**Definisci Modello Waterfall e evidenzia differenze con metodo Agile** Il **modello Waterfall** ( o a cascata) è un approccio con **progressione lineare e unidirezionale** per passare da una fase a quella successiva ( senza possibilità di ritorno), standardizzando gli output di ogni fase. composto da fasi distinte: requisiti, progettazione, implementazione, verifica e manutenzione. Questo modello prevede fasi ben definite:**studio di fattibilità**, che prevede: Descrizione e analisi preliminare del problema, Possibili modi per risolvere il problema, Costi,  **analisi dei requisiti**, la quale produce il RASD (Requirements Analysis and Specification Document), **design**, dove viene prodotto il documento del progetto, l'architettura e lo scopo del progetto,   **implementazione e coding**,  **testing e manutenzione del software**. Al contrario, il **metodo Agile** si basa su un approccio iterativo e incrementale. Utilizza cicli brevi di sviluppo, noti come **sprint**, che consentono di adattarsi rapidamente ai cambiamenti. Questo metodo promuove un alto livello di flessibilità, con un feedback continuo dai clienti e una collaborazione intensa tra team e stakeholder. È particolarmente adatto per progetti in cui i requisiti possono evolversi nel tempo. **Differenze PrincipaliStruttura**: Waterfall è rigido, mentre Agile è adattabile,**Documentazione**: Waterfall richiede documentazione dettagliata a priori; Agile si concentra su comunicazioni dirette e feedback, con una documentazione ridotta,**Rischio**: Waterfall presenta maggiori rischi a lungo termine, mentre Agile riduce i rischi grazie a revisioni frequenti. In sintesi, il modello Waterfall è più appropriato per progetti con requisiti ben definiti, mentre il metodo Agile è ideale per situazioni dinamiche che richiedono adattamenti e risposte rapide.

**Quando è consigliabile utilizzare il modello a cascata**? **Requisiti chiari e stabili**: Ideale quando i requisiti sono ben definiti all'inizio e non ci si aspetta cambiamenti significativi durante lo sviluppo.**Progetto piccolo o poco complesso**: Funziona bene in contesti con poche interdipendenze e funzionalità semplici, dove non è necessaria molta flessibilità.**Richiesta di documentazione dettagliata**: Utile in progetti regolamentati o in settori critici (medico, ingegneristico) dove la tracciabilità è essenziale.**Prodotti software tradizionali**: Adatto a software consolidati in settori che non richiedono cambiamenti frequenti o integrazione rapida di nuove tecnologie.**Budget e tempistiche rigidi**: Offre un approccio prevedibile e ben strutturato, facilitando il rispetto delle scadenze e del budget predefinito.Pianificazione dettagliata. **utenti finali non sono coinvolti** attivamente durante il processo di sviluppo (software governativi)

 1.3 Quali sono vantaggi e svantaggi del modello a cascata e del agile? **Vantaggi del modello a cascata:** Pianificazione chiara e dettagliata, che facilita la gestione del progetto.Documentazione rigorosa, utile per la manutenzione e la formazione.**Svantaggi:**Rigidità, poiché le modifiche ai requisiti possono essere difficili da implementare una volta che una fase è completata.Rischio di scoprire problemi significativi solo nelle fasi finali, quando il costo per correggerli è elevato. **Approcci agili:** Offrono maggiore flessibilità e adattabilità, consentendo modifiche rapide in risposta ai cambiamenti nei requisiti. Tuttavia, possono mancare di una documentazione rigorosa e richiedere una forte collaborazione tra team e clienti.

**Quali sono le caratteristiche principali del modello Agile e in quali contesti è indicato?Il** modello Agile si basa su cicli iterativi e incrementali chiamati sprint, durante i quali il team lavora su piccoli blocchi di funzionalità prioritizzate, con piccoli rilasci frequenti e adattamento continuo ai cambiamenti. Le caratteristiche principali includono la cooperazione attiva tra sviluppatori e clienti, un approccio semplice da apprendere e modificare, e un alto livello di adattabilità ai cambiamenti anche tardivi. Agile è ideale quando i requisiti non sono del tutto definiti all’inizio o sono soggetti a variazioni, e quando i clienti sono coinvolti nel processo di sviluppo, come spesso accade per applicazioni web o prodotti software per il mercato di consumo​. Richiede un team di sviluppo auto-organizzato e autonomo.

