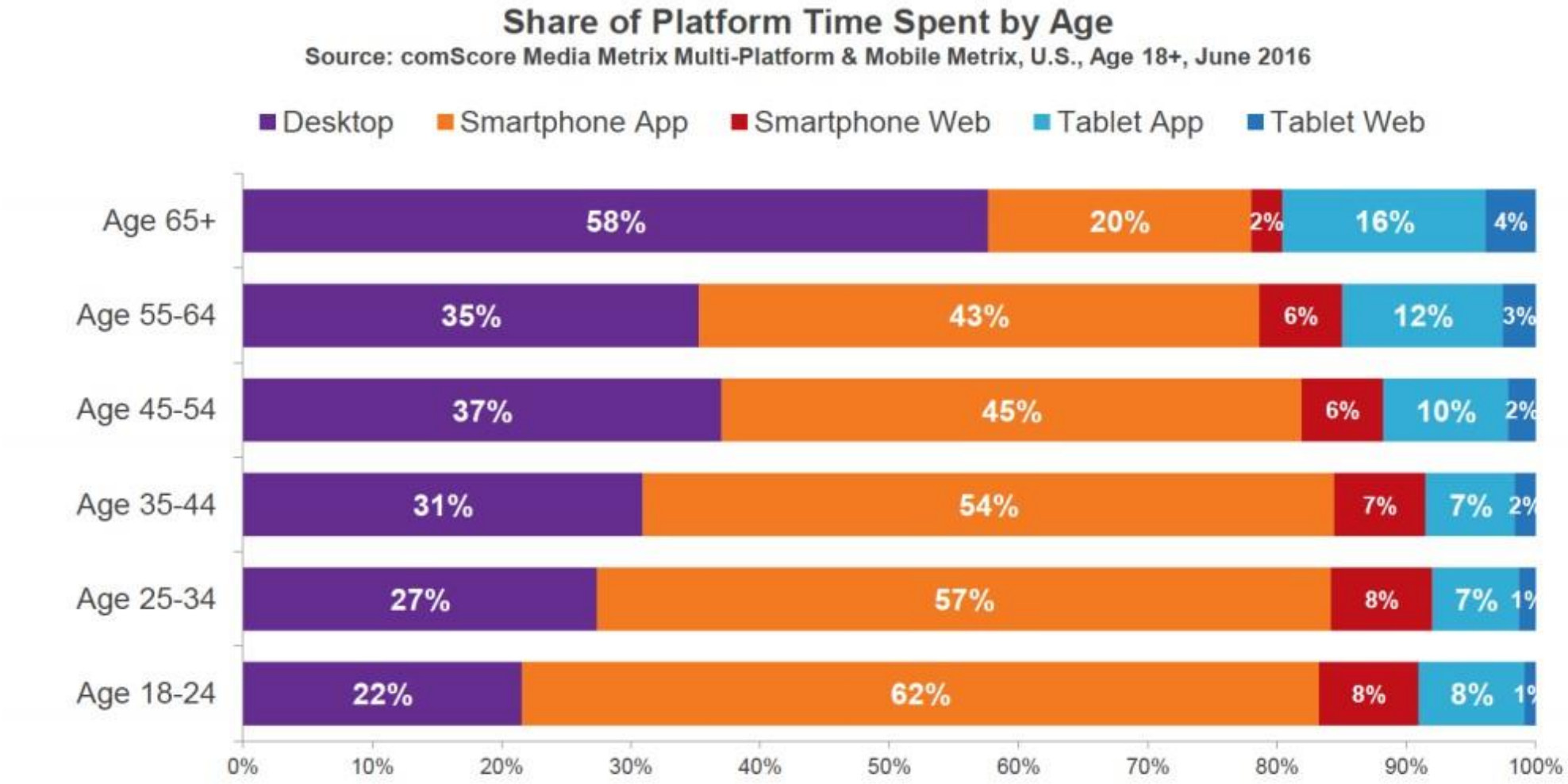
Studio di fattibilità

- Richiede un’attenta analisi del problema, specialmente se il suo output è utilizzato per preparare un’offerta ad un potenziale cliente.
  - Il rischio è di sottostimare le risorse necessarie per lo sviluppo, o di prendere decisioni premature.
- Viene a volte affiancato da uno **studio di mercato**, che serve a capire se lo sviluppo del software è economicamente sostenibile.

Analisi di mercato (es: desktop vs mobile)

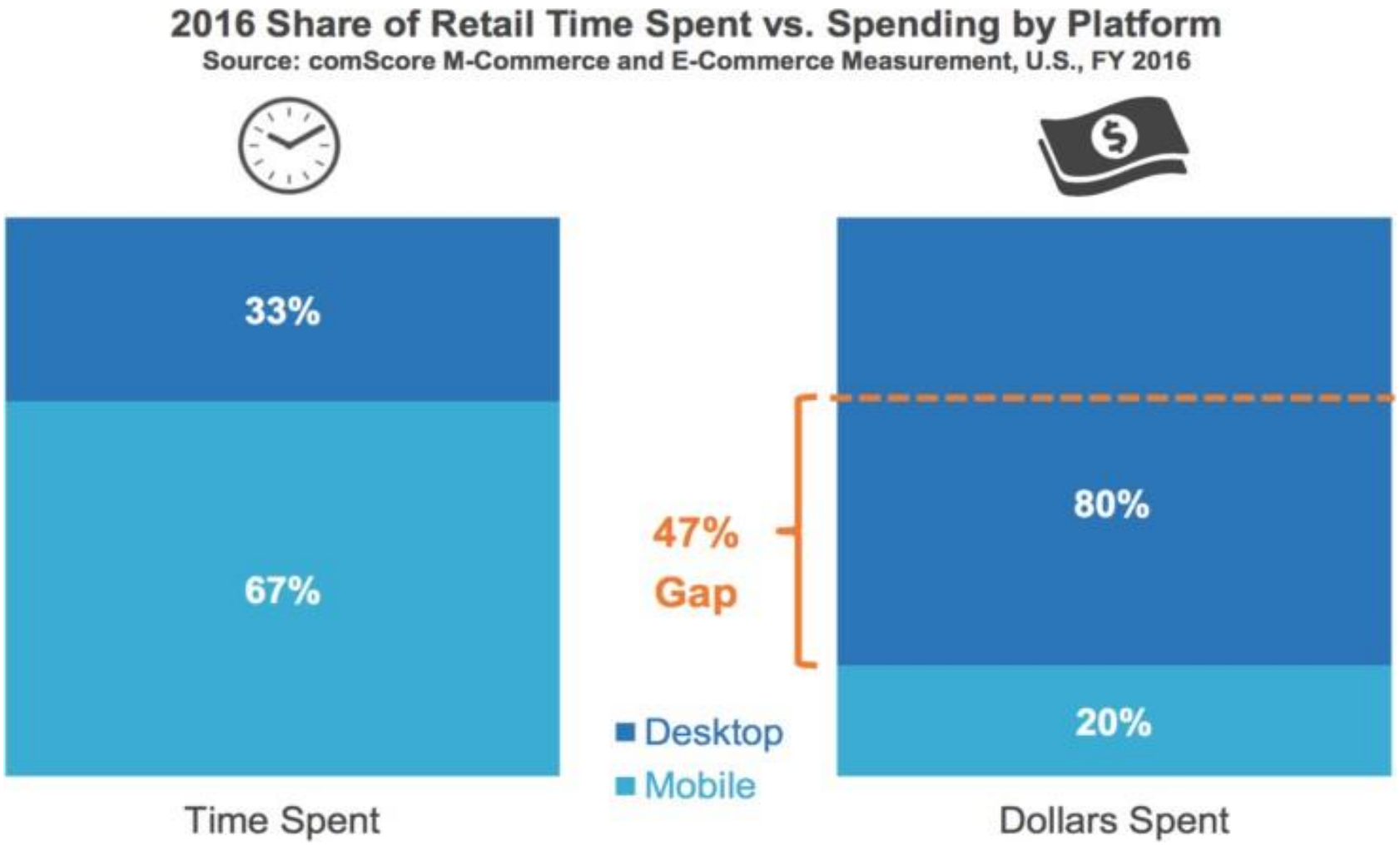


Analisi di mercato (es: desktop vs mobile)



comScore

Analisi di mercato (es: desktop vs mobile)



Specifica

- L'**ingegneria dei requisiti** sviluppa metodi per ottenere, documentare, classificare e analizzare i requisiti.
- Occorre comprendere gli **obiettivi del sistema**, **documentare i requisiti** da soddisfare e **specificare le qualità** richieste alla soluzione software.
  - Vanno descritti requisiti e qualità, ma non come questi vadano raggiunti (*what versus how*).
  - Tale attività non deve vincolare la progettazione e l'implementazione della soluzione con scelte premature.

Specifica

- Il progettista deve capire il dominio applicativo e identificare gli **stakeholder**, ovvero tutti coloro che hanno un interesse nel sistema e che saranno responsabili della sua accettazione.
  - Osservare procedure e sistemi esistenti.
  - Parlare con management, utenti, clienti, e altre parti interessate.
  - Ogni utente ha un punto di vista **parziale** del problema, a volte anche contraddittorio rispetto ad altri.
  - Occorre integrare e riconciliare diversi punti di vista in un'unica vista coerente del sistema.

Specifica

- Una volta raccolte le informazioni, esse vanno analizzate e tradotte in un **documento di specifica dei requisiti**.
- Il documento deve essere:
  - comprensibile
  - preciso
  - completo
  - coerente
  - non ambiguo
  - facilmente modificabile in futuro

Specifica

- Il **documento di specifica** può avere due livelli di dettaglio:
  - Uno di **alto livello**, che sarà analizzato e confermato dagli **utenti** interessati in modo da verificare se coglie tutte le aspettative del **committente**.
  - Uno più **dettagliato**, utilizzato dai **progettisti per sviluppare una soluzione** che soddisfi i requisiti.

