Descrivere i concetti di molteplicità in attributi e associazioni tra classi in UML
La molteplicità di associazione indica il numero min e max di istanze che sono coinvolte nell'associazione.
La molteplicità degli attributi descrive il num min e max dei valori dell'attributo associati ad ogni istanza dell'entità.

Descrivere le categorie dei requisiti non funzionali secondo il modello FURPS+
Usabilità: quanto il sistema è user-friendly.
Affidabilità: è affidabile se fa quello per cui è stato progettato.
Performance: comprende attributi quali tempo di risposta, disponibilità, precisione, throughput.
Supportabilità: capacità del sistema di supportare i cambiamenti dopo essere stato sviluppato.

Descrivere la differenza tra oggetti Entity, Boundary, Control
Entity Object: dati persistenti del sistema
Boundary Object: oggetti che permettono l'interazione tra utente e sistema
Control Object: oggetti che si occupano di gestire la logica del sistema

Descrivere i criteri per la convalida dei requisiti
Correttezza: si rappresenta accuratamente il sistema
Consistenza: non ci sono contraddizioni tra i requisiti funzionali e non
Non ambiguità: è descritto esattamente un sistema
Completezza: è rappresentato tutto il sistema (tutti gli scenari)

Descrivere la differenza tra le relazioni "include" e "extend" in un diagramma dei casi d'uso
"include" è una decomposizione funzionale di un caso d'uso in due o più casi d'uso semplificati, si utilizza o per includere un flusso di eventi all'interno di un caso d'uso oppure per semplificare un caso d'uso complesso.
"extend" è una variante del normale flusso di eventi. Il flusso degli eventi eccezionali è portato fuori dal flusso degli eventi principali per rendere più chiaro il caso d'uso.

Descrivere la differenza tra il modello di processo incrementale ed il modello di processo evolutivo.
Sviluppo incrementale**:** ogni versione aggiunge nuove funzionalità/sottosistemi
Sviluppo iterativo (evolutivo)**:** da subito sono presenti tutte le funzionalità/sottosistemi che vengono successivamente raffinate, migliorate

Descrivere le categorie di requisiti non funzionali (requisiti di qualità e pseudo-requisiti)
Requisiti di qualità:
Usability: è la facilità con cui l'utente può utilizzare il sistema. Include, per esempio, convenzioni adottate per l'interfaccia utente e un manuale di utilizzo.
Reliability: è un requisito di affidabilità. È l'abilità di un sistema o di una componente di compiere le sue funzioni richieste sotto determinate condizioni per un periodo di tempo specificato.

Performance: requisiti che riguardano le caratteristiche del sistema, come: tempo di risposta, throughput, disponibilità e precisione.
Supportability: requisiti che riguardano la facilità di modifiche al sistema dopo la sua distribuzione. Esempi sono l'adattabilità, la manutenibilità, e l'internazionalizzazione.

Pseudo-requisiti:
Requisiti di implementazione: vincoli per la realizzazione del sistema, compreso l'uso di attrezzi specifici, linguaggi di programmazione o piattaforme hardware.
Requisiti di interfaccia: vincoli imposti da sistemi esterni, inclusi sistemi legacy e scambi di formati.
Requisiti di operazione: sono vincoli sulla amministrazione e gestione del sistema.
Requisiti per l'imballaggio(Packaging): vincoli sulla consegna effettiva del sistema.
Requisiti di legge: riguardano le leggi sulle licenze, regolamentazione e certificazione.

Descrivere le caratteristiche dei modelli di processo iterativi e incrementali 
Entrambi i modelli prevedono più versioni successive del sistema. Ad ogni istante, dopo il primo rilascio, esiste una versione del sistema N in esercizio e una versione N+1 in sviluppo.
Modello incrementale:
Le fasi alte del processo sono completamente realizzate. Il sistema così progettato viene decomposto in sottosistemi(incrementi) che vengono implementati, testati, rilasciati, installati e messi in manutenzione secondo un piano di priorità in tempi diversi. Diventa fondamentale la fase, o insieme di attività, di integrazione di nuovi sottosistemi prodotti con quelli già in esercizio.
Modello iterativo:
da subito sono presenti tutte le funzionalità/sottosistemi che vengono successivamente raffinate, migliorate.

Descrivere i concetti di messaggi, attivazione e lifelines in un diagramma di sequenza 
Messaggio: scambio di eventi tra oggetti
Attivazione: tempo in cui un oggetto è impegnato in una determinata operazione
Lifelines: tempo di vita dell'oggetto

Descrivere i concetti di ruolo e qualificatori nelle associazioni tra classi in UML 
Un ruolo è una stringa che etichetta un'associazione. Indica la funzione di ogni classe rispetto all'associazione.
Un qualificatore è un attributo (o insieme di attributi) il cui valore partiziona un insieme di oggetti (detti targets) associati ad un altro oggetto (detto source).