**Che cos’è il RASD** Il **RASD** (Requirements Analysis and Specification Document) è un documento che descrive in modo dettagliato i requisiti di un sistema software da sviluppare. Serve come punto di riferimento per lo sviluppo del sistema, guidando sia gli sviluppatori che gli stakeholder (tutte le entità coinvolte e interessate al progetto) verso una comprensione condivisa del prodotto. Le sue caratteristiche principali includono: **Descrizione dettagliata dei requisiti**: Specifica cosa deve fare il sistema e come deve comportarsi.**Identificazione degli stakeholder**: Chi sono gli utenti e quali sono le loro esigenze.**Vincoli e limitazioni**: Quali sono le condizioni che il sistema deve rispettare.**Casi d'uso e scenari**: Fornisce esempi pratici di come gli utenti interagiranno con il sistema, facilitando la comprensione dei requisiti.**Tracciabilità dei requisiti**: Include una matrice di tracciabilità per garantire che tutti i requisiti siano soddisfatti durante il ciclo di vita del progetto. Il RASD è quindi un componente della documentazione del progetto, focalizzato sui requisiti specifici del sistema o del prodotto da sviluppare.

**Cosa è lo Unit Testing?** L'**unit testing** è una tecnica di testing in cui si verifica una singola funzione o modulo di un programma in isolamento, per assicurarsi che funzioni correttamente. Aiuta a rilevare bug precocemente, facilita la manutenzione e rende più sicuro il refactoring del codice. I test sono automatizzabili e veloci da eseguire. Sia il **black-box testing** che il **white-box testing** possono essere usati negli **unit test**, ma con approcci diversi:**Black-box testing**:è un **testing indiretto** che avviene a livello più alto. Si testa l'unità senza conoscere l'implementazione interna, concentrandosi solo sugli input e sugli output.**White-box testing**: è un **testing diretto** che avviene a livello più basso.Si testa l'unità conoscendo l'implementazione interna, verificando la logica e i percorsi del codice.

**Quali sono le principali caratteristiche della qualità del software secondo lo standard ISO 9126? (CRITERI PER VALUTARE UNA BUONA ARCHITETTURA SOFTWARE)**anche utilizzare un acronimo per aiutare la memoria, come **FAUEMP** (Funzionalità, Affidabilità, Usabilità, Efficienza, Manutenibilità, Portabilità).È uno standard internazionale che si occupa della **qualità del software** che viene definita in base a 6 caratteristiche indipendenti tra loro. **FUNZIONALITA’:** Capacità del software di svolgere ciò che è richiesto. **Appropriatezza**: verifica se le funzionalità sono adatte per compiti specifici. **Accuratezza**: misura la precisione con cui il software esegue le sue funzioni.**Inter-operabilità**: capacità del software di interagire con altri sistemi. **Sicurezza**: protezione contro accessi non autorizzati. **Conformità**: adesione a standard, norme e leggi​. **AFFIDABILITA’:**  Stabilità del software nelle prestazioni nel tempo e per utenti specifici. **Maturità**: misura la frequenza di errori. **Robustezza**: abilità di mantenere il funzionamento anche in presenza di errori. **Ripristinabilità**: capacità di recuperare i dati e ripristinare il funzionamento dopo un malfunzionamento. **Conformità**: rispetta gli standard e le normative rilevanti per la stabilità del software​.**USABILITA’:** Facilità con cui gli utenti possono usare il software. Determinante per una buona esperienza d’utilizzo. **Comprensibilità**: facilità con cui un utente capisce le operazioni da eseguire. **Apprendibilità**: facilità con cui un utente impara a usare il software. **Operabilità**: facilità con cui un utente può eseguire e controllare le operazioni. **Conformità**: adesione agli standard di usabilità e design di interfaccia. **EFFICIENZA:** Rapporto tra prestazioni e risorse utilizzate (numero di transazioni).si misura in **Reattività**: tempi di risposta ed elaborazione delle richieste. **Sfruttamento delle risorse**: quantità di risorse impiegate, come CPU, memoria e larghezza di banda. **Conformità**: adesione agli standard di efficienza, se esistenti. Efficienza è cruciale per evitare un uso eccessivo di risorse e garantire che il software possa funzionare in modo sostenibile in ambienti con capacità limitate​(2). **MANUTENIBILITA’:**  Facilità con cui si possono apportare modifiche al software. **PORTABILITA’:** Capacità del prodotto software di essere trasferito da un ambiente software ad un altro. **Adattabilità**: capacità del software di adattarsi a vari ambienti senza modifiche sostanziali. **Installabilità**: facilità di installazione in ambienti diversi. **Coesistenza**: capacità di coesistere con altre applicazioni condividendo risorse. **Sostituibilità**: possibilità di sostituire altre applicazioni con questo software​. Ogni categoria ha delle misure specifiche. Per esempio, la maturità può essere misurata tramite il tempo medio tra i guasti. Gli sviluppatori possono impostare limiti di accettabilità per ogni misura . Questi criteri forniscono un quadro completo per valutare la qualità dell'architettura software, assicurando che sia robusta, sostenibile e in grado di evolvere in modo controllato nel tempo.

