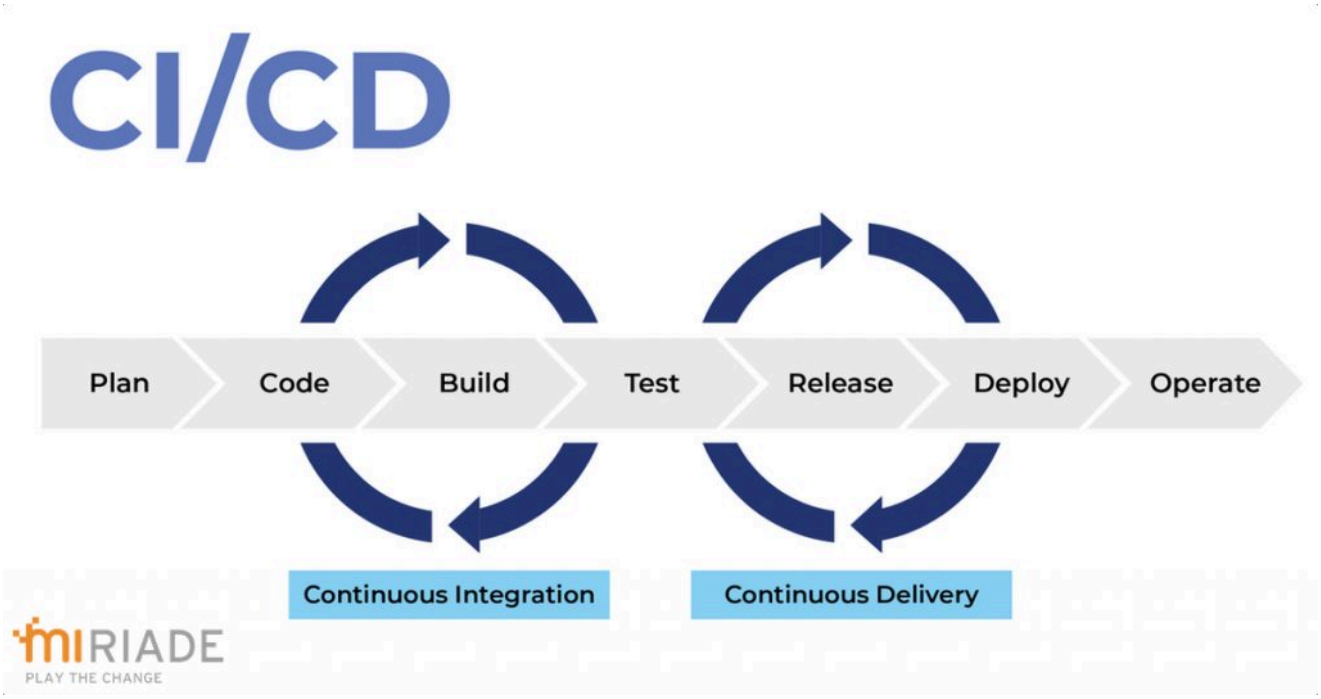
Domanda 1



Based on the course materials, the correct statements about Continuous Delivery are:

A. Il rilascio può essere fatto in ogni momento in ogni ambiente (premendo un bottone). D. Un cambiamento al sistema può potenzialmente essere rilasciato in produzione. E. Continuous Delivery è il passo successivo alla Continuous Integration.

Analysis:

A is correct - This aligns with Martin Fowler's definition: "You can perform push-button deployments of any version of the software to any environment on demand."

B is false - System tests are actually a crucial component of CD. The materials emphasize "Continuous Testing" as a requirement, including automated tests at the system level.

C is false - This describes Continuous **Deployment**, not Continuous **Delivery**. CD enables automated release capability but keeps the actual production deployment as a manual decision ("premendo un bottone").

D is correct - This directly matches the definition: "ogni cambiamento al sistema può potenzialmente essere rilasciato in produzione."

E is correct - The materials explicitly state: "La Continuous Delivery è il passo successivo alla Continuous Integration" and explain how CD builds upon CI by extending beyond just compilation and unit testing to include deployment readiness.

The key distinction is that Continuous Delivery ensures software *can* be released at any time through automated pipelines, while keeping the final production release as a manual trigger.

Continuous Deployment would automate that final step as well.

Domanda 2

Un Issue Tracking System (ITS) è uno strumento fondamentale nel processo di sviluppo software moderno. Si tratta di un sistema che permette di registrare, monitorare e gestire problemi, richieste, bug e attività legate allo sviluppo di un progetto software

Es. GitHub / GitLab

Ogni oggetto è un work item su cui lavorare e risolvere tracciando i problemi.