Spiegare in che modo la prototipazione può supportare la raccolta e la convalida dei requisiti
La realizzazione di un prototipo del sistema o di un sottosistema è un mezzo attraverso il quale si interagisce con il committente per valutare e identificare meglio le specifiche richieste e per verificare la fattibilità del prodotto.

Ddata la seguente tabella delle attività disegnare il relativo diagramma di pert ed individuare il percorso critico

Attività	Durata (gg)	Dipendenze
T1	3	
T2	7	T1
T3	3	T1
T4	4	T2
T5	3	T2,T3
T6	4	T3
T7	5	T4,T5

Critical path: T1->T2->T4->T7

Descrivere la differenza tra aggregazione e composizione in un diagramma delle classi UML
L'aggregazione indica che le parti coinvolte in questa relazione possono esistere in maniera indipendente dall'intero.
La composizione indica che le parti coinvolte nella relazione non possono essere indipendenti dall'intero.

Descrivere le categorie di pseudo-requirements secondo il modello FURPS+
Requisiti di implementazione: vincoli per la realizzazione del sistema, compreso l'uso di attrezzi specifici, linguaggi di programmazione o piattaforme hardware.
Requisiti di interfaccia: vincoli imposti da sistemi esterni, inclusi sistemi legacy e scambi di formati.
Requisiti di operazione: sono vincoli sulla amministrazione e gestione del sistema.
Requisiti per l'imballaggio(Packaging): vincoli sulla consegna effettiva del sistema.
Requisiti di legge: riguardano le leggi sulle licenze, regolamentazione e certificazione.

Quali sono le principali difficoltà che si incontrano durante la raccolta dei requisiti?
Le difficoltà principali sono legate alla comunicazione. Fraintendimenti e omissioni infatti possono portare spesso al fallimento dello sviluppo. Per evitare questi problemi, gli sviluppatori hanno a disposizione una serie di strumenti: convenzioni, infatti se gli sviluppatori si accordano su come rappresentare i modelli, risolvono la maggior parte dei problemi di comunicazione; strumenti CASE che servono per generare modelli e documenti; procedure, validazioni periodiche.

Descrivere il concetto di transizione in un diagramma di stato UML
Una transizione è un passaggio da uno stato ad un altro che si verifica allo scatenarsi di un evento a cui segue un'azione.

Spiegare da quali parti è composto un documento di analisi e specifica dei requisiti
È composto da tre sezioni:

- Modello funzionale (casi d'uso e scenari)
- Modello a oggetti (class diagram)

- Modello dinamico (sequence diagram, statechart diagram)

Cos'è un percorso critico in un diagramma di PERT e qual è la sua importanza?
Il percorso critico è l'insieme di archi che collega il task che non dipende da altri task (non ci sono archi entranti) con un task da cui nessuno dipende (non ci sono archi uscenti). Il percorso è scelto in base alla durata dei task, si scelgono i task con durata maggiore.
È importante perché il ritardo su uno dei task nel percorso, può ritardare di molto tutti gli altri task e quindi tutto il sistema.

Descrivere il meta-modello a spirale per il ciclo di vita del software. Perché lo si definisce meta-modello?
Un modello a spirale è caratterizzato da:
Formalizzazione del concetto di iterazione : ha il riciclo come fondamento. Il processo viene rappresentato come una spirale piuttosto che come una sequenza di attività: ogni giro della spirale rappresenta una fase del processo. Le fasi non sono definite ma vengono scelte in accordo al tipo di prodotto. Ogni fase prevede la scoperta, la valutazione e il trattamento esplicito dei "rischi".
E' un meta – modello : possibilità di utilizzare uno o più modelli. Il ciclo a cascata si può vedere come caso particolare (una sola iterazione).

Descrivere la differenza tra azione e attività in un diagramma di stato
Azione: comportamento atomico che può essere eseguito in uno specifico punto dello stato della macchina.
Attività: è legata ad uno stato. Può durare un lasso di tempo considerevole e può essere interrotta da un evento.

Descrivere i vari tipi di oggetti che è possibile individuare in fase di analisi dei requisiti e in che modo vengono individuati
Durante la fase di analisi è possibile individuare tre oggetti:
Entity Object: è responsabile dei dati persistenti, rappresenta quel dato che deve sopravvivere ad un'esecuzione del sistema.
Boundary Object: è responsabile dell'interazione tra l'utente e il sistema tramite interfaccia.
Control Object: è responsabile della gestione della logica del sistema.
Vengono individuati attraverso l'euristica di Abbot, che effettua un'analisi semantica della descrizione di uno scenario.

