Esercizi di teoria

Domande esami 2018:

- **Cos'è e a cosa serve uno standard?**

Uno standard è una regola o un riferimento di base per il confronto che viene utilizzata per valutare le dimensioni, il contenuto o la qualità di un oggetto o un'attività. Nell'ambito dell'ingegneria del software, gli standard del software racchiudono la conoscenza e le best-practice da attuare durante tutta la fase di sviluppo del software al fine di evitare errori commessi in passato. Gli standard forniscono anche uno strumento per il controllo e la definizione di qualità del prodotto software, in quanto stabiliscono una linea comune di metriche e fattori da tenere in considerazione.

- **Quali aspetti legati al software sono oggetto degli standard illustrati nel corso?**

L'aspetto principale di un software che viene influenzato dagli standard è la qualità. Il processo di gestione della qualità spesso si affida all'utilizzo di standard per verificare appunto, il livello di qualità del prodotto software e dei processi di sviluppo. Avendo gli standard un ruolo importante nella gestione della qualità, di conseguenza hanno anche un ruolo importante in tutti quegli aspetti ad essa legati, come: sicurezza, affidabilità, usabilità, protezione, robustezza, efficienza etc.

- **I requisiti di un sistema software sono spesso incompleti, mal capiti e/o mutevoli; e questa situazione si ripercuote ovviamente sulle specifiche del sistema. Si commenti questa affermazione.**

A livello teorico, in un processo software esiste sempre una fase ben definita di ingegneria dei requisiti (ricerca, analisi, documentazione e verifica dei requisiti) che porta alla creazione di un documento formale che include requisiti dell'utente e del sistema che rappresenta ciò che il team di sviluppo dovrà produrre. In realtà però, il processo di ingegneria dei requisiti è complesso e pieno di difficoltà, a livello pratico quindi, più che essere un processo ben definito in una sola fase è un processo iterativo (nei processi plan-driven è comunque distinguibile una macro-fase di ingegneria dei requisiti, ma questo processo continua comunque durante lo sviluppo del software; nei processi agili invece questa fase è per principio di natura iterativa). Le principali difficoltà nascono dal fatto che i requisiti non sono sempre chiari sin dalla prima fase di ricerca e analisi (i vari stakeholder possono avere idee poco chiare; non sanno cosa vogliono e cosa può offrire loro un prodotto software; vari stakeholder possono avere idee contrastanti; a volte le idee sono chiare ma ci sono dei fraintendimenti tra stakeholder e ingegneri dei requisiti, che portano alla errata comprensione dei requisiti etc.). Tutte queste difficoltà hanno come conseguenza che i requisiti di un sistema variano nel tempo, sia perché durante le prime fasi di progettazione si acquisisce maggiore conoscenza del prodotto da creare, sia per le difficoltà elencate precedentemente; questa continua possibilità di mutazione e variazione dei requisiti fa sì che l'ingegneria dei requisiti sia un processo iterativo, in cui le fasi di ricerca, analisi, documentazione e verifica si ripetano e si intreccino tra loro, con un continuo feedback reciproco. Spesso capita che le operazioni di ing. dei requisiti vengano svolte anche durante la fase di progettazione e di sviluppo (ad es. negli approcci agili allo sviluppo del software). La fase che gestisce le modifiche dei requisiti è detta gestione dei requisiti.

- **Una conseguenza di quanto riportato nella domanda precedente è che sono stati proposti diversi modelli dei processi software che cercano di ovviare a tali criticità. Si illustrino caratteristiche e motivazioni dei principali modelli.**