Specifica

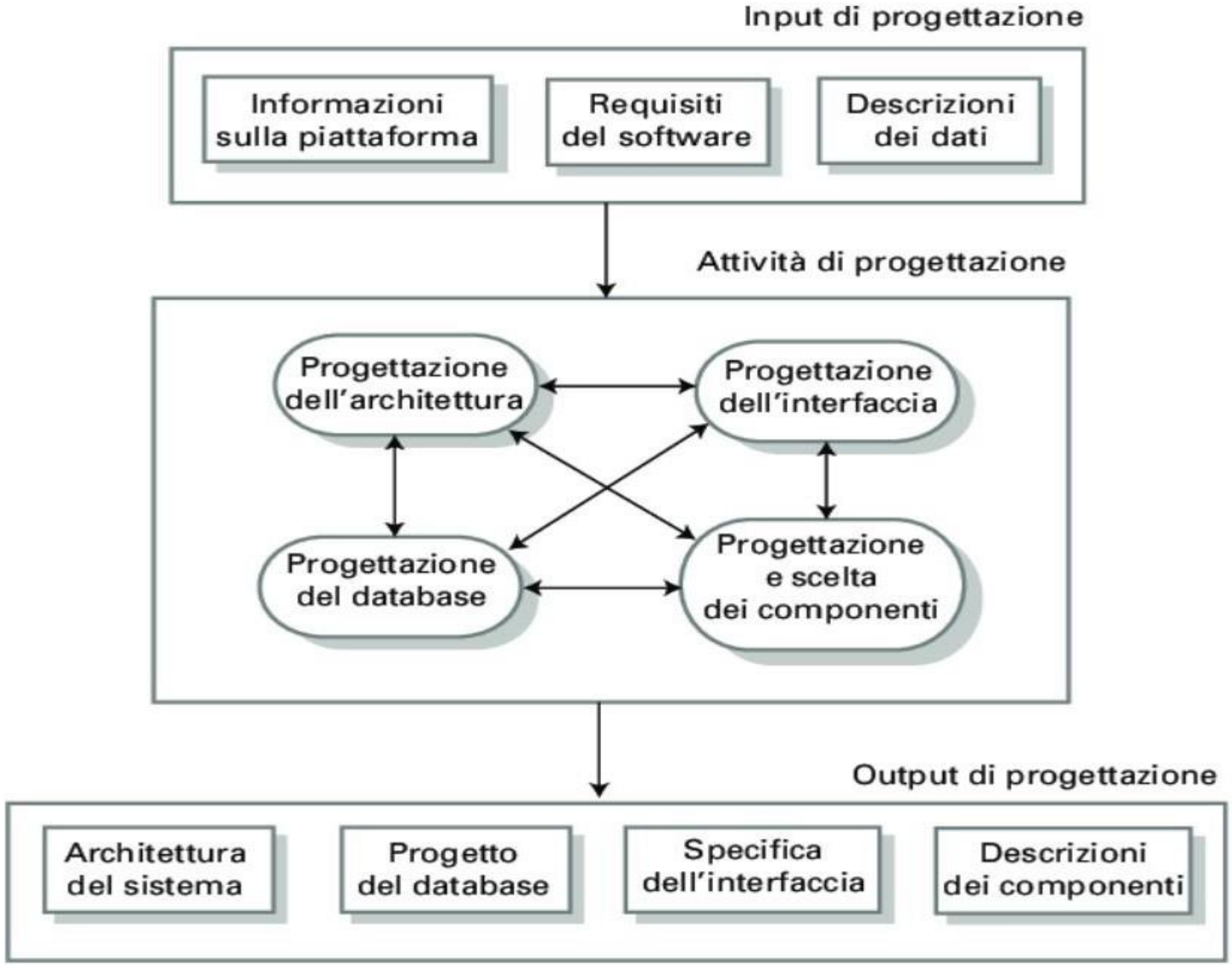
- Possibili contenuti dei **documenti di specifica dei requisiti**:
  - Breve descrizione del **dominio applicativo** e degli obiettivi che dovranno ess raggiunti .
    - Utenti interessati (obiettivi e aspettative).
    - Principali entità del dominio e loro relazioni.
  - **Glossario** e definizioni dei termini e concetti
  - Elenco dei **requisiti funzionali**, ovvero cosa deve fare il prodotto, usando notazioni più o meno formali.
  - Elenco dei **requisiti non funzionali**, quali affidabilità, accuratezza dei risultati, prestazioni, limiti operativi, ecc.
  - Elenco dei **requisiti del processo di sviluppo e manutenzione**, quali procedu il controllo della qualità, priorità di sviluppo, possibili cambiamenti del siste

Sviluppo

- La **progettazione** è l'attività attraverso la quale i progettisti strutturano l'applicazione a diversi livelli di dettaglio.
- La progettazione dell'**architettura** del software consistite nell'identificare la struttura complessiva del sistema.
  - Sfrutta il principio di modularità, applicandolo a più livelli di astrazione.

Sviluppo

- Le strutture dati del sistema e le loro rappresentazioni vengono definite nella progettazione del **database**.
- La progettazione dell'**interfaccia** definisce le interfacce tra i componenti del sistema in maniera non ambigua, in modo che possano essere sviluppati indipendentemente.
- Infine, vengono ricercati i componenti riutilizzabili e, se non disponibili, vengono progettati nuovi componenti.


Input di progettazione
Informazioni sulla piattaforma
Requisiti del software
Descrizioni dei dati
Attività di progettazione
Progettazione dell'architettura
Progettazione dell'interfaccia
Progettazione del database
Progettazione e scelta dei componenti
Output di progettazione
Architettura del sistema
Progetto del database
Specifica dell'interfaccia
Descrizioni dei componenti

Sviluppo

- La produzione di codice e il test dei moduli sono poi le attività che seguono la progettazione.
- Sebbene la programmazione è tipicamente un'attività individuale, può essere soggetta a standard sulla struttura dei programmi, sui commenti, sulle convenzioni di naming, ecc.
- Il test serve poi a stabilire l'esistenza di difetti, che possono poi essere risolti mediante *debug* del codice.

Convalida

- La **convalida** del software è intesa a mostrare che un sistema:
  - è conforme alle sue specifiche
  - soddisfa le aspettative del cliente

Convalida

- Avviene tipicamente in tre fasi.
  1. **Test dei singoli componenti** da parte di chi ha sviluppato il modulo attraverso dei *test case* automatici.
  2. **Integrazione dei componenti** per formare il sistema completo, al fine di identificare i problemi con le interfacce e verificare la conformità dei requisiti.
  3. **Esecuzione dei test cliente**, con dati reali, al fine di misurare quanto il software soddisfa le esigenze del cliente.

Convalida

- Gli **alpha test** sono dei test realizzati internamente alla società che sviluppa il software, in condizioni comunque realistiche (ad esempio con molti utenti attivi e con molti dati simulati).
- I **beta test** sono invece dei test realizzati con un insieme selezionato di utenti reali del sistema e con dati reali. Gli utenti dovrebbero essere motivati, critici e attivi nell'utilizzo del sistema.

Evoluzione

- Una volta in servizio, il sistema potrà evolvere, al fine di:
  - risolvere difetti non evidenziati nella fase di convalida
  - aggiungere nuove funzionalità
  - adattarsi a cambiamenti dell’ambiente operativo
  - ecc.

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Modello a cascata: descrizione

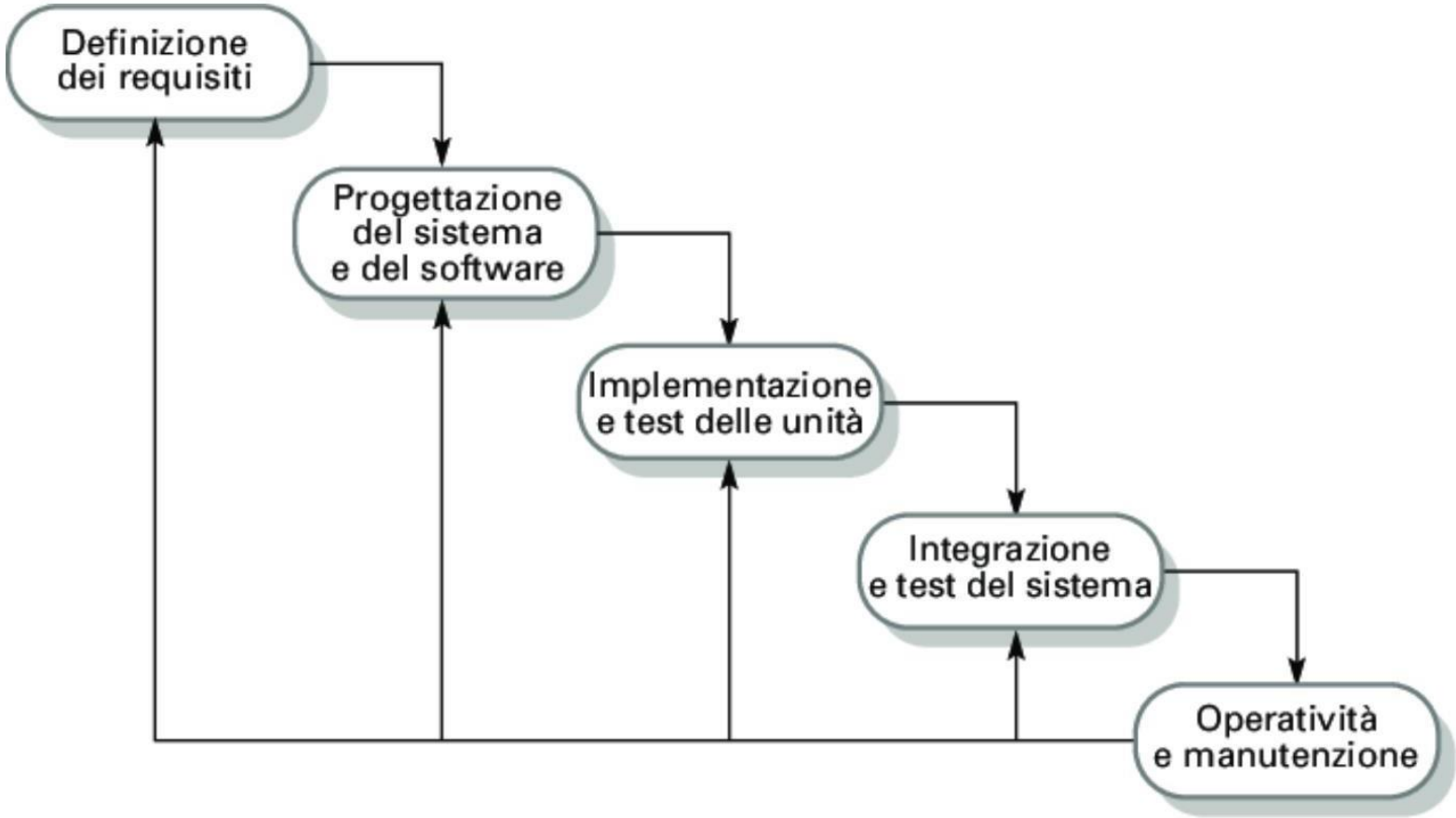
- Il **modello a cascata** fu introdotto negli anni ‘50 e divenne popolare negli anni ’70.
  - È ancora oggi uno dei modelli di riferimento.

- Le attività sono strutturate come una **cascata lineare di fasi.**
  - L’output di una fase diventa l’input della fase successiva.
  - Ogni fase è strutturata in sotto-attività che possono essere svolte contemporaneamente.

Modello a cascata: descrizione

- Esistono varie versioni del modello, ma tutte condividono alcuni aspetti essenziali:
  - L'**intero processo** va **pianificato in anticipo**.
  - Fasi che sono **sequenziali**, ovvero una fase inizia solo dopo che la precedente è terminata.
  - fasi **basate su documenti** (*document driven*).
- nei primi modelli a cascata feedback assente, alcune istanze del modello consentono ***document driven*** dei feedback da una fase a quella strettamente precedente, o ad una fase antecedente qualsiasi.