**Descrivi le Reviews** È un processo in cui il codice, la progettazione o altri elementi legati allo sviluppo vengono esaminati per verificarne la qualità, correttezza e l’aderenza degli standard. Aiutano a individuare errori o potenziali problemi prima che il software venga rilasciato. Ci sono vari tipi di revisioni: **Ispezioni:** Sono formali e sistematiche, con un moderatore addestrato che segue procedure documentate.  **Walkthroughs:** Più informali, condotti dall'autore del documento che presenta il suo lavoro a un gruppo di pari per ottenere feedback. **Pair programming:** Due sviluppatori lavorano insieme sullo stesso codice, effettuando revisioni continue e incrementali.    **Ad-hoc reviews:** Revisioni informali in cui un programmatore chiede a un collega di esaminare il proprio codice per ottenere una prospettiva differente.Le fasi principali di un processo di revisione formale includono:**Pianificazione:** Selezione dei partecipanti e assegnazione dei ruoli.**Preparazione individuale:** Ogni revisore esamina il documento da solo prima dell'incontro.**Incontro di revisione:** Discussione tra i partecipanti, durante la quale vengono identificati difetti e vengono fatti commenti.**Rework:** L'autore corregge i difetti trovati durante la revisione.**Follow-up:** Si verifica se i difetti sono stati corretti e si raccolgono dati per migliorare il processo.I ruoli principali in una revisione formale includono:**Moderatore:** Conduce la revisione e garantisce che il processo venga seguito correttamente.**Autore:** Colui che ha prodotto il documento o il codice in esame, è un partecipante passivo.**Revisori:** Analizzano il documento e identificano difetti o problemi.**Scriba:** Registra i difetti e le decisioni prese durante l'incontro.**Manager:** Decide se gli obiettivi della revisione sono stati raggiunti.