I principali modelli dei processi software che cercano di ovviare a queste criticità sono quelli basati su un approccio agile (sviluppo rapido del software; ad esempio l'extreme programming). La principale caratteristica legata ai requisiti del sistema che caratterizza i modelli agili è che i processi di ingegneria del software, progettazione e implementazione sono intrecciati; di conseguenza NON c'è una macro-fase iniziale di ingegneria dei requisiti; è assente una specifica di sistema dettagliata e i documenti dei progetti sono minimali. Il software viene sviluppato con un approccio incrementale e

gli utenti (stakeholder finali) sono spesso resi partecipi nella specifica e valutazione di ogni incremento. Questa forte collaborazione fa sì che si crei un rapporto informale che però permette di controllare meglio la natura mutevole dei requisiti. I metodi agili ritengono che "la risposta al cambiamento" sia più importante di seguire un piano ben definito. Per gestire la mutazione dei requisiti, i metodi agili hanno sviluppato la tecnica delle storie utente, che permette di integrare la deduzione dei requisiti con lo sviluppo. Una storia utente è uno scenario d'uso in cui potrebbe trovarsi un utente finale; queste storie utente sono la rappresentazione di ciò che il sistema deve fare e di conseguenza sono analoghi ai requisiti.

- **Si illustri la strutturazione e gli scopi del processo di progettazione architetturale, collocandolo nell'ambito del ciclo di vita del software.**

La progettazione architetturale si occupa dell'organizzazione di un sistema software e della progettazione della sua struttura complessiva. Nell'ambito di ciclo di vita del software, la progettazione architetturale si colloca tra ingegneria dei requisiti e la fase di progettazione del software, in quanto identifica i principali componenti di un sistema e le loro relazioni, analizzando i requisiti ottenuti. L'output del processo di progettazione architetturale è un modello architetturale che descrive come il sistema sarà organizzato in funzione dei componenti principali. La scelta di come strutturare il software è importante perché influenzerà le prestazioni, la robustezza, la manutenibilità etc. del sistema; infatti l'architettura ha un influenza predominante sulle caratteristiche non funzionali del sistema; mentre i singoli componenti su quelle funzionali (in quanto implementano i singoli servizi; quindi i singoli requisiti funzionali). Il processo di progettazione architetturale dipende fortemente dal sistema che si sta sviluppando e dall'esperienza del team; esso è più una serie di decisioni da prendere che una serie di attività ben definite da seguire. L'architettura di un sistema software può basarsi su un particolare schema o stile architetturale. Uno schema architetturale è una descrizione dell'organizzazione del sistema, per esempio un'organizzazione client-server o un'architettura a strati (esempi di schemi architetturali: a strati, client-server, MVC, repository, orientato ai servizi, per sistemi distribuiti). Per scomporre le unità strutturali del sistema, bisogna scegliere la strategia di scomposizione dei componenti in sottocomponenti; infine, nel processo di modellazione del controllo, occorre sviluppare un modello generale delle relazioni di controllo tra le varie parti del sistema e decidere come controllare l'esecuzione dei componenti. A causa della stretta relazione tra le caratteristiche non funzionali del sistema e l'architettura hardware sottostante, la scelta dello schema architetturale e della struttura dipende pesantemente dai requisiti non funzionali del sistema.

- **Si illustrino caratteristiche, differenze, ambiti applicativi, pro e contro dei due modelli repository e client server.**

L'architettura client-server è organizzata come un insieme di servizi e server associati e di client che accedono e utilizzano tali servizi. Le architetture client-server sono tipicamente utili negli ambiti di sistemi distribuiti (anche se possono essere implementate architetture client-server su una singola macchina). Questa architettura permettere di separare e rendere indipendenti i vari componenti del sistema, di conseguenza questa architettura ha un'elevata manutenibilità.

L'architettura a repository concettualizza una struttura in cui più componenti interattivi condividano i dati, salvati tipicamente in un database centrale (repository). Questo fa sì che i componenti non interagiscano tra loro ma solo mediante la repository. I componenti sono quindi indipendenti, e questo risulta essere un vantaggio. Il punto debole di un'architettura a repository è la repository stessa, un guasto della repository renderebbe inutilizzabile l'intero sistema.

❖ **Si inquadri e si illustri l'utilizzo della tecnica equivalence partitioning nell'ambito del black-box testing. Quali le motivazioni che giustificano il suo utilizzo e quali i vantaggi?**

Il test black-box è un metodo di test del software che esamina la funzionalità di un'applicazione senza scrutare nelle sue strutture o funzionamenti interni (tipico dei sistemi legacy).

Il partizionamento di equivalenza o il partizionamento di classe di equivalenza (ECP) è una tecnica di test del software che divide i dati di input di un'unità software in partizioni di dati equivalenti da cui è possibile derivare casi di test. In linea di principio, i casi di test sono progettati per coprire ogni partizione almeno una volta.