Modello a cascata: schema



Modello a cascata: vantaggi

- Impone una forte disciplina durante tutto il ciclo di vita.
- Permette di definire da subito dei **deliverable** associati a date di consegna, i documenti delle prime fasi.
- È adatto per quei sistemi in cui è possibile e importante conoscere a fondo le specifiche prima di iniziare la fase di progettazione.
  - Ad esempio **sistemi critici** o **sistemi embedded**.

Modello a cascata: svantaggi

- Fasi rigide, requisiti e specifiche di progetto congelati prima della fase di sviluppo.
  - **Interazioni** intermedie tra **cliente** e **sviluppatori** non previste.
    - Il **cliente approva il documento di specifica dei requisiti**, ma questo non garantisce che il prodotto finale lo soddisfi.
  - Eventuali **errori in fase di analisi** potrebbero non emergere fino al rilascio del prodotto.
  - Alcuni **requisiti potrebbero non essere più attuali** se passa molto tempo tra le prime fasi e la consegna.
  - I costi di manutenzione potrebbero lievitare a causa di incomprensioni o cambiamenti non previsti.

Modello a cascata: svantaggi

- Il modello a cascata non è applicabile in particolare a quei contesti in cui i requisiti non sono completamente noti a priori.
- L'approccio document driven può risultare troppo burocratizzato e oneroso per alcuni tipi di progetti.

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili

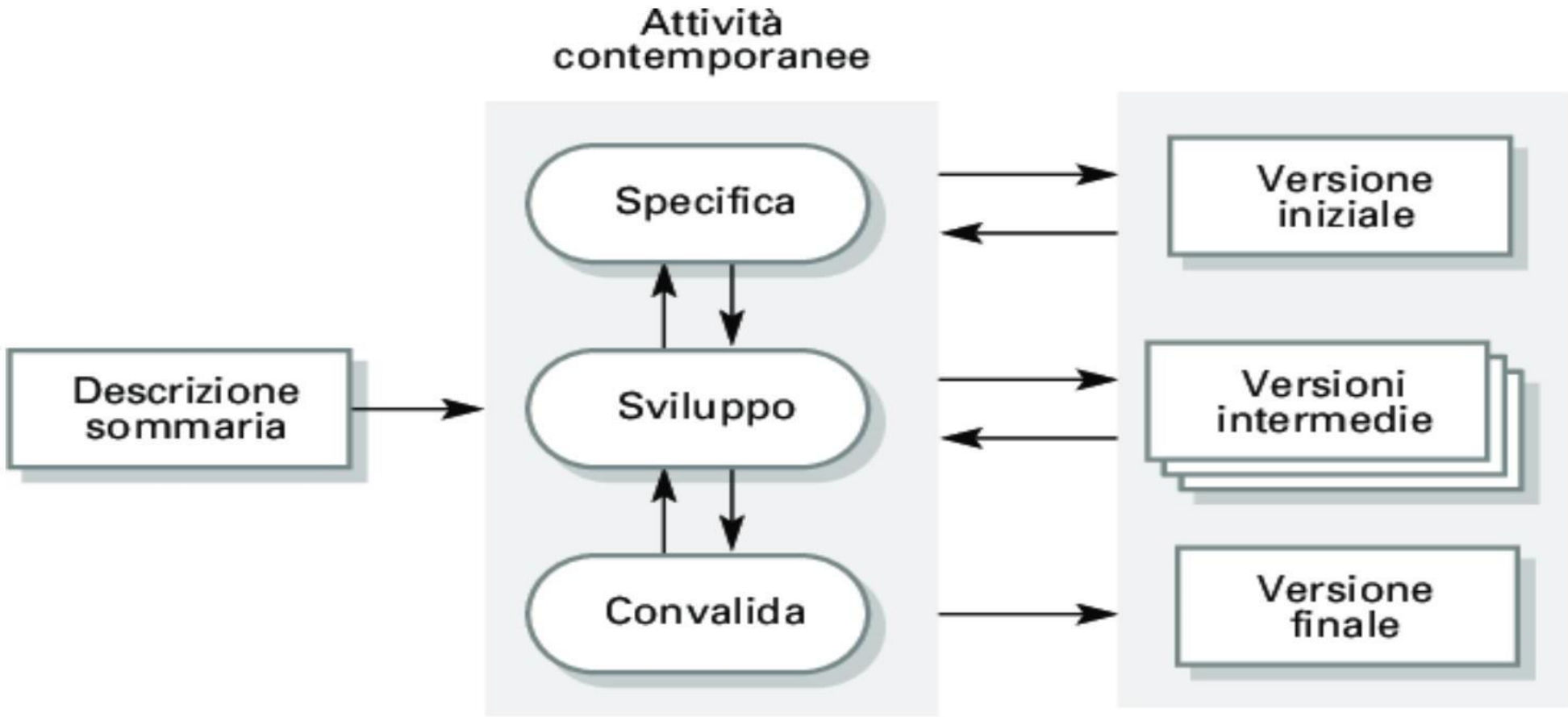
Modelli evolutivi: descrizione

- I modelli **evolutivi** o **incrementali**, sono tipicamente meno plan-driven e più agili.
- Si produce da subito una versione iniziale del software che viene poi subito esposta agli utenti al fine di ottenere dei **feedback immediati**.
  - Tipicamente contiene solo un sottoinsieme delle funzionalità.

Modelli evolutivi: descrizione

- La **versione iniziale** viene poi r**affinate** attraverso varie versioni, fino a raggiungere una **versione stabile finale**.
- Le attività di specifica, sviluppo e convalida si intrecciano tra loro, con **feedback continui** tra una attività e l'altra.

Modelli evolutivi: descrizione



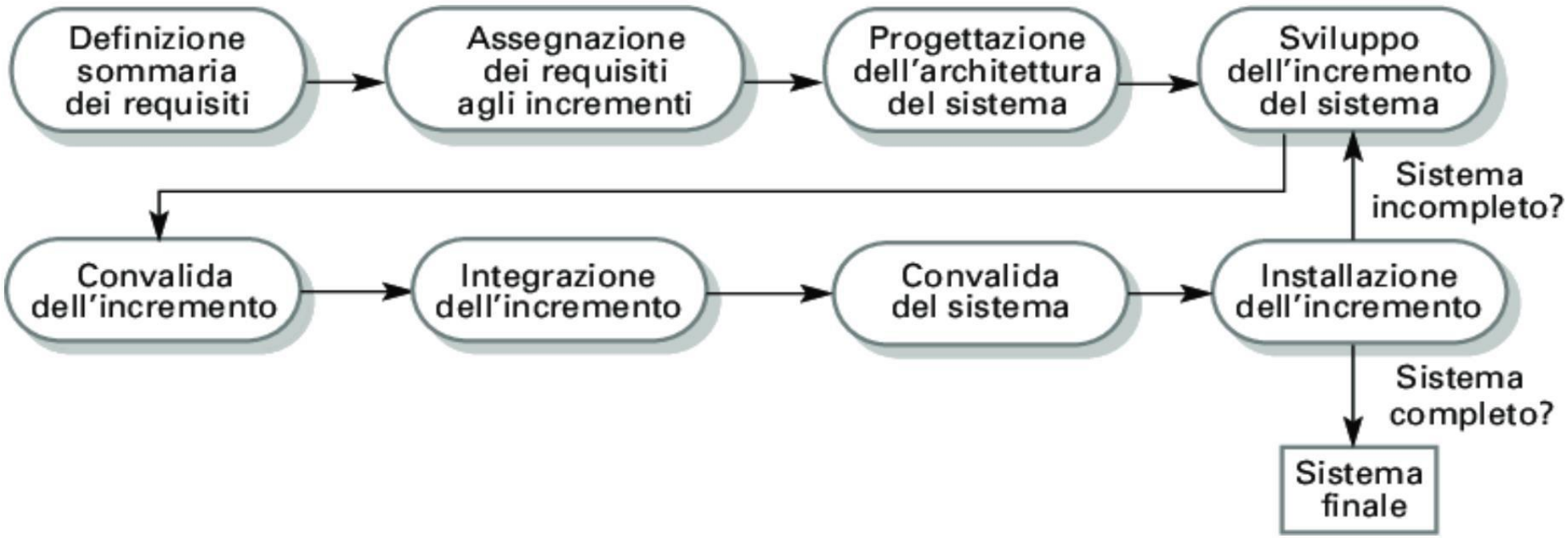
Modelli evolutivi: descrizione

- La strategia di sviluppo dietro un modello evolutivo può essere espressa così:
  1. **Rilascio** di una **funzionalità** all'utente.
  2. **Misura del valore aggiunto** per il cliente.
  3. **Aggiornamento** sia del **progetto** che degli **obiettivi**, in base a quanto osservato.

Modelli evolutivi: descrizione

- Nel modello a cascata il cambiamento si manifesta come un'attività post-sviluppo (manutenzione).
- In un modello evolutivo i cambiamenti fanno parte invece dello sviluppo e tipicamente si ricorre ad una **consegna incrementale** del prodotto.

Consegna incrementale



Modelli evolutivi: vantaggi

- Il costo delle **modifiche dei requisiti** è ridotto.
- Risulta più semplice ottenere dei **feedback** dal **cliente**.
- Il cliente può **iniziare a utilizzare il prodotto** in anticipo rispetto al rilascio finale (consegna incrementale).

Modelli evolutivi: svantaggi

- Non è economico produrre molta documentazione per ogni incremento del prototipo.
  - Il processo può risultare meno visibile da un punto di vista manageriale.
- La struttura del programma tende a degradarsi e il codice si complica quando vengono aggiunti nuovi incrementi.
  - Occorre una costante riorganizzazione (refactoring) del codice.