**Cos'è l'analisi del software e quali tipi di artefatti coinvolge con vantaggi e svantaggi?** L'analisi del software è il processo di esame e valutazione del software per migliorare la qualità, rilevare bug e garantire che soddisfi i requisiti. Coinvolge vari artefatti del software, come il codice sorgente, la documentazione, l'architettura e i test.**Artefatti coinvolti**: **Codice sorgente**: Analizzato per rilevare bug, performance e sicurezza.**Documenti di requisiti**: Verificano la completezza e la correttezza delle specifiche.**Architettura**: Valuta la struttura del software per scalabilità e manutenibilità. **Casi di test e log di esecuzione**: Monitorano e verificano i comportamenti attesi.**Tipi di analisi** **Analisi statica**: Esegue un'analisi del codice senza eseguire il programma  basandosi sulla struttura e qualità del codice. Vantaggiosa per rilevare errori precoci e problemi di stile, ma non rileva bug di runtime. **Analisi dinamica**: Si esegue durante l'esecuzione del software per valutare il comportamento effettivo del sistema. Efficace per trovare bug di runtime, ma più costosa e complessa.**Vantaggi** **Rilevazione precoce di bug**,**Miglioramento** della qualità e manutenibilità,**Prevenzione** di vulnerabilità di sicurezza**Svantaggi** Richiede risorse e tempo, Potenziali falsi positivi o negativi, Non sempre rileva problemi runtime complessi (analisi statica). L'**analisi del software** e il **software testing** svolgono ruoli diversi. L'**analisi del software** si concentra sull'esame del codice, della progettazione e dell'architettura per assicurarsi che il software sia costruito correttamente. Il **software testing** verifica che il software funzioni come previsto in fase di esecuzione.Entrambi i processi si completano a vicenda, contribuendo a migliorare la qualità complessiva del software e a ridurre i difetti che potrebbero emergere durante o dopo il rilascio

ANALISI AD-HOC? e differenza con ad-hoc reviews L'**analisi ad-hoc** è un ***testing*** informale, non pianificato, in cui il tester esplora il software liberamente, basandosi su intuito ed esperienza per individuare bug. È rapido e flessibile, ma non garantisce copertura completa poiché non segue una procedura strutturata e può rendere difficile replicare i risultati se non ben documentato.Le **ad-hoc reviews**, invece, sono revisioni informali di codice o documentazione, dove i revisori esaminano liberamente il contenuto per identificare errori o suggerire miglioramenti, senza seguire criteri prestabiliti. In entrambi i casi, il termine "ad-hoc" indica un approccio non strutturato e flessibile, ma la differenza principale è che le **ad-hoc reviews** riguardano la revisione di codice o documentazione, mentre l'**analisi ad-hoc** si concentra sul testing del software.

**Quali sono i vantaggi e svantaggi di far testare il codice agli sviluppatori stessi?Vantaggi:** Conoscenza dettagliata del programma da testare. Feedback immediato sui difetti. Guida per il debug. Nessun costo di comunicazione tra tester e sviluppatori. Maggiore fiducia nel proprio codice. **Svantaggi:** C'è una "cecità" nei confronti dei propri errori. Lo sviluppatore viene a mancare nella sua mansione di sviluppatore stesso per testare codice: segue uno spreco di risorse economiche e materiali.Potrebbe mancare una mentalità critica per trovare difetti. Problemi causati da malintesi delle specifiche potrebbero non essere individuati.

**Qual è la differenza tra verifica e validazione in Agile e perché sono importanti?**La verifica e la validazione sono processi fondamentali nel modello Agile per garantire la qualità del prodotto. La **validazione** risponde alla domanda "Stiamo costruendo il prodotto giusto?" assicurando che il software soddisfi le esigenze del cliente.La **verifica** risponde a "Stiamo costruendo il prodotto nel modo giusto?", controllando che il software sia conforme alle specifiche tecniche stabilite. In Agile, la verifica e la validazione sono continue e basate sul feedback del cliente, permettendo di intercettare errori o incongruenze prima del completamento​.

**Qual è il ruolo della documentazione (del progetto) nel ciclo di vita del software?**La documentazione è essenziale in tutto il ciclo di vita del software per garantire chiarezza e coerenza. Include il manuale utente preliminare, i piani di test, e la documentazione di ogni modulo implementato.  La documentazione permette ai membri del team di seguire uno standard comune e facilita la gestione delle modifiche future. Una documentazione incompleta o non aggiornata può portare a errori e confusione, specialmente nei progetti complessi​.