❖ **Si illustri il concetto di dependability del software e le sue quattro principali dimensioni.**

La fidatezza di un sistema software è una proprietà che rispecchia il livello di fiducia che l'utente può riporre in esso. La fidatezza (dependability) è un termine che fa riferimento e rappresenta l'unione di quattro principali caratteristiche di un software, ovvero:

- Disponibilità: indica la probabilità che un sistema sia attivo e in grado di fornire i servizi utili per il quale è stato sviluppato.
- Affidabilità: indica la probabilità che il sistema in un determinato periodo di tempo fornisca correttamente i servizi secondo le aspettative dell'utente.
- Sicurezza: indica la stima delle probabilità che il sistema possa causare danni a persone o all'ambiente.
- Protezione: indica la stima delle probabilità che il sistema possa resistere a intrusioni accidentali o deliberate.

Un sistema si dice quindi fidato (elevata fidatezza) se ha buone qualità nelle caratteristiche appena elencate. La fidatezza è una caratteristica fondamentale di tutti i sistemi, ma soprattutto di quelli critici: sistemi dove i guasti hanno un elevato rischio di causare danni a persone, danni economici e ambientali etc. (vedi software di pilotaggio).

❖ **Si definisca e si illustri una metrica per misurare l'availability (disponibilità).**

Per misurare le caratteristiche della fidatezza sopra elencate, tipicamente vengono utilizzate delle metriche: una metrica del software è una caratteristica del software può essere effettivamente misurata. Le metriche che spesso vengono usate per misurare la disponibilità del software sono:

- Probabilità di fallimento su richiesta (POFOD): probabilità che la richiesta di un servizio provochi il suo fallimento.
- Tasso di occorrenza dei fallimenti (ROCOF): numero di fallimenti del sistema che statisticamente si verificano in un certo lasso di tempo.
- Tempo medio al fallimento (MTTF): tempo medio tra due fallimenti del sistema consecutivi.
- Disponibilità (AVAIL): probabilità che un sistema sarà operativo quando viene richiesto un servizio.

❖ **Cosa significa validazione del software? Come si fa?**

La validazione (o convalida) fa parte di un processo più generale di verifica e convalida del software (V&V, Verification and Validation). Lo scopo della validazione del software è garantire che esso rispetti le attese del cliente. Il suo scopo va ben oltre il semplice controllo di conformità alla specifica per dimostrare che il software fa ciò che il cliente ha richiesto. La convalida è essenziale perché, le definizioni dei requisiti non sempre riflettono i reali desideri o necessità dei clienti. L'obiettivo ultimo dei processi di verifica e convalida è stabilire, con un buon grado di confidenza, che il software è adatto al suo scopo. Ciò significa che il sistema deve essere adatto all'uso cui è destinato. Il principale approccio per eseguire la validazione del software è il testing: l'obiettivo dei test del software è dimostrare che un programma svolge i compiti per i quali è stato sviluppato e identificare eventuali

errori. I test hanno l'obiettivo di dimostrare che i requisiti funzionali e non funzionali siano rispettati. I test di convalida possono anche richiedere ispezioni e revisioni statiche del software. Anche il controllo della qualità fa parte della convalida del software.

❖ **Cosa la differenzia dalla verifica del software e cosa la accomuna?**

Come accennato nella risposta precedente, la verifica e la validazione (convalida) fanno parte di un processo più generale di verifica e convalida (V&V) del software. Il ruolo dei processi di verifica e convalida è essenzialmente controllare che il software che si sta sviluppando soddisfi le sue specifiche e offra le funzionalità richieste da chi ha acquistato il software. La differenza sostanziale tra verifica e convalida è che:

- la verifica è il processo che controlla se il software soddisfa i requisiti funzionali e non funzionali. Fornisce l'obiettiva conferma che il software soddisfa le specifiche.
- la validazione è un processo più generale che mira a garantire che il software rispetti le attese del cliente. Fornisce la conferma che il software rispetta i requisiti e aspettative del cliente.