Programma

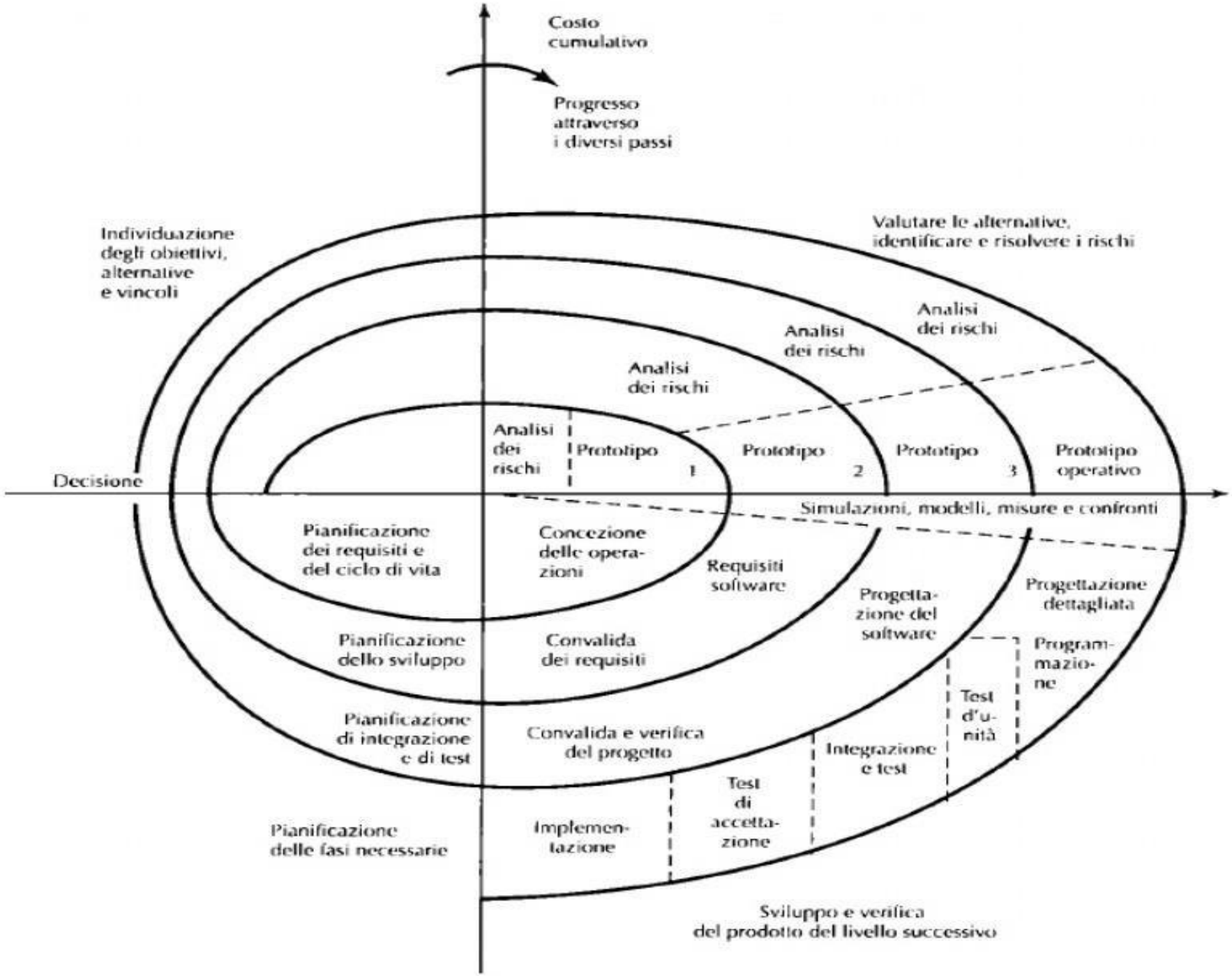
- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Modello a spirale: descrizione

- Il **modello a spirale** è un «metamodello» in grado di descrivere qualsiasi modello di processo di sviluppo.
- L'obiettivo è fornire un quadro di riferimento per la progettazione dei processi, al fine di **minimizzare i rischi** presenti in un progetto in un dato momento.
  - I rischi sono circostanze potenzialmente avverse, in grado di pregiudicare il processo di sviluppo e la qualità del prodotto.

Modello a spirale: descrizione

- La **gestione dei rischi** è una disciplina i cui obiettivi sono *identificare*, *affrontare* ed *eliminare i rischi* in cui il software può incorrere prima che diventino minaccia seria o causa di re-implementazioni costose.


Costo cumulativo
Progresso attraverso i diversi passi
Individuazione degli obiettivi, alternative e vincoli
Valutare le alternative, identificare e risolvere i rischi
Analisi dei rischi
Analisi dei rischi
Analisi dei rischi
Analisi dei rischi
Prototipo 1
Prototipo 2
Prototipo 3
Prototipo operativo
Decisione
Simulazioni, modelli, misure e confronti
Pianificazione dei requisiti e del ciclo di vita
Concezione delle operazioni
Requisiti software
Progettazione del software
Progettazione dettagliata
Pianificazione dello sviluppo
Convalida dei requisiti
Programmazione
Test d'unità
Pianificazione di integrazione e di test
Convalida e verifica del progetto
Integrazione e test
Test di accettazione
Pianificazione delle fasi necessarie
Implementazione
Sviluppo e verifica del prodotto del livello successivo

Modello a spirale: descrizione

- Il **modello** è **ciclico** anziché lineare, **ogni ciclo** della spirale consiste di **quattro fasi** e ciascuna è rappresentata da un quadrante del diagramma Cartesiano.
- Il **raggio della spirale** rappresenta il **costo accumulato** dal processo fino a quel momento.
- La **dimensione angolare della spirale** rappresenta il **progresso del processo**.

Modello a spirale: descrizione

- La **prima** fase **identifica gli obiettivi della porzione di prodotto** che si sta considerando, in termini di qualità da ottenere. Identifica inoltre le alternative (ad esempio, *buy versus make*) e i vincoli rispetto alla scelta delle alternative.
- La **seconda** fase **valuta le alternative,** evidenziando e affrontando le **potenziali aree di rischio**.

Lo studio dei rischi può richiedere la pianificazione di attività quali la prototipazione o la simulazione.

Modello a spirale: descrizione

- La **terza** fase consiste nello **sviluppo** e nella **verifica** del prodotto o porzione.

- La **quarta** fase consiste nella **revisione dei risultati** delle tre fasi attraversate e nella **pianificazione** della **prossima iterazione** della spirale.

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Sviluppo agile: contesto

- Oggi le aziende lavorano in un ambiente globale caratterizzato da **cambiamenti rapidi**.
  - Occorre cogliere nuove opportunità.
  - Serve rispondere a nuovi mercati, condizioni economiche variabili, presenza di prodotti e servizi concorrenti.
- La **rapidità (produttività) dello sviluppo** diventa quindi il requisito più critico per la maggior parte dei sistemi software aziendali.

Sviluppo agile: contesto

- I requisiti dei clienti **cambiano** perché è impossibile prevedere come **un sistema influenzerà le pratiche operative**, come interagirà con gli altri sistemi e quali operazioni degli utenti dovranno essere automatizzate.

I **requisiti reali** diventano chiari solo **dopo** che il sistema è in utilizzo.

*«Il medium **è** il messaggio» (Marshall McLuhan)*

- I processi di sviluppo plan-driven non sono adatti a questo scenario, poiché frequenti cambiamenti ai requisiti allungano i tempi di sviluppo notevolmente.

«Manifesto per lo Sviluppo Agile»

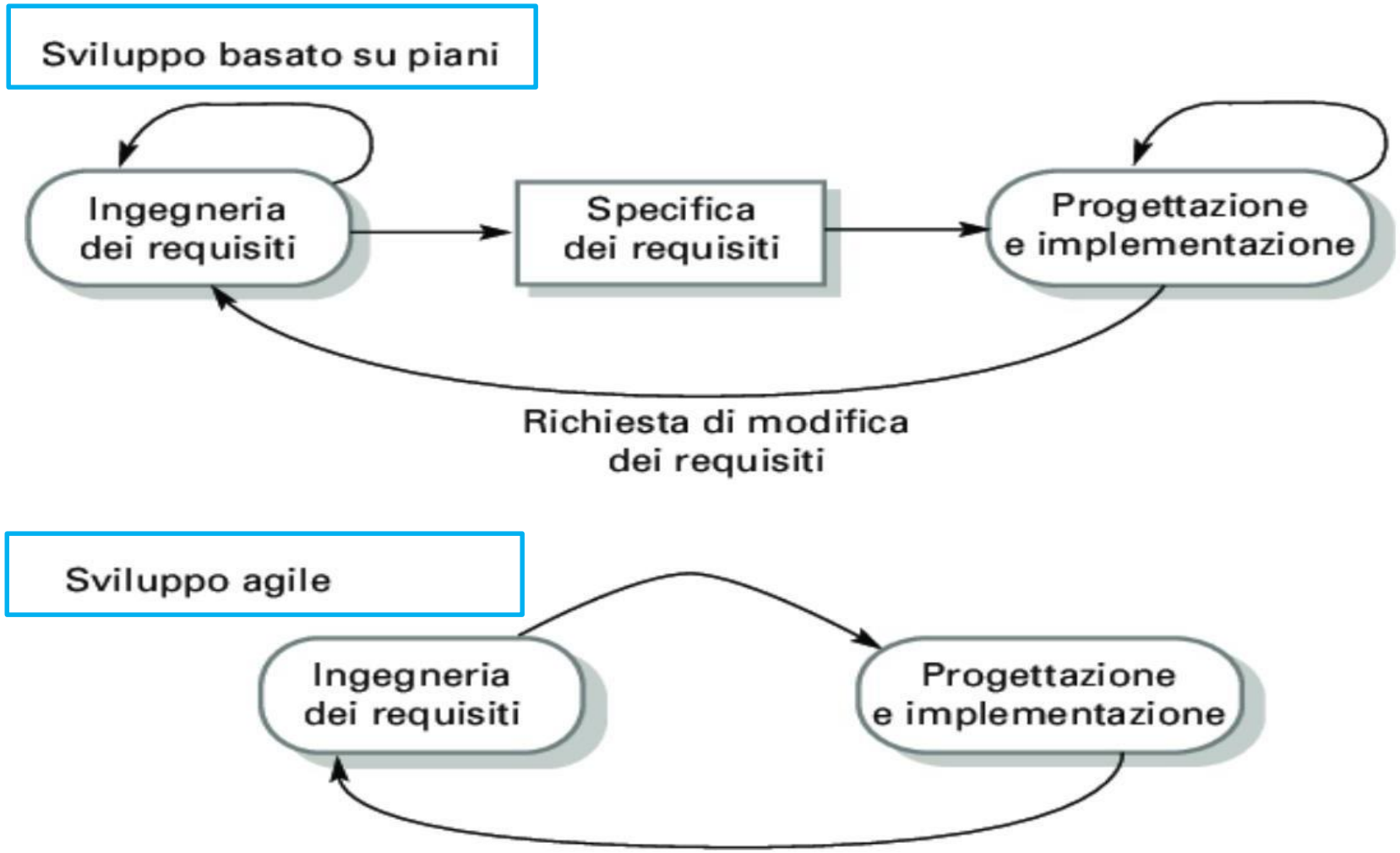
- *Gli individui e le interazioni più che i processi e gli strumenti*
- *Il software funzionante più che la documentazione esaustiva*
- *La collaborazione col cliente più che la negoziazione dei contratti*
- *Rispondere al cambiamento più che seguire un piano*

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

Sviluppo agile: descrizione

- Caratteristiche comuni dei metodi di **sviluppo agile** (anni ’90):
  - Processi di specifica, progettazione e implementazione intrecciati.
  - Specifica non dettagliata e documentazione minimizzata (spesso prodotta automaticamente).
  - Sistema prodotto incrementalmente. Utenti finali coinvolti nella specifica e nella valutazione di ogni incremento.
  - Ricca gamma di strumenti a supporto del processo di sviluppo (automazione dei test, gestione della configurazione, integrazione del sistema, produzione automatica di interfacce utente).

