

1. ISSUE TRACKING SYSTEM (ITS)

Domande Teoriche

1.1 Dare una definizione di Issue Tracking System (ITS).

Risposta: Un ITS è un software che gestisce e mantiene liste di issue secondo le necessità di un'organizzazione. Facilita la gestione del processo di sviluppo e di change management attraverso la gestione di attività diverse. Un work item è gestito in un workflow e mantenuto all'interno di un'unica piattaforma e repository.

1.2 Quali sono i tre elementi fondamentali di un ITS?

Risposta:

- **Work Item:** unità minima del progetto (analisi requisiti, sviluppo, test, bug, release, deploy, change request)
- **Workflow:** insieme di stati e transizioni che un work item attraversa durante il suo ciclo di vita
- **Repository:** dove vengono memorizzati i work item

1.3 Elenca e spiega i 5 obiettivi principali di un ITS.

Risposta:

- **Condividere:** condivisione informazioni tra team di sviluppo, project manager e cliente
- **Processo:** implementare un processo per misurare la qualità del progetto
- **Informazioni:** avere un'istantanea dello stato del progetto
- **Rilascio:** decidere quando e cosa rilasciare
- **Priorità:** assegnare priorità alle attività tramite priorità work item

Domande Pratiche

1.4 Descrivi un esempio di workflow tipico per un bug report.

Risposta: OPEN → IN PROGRESS → RESOLVED → CLOSED (con possibili transizioni: Stop Progress, Close Issue, Resolve Issue, Reopen Issue)

1.5 Quali sono le principali funzionalità di gestione, integrazione e condivisione di un ITS?

Risposta:

- **Gestione:** Ricerca avanzata, salvataggio ricerche, esportazione, reporting
- **Integrazione:** con Source Code Management, con ambiente di sviluppo

- **Condivisione:** Notifiche, bacheche/board, dashboard, definizione road map e release notes
-

2. VERSION CONTROL SYSTEM (VCS)

Domande Teoriche

2.1 Definire un Version Control System e le sue funzionalità principali.



Risposta: Un VCS è un software che tiene traccia di tutti i cambiamenti avvenuti a un insieme di file. Funzionalità:

- **Backup e Restore:** mantenere storico delle versioni
- **Sincronizzazione:** permettere a più persone di lavorare sugli stessi file
- **Short-term undo:** tornare a versioni precedenti
- **Long-term undo:** recuperare versioni molto vecchie
- **Track Changes:** vedere cosa è cambiato e quando
- **Track Ownership:** vedere chi ha fatto cosa
- **Sandboxing:** possibilità di lavorare in parallelo senza interferenze
- **Branching e Merging:** permettere sviluppo parallelo
- **Tracciabilità:** riferimento alle attività nel ITS



2.2 Confronta CVCS vs DVCS: vantaggi e svantaggi.

Risposta:

CVCS (Centralized):

-  Apprendimento più semplice, lock file
-  Meno recenti, centralized workflow, commit più lenti, single point of failure

DVCS (Distributed):

-  Più recenti, distribuiti, migliori workflow, commit veloci, no lock file
-  Apprendimento difficile

Domande Pratiche

2.3 Spiega il concetto di Forking Workflow.

Risposta: Concetti di push forward del file system distribuito. Ogni utente fa il fork del repo principale e può proporre richieste di pull tra i repo. Gestione autorizzazioni migliorata. Autonomia per processo di collaborazione. Decentrato per nuovi modelli.

2.4 Cos'è un branch e quali sono i vantaggi del branching?

Risposta: Un branch è un pointer a un singolo commit. Vantaggi: sviluppo parallelo di feature, isolamento del lavoro, possibilità di sperimentare senza impatti sul main branch.