Sono due fasi che in comune hanno l'obiettivo ultimo, ovvero: stabilire con un buon grado di confidenza, che il software è adatto al suo scopo. Ciò significa che il sistema deve essere adatto all'uso cui è destinato.

❖ **Si illustri lo scopo della scomposizione modulare.**

Dopo aver scelto l'organizzazione generale del sistema, si deve decidere l'approccio da usare per la scomposizione dei sottosistemi in moduli. Un modulo è solitamente un componente di un sistema che fornisce uno o più servizi ad altri moduli, fa uso dei servizi forniti da altri moduli, e non è normalmente considerato un sistema indipendente. Tipicamente i moduli sono formati da una serie di componenti più piccoli (che sono indipendenti). Il principale scopo della scomposizione modular è l'aumento della manutenibilità del sistema e della riusabilità

❖ **Si illustrino i due concetti di coesione e di accoppiamento. Si evidenzi anche perché è importante progettare moduli con forte coesione e basso accoppiamento.**

La coesione è una misura del livello di correlazione tra diverse funzionalità presenti all'interno di un modulo, ossia del suo livello di omogeneità funzionale. L'accoppiamento è una misura delle connessioni (dipendenze) tra due componenti del sistema. È importante progettare moduli con forte coesione perché questo indica un elevato livello di manutenibilità del sistema, ogni modifica è più rapida ed economica se svolta in un componente unico, invece che andare a modificare più componenti del sistema con il rischio di dover eseguire molte più modifiche. Un buon metodo per avere alta coesione nei moduli è applicare i principi solidi (principalmente il SRP). Avere basso accoppiamento indica che i moduli del sistema sono prevalentemente indipendenti e hanno scarse relazioni con altri moduli, questo basso livello di accoppiamento fa si che una modifica non vada a impattare su altri moduli. Decentralizzando lo stato e facendo comunicare i moduli tramite messaggi (e non condividendo variabili) si raggiunge un basso livello di accoppiamento (loose coupling).

❖ **Si illustri il concetto di requisiti software.**

I requisiti di un sistema sono la descrizione dei servizi che il sistema deve fornire e dei suoi vincoli operativi. Il processo di riceva, analisi, documentazione e verifica di questi servizi e vincoli è chiamato ingegneria dei requisiti. I requisiti possono essere suddivisi in requisiti utente (requisiti di alto livello che dichiarano quali servizi il sistema deve fornire) e requisiti di sistema (forniscono la descrizione dettagliata delle funzioni, dei servizi e dei vincoli operativi, cosa effettivamente deve essere implementato).

❖ **Cosa sono rispettivamente la definizione dei requisiti e la specificazione dei requisiti? Quali le differenze? Come se ne rappresentano rispettivamente i risultati?**

La definizione dei requisiti e la specificazione dei requisiti sono differenti fasi del processo di ingegneria dei requisiti, inoltre differiscono nel livello di dettaglio dei "requisiti" raccolti. La definizione dei requisiti consiste nella scoperta e analisi dei requisiti utente e requisiti di sistema (descritti nella risposta precedente) tramite l'interazione con gli stakeholder, mentre la specificazione dei requisiti consiste nella produzione (conversione dei requisiti in un formato standard – le specifiche del sistema) di un documento dei requisiti, con descrizioni tecniche/dettagliate del software e di cosa deve fare, che verrà poi utilizzato dagli ingegneri come punto di partenza per la fase di progettazione e implementazione e come documento formale rappresentante i requisiti utenti e del sistema.

❖ **Si illustrino gli scopi dei due processi fondamentali della gestione della qualità del software, ossia**
(1) l'assicurazione della qualità (software quality assurance)
(2) il controllo di qualità (software quality control).

La gestione della qualità del software (QM – Quality Management) si occupa di garantire che i sistemi software sviluppati siano conformi agli scopi prestabiliti, ovvero che i sistemi soddisfino le esigenze dei loro utenti, che i processi siano eseguiti in modo efficiente e affidabile e che il software sia consegnato entro i limiti di tempo e budget previsti. La garanzia (assicurazione / assurance) della qualità è la definizione di processi e standard che dovrebbero essere poi seguiti al fine di produrre prodotti di alta qualità. Il controllo della qualità è l'applicazione di questi processi e standard per eliminare quei prodotti che non hanno raggiunti i livelli di qualità voluti. Il processo di assicurazione della qualità è più generale ed è relativo a come ci si organizza per raggiungere i requisiti di qualità, mentre il controllo è più specifico ed è relativo a come si procede nelle singole valutazioni necessarie a misurare la qualità e nelle esecuzioni di processi di controllo.