Sviluppo agile: descrizione



Metodi agili

- I **metodi agili** risultano particolarmente utili per:
  - Prodotti di piccole o medie dimensioni sviluppati da software house.
  - Sviluppo personalizzato di sistemi all'interno di un'organizzazione.
    - Chiaro impegno da parte del cliente di essere coinvolto nel processo di sviluppo.
    - Pochi stakeholder e pochi regolamenti esterni che possono influire sul software.

I principi del «Manifesto Agile»

- La nostra massima priorità è **soddisfare il cliente** rilasciando software di valore, fin da subito e in maniera continua.
- **Accogliamo i cambiamenti** nei requisiti, anche a stadi avanzati dello sviluppo.
- **Consegniamo frequentemente** software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi.
- Committenti e sviluppatori devono **lavorare insieme** quotidianamente per tutta la durata del progetto.

I principi del «Manifesto Agile»

- Fondiamo i progetti su **individui motivati**. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine.
- Una **conversazione faccia a faccia** è il modo più efficiente e più efficace per comunicare con il team ed all'interno del team.
- Il **software funzionante** è il principale metro di misura di progresso.
- I processi agili promuovono uno **sviluppo sostenibile**. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.

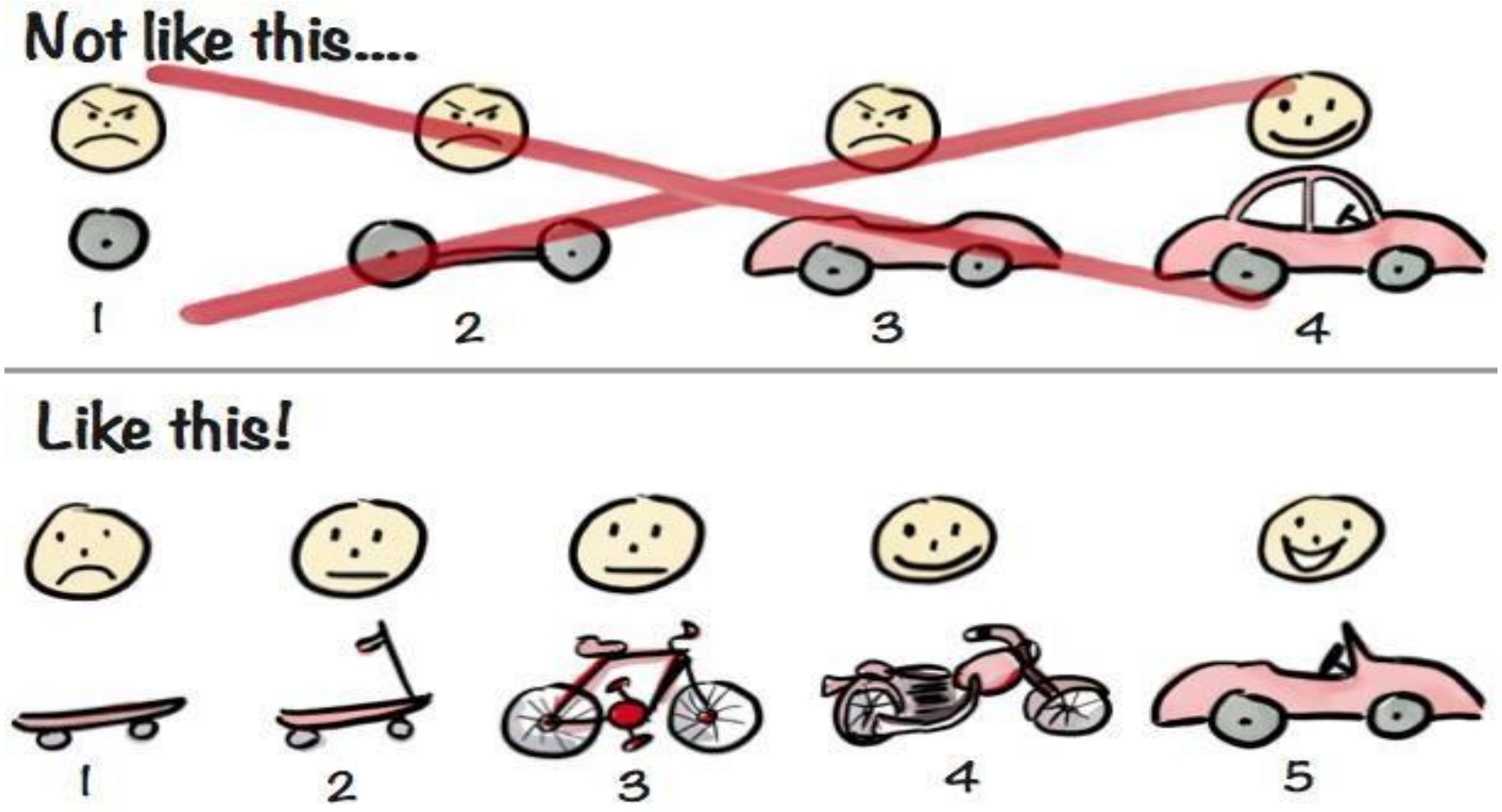
I principi del «Manifesto Agile»

- La continua attenzione all'**eccellenza tecnica** e alla buona progettazione esaltano l'agilità.
- La **semplicità** - l'arte di massimizzare la quantità di lavoro non svolto - è essenziale.
- Le architetture, i requisiti e la progettazione migliori emergono da team che si **auto-organizzano**.
- A intervalli regolari il team **riflette** su come diventare più efficace, dopodiché regola e adatta il proprio comportamento di conseguenza.

Minimal viable product

- Nello sviluppo di un nuovo prodotto, il prodotto minimo funzionante (**Minimum Viable Product** o **MVP**) è il prodotto con il più alto ritorno sugli investimenti rispetto al rischio.
- È una strategia (processo) mirata a:
  - evitare di costruire prodotti che i clienti non vogliono,
  - massimizzare le informazioni apprese sul cliente per ogni euro speso.

Minimal viable product



eXtreme Programming

- La tecnica di **extreme programming (XP)** è uno degli approcci agili più noti.
- **Sviluppo incrementale** supportato attraverso **piccole e frequenti release** del sistema.
- **Requisiti** basati su **semplici scenari** utilizzati come base per decidere quale funzionalità deve essere inclusa in un incremento del sistema.

eXtreme Programming

- **Coinvolgimento dell'utente** supportato attraverso l'impegno costante del cliente nel team di sviluppo.

- **Persone** (anziché processo) supportate da:
  - programmazione in coppia,
  - possesso collettivo del codice sorgente,
  - un processo di sviluppo sostenibile che non richiede tempi di lavoro eccessivamente lunghi.

Storie utente

- Requisiti dedotti attraverso **storie utente**, ovvero scenari d'uso in cui potrebbe trovarsi un utente del sistema.
- Cliente e sviluppatori redigono una **story card** che raccoglie le esigenze dell'utente.

Story card: esempio

Prescrizione dei farmaci
Kate è un dottore che desidera prescrivere un farmaco a un paziente che frequenta una clinica. La cartella clinica del paziente è già visualizzata sul suo computer, quindi fa clic sul campo del farmaco e può selezionare "farmaco corrente", "nuovo farmaco" o "formulario".
Se sceglie "farmaco corrente", il sistema le chiede di controllare la dose. Se Kate vuole cambiare la dose, digita la nuova dose e conferma la prescrizione.
Se sceglie "nuovo farmaco", il sistema suppone che Kate conosca quale farmaco prescrivere. Kate digita le prime lettere del nome del farmaco. Il sistema visualizza una lista di possibili farmaci che iniziano con quelle lettere. Kate seleziona il farmaco richiesto e il sistema risponde chiedendole di verificare se il farmaco selezionato è corretto. Lei digita la dose e conferma la prescrizione.
Se sceglie "formulario", il sistema visualizza una finestra di ricerca per il formulario approvato. Kate può cercare il farmaco richiesto e poi lo seleziona. Il sistema le chiede di verificare se il farmaco scelto è corretto. Lei digita la dose e conferma la prescrizione.
Il sistema controlla sempre che la dose sia entro i limiti consentiti; in caso contrario, chiede a Kate di cambiare la dose.
Dopo che Kate ha confermato la prescrizione, dovrà controllarla e poi fare clic su "OK" o su "Cambia". Se fa clic su "OK", la prescrizione viene memorizzata nel database di controllo. Se fa clic su "Cambia", il sistema riavvia il processo di prescrizione dei farmaci.

Task

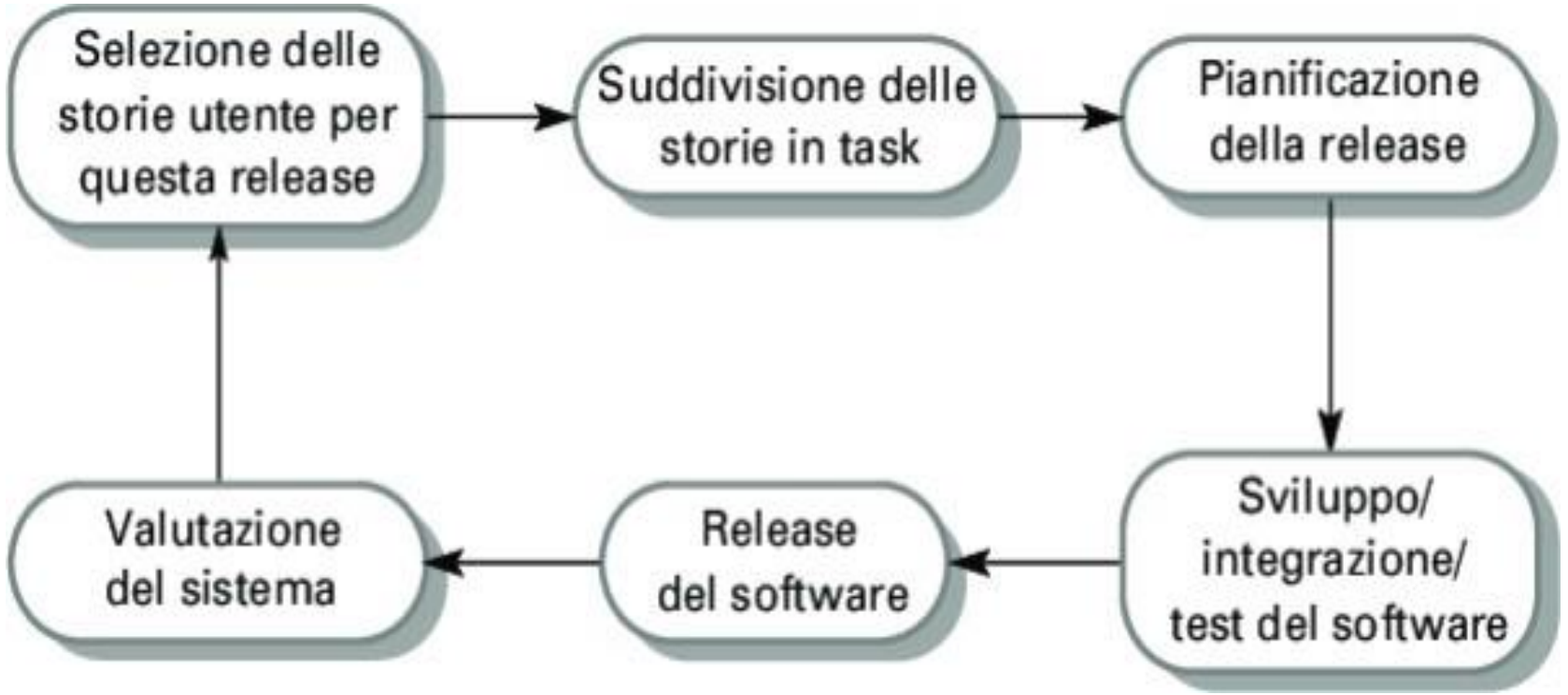
- Ogni **story card** viene **suddivisa in task**.
- **Stima** delle **risorse** e degli **sforzi richiesti** per implementare ciascun task.
- I task sono le **unità principali** dell'implementazione.
- Storie ordinate per **priorità** dal cliente, iniziando da quelle che possono fornire immediatamente un supporto utile all'azienda.