Spiegare la differenza tra requisiti funzionali, requisiti non funzionali e pseudo-requirements
Il requisito funzionale è una funzione che il sistema deve supportare.
Il requisito non funzionale indica gli aspetti che non sono legati direttamente alle funzionalità del sistema.
Gli pseudo-requirements sono specifici strumenti e tecnologie che il sistema deve utilizzare e rispettare.

Descrivere i vantaggi e gli svantaggi del modello di ciclo di vita a cascata e le varianti proposte per risolverli

il modello a cascata esegue i vari passi del processo di sviluppo software in maniera sequenziale.
Vantaggi: ogni fase è ben definita e non vengono mai inserite funzionalità indesiderate.
Svantaggi: i requisiti vengono indicati all'inizio e non controllati durante le varie fasi,ciò può comportare alti rischi e in alcuni casi porta al fallimento. Un altro svantaggio sta nel fatto che i gruppi non lavorano in parallelo quindi potrebbero esserci gruppi inattivi in attesa del compimento di una fase precedente.
Le varianti proposte per risolvere i problemi legati agli svantaggi si trovano nel modello iterativo.

Spiegare la differenza tra le seguenti qualità del software: robustezza, affidabilità e correttezza.

Robustezza: il grado con cui un sistema o una sua componente può funzionare correttamente in presenta di input invalidi o di condizioni stressanti dell'ambiente esterno.
Affidabilità: la probabilità che il software non causerà fallimenti per un periodo specificato di tempo.
Correttezza: il sistema viene rappresentato secondo le specifiche dell'utente.

Descrivere cosa si intende per mantenere la tracciabilità tra i requisiti e qual è la sua importanza

La tracciabilità indica il fatto che ogni funzione all'interno del codice sorgente deve poter essere tracciata nel corrispondente insieme di requisiti e viceversa.

Descrivere i vari tipi di prototipazione

mock – ups : produzione completa dell'interfaccia utente. Consente di definire con completezza e senza ambiguità i requisiti (si può, già in questa fase, definire il manuale di utente).
Breadboards : implementazione di sottoinsiemi di funzionalità critiche del SS, non nel senso della fattibilità ma in quello dei vincoli pesanti che sono posti nel funzionamento del SS (carichi elevati, tempo di risposta,...), senza le interfacce utente. Produce feedback su come implementare la funzionalità (in pratica si cerca di conoscere prima di garantire).
"Throw – away" : Lo scopo è quello di identificare meglio le specifiche richieste dall'utente sviluppando dei prototipi che sono funzionanti. Il prototipo sperimenta le parti del sistema che non sono ancora ben comprese oppure serve per valutare la fattibilità di un approccio.
"Esplorativa" : pervenire ad un prodotto finale partendo da una descrizione di massima e lavorando a stretto contatto con il committente. Lo sviluppo dovrebbe avviarsi con la parte dei requisiti meglio compresa. Consente uno sviluppo incrementale.

Cosa sono scenari e casi d'uso? Quale relazione esiste tra essi?

Uno scenario rappresenta un flusso di eventi dal punto di vista dell'utente. Un caso d'uso identifica una funzione che il sistema deve supportare. Uno scenario è un'istanza di un caso d'uso e un caso d'uso rappresenta tutti i possibili scenari.

Descrivere i concetti di accoppiamento e coesione
L'accoppiamento indica le dipendenze tra i sottosistemi. Se due sistemi sono "fortemente accoppiati" cambiamenti in uno dei due si ripercuotono fortemente sull'altro. Viceversa, se sono "debolmente" accoppiati, cambiamenti in uno dei due hanno impatto lieve sull'altro (sono quasi indipendenti). Si vuole realizzare un sistema che minimizzi l'accoppiamento.
La coesione indica le dipendenze tra elementi dello stesso sottosistema. Si vuole progettare un sistema con alta coesione, in quanto se c'è coesione tra gli elementi si tendono a svolgere le stesse operazioni.

Descrivere i concetti di stratificazione e partizionamento

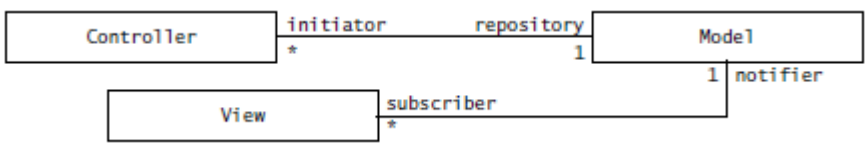
Stratificazione: suddivisione orizzontale in layer, in cui ogni layer rappresenta un'astrazione di un concetto e fornisce servizi al layer immediatamente superiore.