❖ **Cos'è un sistema di qualità (quality system))?**

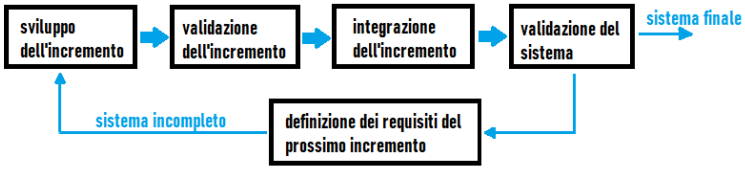
Un sistema di qualità è un sistema dove la qualità è un concetto critico. Un quality system è un sistema che segue una serie di standard e normative al fine di ottenere elevati livelli di qualità.

❖ **Cosa è il manuale della qualità (quality manual)?**

❖ **Si fornisca un modello dei processi relativamente allo sviluppo incrementale (consegna incrementale o incremental software development): si utilizzi un grafico per illustrare struttura e legami dei vari processi e si descrivano gli scopi di ciascun processo.**

Lo sviluppo incrementale si basa sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni, finché non si ottiene il sistema richiesto. Le attività di specifica, sviluppo e convalida sono intrecciate anziché separate, con molti feedback veloci tra le varie attività. Questo approccio può essere plan-driven, agile, o una combinazione dei due. In un approccio plan-driven gli incrementi sono pianificati precedentemente, nell'approccio agile invece dipendono dalla situazione generale di sviluppo. Questo approccio è pensato per quei software in cui i requisiti e le specifiche di sistema variano costantemente. Ciascun incremento (versione) del sistema apporta qualche funzionalità aggiuntiva richiesta dall'utente.

Un modello dei processi che utilizza un approccio incrementale è di natura ciclica (ad esempio l'XP):



❖ **Cos'è un incremento e come ad esso vengono assegnati i requisiti?**

Quando si parla di incrementi, stiamo parlando dei modelli di sviluppo software basati sullo sviluppo incrementale. Lo sviluppo incrementale si basa sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni, attuando un processo iterativo fino a quando non si ottiene il sistema richiesto. Un incremento è quindi l'aggiunta di una nuova funzionalità alla versione precedente, che viene poi testata e convalidata. I requisiti, negli approcci incrementali vengono scelti e discussi con il cliente (negli approcci agili) o pianificati in precedenza (negli approcci plan-driven) prima della progettazione e sviluppo dell'incremento stesso.

❖ **Che dimensioni può avere un incremento in termini di tempo necessario per il suo sviluppo?**

Tipicamente lo sviluppo incrementale punta a sviluppare il software in piccoli incrementi, implementabili in 1-2 settimane. Questo perché punta a uno sviluppo rapido del software (metodi agili).

<u>Domande esami 2017:</u>

❖ **Quali sono i principali elementi che caratterizzano i metodi agili?**

I metodi agili (o metodi di sviluppo agile del software) nascono dall'esigenza di produrre software utile e di qualità velocemente, in una realtà commerciale competitiva dove i tempi di consegna sono un fattore fondamentale. I processi di sviluppo sono caratterizzati da fasi di specifica, progettazione e implementazione intrecciate (a differenza dei processi plan-driven dove sono fortemente sequenziali). Con un approccio agile, un sistema viene sviluppato tipicamente seguendo un approccio di sviluppo incrementale, in cui gli utenti finali (in generale gli stakeholder) sono coinvolti nella specifica e valutazione di ciascun incremento. I principi dei metodi agili possono essere riassunti dai punti elencati dal manifesto dello sviluppo agile del software:

- individui e interazioni sono più importanti dei processi e strumenti;
- il software funzionante è più importante della documentazione dettagliata;
- la collaborazione con il cliente è più importante della negoziazione dei contratti;
- rispondere al cambiamento (in riferimento ai requisiti che mutuano velocemente) è più importante che seguire un piano.