3. FRAMEWORK SCRUM

Domande Teoriche

3.1 Definire Scrum e le sue caratteristiche principali.

Risposta: Scrum è un processo agile per sviluppo di progetti complessi che permette di concentrarsi sulla consegna del maggior valore business nel minor tempo. Caratteristiche:

- **Leggero**
- **Facile da capire**
- **Difficile da padroneggiare**

3.2 Quali sono i 3 pilastri di Scrum?

Risposta:

- **Trasparenza:** linguaggio comune per conoscenza condivisa, definition of done
- **Controllo:** ispezioni pianificate per prevenire variazioni indesiderate
- **Adattamento:** aggiustamenti per minimizzare deviazioni tramite feedback continuo

3.3 Descrivi il ruolo del Product Owner.

Risposta: Il Product Owner:

- Rappresenta il desiderio del committente
- Espone le specifiche del prodotto
- Dà priorità in base al valore sul mercato
- Si preoccupa della redditività del prodotto
- Definisce le tempistiche
- Accetta o rifiuta il lavoro svolto

Domande sui Ruoli e Eventi

3.4 Chi sono i tre ruoli principali in Scrum?

Risposta: Product Owner, Scrum Master, Development Team

3.5 Cosa sono Sprint Planning, Daily Scrum, Sprint Review e Sprint Retrospective?

Risposta:

- **Sprint Planning:** riunione preliminare per stabilire sprint goal e sprint backlog
- **Daily Scrum:** meeting giornaliero di 15 min per coordinamento
- **Sprint Review:** presentazione del lavoro fatto alla fine dello sprint
- **Sprint Retrospective:** analisi del processo per miglioramenti

Domande Pratiche

3.6 Cosa caratterizza uno Sprint in Scrum?

Risposta: Durata tipica 2-4 settimane costante. Il prodotto è progettato, realizzato e testato durante lo sprint. Non si cambia durante lo sprint. Dimensione team tipica: 7 ± 2 persone.

4. SOFTWARE TESTING

Domande Teoriche

4.1 Definire Software Testing secondo il corso.

Risposta: Software testing è un'indagine per fornire informazioni sulla qualità del software. È il processo di tutte le attività statiche e dinamiche per determinare che i prodotti soddisfino i requisiti specificati e per rilevare difetti.

4.2 Perché è necessario il Software Testing?

Risposta:

- L'uomo non è perfetto
- Il codice è scritto dagli uomini
- Il codice può contenere errori (bug)
- I bug possono avere conseguenze
- Per questo il software deve essere testato

4.3 Chi può inserire difetti nel software e con quale percentuale?

Risposta: Il Programmatore (45%) - fa errori durante sviluppo, inserisce difetti, che causano comportamenti inattesi quando si verificano condizioni non considerate.

Domande sui Tipi di Test

4.4 Distingui tra Unit Testing, Integration Testing e System Testing.

Risposta:

- **Unit Testing:** test della singola unità del codice
- **Integration Testing:** test dell'integrazione tra componenti
- **System Testing:** test dell'intero sistema

4.5 Cos'è il TDD (Test Driven Development)?

Risposta: Metodologia di sviluppo dove si scrivono prima i test e poi il codice per farli passare.

5. BUILD AUTOMATION

Domande Teoriche

5.1 Definire Build Automation e le caratteristiche CRISP.

Risposta: Processo di automazione della creazione di software build. Caratteristiche CRISP:

- **Completo:** indipendente da fonti non specificate
- **Ripetibile:** esecuzione ripetuta dà stesso risultato
- **Informativo:** fornisce info sullo stato del prodotto
- **Schedulabile:** può essere programmato automaticamente
- **Portabile:** indipendente dall'ambiente di esecuzione

5.2 Differenza tra scripting tools e artifact oriented tools nella build automation.

Risposta:

- **Scripting tools:** definiscono automatismi tramite linguaggi di scripting (.sh, .bat, Make, Gradle)
- **Artifact oriented tools:** si basano su definizione e creazione di artefatti (Apache Maven, NPM)

Domande su Maven

5.3 Definire Apache Maven e il paradigma "Convention over Configuration".

Risposta: Maven è uno strumento di gestione progetti software basato su Project Object Model (POM). "Convention over configuration" prevede configurazione minima, obbligando a configurare solo aspetti che si differenziano dalle implementazioni standard.

5.4 Quali sono le fasi del Default Build Lifecycle di Maven?

Risposta:

1. **Validate**: convalidare correttezza progetto
2. **Compile**: compilare codice sorgente
3. **Test**: testare con framework unit testing
4. **Package**: confezionare in formato distribuibile (JAR)
5. **Verify**: controlli sui test di integrazione
6. **Install**: installare nel repository locale
7. **Deploy**: copiare nel repository remoto

5.5 Cos'è il POM (Project Object Model)?

Risposta: File XML che contiene informazioni sul progetto e dettagli di configurazione usati da Maven per build. Specifica: dipendenze, plugin/goal, profili di build, altre info progetto.