Domanda 3

Le attività del Project Manager nella fase di **configurazione** di un ITS sono:

A. Crea un nuovo progetto. C. Aggiunge gli utenti e assegna ruoli/permessi. F. Definisce il processo da seguire (tipi di work item, campi custom, workflow, collegamenti).

Le altre opzioni (B, D, E) sono attività della fase di **utilizzo**, non di configurazione. Durante la configurazione si impostano struttura, processi e permessi; durante l'utilizzo si gestiscono i work item operativamente.

Domanda 4

Risposta corretta: A

A. Corretto - I test di integrazione verificano proprio i contratti di interfaccia tra più moduli o subsystem, assicurandosi che le diverse componenti comunichino correttamente tra loro.

B. Falso - I test di integrazione sono più lenti da configurare ed eseguire rispetto ai test di unità perché coinvolgono multiple componenti e possono richiedere setup più complessi (database, servizi esterni, ecc.).

C. Falso - Questa è la definizione dei **test di sistema**, che verificano l'intero sistema nel suo complesso.

D. Falso - Questa è la definizione dei **test di unità**, che verificano la singola unità più piccola testabile (classe/metodo).

Spiegazione teorica: I test di integrazione si posizionano tra i test di unità e i test di sistema nella piramide del testing. Il loro scopo è verificare che i diversi moduli, già testati individualmente, funzionino correttamente quando integrati insieme. Possono essere di due tipi: integrazione tra subsystem interni (più veloci) o tra subsystem interni ed esterni come database (più lenti e complessi).

Domanda 5

16) L'acronimo CRISP descrive le caratteristiche del processo di build che sono: 0 punti

- ○ A. Coeso, Ripetibile, Indipendente dall'ordine di esecuzione, Schedulabile, Portabile.
- ◉ B. Completo, Ripetibile, Informativo, Stoppable (che si può interrompere), Portabile.
- ○ C. Completo, Ripetibile, Informativo, Schedulabile, Portabile.
- ○ D. Cheap (economico di risorse), Rapido, Indipendente dall'ordine di esecuzione Schedulabile, Portabile.

Cancella selezione

Domanda 6

L'analisi statica del codice è un metodo di debug che esamina il codice sorgente senza eseguirlo. Questa tecnica permette di identificare potenziali problemi, vulnerabilità e violazioni degli standard di codifica nelle prime fasi del ciclo di sviluppo, quando sono più facili ed economici da correggere.

17) A quali categorie di test appartiene l'analisi statica? 0 punti

- ☐ A. Non funzionale
- ☐ B. Black box: non è a disposizione il codice sorgente del progetto
- ☑ C. Funzionale
- ☑ D. Statico
- ☐ E. Dinamico
- ☑ F. White box: è a disposizione il codice sorgente del progetto

Plugin di analisi statica: CheckStyle / ESLint

Domanda 7

18) Quale tra le seguenti è la migliore definizione di Continuous Integration? (1) 0 punti

- (●) A. Nell'ingegneria del software l'integrazione continua o Continuous Integration è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di versioning. Consiste nell'esecuzione di un processo che costruisce il prodotto, esegue i test di unità, installa il prodotto in ambienti di test dove vengono eseguiti i test di sistema e infine rilascio automatico nell'ambiente di produzione.
- ( ) B. Nell'ingegneria del software l'integrazione continua o Continuous Integration è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di versioning. Consiste nell'allineamento frequente degli ambienti di lavoro degli sviluppatore verso l'ambiente condiviso (mainline).
- ( ) C. Nell'ingegneria del software l'integrazione continua o Continuous Integration è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di versioning. Consiste nell'allineamento e verifica giornaliero, di solito durante la notte, degli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso(mainline).

Domanda 8

UAT = User Acceptance Testing --> Collaudo
SUT = System Under Testing

Risposte corrette: A e B

A. Vero - I test di accettazione sono anche conosciuti come "UAT" (User Acceptance Testing) o "End User testing".

B. Vero - Nei test di accettazione il SUT è considerato black box perché si testano i casi d'uso e i requisiti dal punto di vista dell'utente finale, senza conoscere l'implementazione interna.

C. Falso - I test di accettazione sono di tipo **dinamico** perché richiedono l'esecuzione del software.

D. Falso - Come detto sopra, il SUT è considerato black box, non white box.

E. Falso - I test di accettazione fanno parte della categoria **"Validazione"** (verificano che il prodotto soddisfi i requisiti dell'utente), non "Verifica" (che controlla la conformità alle

specifiche tecniche).