I metodi agili sono molto efficaci nello sviluppo di software di piccole/medie dimensioni e dove c'è chiara necessità di gestire numerose mutuazioni dei requisiti del sistema. I metodi agili sono supportati da una serie di tecniche e metodologie, tra queste, le più importanti sono:

- proprietà collettiva: il software è sotto la responsabilità di ogni sviluppatore della software house.
- integrazione continua: appena un task è stato implementato, va integrato e validato nel sistema.
- pianificazione incrementale: invece che seguire un approccio guidato da piani, viene eseguita una pianificazione dei task da implementare per ciascuna release del sistema.
- cliente on-site: il cliente deve essere sempre disponibile per mediare e aiutare nello sviluppo del sistema.
- utilizzo della programmazione a coppie
- refactoring
- progettazione semplice, volta a soddisfare i requisiti correnti, niente di più
- piccole release
- sviluppo con test iniziali
- user stories: i requisiti del software cambiano sempre. Per gestire questi cambiamenti, i metodi agili non hanno un'apposita attività di ingegneria dei requisiti, ma integrano la deduzione dei requisiti con lo sviluppo. Per semplificare questo approccio, fu sviluppata l'idea delle storie utente, dove una storia è uno scenario d'uso in cui potrebbe trovarsi un utente finale del sistema.

❖ **Come si può descrivere l'Extreme Programming in termini di Incremental development?**

L'extreme programming incarna totalmente l'essenza dello sviluppo rapido del software, attuando praticamente in totalità le metodologie e tecniche proposte dallo sviluppo agile. L'XP segue quindi un approccio incrementale: nell'XP i requisiti del sistema sono rappresentati su story card (user stories) e per ciascun incremento vengono determinate quali task (derivati dalle user stories) implementare per ciascun incremento. Appena un task è implementato esso viene integrato nel sistema (tecnica dell'integrazione continua). Dopo ogni integrazione di codice nel sistema, esso deve superare tutti i test.

❖ **Si illustrino le due principali tecniche di verifica del software.**

La verifica fa parte di un processo più generale di "verifica e convalida" del software (V&V, Verification and Validation). Il processo di V&V ha lo scopo di controllare che il software sviluppato offra le funzionalità richieste e che soddisfi le aspettative del cliente. La fase di verifica è il processo che controlla che il software soddisfi tutti i requisiti funzionali e non funzionali; mira a fornire quindi l'obiettiva conferma che il software è stato realizzato correttamente e che rispetta tutte le specifiche. Le principali tecniche di verifica del software sono il testing e le ispezioni (e le revisioni).

Il testing mira a dimostrare che il sistema funzioni correttamente utilizzando una serie di test case che riflettono l'uso previsto del sistema (test di convalida) e scovare difetti (bug) nel software (test dei difetti). Esistono tre stadi di test:

- Test dello sviluppo: include tutte le attività di test che sono svolte dal team di sviluppo di un sistema software. Si suddivide in: test delle unità, dei componenti e del sistema.
- Test della release: è il processo che testa una particolare release di un sistema che dovrà essere utilizzata all'esterno del team di sviluppo (release per il cliente o per un team che sviluppa sistemi correlati).
- Test degli utenti: gli utenti testano il sistema, dando i propri input e suggerimenti (alpha/beta test, test di accettazione in cui il cliente accetta e completa l'acquisto del software).

Il testing è una tecnica di verifica dinamica in quanto richiede che il software venga eseguito.

Le ispezioni e le revisioni sono tecniche di verifica statica in quanto non è richiesto che il software venga eseguito. Consistono nell'analisi e controllo i requisiti del sistema, gli schemi di progettazione, il codice sorgente e i test proposti per il sistema. Hanno dei vantaggi in quanto possono scovare errori non facilmente scovabili via testing: errori di dipendenza tra classi/componenti ad esempio. Le ispezioni e revisioni inoltre possono controllare aspetti non "testabili" come: qualità del codice, facilità di manutenzione, errate scelte di design etc.