Esempio di Task

Task 1: cambia la dose del farmaco prescritto
Task 2: scelta dal formulario
Task 3: controllo della dose
Il controllo della dose è una precauzione per verificare che il dottore non abbia prescritto una dose pericolosamente piccola o grande.
Utilizzando il codice del formulario come nome del farmaco, ricerca nel formulario le dosi minima e massima consigliate.
Confronta la dose prescritta con la minima e la massima. Se la dose è fuori dal range ammesso, visualizza un messaggio di errore per segnalare che la dose prescritta è troppo piccola o troppo grande. Se è all'interno del range, abilita il pulsante "Conferma".


Organize Email
Manage Email
Manage Calendar
Manage Contacts
Search Email
File Emails
Compose Email
Read Email
Delete Email
View Calendar
Create Appt
Update Appt
View Appt
Create Contact
Update Contact
Delete Contact
Search by Keyword
WIP
Move Emails
Create and send basic email
Done
Open basic email
Done
Delete email
View list of appts
Done
Create basic appt
Done
Update contents /location
View Appt
Done
Create basic contact
Done
Update contact info
WIP
Create sub folders
Done
Send RTF e-mail
Open RTF e-mail
View Monthly formats
WIP
Create RTF appt
Accept/ Reject/T entative
Release 1
Limit Search to one field
Send HTML e-mail
Open HTML e-mail
Empty Deleted Items
View Daily Format
Create HTML appt
Propose new time
Add address data
Update Address Info
Delete Contact
Limit Search to 1+ fields
Set email priority
Open Attachments
Mandatory/Optional
Example story map created by Steve Rogalsky
http://winnipegagilist.blogspot.com
Release 2
Search attachments
Get address from contacts
View Weekly Formats
Get address from contacts
View Attachments
Import Contacts
Search sub folders
Send Attachments
Search Calendar
Add Attachments
Export Contacts
Release 3

Il ciclo di rilascio



Evoluzione delle storie utente

- Il problema principale delle storie utente è che potrebbero essere incomplete o poco veritiere.
  - Se i **requisiti variano**, le storie non implementate possono cambiare o essere scartate.
  - Se sono **richieste modifiche** per un sistema già consegnato, **nuove story card** vengono sviluppate.

Test e refactoring

- Modifiche supportate da:
  - regolari release del sistema,
  - **sviluppo preceduto da test**,
  - **refactoring**,
  - integrazione continua di nuove funzionalità.
- Mantenimento della semplicità supportate da:
  - costante refactoring che migliora la qualità del codice,
  - uso di semplici progetti che non necessariamente prevedono future modifiche del sistema.

Refactoring

- Il **refactoring** è una pratica che richiede al team di programmazione di ricercare possibili miglioramenti del software e di implementarli immediatamente (anche se non c'è una immediata necessità).
  - La programmazione estrema preferisce il refactoring alla progettazione che anticipa il cambiamento, considerata uno sforzo quasi inutile quando i requisiti sono molto instabili.

Refactoring

- Il refactoring migliora la **struttura** e la **leggibilità** del codice, contrastando il deterioramento strutturale che si verifica naturalmente quando si modifica il software.
- Esempi:
  - Riorganizzazione della gerarchia delle classi per eliminare codice duplicato.
  - Riordino e ridenominazione di attributi e metodi.
  - Sostituzione di sezioni simili con chiamate a metodi definiti in una libreria di programma, ecc.

Sviluppo con test iniziali

- I test vengono scritti **prima** del codice da testare.
  - Le interfacce e le specifiche comportamentali per le funzionalità da sviluppare sono quindi ben definite.
  - Si risolvono subito eventuali ambiguità e omissioni nelle specifiche.

Sviluppo con test iniziali

- Ogni task genera uno o più test.
- I test sono automatizzati e lo sviluppo non può procedere finché tutti i test non sono stati superati con successo.
  - Esistono vari framework per l'automatismo dei test, come ad esempio *Junit* per Java.
- Il cliente aiuta a sviluppare i **test di accettazione** delle storie.
  - Fanno uso di dati del cliente per verificare se il sistema soddisfa le reali esigenze del cliente.

Sviluppo con test iniziali

- Test automatici eseguiti ogni volta che viene **aggiunto nuovo codice**.
- Problemi con **vecchi e nuovi test** identificati immediatamente.
- **Test aggiornati** ogni volta che vengono trovati nuovi problemi non precedentemente testati (lo stesso problema non si deve verificare mai due volte).

Esempio di test case

Task 4: controllo della dose
Input: 1.Un numero in mg rappresenta una singola dose di farmaco. 2.Un numero rappresenta il numero di singole dosi giornaliere. Test: 1.Controlla l'input nel quale la singola dose è corretta, ma la frequenza è troppo alta. 2.Controlla l'input nel quale la singola dose * la frequenza è troppo grande o troppo piccola. 3.Controlla l'input nel quale la singola dose * la frequenza è nel range consentito. Output: OK o messaggio di errore per segnalare che la dose è fuori dal range consentito.

Programmazione a coppie

- Coppie di programmatori siedono alla stessa postazione di lavoro per sviluppare il software.
    - La proprietà e la responsabilità del sistema è collettiva.
        - Se c'è un problema, tutto il team ne è responsabile.
    - Refactoring non percepito come un'attività che danneggia la produttività del singolo programmatore.

Legge di Linus:
Dato un numero sufficiente di occhi, tutti i bug vengono a galla.

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Gestione della progettazione

- In contrasto rispetto ai processi plan-driven, i **metodi agili** inizialmente supportavano una **pianificazione informale** e **un'organizzazione autonoma dei team**, i quali producevano poca documentazione e lavoravano in cicli di sviluppo molto brevi.
- In qualsiasi organizzazione, i manager hanno però bisogno di sapere che cosa sta accadendo, se un progetto potrà raggiungere i suoi obiettivi e se il software sarà consegnato in tempo con il budget previsto.

Scrum

- Per risolvere questo problema, nei primi anni 2000 fu sviluppato il metodo agile **Scrum**:
  - Framework per organizzare progetti agili.
  - Buona visibilità esterna su ciò che sta accadendo durante il processo di sviluppo.

Scrum

- Al fine di integrarsi più facilmente con i metodi già esistenti in un'organizzazione, scrum non richiede necessariamente l'uso di specifiche pratiche di sviluppo, come la programmazione a coppie e lo sviluppo con test iniziali.
- Per questi motivi, scrum è oggi il metodo più utilizzato per la produzione di software.

Product backlog

- Ogni **team di sviluppo** è un'**organizzazione autonoma** che non dovrebbe avere più di sette persone al suo interno.
- Un **product backlog** è una lista di elementi di cui si deve occupare il team.
  - Definizione di **caratteristiche per il software**, **requisiti del software**, **storie utente**, definizione dell'**architettura**, **documentazione** per l'**utente**.

Product owner

- Ogni prodotto ha un **product owner** che:
  - Identifica **caratteristiche** e **requisiti** del prodotto.
  - Stabilisce le priorità, e rivede continuamente il backlog per tenerlo allineato.
- Può essere un **rappresentante del cliente** o un **manager** dell'azienda.

Sprint

- L’input del processo scrum è dunque **il product backlog**.
- Ogni iterazione del processo, detta **sprint**, genera un **incremento del prodotto** testato e in teoria pronto per essere **rilasciabile al cliente**.
  - Ogni sprint dura 2 - 4 settimane.
  - All’inizio di ogni sprint il product owner stabilisce le priorità nel product backlog.
  - Gli elementi non completati nel periodo assegnato vengono restituiti al product backlog e saranno realizzati in uno sprint successivo.

Story points

- Lo **story point** è una unità di misura della «dimensione» di una storia utente.
  - Stima le **risorse** (tempo) necessarie e il **rischio** associato.

- Dopo aver stimato la complessità delle storie, si **stima** la **velocità dei team** per avere una idea del **budge**t e del t**empo** necessario.

Planning Poker Game

- Il team assegna gli story points a ciascuna storia usando la serie di Fibonacci:
  - Esempio: 1, 2, 3, 5, 8, 13 , 21, BIG
- Si sceglie la storia più semplice e meno costosa e la si pesa 1 story point.
- Dopo di che si prende una storia media e la si mostra al team descrivendola.
  - Ciascun membro del team sceglie poi di nascosto i valore che pensa sia corretto per quella storia in base all'1 scelto.
  - Se non si riesce a valutare si sceglie "?".

Planning Poker Game

- Una volta che tutti hanno fatto la loro scelta si girano le carte.
  - Se tutti hanno scelto lo stesso valore si assegna il valore concordato.
  - Altrimenti le persone che hanno messo il valore più basso e quello più alto si confrontano.
    - Gli altri membri del team possono collaborare al dialogo chiedendo chiarimenti al Product Owner.
- Dopo qualche minuto si riprova a giocare.
  - Dopo qualche tentativo si dovrebbe arrivare alla convergenza.

Planning Poker Game

- Una storia che crea troppa discussione vuol dire che ha bisogno di approfondimenti e quindi non è pronta per gli sprint imminenti.

Legge di Sayre:
in ogni controversia l'intensità del sentimento è inversamente proporzionale al valore dell'argomento trattato.