Parla della documentazione del progetto:La **documentazione del progetto** è una raccolta di documenti che forniscono una panoramica completa di vari aspetti di un progetto. Essa è essenziale per garantire che tutte le parti interessate abbiano una comprensione chiara degli obiettivi e delle aspettative. Ecco alcuni degli elementi chiave che può includere: **Obiettivi del progetto**: Definiscono cosa si vuole raggiungere con il progetto, delineando i risultati attesi. **Pianificazione**: tempistiche scadenze del progetto. **Risorse necessarie**: Dettaglia le risorse umane, tecniche e materiali necessarie per completare il progetto. **Fasi e deliverable**: Identifica le diverse fasi del progetto e i deliverable (prodotto pronto ad essere spedito alla fine di una fase) associati a ciascuna fase. **Gestione dei rischi**: Analisi dei rischi e strategie di mitigazione. **Criteri di successo**: Stabilisce come verrà misurato il successo del progetto, definendo indicatori chiave di performance (KPI→ key performance indicator). **Report di avanzamento**: Documenti periodici che monitorano il progresso del progetto rispetto ai piani stabiliti. Questa documentazione serve come guida generale per l'intero ciclo di vita del progetto, facilitando la comunicazione e la collaborazione tra il team di progetto e le parti interessate. ***IN SINTESI***→ **Documentazione del progetto**: Comprende una varietà di documenti che coprono l'intero progetto, fornendo una visione d'insieme sugli obiettivi, la pianificazione, le risorse, i rischi e i criteri di successo.

**Quali sono le principali differenze tra manutenzione correttiva, adattiva e perfettiva del software?** La **manutenzione correttiva** è finalizzata alla correzione di errori o bug che emergono durante l'uso del software. . La **manutenzione adattiva** riguarda l’adeguamento del software a nuovi requisiti ambientali, ad esempio cambiamenti nelle leggi fiscali. La **manutenzione perfettiva** prevede l'aggiunta di nuove funzionalità o miglioramenti, come l'ottimizzazione delle prestazioni.

**Che cos'è un "bug software" e come viene generalmente classificato?** Un *bug software* è un errore, difetto o malfunzionamento in un programma che causa comportamenti indesiderati o imprevedibili. I bug possono essere classificati come:**Errori (Error):** Azioni umane che producono risultati sbagliati (ad esempio, un errore logico nel codice).**Difetti (Fault):** Passi o processi sbagliati che possono causare il fallimento di un componente del sistema (es., scrittura errata di una variabile). **Fallimenti (Failure):** Quando un difetto si manifesta durante l'esecuzione e il sistema devia dalle sue specifiche richieste (ad esempio, crash del sistema o un risultato errato)​.

**Qual è l'importanza del testing nel software development e perché è spesso costoso?**Il testing è cruciale per garantire la qualità del software, verificare che funzioni come previsto e prevenire errori critici in produzione. Tuttavia, è costoso per diverse ragioni:. Richiede molto tempo per coprire tutti i possibili casi.È complesso, soprattutto quando si testano sistemi concorrenti o distribuiti. La riproduzione di errori non sempre è immediata, il che può aumentare i costi di debug e correzione​