❖ **Cos'è e a cosa serve il path testing?**

In ingegneria del software, il path tesing (test del percorso) è un metodo "white box" per la progettazione di test case. Il metodo analizza il diagramma di flusso di controllo di un programma per trovare un insieme di percorsi di esecuzione linearmente indipendenti.

❖ **Cos'è la complessità ciclotomica e come può essere utile per il path testing?**

La complessità ciclomatica è una metrica software utilizzata per misurare la complessità strutturale di un programma. Tale metrica misura direttamente il numero di cammini linearmente indipendenti che caratterizzano il grafo del flusso di controllo di un programma di conseguenza è utile per dare un valore concreto da utilizzare nel metodo del path testing.

❖ **Cos'è e perché è importante il processo di Software Project Management?**

La gestione dei progetti (project management) è una parte essenziale dell'ingegneria del software. I progetti devono essere gestiti perché l'ingegneria del software professionale è sempre soggetta a

vincoli aziendali di budget e di tempi. La buona gestione non può garantire il successo di un progetto, ma la cattiva gestione di solito ne determina il fallimento.

❖ **In cosa consistono le specifiche attività eseguite da un SW Project Manager?**

È difficile fornire una descrizione dei compiti standard di un project manager. I compiti variano notevolmente in funzione dell'azienda e del tipo di software che si sta sviluppando. Tuttavia, alcune attività di gestione dei progetti sono comuni a tutte le aziende: pianificazione dei progetti, risk management, gestione del personale e reporting. I project manager sono responsabili della pianificazione della stima e della tempistica dello sviluppo dei progetti; inoltre devono assegnare i compiti al personale. Hanno la supervisione del lavoro per garantire che esso sia svolto secondo gli standard, e tengono sotto controllo l'avanzamento del lavoro per verificare che esso venga sviluppato in tempo ed entro il budget previsto. I project manager devono definire i rischi che potrebbero influenzare un progetto, monitorare questi rischi e svolgere le azioni appropriate quando si presenta qualche problema. I project manager sono responsabili della gestione di una squadra di persone. Devono selezionare le persone della loro squadra e definire le modalità operative che possono migliorare l'efficienza del lavoro di gruppo. I project manager hanno il compito di documentare il progresso di un progetto in appositi report da inviare ai clienti e ai manager della società che sviluppa il software.

❖ **Quali sono le differenze tra i concetti di Deliverable e di Milestone?**

Sono due termini usati nell'ambito della pianificazione della progettazione. Con deliverable si intende un risultato di progetto concerto, che si può fornire a un cliente: un report, un prototipo, il software finale etc. Con milestone (in italiano pietra miliare) si indica il punto finale di un'attività di processo (il raggiungimento di un obiettivo prefissato).

❖ **Nell'ambito del concetto di Dependability, si illustrino i seguenti tre concetti e le relative relazioni: errore umano (human error), bug (errore), failure (malfunzionamento), evidenziando anche in che specifico momento si possono verificare.**

Il modello difetto-errore-fallimento si basa sul concetto che gli errori umani causano difetti nel sistema, i difetti causano errori, gli errori causano fallimenti.

- Errore umano: comportamento umano che si traduce nell'introduzione di difetti nel sistema (durante la fase di progettazione, testing, implementazione, specifica etc.)
- Difetti del sistema (bug): caratteristica di un sistema software che può portare a un errore del sistema (a run-time)
- Errore del sistema: comportamento del sistema non atteso dagli utenti (a run-time)
- Fallimento del sistema: evento che si verifica nel momento in cui il sistema non fornisce il servizio che gli utenti si aspettano (a run-time).

❖ **Si definisca il concetto di affidabilità del software e si forniscano due metriche per la sua misura.**

L'affidabilità è uno delle quattro dimensioni fondamentali della fidatezza. Se pensiamo ai sistemi software come strumenti che forniscono qualche tipo di servizio, l'affidabilità può essere informalmente definita come la capacità del sistema di fornire i risultati corretti. Più precisamente, l'affidabilità è la probabilità di un'operazione di essere svolta senza che il sistema fallisca in un determinato periodo di tempo, in un dato ambiente, per uno scopo specifico. Delle metriche per misurare l'affidabilità sono:

- Probabilità di fallimento su richiesta (POFOD): definisce la probabilità che la richiesta di un servizio provochi il fallimento del sistema.