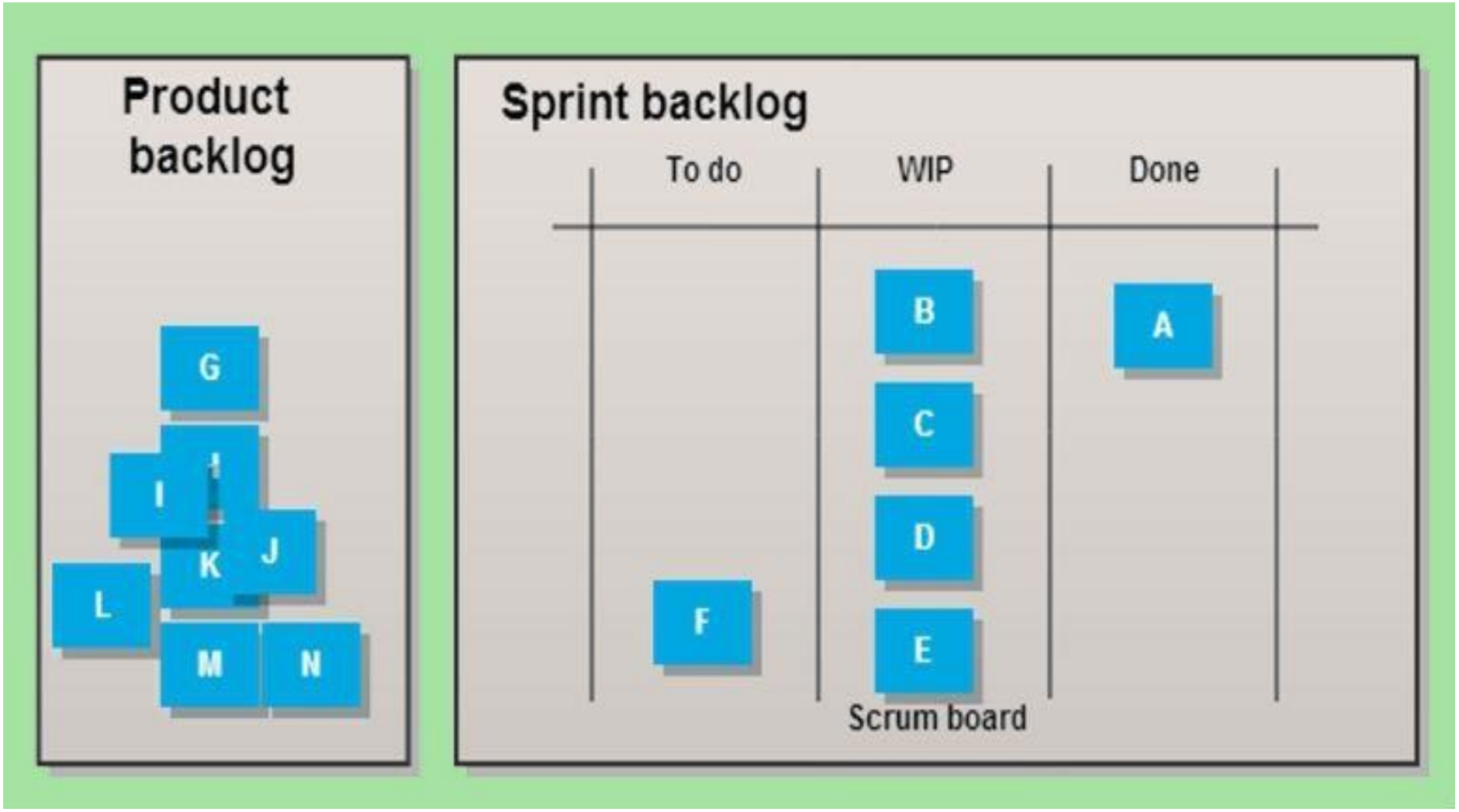
Sprint backlog e scrum

- Lo **sprint backlog** è un piano di lavoro per lo sprint (priorità degli elementi, stima risorse).
- Lo **scrum** è una breve riunione giornaliera in cui il team:
  - Esamina l'avanzamento del lavoro.
  - Stabilisce le priorità del lavoro da svolgere in quel giorno.
  - Discute eventuali problemi riscontrati nello sviluppo.

Sprint backlog e scrum

- Le interazioni giornaliere possono essere coordinate attraverso una lavagna che riporta note sullo sprint backlog, sul lavoro svolto, sull'indisponibilità delle persone, ecc.
- Alla fine di ogni sprint si tiene un riunione di verifica, utile a migliorare la qualità del processo, e a capire lo stato di avanzamento del prodotto al fine di revisionare il product backlog.

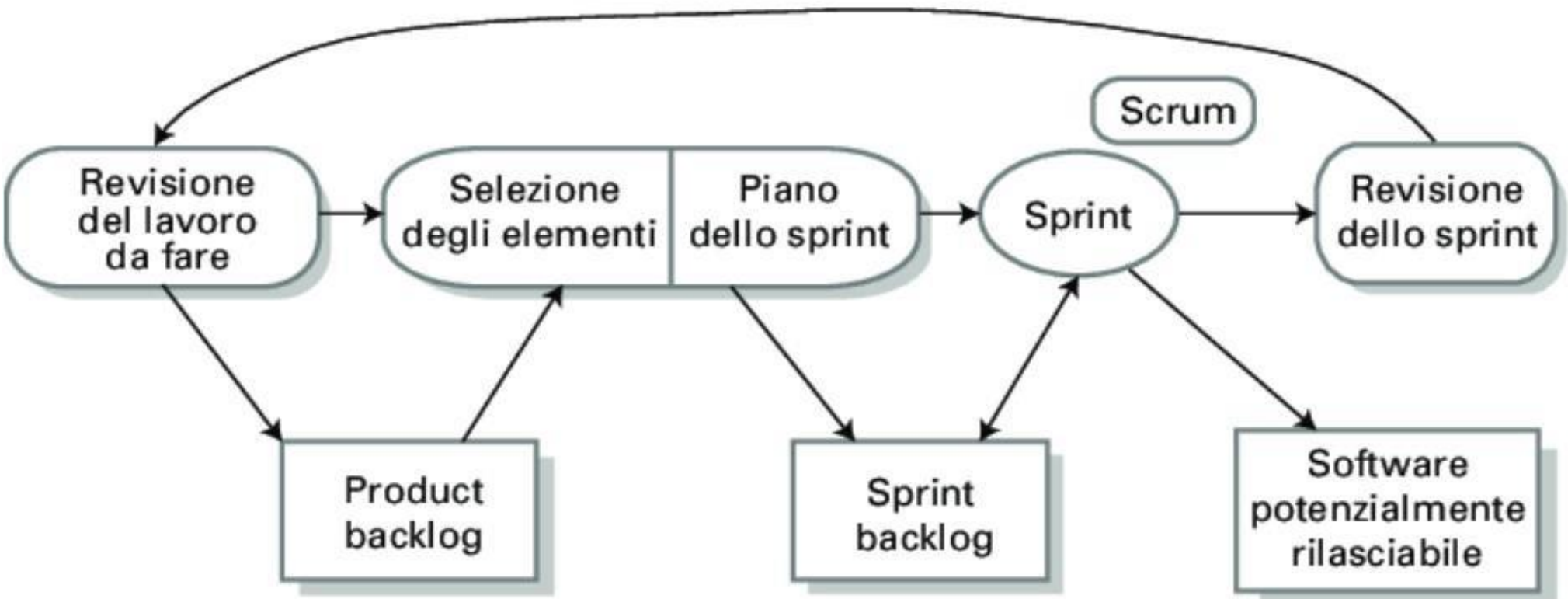
Sprint backlog e scrum



Che significa «done»?

- Tipicamente:
  - Codifica della funzionalità richiesta completata
  - Test di unità scritti e superati
  - Test di integrazione superato
  - Test prestazionale superato
  - Documentazione (minimale) scritta
  - Approvato dal product owner

Scrum


Scrum
Revisione del lavoro da fare
Selezione degli elementi
Piano dello sprint
Sprint
Revisione dello sprint
Product backlog
Sprint backlog
Software potenzialmente rilasciabile

ScrumMaster

- Lo **ScrumMaster:**
  - Ha la responsabilità di garantire che il processo di scrum sia seguito e sia efficiente.
  - Si interfaccia inoltre con il resto dell'organizzazione, riferendo sullo stato del progetto e partecipando alla sua pianificazione.

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Problemi pratici dei metodi agili

- L'**informalità** dello **sviluppo agile** è spesso incompatibile con l'**approccio legale** alla **definizione dei contratti** che di solito si usano nelle grandi società.
  - Non è possibile **stabilire i requisiti definitivi** in anticipo.
  - Ci si deve basare su contratti in cui è il **tempo di sviluppo ad essere pagato**, piuttosto che un insieme di requisiti.

Problemi pratici dei metodi agili

- I metodi agili sono più indicati per lo sviluppo di nuovo software, mentre sono poco adatti alla manutenzione.
  - Mancanza di documentazione.
  - Difficoltà nel mantenere coinvolto il cliente.
  - Difficoltà nel garantire continuità al team di sviluppo.
- I metodi agili sono pensati per piccoli team che lavorano fisicamente insieme.

Scalabilità dei metodi agili

- Principali problematiche legate alla **produzione di sistemi software su vasta scala**:
  - **Team diversi** spesso distanti geograficamente.
    - Difficoltà di **comunicazione**, e di **condivisione della responsabilità**.
  - **Interazione/integrazione con altri sistemi esterni.**
    - Molti requisiti riguardano questa interazione e non si prestano allo sviluppo incrementale.
    - **Attività di configurazione**, anch'essa poco adatta allo sviluppo incrementale e all'integrazione continua.
  - **Regole e norme esterne** che potrebbero richiedere la produzione di alcuni tipi di documenti e **specifici requisiti di conformità**.
  - **Tempi lunghi di sviluppo**, difficile garantire continuità ai team.
  - Molti **stakeholder con prospettive e obiettivi diversi**. Non è possibile, e spesso comunque controproducente, coinvolgerli tutti nello sviluppo.

Scalabilità dei metodi agili

- Esistono vari approcci per **adattare i metodi agili** alla produzione di sistemi su **vasta scala**.
- Punti comuni:
  - Approccio completamente incrementale all'ingegneria dei requisiti non sostenibile.
    - Lavoro preliminare volto a individuare le diverse parti del sistema, i cui requisiti saranno alla base del contratto col cliente.
  - Un solo **product owner non sufficiente**.
    - Coinvolgimento di più rappresentanti del cliente e di più persone nello sviluppo che comunicano e negoziano.
  - **Meccanismi di comunicazione tra team** usando vari strumenti (telefono, e-mail, wiki, social network).
  - **Integrazione continua**, ovvero la ricostruzione dell'intero sistema ad ogni modifica di un componente, non sostenibile.
    - **Integrazione frequentemente e release regolari**.
    - **Utilizzo di strumenti di gestione della configurazione** che supportano più team, multiteam,

Scrum multi-team

- Per la produzione di software su vasta scala, il metodo scrum viene adattato per funzionare con più team.
- Ogni team ha un suo product owner e un suo ScrumMaster.
- Potrebbero esserci un product owner e uno ScrumMaster anche per l'intero progetto.
- Ogni team ha un **architetto di prodotto**, che collabora alla progettazione e al progresso dell'intera architettura.