**16.**  **Cos’è il "software testing" e quali sono le sue principali tipologie?** Il software testing è il processo di esecuzione del software per verificarne il corretto funzionamento, individuare bug e assicurarsi che soddisfi i requisiti. Il testing garantisce che il software operi correttamente in diversi scenari e sia privo di difetti significativi. **Principali tipologie di software testing**. **Unit Testing:** Testa singole unità di codice (funzioni, metodi), Verifica il corretto funzionamento delle singole componenti.Integration Testing: Valuta l'interazione tra moduli o componenti del sistema,Verifica che i moduli funzionino correttamente quando combinati.System Testing: Testa l'intero sistema software come un unico prodotto. Verifica che l'intero sistema soddisfi i requisiti specificati.Acceptance Testing (UAT):Effettuato dagli utenti finali per confermare che il software soddisfi le loro aspettative e requisiti.**Black-box Testing:**Il tester non conosce il codice sorgente.Si basa sugli input e sui risultati attesi per testare il comportamento del software.**White-box Testing:**Il tester ha accesso al codice sorgente.Si concentra sulla verifica dei flussi logici e della copertura del codice.Performance Testing:Valuta l'efficienza del sistema, come velocità, scalabilità e utilizzo delle risorse.**Regression Testing:**Verifica che nuove modifiche o correzioni non introducano bug in parti già testate. L'**analisi del software** e il **software testing** svolgono ruoli diversi. L'analisi **si concentra sull'esame del codice**, della **progettazione** e dell**'architettura** per assicurarsi che il software sia costruito correttamente. Il testing **verifica che il software funzioni come previsto in fase di esecuzione**. Entrambi i processi si completano a vicenda, contribuendo a migliorare la qualità complessiva del software e a ridurre i difetti che potrebbero emergere durante o dopo il rilascio. **Vantaggi del software testing** Individuazione (ma non risoluzione) di bug prima della produzione.Garanzia di conformità ai requisiti.Miglioramento della qualità e della sicurezza del software.**SVANTAGGI** Costi elevati e richiede tempo per garantire una copertura adeguata.Non può garantire l'assenza di tutti i bug. In sintesi, il software testing è fondamentale per garantire che il software funzioni correttamente e soddisfi le esigenze degli utenti. Le varie tipologie di testing affrontano diverse aree critiche del comportamento del sistema

**Cos’è un modello di progetto (design pattern) e qual è la sua importanza nel software engineering?** Un modello di progetto è una soluzione standard a un problema comune di progettazione software che si ripete in contesti diversi. L’importanza dei design pattern sta nella loro capacità di fornire soluzioni comprovate, migliorare la leggibilità e la manutenzione del codice, e facilitare la comunicazione tra sviluppatori utilizzando un linguaggio comune. Esempi includono Singleton, Observer, Factory, Strategy, ecc. Diversi tipi di pattern: Il pattern Singleton assicura che una classe abbia solo un’istanza e fornisce un punto di accesso globale a tale istanza. Questo modello è spesso utilizzato per gestire risorse condivise come connessioni al database o logger, dove è importante garantire che esista un solo oggetto di tale risorsa.**Esempio**: Implementare un gestore di connessioni al database in un’applicazione può essere fatto con il Singleton per assicurarsi che tutte le connessioni utilizzino lo stesso oggetto di connessione. Il Factory Method è un pattern creazionale che definisce un’interfaccia per creare un oggetto, ma lascia alle sottoclassi la decisione di quale classe concreta istanziare. Questo modello è utile quando il processo di creazione degli oggetti può variare, o quando si vuole evitare la creazione diretta di oggetti all’interno del codice. **Applicazione**: Viene utilizzato in framework dove le classi devono essere estensibili e la logica di creazione degli oggetti è decentralizzata. Il pattern Composite è consigliato quando hai bisogno di trattare oggetti individuali e composizioni di oggetti in modo uniforme. Questo modello consente di costruire strutture ad albero, dove i nodi possono essere sia oggetti singoli che composizioni di altri oggetti, semplificando le operazioni su intere gerarchie. Il pattern Decorator consente di aggiungere funzionalità a un oggetto in modo dinamico senza modificare il suo codice. A differenza dell’ereditarietà, che aggiunge funzionalità a livello di classe, il decorator aggiunge funzionalità a livello di istanza, rendendolo più flessibile. Il pattern Command incapsula una richiesta in un oggetto, permettendo di parametrizzare i metodi con oggetti di richiesta, mettere in coda richieste o annullarle. Viene utilizzato per implementare operazioni come l’annullamento (undo), la ripetizione (redo) e la gestione delle transazioni. Il pattern Proxy fornisce un sostituto o surrogato per un altro oggetto per controllarne l’accesso. Viene utilizzato quando l’accesso diretto all’oggetto reale è costoso o non sicuro. Un proxy può essere utilizzato per implementare cache, controlli di accesso o caricamento ritardato.**Qual è la differenza tra un pattern strutturale, comportamentale e creazionale?** **Pattern strutturali**: Questi pattern riguardano la composizione di classi o oggetti per formare strutture più grandi. Esempi sono Adapter, Composite e Facade. **Pattern comportamentali**: Si concentrano sulle interazioni tra oggetti e su come comunicano tra loro. Esempi includono Observer, Strategy e Command. **Pattern creazionali**: Questi pattern si occupano del processo di creazione degli oggetti, fornendo meccanismi per gestire la complessità. Esempi sono Factory, Singleton e Builder.