Spiegazione teorica: I test di accettazione rappresentano l'ultimo livello della piramide del testing. Sono svolti con l'utente finale/cliente per validare che il sistema soddisfi effettivamente le sue esigenze e i requisiti concordati. Si concentrano sui casi d'uso reali e sulla user experience, trattando il sistema come una "scatola nera" di cui testare solo il comportamento esterno.

Domanda 9

20) In Git, un insieme di modifiche a uno o più file (diff) che sono state esplicitamente validate (nell'area di staging) e salvate nel repository rappresentano un: 0 punti

- ○ A. Merge
- ○ B. Pull Request
- ◉ C. Commit
- ○ D. Branch

Domanda 10

Risposte corrette: B e D

A. Falso - Le priorità non fanno parte del workflow; il workflow riguarda stati e transizioni, non priorità.

B. Vero - Definizione corretta: il workflow è un insieme di **stati e transizioni** che un work item attraversa durante il suo ciclo di vita (es. Open → In Progress → Closed).

C. Falso - Il workflow non serve per "registrare" il lavoro svolto, ma per **guidare** il processo di completamento.

D. Vero - Il workflow permette di implementare un processo strutturato per completare l'attività, definendo i passaggi obbligatori e le verifiche necessarie.

Spiegazione teorica: Il workflow in un ITS è un meccanismo di controllo del processo che definisce:

- Gli **stati** possibili di un work item (To Do, In Progress, Testing, Done)
- Le **transizioni** permesse tra questi stati
- Le **regole** e i **controlli** per ogni passaggio di stato

Questo assicura che ogni attività segua un processo standardizzato e che vengano rispettate le procedure aziendali. Il workflow può essere associato a diversi tipi di work item (Bug, Task, Epic) con regole specifiche per ognuno.

Domanda 11

Risposta corretta: A

A. Vero - L'idempotenza significa che applicando la stessa operazione più volte si ottiene sempre lo stesso risultato, indipendentemente dal numero di esecuzioni.

B. Falso - Questa definizione non ha senso nel contesto dell'idempotenza. Operazioni diverse possono produrre risultati diversi.

C. Falso - Questo è l'opposto dell'idempotenza.

Spiegazione teorica: Nel Configuration Management, l'idempotenza è un principio fondamentale che garantisce la **prevedibilità** e **affidabilità** delle operazioni di configurazione.

Esempi pratici:

- **Ansible**: eseguire lo stesso playbook più volte porta sempre allo stesso stato del sistema
- **Docker**: ricostruire la stessa immagine produce sempre lo stesso risultato
- **Installazione software**: reinstallare lo stesso pacchetto non cambia lo stato finale

Questo principio è cruciale per:

- **Riproducibilità** degli ambienti
- **Sicurezza** nelle operazioni (posso ri-eseguire senza timore)
- **Automazione** affidabile delle deployment pipeline

L'idempotenza elimina gli effetti collaterali indesiderati e garantisce che il sistema raggiunga sempre lo stato desiderato, indipendentemente dalle condizioni iniziali.

Domanda 12

- pom.xml
    - File di configurazione usato da Maven (Build automation)

```
    </plugins>
    <finalName>MavenEnterpriseApp-ear</finalName>
  </build>
    <dependencies>
        <dependency>
            <groupId>com.mycompany</groupId>
            <artifactId>MavenEnterpriseApp-ejb</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>ejb</type>
        </dependency>
        <dependency>
            <groupId>com.mycompany</groupId>
            <artifactId>MavenEnterpriseApp-web</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>war</type>
        </dependency>
    </dependencies>
</project>
```

23) Quali tra le seguenti sono configurazioni che possono essere specificate nel file pom.xml? 0 punti

- [x] A. Le configurazioni dei plugin da utilizzare nel processo di build.
- [ ] B. Risultati delle esecuzioni dei test di unità.
- [x] C. Le dipendenze del progetto.
- [x] D. L'identificativo del progetto (Gruppo, Artefatto e Versione).
- [ ] E. Issue, bug e attività svolte in una determinata versione.
- [ ] F. I file da ignorare/non salvare nel VCS.