Scrum multi-team

- Le date delle release dei singoli team vengono allineate, al fine di produrre un sistema dimostrativo completo.
- Ogni giorno si svolge uno **scrum di scrum**, dove i rappresentanti dei vari team si incontrano per discutere l'avanzamento dei lavori, identificare i problemi e pianificare il lavoro da fare in quel giorno.

Oltre i metodi agili: Software Craftsmen

- Non solo software funzionante, ma anche **software ben realizzato**
- Non solo rispondendo al cambiamento, ma **aggiungendo costantemente valore**
- Non solo individui e interazioni, ma anche una **comunità di professionisti**
- Non solo collaborazione con i clienti, ma anche **collaborazioni produttive**

Cioè, nella ricerca degli oggetti a sinistra abbiamo trovato indispensabili gli oggetti a destra.

Programma

- I processi per la produzione di software
- Modello a cascata
- Modelli evolutivi
- Modello a spirale
- Sviluppo agile
- Scrum
- Problemi e scalabilità dei metodi agili
- DevOps

Cicli di rilascio brevi

- Lo sviluppo agile del software promuove la collaborazione e introduce **brevi cicli di rilascio**.
  - Rilasciando una versione di **software ancora incompleto ma già utilizzabile**, il **feedback reale** dell'utente o del cliente diventa subito disponibile.
  - Questo feedback viene quindi utilizzato per meglio **indirizzare lo sforzo di sviluppo**, aumentando l'**affidabilità** del processo.
  - Anche il rendimento è aumentato, perché viene minimizzato lo sforzo messo in caratteristiche non necessarie.

Cicli di rilascio brevi

- Circa 10 anni fa si comprese che i metodi agili potevano essere ancora migliorati al fine di **ridurre ulteriormente i tempi** che intercorrono tra i **rilasci**.
- I benefici che si ottengono sono:
  - **Maggiore tempesitività** del processo di produzione
  - **Maggiore flessibilità** rispetto a cambiamenti nei requisiti
  - **Release più piccole** e più stabili
  - **Rilascio di fix** più rapido

Fusione tra sviluppo e rilascio

- L’obiettivo è migliorare i livelli di condivisione e di integrazione tra sviluppatori e sistemisti per accelerare i tempi di progettazione, testing e di rilascio delle soluzioni applicative aziendali sia in ambienti tradizionali che in ambienti cloud.
- Il **DevOps** è un set di pratiche e di cambiamenti di processo supportati da strumenti automatici, al fine di **automatizzare il rilascio del software** rispetto alla sua catena di produzione, ottenendo un software di qualità superiore e sicuro in modo estremamente più rapido.

Fusione tra sviluppo e rilascio

- Vengono ridisegnati i **team di lavoro** per fondere sviluppo (development) la produzione (operation).
- I classici **team raggruppati per specializzazioni** funzionali (database, front-end, back-end,…) vengono sostituiti da **team inter-funzionali**, normalmente piccoli (8-12 persone) in modo che possano mantenere un focus su un aspetto specifico del prodotto complessivo.

Werner Vogels (Amazon CTO):
as you build it, you run it.

DevOps

- Secondo il modello DevOps**, sviluppo** e **produzione** sono fusi in un'unica unità in cui i **tecnici sono attivi** lungo tutto il ciclo di vita dell'applicazione (sviluppo- test-distribuzione-produzione), e acquisiscono una serie di competenze non limitate da una singola funzione.

  - In alcuni modelli DevOps, anche i team dedicati a **controllo della qualità e sicurezza** spesso sono coinvolti più direttamente nelle fasi di sviluppo e gestione in tutto il ciclo di vita dell'applicazione.

  - Quando in un team di DevOps la sicurezza è il punto focale di tutti, a volte prende il nome di **DevSecOps**.

Prassi DevOps

- Integrazione continua
- Distribuzione continua
- Microservizi
- Infrastruttura come codice
- Monitoraggio e accessi
- Comunicazione e collaborazione

Integrazione continua

- Come visto in precedenza, l'**integrazione continua** è una pratica di sviluppo software in cui gli sviluppatori aggiungono regolarmente modifiche al codice in un **repository centralizzato**, quindi la creazione di **build** e i **test** vengono eseguiti **automaticamente**.
- Gli **obiettivi principali** dell'integrazione continua sono **individuare e risolvere i bug** con maggiore tempestività, **migliorare la qualità del software** e **ridurre il tempo** richiesto **per convalidare** e **pubblicare** nuovi aggiornamenti.

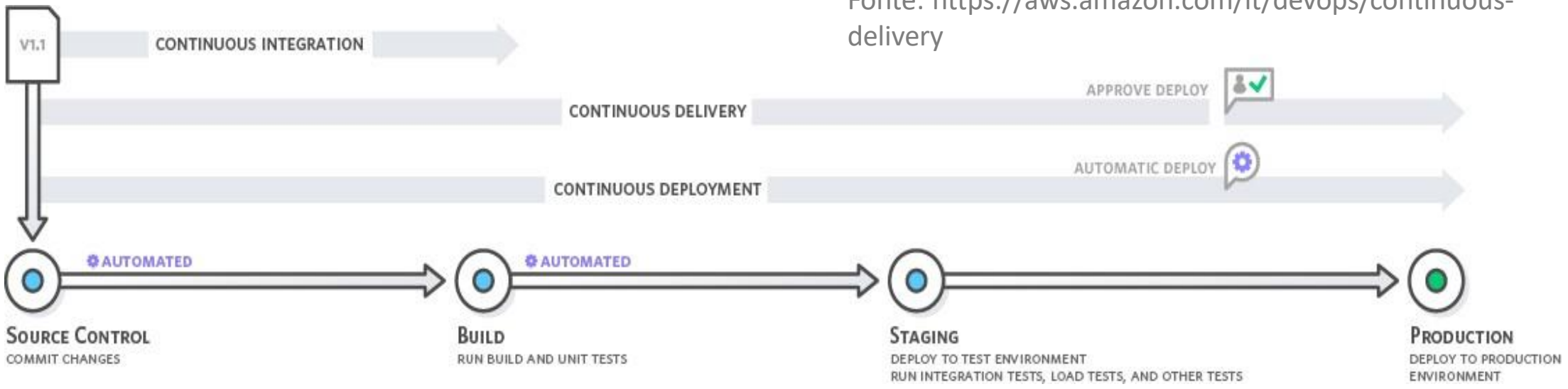
Distribuzione continua

- La **distribuzione continua** prevede che le **modifiche** al codice vengano **applicate a una build**, **testate** e preparate per il **rilascio in produzione** in modo **automatico**.
- Estende l'**integrazione continua** distribuendo tutte le modifiche al codice all'ambiente di testing e/o di produzione dopo la fase di creazione di build.
- Gli sviluppatori hanno sempre a disposizione una **build temporanea** pronta per la distribuzione che ha già passato un **processo di testing standardizzato**.

Distribuzione continua

- Ogni nuova versione del codice inoltrata attiva un flusso di lavoro automatizzato che applica a una build, testa e approva temporaneamente l'aggiornamento. La decisione finale per implementare il nuovo software nell'ambiente di produzione attivo dipende poi dallo sviluppatore.

Fonte: https://aws.amazon.com/it/devops/continuous-delivery



Microservizi

- L'architettura a microservizi è un approccio alla progettazione in cui un'**applicazione** si **basa su un gruppo di servizi di piccole dimension**i (ci torneremo più avanti).
- Per **scrivere microservizi** e **distribuirli** (in modo indipendente, come servizio singolo o come gruppo di servizi) è possibile utilizzare diversi framework o linguaggi di programmazione.

Infrastruttura come codice

- **Infrastructure as Code** (IaC) è una prassi per l'automatizzazione via codice delle fasi di provisioning e gestione dell'infrastruttura.
- Si basa su sistemi di containerizzazione o su modelli di gestione cloud tramite API, che permettono agli sviluppatori di interagire con l'infrastruttura in modo programmatico e su larga scala, piuttosto che tramite l'impostazione e la configurazione manuale delle risorse.

Infrastruttura come codice

- Poiché possono essere definiti tramite codice, l'infrastruttura e i server possono essere distribuiti con la massima rapidità tramite modelli standardizzati, aggiornati con patch e versioni più recenti o duplicati in modo iterabile.
- Esempi di tecnologie e piattaforme che supportano IaC: Docker, Amazon AWS, Google Cloud Platform,…

Monitoraggio e accessi

- Diventa fondamentale **monitorare** i **parametri** e i **log del sistema** per scoprire in che modo le prestazioni di applicazione e infrastruttura influiscono sull'esperienza dell'utente finale.
- Per capire **l'impatto di modifiche o aggiornamenti** sugli utenti finali, si devono acquisire dati e log generati da applicazioni e infrastruttura, suddividerli in categorie e analizzarli, esaminando le possibili cause primarie dei problemi o le modifiche impreviste.
- Anche la **creazione di allarmi o l'analisi dei dati** in tempo reale sono attività che aiutano a **monitorare proattivamente** i propri servizi.

Monitoraggio e accessi

aws Azure AWS Lambda Apache Kafka kubernetes mongoDB

PRODUCT FEATURES

Three pillars of observability in one platform

- Seamlessly unites metrics, traces, and logs
- Aggregate metrics and events from 400+ technologies
- Search, analyze, and explore enriched log data
- Trace requests across distributed systems and alert on app performance
- Seamlessly pivot between correlated data for rapid troubleshooting

Fonte: https://www.datadoghq.com/

Comunicazione e collaborazione

- Uno degli aspetti cruciali dell'approccio DevOps è l'aumento di comunicazione e collaborazione interne all’organizzazione.
- L'uso di strumenti DevOps e l'automatizzazione dei processi di distribuzione software prevede maggiore collaborazione tramite l'unione di flussi di lavoro e responsabilità in genere distribuiti tra sviluppatori e produzione.

Approfondimenti

- Ghezzi, Jazayeri, Mandrioli: «Ingegneria del software», 2° Edizione, Pearson (2004)
  - Capitolo 7
- Sommerville: «Software Engineering», 10° Edizione, Pearson (2016)
  - Capitoli 2 e 3
- http://agilemanifesto.org/iso/it/manifesto.html
- http://manifesto.softwarecraftsmanship.org/
- https://aws.amazon.com/it/devops/what-is-devops/