**Cos'è una race condition e come può influenzare un sistema software?** Una *race condition* si verifica quando il comportamento di un sistema dipende dall'ordine di esecuzione di eventi concorrenti. Questo accade spesso in software (che gestiscono thread o processi paralleli), dove un'operazione si basa sull'esecuzione di un'altra che potrebbe non avvenire nel giusto ordine. Ad esempio, se due thread accedono e modificano la stessa variabile senza un'adeguata sincronizzazione, possono causare comportamenti imprevedibili come errori di logica o corruzione dei dati​

**Che cos'è il debugging e come si differenzia dal testing?** Il ***debugging*** è l'attività di sviluppo che identifica le cause immediate (difetti) dei fallimenti esistenti o potenziali e risolve i problemi. Viene **eseguito dagli sviluppatori** e non dai tester. Il debugging riguarda: Identificare la causa radice degli errori.Correggere i difetti riscontrati. Mantenere la coerenza di tutti gli artefatti del software. Il ***testing***, invece, cerca di rilevare difetti, ma non si concentra sulla risoluzione. ***“Mentre il testing è un processo di verifica, il debugging è un processo di correzione​.”***

**Qual è la differenza tra "test data" e "test case"?Test data:** si riferisce agli input utilizzati durante il testing. Sono relativamente facili da generare e possono essere creati automaticamente.**Test case:** è una combinazione di input e output attesi. Creare test case è molto più complesso e di solito non può essere automatizzato. I test case rappresentano situazioni specifiche per verificare il comportamento del software​.

**Cos’è il “Software Design in-the-small”?** Il “Software Design in-the-small” riguarda lo sviluppo di componenti individuali di un sistema software. Questi componenti vengono descritti in termini astratti, ma pronti per l’implementazione senza ulteriori modifiche di design. Il focus è sulla qualità, riducendo tempi e sforzi di sviluppo. Gli obiettivi principali includono la definizione di dati, metodi e politiche .

**In cosa consiste uno studio di fattibilità e quali sono i principali elementi che deve includere?**Uno **studio di fattibilità** è un'analisi approfondita di un progetto per determinare se è praticabile e conveniente realizzarlo. Serve a identificare potenziali ostacoli, costi e benefici, nonché a garantire che il progetto soddisfi gli obiettivi aziendali. Gli elementi principali includono: analisi dei costi-benefici, la descrizione preliminare del problema, gli scenari delle possibili soluzioni, i costi, tempistiche, scelta del modello di sviluppo (Agile, Waterfall, ecc.)

22.   **Quali sono i criteri di copertura strutturale più comuni utilizzati nel testing del software** I criteri di copertura strutturale più comuni includono:**Statement coverage**: copertura di tutte le istruzioni.**Branch coverage**: copertura di tutti i rami nelle istruzioni di controllo.**Condition coverage**: copertura di tutte le condizioni elementari.**Path coverage**: copertura di tutti i percorsi unici nel programma.**Data flow coverage**: copertura dei percorsi definiti dall'uso delle variabili nel codice.