6. CONTINUOUS INTEGRATION (CI)

Domande Teoriche

6.1 Definire Continuous Integration.

Risposta: Pratica di sviluppo software dove i membri del team integrano almeno quotidianamente, portando a multiple integrazioni per giorno. Permette di intensificare attività di sviluppo e test integrando gli sviluppi nel VCS il più spesso possibile.

6.2 Quali sono i prerequisiti per implementare CI?

Risposta:

1. Codice del progetto gestito in VCS
2. Processo di build automatico
3. Processo di build esegue verifiche automatiche (unit test, integration test, analisi statica)
4. Team adotta correttamente questa pratica
5. Sistema automatico per eseguire build ad ogni integrazione [Opzionale]

6.3 Descrivi il processo CI dettagliato in 6 passi.

Risposta:

1. Controllo se build è in esecuzione nel sistema CI
2. Se build ha successo, aggiornare workspace con codice VCS e integrare localmente
3. Eseguire build locale per verificare funzionamento
4. Aspettare risultati del processo di build
5. Inviare modifiche al VCS se build locale ha successo
6. Attendere sistema CI e fermarsi se build fallisce per sistemare problemi

Domande su Strumenti

6.4 Confronta Jenkins vs GitHub Actions.

Risposta:

Jenkins:

- Più complesso da configurare ma più flessibile
- Scalabile con nodi distribuiti
- Supporta pipeline scriptate e dichiarative
- Adatto a progetti enterprise

GitHub Actions:

- Integrazione nativa con GitHub
 - Più semplice per progetti GitHub
 - Utilizza runner GitHub o self-hosted
 - Limitazioni infrastruttura GitHub
-

7. CONTINUOUS DELIVERY (CD)

Domande Teoriche

7.1 Fornire 3 definizioni di Continuous Delivery.

Risposta:

1. **Definizione 1:** Approccio ingegneristico dove team producono software in cicli brevi, assicurando rilascio affidabile in qualsiasi momento con rilascio manuale
2. **Definizione 2:** Disciplina dove si costruisce software rilasciabile in produzione in qualsiasi momento, prioritizzando mantenimento deployabilità
3. **Definizione 3:** Capacità di ottenere cambiamenti di tutti i tipi in produzione in modo sicuro e veloce

7.2 Quali sono i requisiti per realizzare Continuous Delivery?

Risposta:

- **Continuous Integration:** VCS, Build automation, Unit Testing
- **Artifact Repository:** gestione dipendenze e artefatti binari
- **Configuration Management:** gestione configurazione ambienti tramite codice
- **Continuous Testing:** test automatici di sistema funzionali e non funzionali
- **Orchestratore:** sistema per modellare esecuzioni pipeline (es. Jenkins)

Deployment Pipeline Practices

7.3 Spiega la pratica "Only Build Your Binaries Once".

Risposta: Eseguire build solo una volta garantisce stesso artefatto per tutte le verifiche in ogni ambiente. Vantaggi:

- Artefatto rilasciato in produzione è lo stesso verificato nelle stages
- Ottimizzazione (lavoro fatto una volta sola)
- Richiede artifact repository e artefatti indipendenti dall'ambiente

7.4 Cosa significa "Deploy the Same Way to Every Environment"?

Risposta: Utilizzare stesso script per rilascio in ambienti differenti. Lo script sarà più solido perché verificato maggiormente.

7.5 Cos'è uno Smoke Test nel contesto CD?

Risposta: Test per verificare se rilascio è andato bene. Devono verificare anche corretto funzionamento dei sub-system esterni (DB). Obiettivo è fail fast.

8. CONFIGURATION MANAGEMENT

Domande Teoriche

8.1 Definire Configuration Management secondo ITIL.

Risposta: Processo per fornire modello logico dell'infrastruttura attraverso identificazione, controllo, gestione e verifica di tutte le versioni di Configuration Items esistenti.

8.2 Cos'è un Configuration Item (CI)?

Risposta: Unità di configurazione che può essere gestita individualmente (computer, router, server, software, etc.).

8.3 Cos'è il CMDB e quali sono i benefici del Configuration Management?

Risposta: **CMDB:** Configuration Management Database per tracciare tutti i CI e relazioni tra loro.