Complementare:

49) dai sorgenti di un progetto software com'è possibile capire che il processo di build è gestito con maven? 0 punti

- ( ) dalla presenza del file jenkinsfile nella root del progetto
- (•) dalla presenza del file pom.xml nella root del progetto
- ( ) dalla presenza del file travis.yml nella root del progetto
- ( ) non è possibile capirlo da sorgenti. di solito le configurazioni di un progetto maven non vengono salvate nel vsc

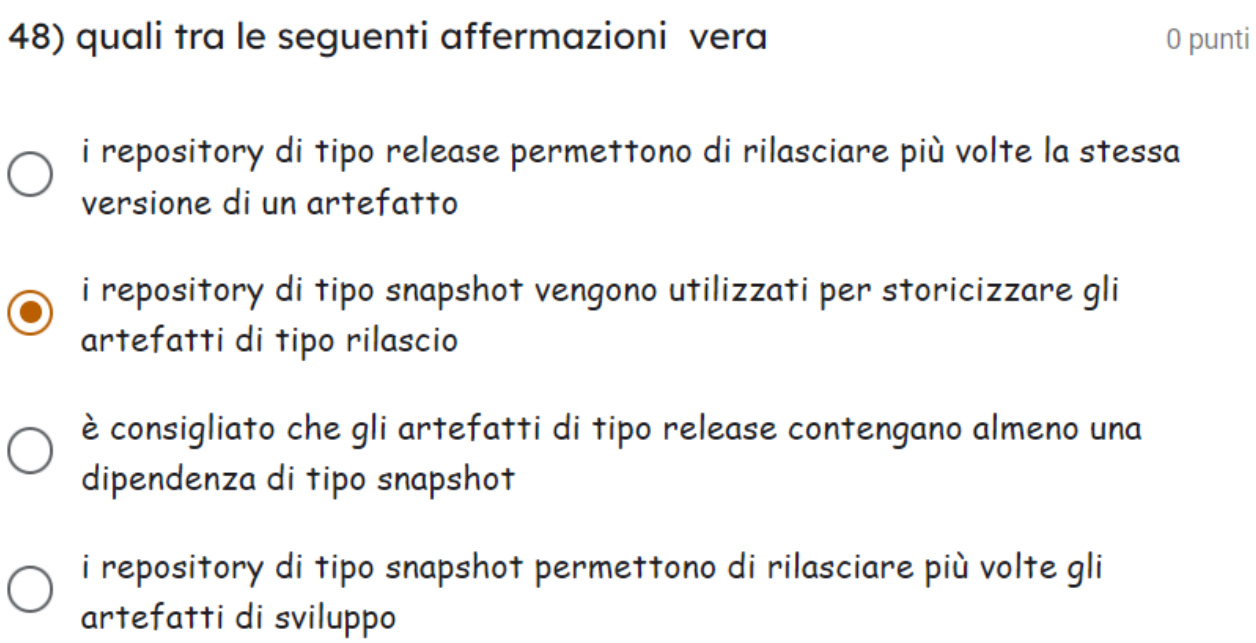
Domanda 13

La Build Automation, o automazione del processo di build, è una pratica fondamentale nello sviluppo software moderno che consiste nell'automatizzare il processo di trasformazione del codice sorgente in software eseguibile.

Questo processo include la
compilazione del codice, l'esecuzione di test, la generazione di documentazione e la creazione di pacchetti distribuibili.
L'automazione del processo di build è diventata essenziale con l'aumentare della complessità dei progetti software e con l'adozione di pratiche di sviluppo agile e DevOps.

Un processo di build automatizzato garantisce coerenza, riproducibilità e affidabilità, riducendo gli errori umani e liberando tempo prezioso per gli sviluppatori.

CRISP / Build Lifecycle --> Keywords



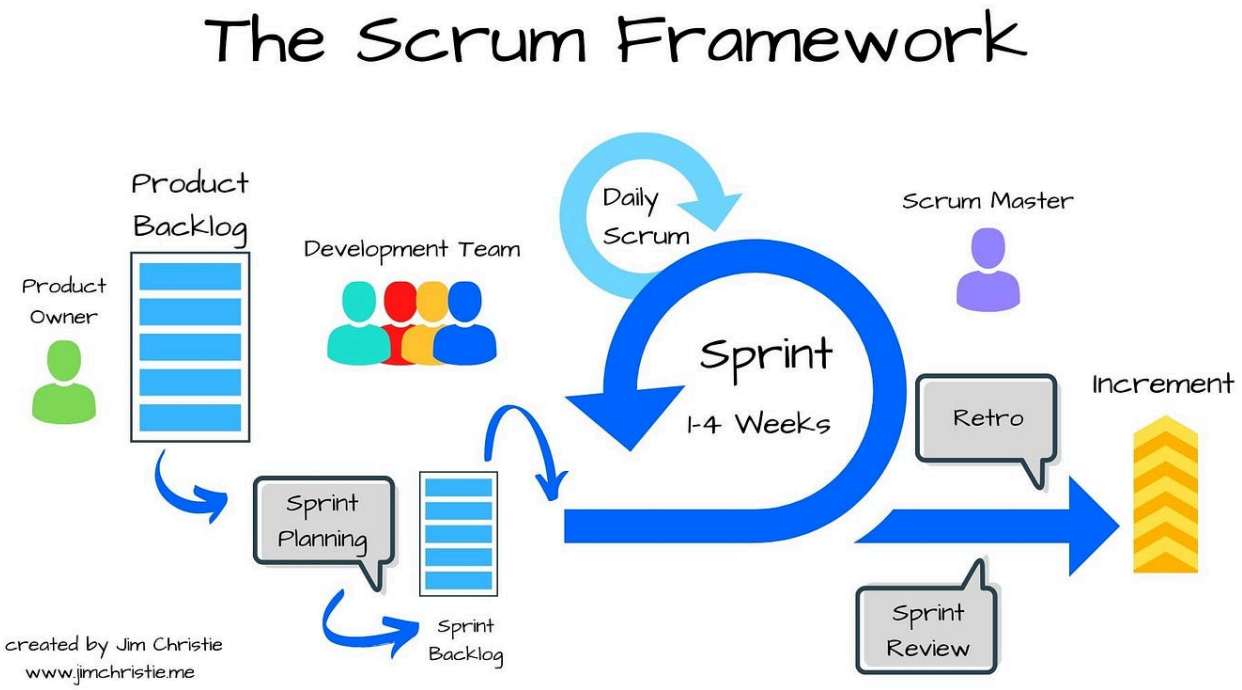
Spiegazione teorica:

- **Repository Release**: per artefatti stabili e finali, versioni immutabili
- **Repository Snapshot**: per artefatti in sviluppo, versioni mutabili con timestamp
- Gli snapshot contengono la keyword "SNAPSHOT" e permettono aggiornamenti frequenti durante lo sviluppo

Domanda 14

36) in quale evento del framework scrum: 0 punti
-il gruppo di lavoro presenta ciò che ha realzzato durante lo sprint
-viene validato e accettato quanto realizzato

- ○ sprint review
- ◉ sprint retrospective
- ○ sprint planning
- ○ daily scrum



Domanda 15

TDD = Test Driven Development

Risposta corretta: 3-1-2

L'ordine corretto del ciclo TDD è:

1. **RE(WRITE) A TEST** (fase rossa)
2. **WRITE PRODUCTION CODE** (fase verde)
3. **CLEAN UP CODE** (fase grigia/refactoring)

Spiegazione del ciclo TDD:

Fase Rossa: Si scrive prima il test per la nuova funzionalità (che deve fallire perché la funzione non esiste ancora)

Fase Verde: Si scrive il codice minimo necessario per far passare il test

Fase Grigia: Si effettua il refactoring del codice per migliorarne la qualità mantenendo i test verdi

Principio fondamentale: "Write new code only if an automated test has failed" (Kent Beck)

Questo approccio garantisce:

- Copertura di test completa
- Codice più pulito e modulare
- Sviluppo guidato dai requisiti
- Cicli di feedback rapidi

Il ciclo si ripete per ogni nuova funzionalità da implementare.

Commenti finali (non detti oggi)

- I container sono unità software standardizzate che impacchettano il codice e tutte le sue dipendenze, permettendo all'applicazione di funzionare in modo rapido e affidabile da un ambiente di computing all'altro.

Esempio verbale pratico:

Immagina di dover far girare un'applicazione web Java su diversi computer (il tuo laptop, il server di test, la produzione). Senza container avresti questi problemi:

- Sul tuo laptop hai Java 11, sul server di test Java 8, in produzione Java 17
- Ognuno ha versioni diverse di librerie, database, configurazioni
- L'app funziona da te ma non sugli altri ambienti ("ma da me funziona!")

Con i container: Invece di installare tutto su ogni macchina, "impacchetti" la tua app con tutto ciò che serve (Java, librerie, configurazioni) in un **container**. È come una valigia che contiene tutto il necessario.

Esempio pratico con Docker:

```
# Dockerfile - "ricetta" per creare il container
FROM java:11                    # Parte da Java 11
COPY myapp.jar /app/            # Copia la tua app
EXPOSE 8080                     # Espone la porta 8080
CMD ["java", "-jar", "/app/myapp.jar"]  # Comando per avviarla
```

Uso:

```
docker build -t myapp .         # Costruisce il container
docker run -p 8080:8080 myapp  # Lo avvia
```

Risultato: La tua app gira identica su qualsiasi macchina che abbia Docker, indipendentemente dal sistema operativo o dalle configurazioni locali.

Analogia: È come avere un appartamento portatile completamente arredato che puoi piazzare ovunque, invece di dover comprare mobili ogni volta che cambi casa.