- Tasso di occorrenza di fallimenti (ROCOF)): definisce il numero di fallimenti del sistema che statisticamente si verificheranno in un certo lasso di tempo.
- Tempo medio al fallimento (MTTF): tempo medio tra l'occorrenza di due fallimenti.
- Disponiblità (AVAIL): probabilità che un sistema sarà operativo all'effettuazione di una richiesta di servizio.

❖ Si illustrino i passaggi (step/fasi) che caratterizzano la progettazione orientata alle funzioni (function-oriented design), evidenziando i ruoli del diagramma DFD (Data Flow Diagram) di partenza, del diagramma strutturale e delle minispec prodotte come risultato.

❖ In riferimento alla domanda precedente: in quale fase del processo generale di progettazione si inseriscono queste attività?

❖ In quali situazioni e per quali tipologie di sistemi di elaborazione è adatto l'approccio function-oriented design?

Domande esami 2015:

❖ Si fornisca una definizione dell'architettura client server, indicando anche pregi e difetti ed in quale parte della progettazione architetturale viene considerata.

❖ Si indichino le differenze tra le seguenti tipologie di architetture client server: thin-client, fat-client, three-tier e multi-tier.

❖ Si illustri il concetto di software testing, evidenziandone gli elementi fondamentali.

❖ Nel corso sono state esaminate numerose tecniche di testing, classificate in gruppi e differenziate secondo diversi criteri. Si spieghino i principali criteri utilizzati.

Domande esami 2014:

❖ Si illustri il concetto ed il ruolo degli stakeholder (portatori di interesse) nell'ambito del processo di sviluppo software.

❖ In riferimento alla domanda precedente: In quali fasi del ciclo di vita intervengono?

❖ In riferimento alla domanda precedente: Quali sono le criticità?

❖ Si illustri sinteticamente in cosa consistono le principali management activity (attività di gestione) nell'ambito di un progetto software?

- Cos'è e a cosa serve un activity diagram o network (rete di attività)?

- Cos'è il percorso critico (critical path)?

- Nell'ambito dell'approccio delineato dallo standard ISO 9000, si illustrino i due distinti processi di Quality Assurance (Assicurazione della Qualità) e di Quality Control (Controllo della Qualità).

- Nell'ambito del ciclo di vita del software, quando viene svolto il processo di Quality Control e come?

Domande esami 2013:

- Si illustri il processo di ingegnerizzazione dei requisiti software.

- Si illustrino le tecniche di elicitazione e di analisi dei requisiti.

- Si illustri l'idea delle Tecniche Agili per lo sviluppo del SW, fornendone la principale motivazione.

- Cos'è e quali sono gli elementi principali dell'eXtreme Programming.

Domande esami 2012:

- Come e quando i 2 distinti processi di SW Quality – SQA (Assicurazione della Qualità) e di Quality Control (Controllo delle Qualità) contribuiscono in una software house alla produzione di software di qualità?

- Cos'è un processo software (software process)? Si descriva il processo di sviluppo del software basato su incrementi successivi (incremental software development). Si illustri come tale approccio rappresenti una combinazione dei due approcci evolutivo e waterfall, evidenziandone poi pregi e criticità.

Domande esami 2011:

- Si illustri il concetto di software specification (specifica software)? In che relazione è con il processo di specificazione dei requisiti? Da cosa si parte per ottenere le specifiche software e come si procede?

- Cosa significa architettura distribuita? Quali sono i principali tipi di architetture a strati (tier-architecture) e come si differenziano? Cos'è il middleware?

- **Progettazione architetturale: si indichi quali fasi del ciclo di vita lo precedono e quali lo seguono? Si illustri lo scopo dei tre sottoprocessi che lo compongono.**

- **Si illustri il processo di statistical testing utilizzato per misurare l'affidabilità.**

- **A cosa serve il profilo operativo (operational profile)?**

Il profilo operativo del software riflette come questo verrà utilizzato nella pratica. Consiste in una specifica delle classi di input e nella probabilità che questi avvengano.