Benefici:

- Informazioni accurate sull'infrastruttura IT
- Maggiore controllo sulle CI
- Maggiore aderenza alle leggi (licenze software)

- Miglior supporto a Incident e Problem Management
-

9. CONTAINER PLATFORM (DOCKER)

Domande Teoriche

9.1 Definire Docker e i suoi vantaggi principali.

Risposta: Docker è una piattaforma aperta per sviluppo, spedizione ed esecuzione di applicazioni. Permette di separare applicazioni dall'infrastruttura per distribuire rapidamente software. Gestisce infrastruttura come le applicazioni.

9.2 Differenza tra Container e Virtualizzazione.

Risposta:

- **Container:** condividono kernel OS, più leggeri, avvio più veloce
- **VM:** ogni VM ha proprio OS completo, più pesanti, maggior isolamento

9.3 Spiegare l'architettura Build, Ship, Run di Docker.

Risposta:

- **BUILD:** Creare Docker image con Dockerfile
- **SHIP:** Distribuire image tramite Registry
- **RUN:** Eseguire container da image

Domande sui Concetti Docker

9.4 Definire i seguenti concetti Docker:

Risposta:

- **Docker Engine:** runtime per gestire container
- **Docker Image:** template immutabile per container
- **Dockerfile:** file di istruzioni per creare image
- **Docker Container:** istanza running di una image
- **Docker Registry:** repository per image (es. Docker Hub)

9.5 Cos'è un Docker Network?

Risposta: Sistema per permettere comunicazione tra container, fornendo isolamento e connettività di rete.

10. ARTIFACT REPOSITORY

Domande Teoriche

10.1 Cos'è un Artifact Repository e le sue caratteristiche principali.

Risposta: Sistema per depositare e pubblicare prodotti software. Caratteristiche:

- Ambiente per depositare/pubblicare prodotti
- Intermediario per scaricare da depositi esterni
- Ricerca e reperimento informazioni prodotti
- Gestione permessi d'accesso
- Segnalazione vulnerabilità
- Verifica problemi licenze
- Documentazione artefatti con metadati

10.2 Cosa significa G.A.V negli artefatti?

Risposta: Group, Artifact, Version - sistema per distinguere univocamente un artefatto.

10.3 Quali sono i tipi di artefatti gestiti?

Risposta: Binari (JAR, WAR), immagini Docker, pacchetti NPM, gem Ruby, etc.

Domande Pratiche

10.4 Vantaggi di Maven Repository Management.

Risposta:

- **Proxy Remote Repositories:** recuperare artefatti da repository esterni
- Distribuire versioni ufficiali certificate
- Analisi compatibilità licenze
- Facilitare collaborazione evitando creazione da VCS

11. FEEDBACK LOOP

Domande Teoriche

11.1 Cos'è un Feedback Loop nel contesto CD?

Risposta: Insieme di strumenti che:

- Forniscono stato di ogni passo nella pipeline

- Inviano messaggio giusto alla persona giusta
- Permettono di misurare il processo

11.2 Come implementare un feedback loop efficace?

Risposta: Valutare:

- **Quali attori notificare:** Developers, Operations, Project Manager, Business
 - **Che media utilizzare:** Direct Visualization, E-Mail, IRC, Chat, Video messaging, Extreme Feedback Devices
 - **Che evento/priorità/frequenza:** modifica stato Stage, gray-listing, feedback Production vs Testing
-

12. DOMANDE TRASVERSALI E INTEGRATIVE

Architettura e Integrazione

12.1 Spiega la relazione tra CI e CD Pipeline.

Risposta: CI si concentra su integrazione frequente del codice con build e test automatici. CD estende CI permettendo rilascio continuo attraverso pipeline automatizzate che portano il software dalla build alla produzione.

12.2 Come si integrano ITS, VCS e CI/CD?

Risposta: ITS traccia work item, VCS gestisce versioni codice con riferimenti a issue ITS, CI/CD automatizza build/test/deploy triggerate da commit VCS, con feedback loop che aggiorna ITS.

Best Practices

12.3 Principi dell'Agile Manifesto.

Risposta:

- Persone e interazioni più che processi e strumenti
- Software funzionante più che documentazione ampia
- Collaborazione con cliente più che negoziazione contratto
- Risposta al cambiamento più che seguire un piano

12.4 Cosa significa "Fail Fast" e perché è importante?

Risposta: Individuare errori il prima possibile per ridurre costi di correzione e tempo di feedback. Importante per mantenere qualità e velocità di sviluppo.