**Cos’è un modello di progetto (design pattern) e qual è la sua importanza nel software engineering?** Un modello di progetto è una soluzione standard a un problema comune di progettazione software che si ripete in contesti diversi. L’importanza dei design pattern sta nella loro capacità di fornire soluzioni comprovate, migliorare la leggibilità e la manutenzione del codice, e facilitare la comunicazione tra sviluppatori utilizzando un linguaggio comune. Esempi includono Singleton, Observer, Factory, Strategy, ecc. Diversi tipi di pattern: Il pattern Singleton assicura che una classe abbia solo un’istanza e fornisce un punto di accesso globale a tale istanza. Questo modello è spesso utilizzato per gestire risorse condivise come connessioni al database o logger, dove è importante garantire che esista un solo oggetto di tale risorsa.**Esempio**: Implementare un gestore di connessioni al database in un’applicazione può essere fatto con il Singleton per assicurarsi che tutte le connessioni utilizzino lo stesso oggetto di connessione. Il Factory Method è un pattern creazionale che definisce un’interfaccia per creare un oggetto, ma lascia alle sottoclassi la decisione di quale classe concreta istanziare. Questo modello è utile quando il processo di creazione degli oggetti può variare, o quando si vuole evitare la creazione diretta di oggetti all’interno del codice. **Applicazione**: Viene utilizzato in framework dove le classi devono essere estensibili e la logica di creazione degli oggetti è decentralizzata. Il pattern Composite è consigliato quando hai bisogno di trattare oggetti individuali e composizioni di oggetti in modo uniforme. Questo modello consente di costruire strutture ad albero, dove i nodi possono essere sia oggetti singoli che composizioni di altri oggetti, semplificando le operazioni su intere gerarchie. Il pattern Decorator consente di aggiungere funzionalità a un oggetto in modo dinamico senza modificare il suo codice. A differenza dell’ereditarietà, che aggiunge funzionalità a livello di classe, il decorator aggiunge funzionalità a livello di istanza, rendendolo più flessibile. Il pattern Command incapsula una richiesta in un oggetto, permettendo di parametrizzare i metodi con oggetti di richiesta, mettere in coda richieste o annullarle. Viene utilizzato per implementare operazioni come l’annullamento (undo), la ripetizione (redo) e la gestione delle transazioni. Il pattern Proxy fornisce un sostituto o surrogato per un altro oggetto per controllarne l’accesso. Viene utilizzato quando l’accesso diretto all’oggetto reale è costoso o non sicuro. Un proxy può essere utilizzato per implementare cache, controlli di accesso o caricamento ritardato.**Qual è la differenza tra un pattern strutturale, comportamentale e creazionale?** **Pattern strutturali**: Questi pattern riguardano la composizione di classi o oggetti per formare strutture più grandi. Esempi sono Adapter, Composite e Facade. **Pattern comportamentali**: Si concentrano sulle interazioni tra oggetti e su come comunicano tra loro. Esempi includono Observer, Strategy e Command. **Pattern creazionali**: Questi pattern si occupano del processo di creazione degli oggetti, fornendo meccanismi per gestire la complessità. Esempi sono Factory, Singleton e Builder.

**Quali sono le principali categorie di bug software secondo la classificazione di Kaner?**Le principali categorie di bug secondo Kaner includono:**Gestione degli errori:** Prevenzione, rilevazione e recupero dagli errori.**Condizioni limite:** Errori legati a confini numerici, temporali o spaziali (es., superamento di limiti in loop).**Gestione dei dati:** Errori nei tipi di dati, parametri fuori ordine o valori errati in tabelle.**Interfaccia utente:** Bug legati a funzionalità, prestazioni o mancanza di comandi.**Flusso di controllo:** Errori in sequenze di esecuzione, condizioni 'if-then-else', loop mal gestiti​(3).

**Quali sono i requisiti funzionali e non funzionali di un software, e come si distinguono?**I **requisiti funzionali** definiscono le funzioni specifiche che il software deve eseguire, come la gestione delle prenotazioni in un sistema di gestione alberghiera. I **requisiti non funzionali**, invece, descrivono come il sistema esegue queste funzioni, come velocità, usabilità, affidabilità e sicurezza. La distinzione tra i due è cruciale poiché i requisiti non funzionali influenzano le prestazioni globali del software e ne determinano l’accettabilità da parte degli utenti​(1).