Domanda 1 - Change Management (CM)

Il Configuration Management è come essere il bibliotecario perfetto di tutto ciò che compone il tuo software. Immagina di gestire una ricetta complessa con centinaia di ingredienti: devi sapere esattamente quale versione di ogni ingrediente hai usato, quando, e come ricreare esattamente lo stesso piatto.

Nel software, il CM si occupa di **tracciare e controllare sistematicamente** tutti gli elementi che compongono il sistema: codice sorgente, configurazioni, dipendenze, script di deployment, documentazione. È come avere un DNA completo del tuo software in ogni momento.

Pensa a quattro pilastri fondamentali:

Identificazione: ogni componente ha un'identità precisa (come i codici a barre nei supermercati). Il tuo JAR versione 2.3.1 è diverso dalla 2.3.0 e devi saperlo sempre.

Controllo: nessuna modifica avviene a caso. È come avere un portiere che controlla chi entra e esce, registrando tutto.

Tracciabilità: puoi sempre rispondere a "da dove viene questo bug?" risalendo esattamente a quale commit, quale versione, quale configurazione lo ha introdotto.

Audit: verifichi che quello che hai in produzione corrisponda esattamente a quello che pensi di avere.

Senza CM è come cucinare senza ricetta: magari funziona una volta, ma quando devi rifare il piatto (o risolvere un problema) sei completamente perso. Con il CM, hai sempre il controllo totale del tuo ecosistema software.

> Esempio di CM Tool: Maven (tracci dipendenze, hai le configurazioni, etc.)

Domanda 2: Scrum vs Agile

Agile è una **filosofia di sviluppo software** nata dalla frustrazione verso i metodi tradizionali "a cascata". È come passare dal costruire una cattedrale (progetto rigido, tutto pianificato nei minimi dettagli per anni) al fare giardinaggio (adattivo, che cresce e si modifica in base alle condizioni).

La filosofia Agile si basa su quattro valori fondamentali: privilegiare le **persone e le interazioni** rispetto a processi e strumenti, il **software funzionante** rispetto alla documentazione esaustiva, la **collaborazione con il cliente** rispetto alla negoziazione contrattuale, e la **risposta al cambiamento** rispetto al seguire un piano rigido.

È un **mindset** che abbraccia l'incertezza e la vede come opportunità. L'idea è che i requisiti cambiano, gli utenti scoprono cosa vogliono veramente solo vedendo il prodotto, e quindi è meglio essere pronti ad adattarsi piuttosto che seguire ciecamente un piano iniziale.

Scrum, invece, è una **metodologia specifica** che implementa i principi Agile. È come prendere la filosofia "facciamo giardinaggio" e dire "ecco esattamente come organizziamo il lavoro": Sprint di durata fissa, ruoli definiti (Product Owner, Scrum Master, Development Team), cerimonie precise (Sprint Planning, Daily Standup, Sprint Review, Retrospective).

Agile ti dice **cosa fare** (collabora, adattati, consegna valore frequentemente), Scrum ti dice **come farlo** (in cicli di 2-4 settimane, con questi ruoli, seguendo queste regole). Agile è il "perché", Scrum è il "come".

Esistono altri framework Agile come Kanban o XP, ognuno con la propria interpretazione di come mettere in pratica la filosofia.

> Nota: Esistono vari framework Agile, Scrum è uno!
> https://www.parabol.co/resources/agile-frameworks-guide/

Domanda 3: Analisi statica

L'**analisi statica** è come avere un correttore di bozze esperto che esamina il tuo codice senza mai eseguirlo. È l'opposto dell'analisi dinamica: invece di far girare il programma per vedere cosa succede, "legge" il codice sorgente per individuare problemi potenziali.

Pensa a un revisore che controlla un manoscritto: può trovare errori di grammatica, stile inconsistente, frasi ambigue, senza mai dover "eseguire" il testo. L'analisi statica fa lo stesso con il codice.

Cosa rileva:

- **Errori sintattici** e violazioni delle convenzioni di codifica
- **Vulnerabilità di sicurezza** (buffer overflow, SQL injection)
- **Code smells** (codice duplicato, metodi troppo lunghi, classi troppo complesse)
- **Bug pattern** comuni (variabili non inizializzate, dead code, memory leak potenziali)
- **Metriche di qualità** (complessità ciclomatica, accoppiamento tra classi)

Esempi di strumenti:

- **SonarQube**: il più completo, offre dashboard per monitorare la qualità nel tempo
- **Checkstyle**: si concentra sulle convenzioni di stile e formattazione
- **FindBugs/SpotBugs**: specializzato nel trovare bug pattern in Java
- **PMD**: rileva problemi di design e code smell
- **ESLint** per JavaScript, **RuboCop** per Ruby

Vantaggio principale: trova problemi **prima** che il codice vada in produzione, senza dover scrivere test specifici. È come avere un metal detector che trova problemi nascosti nel terreno del tuo codice.

L'analisi statica è **white box** (hai accesso al sorgente), **non funzionale** (non testa le funzionalità) e **statica** (non richiede esecuzione).