13. QUIZ A SCELTA MULTIPLA

13.1 Quale NON è una caratteristica CRISP della Build Automation? a) Completo b) Ripetibile
c) Configurabile d) Portabile

Risposta: c) Configurabile

13.2 In Scrum, la durata tipica di uno Sprint è: a) 1 settimana b) 2-4 settimane c) 1-2 mesi d) Variabile

Risposta: b) 2-4 settimane

13.3 Nel processo CI, se la build fallisce: a) Si continua con lo sviluppo b) Si ritorna all'ultima versione funzionante c) Si aspetta il giorno successivo d) Si notifica solo il project manager

Risposta: b) Si ritorna all'ultima versione funzionante

13.4 Docker utilizza quale tecnologia per l'isolamento? a) Virtualizzazione completa b) Kernel namespaces e cgroups c) Macchine virtuali d) Emulazione hardware

Risposta: b) Kernel namespaces e cgroups

13.5 Il comando Maven per eseguire il default lifecycle completo è: a) mvn compile b) mvn test c) mvn install d) mvn deploy

Risposta: c) mvn install

14. DOMANDE VERO/FALSO

14.1 DVCS ha sempre lock file. **FALSO**

14.2 Il Product Owner in Scrum può cambiare lo Sprint Goal durante lo Sprint. **FALSO**

14.3 Continuous Deployment e Continuous Delivery sono la stessa cosa. **FALSO**

14.4 Un container Docker condivide il kernel con l'host. **VERO**

14.5 Maven può gestire solo progetti Java. **FALSO**

14.6 L'artifact repository deve contenere solo binari. **FALSO**

14.7 Il feedback loop deve notificare sempre tutti gli stakeholder. **FALSO**

14.8 TDD significa scrivere prima i test del codice. **VERO**

14.9 In Git, un branch è un pointer a un commit. **VERO**

14.10 CI richiede necessariamente un sistema di build automation. **VERO**

15. DOMANDE PRATICHE E SCENARI

15.1 Scenario: Sei project manager di un team che deve implementare CI/CD. Descrivi i passi e gli strumenti necessari.

Risposta:

1. Implementare VCS (Git)
2. Configurare ITS per tracking
3. Implementare build automation (Maven)
4. Configurare artifact repository
5. Implementare CI server (Jenkins)
6. Creare pipeline CD con stages
7. Implementare configuration management
8. Configurare feedback loop
9. Training del team su processo

15.2 Scenario: La build CI fallisce frequentemente. Analizza possibili cause e soluzioni.

Risposta: Cause:

- Test instabili
- Dipendenze non gestite correttamente
- Configurazione ambiente inconsistente
- Conflitti merge non risolti

Soluzioni:

- Fix broken builds immediately
- Migliorare test automation
- Standardizzare environment configuration
- Implementare code review process

15.3 Scenario: Come implementeresti containerizzazione in pipeline esistente?

Risposta:

1. Creare Dockerfile per applicazione
2. Integrare build Docker image in pipeline

3. Configurare Docker registry
 4. Aggiornare deployment scripts per container
 5. Implementare orchestrazione (se necessario)
 6. Aggiornare monitoring e logging
-

16. DEFINIZIONI RAPIDE (Per ripasso veloce)

- **Artifact:** Prodotto software distribuibile
 - **Branch:** Pointer a un commit
 - **Build:** Processo di trasformazione codice in prodotto
 - **CI:** Integrazione frequente del codice
 - **CD:** Capacità di rilascio continuo
 - **Container:** Ambiente di esecuzione isolato e leggero
 - **DoD:** Definition of Done
 - **DVCS:** Distributed Version Control System
 - **ITS:** Issue Tracking System
 - **POM:** Project Object Model (Maven)
 - **Sprint:** Periodo di sviluppo timeboxed in Scrum
 - **TDD:** Test Driven Development
 - **VCS:** Version Control System
 - **Workflow:** Insieme di stati e transizioni
-

Questa raccolta copre tutti gli argomenti del corso MTSS. Studia le definizioni, comprendi i concetti, pratica con gli scenari e verifica la tua preparazione con i quiz. Buona fortuna per l'esame!