Partizionamento: decomposizione verticale di un sistema in sottosistemi alla pari, ognuno responsabile di una diversa classe di servizi.

Descrivere la differenza tra architetture aperte e architetture chiuse
In un'architettura aperta, ogni layer può comunicare con i layer dei livelli inferiori. Mentre in un'architettura chiusa, ogni layer può comunicare solamente con il layer immediatamente inferiore. Entrambi però forniscono servizi al layer immediatamente superiore.

Descrivere lo stile architetturale Repository
I sottosistemi fanno riferimento ad un "deposito" centralizzato che mantiene tutte le informazion.
Essi si rivolgono al repository per operare sui dati.
Vantaggi: sono adatti ad applicazioni con cambiamento costante.
Svantaggi: il repository centralizzato diventa un bottleneck in caso di un grande numero di accessi.
Ogni cambiamento sul repository ha impatto sui sottosistemi.

Descrivere lo stile architetturale Model/View/Controller
Questo stile è un caso particolare dell'architettura repository. Possiamo individuare tre tipi di sottosistemi:
Model: sottosistema che mantiene la conoscenza del dominio applicativo.
View: sottosistema che si occupa di mostrare all'utente il dominio applicativo.
Controller: sottosistema che gestisce le interazioni con l'utente.



Descrivere lo stile architetturale client/server
Il sottosistema server fornisce dei servizi al sottosistema (o ai sottosistemi) client. Questo stile è vantaggioso quando bisogna interagire con un gran numero di dati distribuiti in vari sottosistemi. Gli svantaggi riguardano il fatto che i due sottosistemi client e server hanno ruoli diversi.

Descrivere lo stile architetturale Peer-to-Peer
Questo stile è una generalizzazione del client/server, in quanto i sottosistemi possono essere sia client che server, nel senso che ognuno può richiedere o offrire servizi.

Descrivere gli stili architetturali a tre e quattro lvelli
Lo stile a tre livelli è il three tier in cui individuiamo:

- Livello dell'interfaccia: si occupa dell'interazione con l'utente e contiene tutti gli oggetti che interagiscono con esso.
- Livello delle applicazioni logiche: include gli oggetti di controllo.
- Livello di storage: mantiene i dati persistenti.

Il vantaggio sta nel fatto che vi è una debole dipendenza tra i layer.
Lo stile a quattro livelli è uguale al three tier ma decompone il livello dell'interfaccia in Presentation Client layer e Presentation Server layer. Il primo è allocato sulla macchina client, il secondo può essere allocato su uno o più server.

Descrivere i design goal relativi alle Performance
Includono i requisiti imposti sul sistema in termini di spazio e velocità:
Tempo di risposta : il tempo entro quanto una richiesta da parte di un utente deve essere soddisfatta.
Troughput : il numero di task che il sistema porta a compimento in un periodo di tempo prefissato.
Memoria : lo spazio richiesto affinché il sistema possa funzionare.

Descrivere i design goal relativi a Dependability
Determinano quanto sforzo deve essere speso per minimizzare i crash del sistema e le loro conseguenze. Includono i concetti di:
robustezza, affidabilità, disponibilità, tolleranza ai guasti, sicurezza.

Descrivere i design goal relativi a Maintenance
Determinano la difficoltà nel cambiare il sistema dopo la distribuzione. Includono i concetti di: estensibilità, modificabilità, adattabilità, portabilità, tracciabilità dei requisiti, leggibilità.

Descrivere le diverse strategie per la gestione della memorizzazione
Una volta identificati i dati persistenti, ci sono tre strategie per memorizzarli:
Flat files: astrazione di memorizzazione fornita dai sistemi operativi.
Database relazionali: strutture dati che contengono informazioni memorizzate in tabelle con uno schema predefinito. Il DBMS permette di manipolare i dati.

Database orientati agli oggetti: simile ai database relazionali ma i dati vengono memorizzati come oggetti e associazioni.

Descrivere vantaggi e svantaggi nella scelta tra flat files, database relazionali ed object-oriented
I flat files sono indicati quando si ha un gran numero di dati momentanei.
I database sia relazionali che object-oriented sono preferibili quando si hanno accessi multipli o concorrenti.
In particolare:
si preferiscono i database relazionali quando abbiamo query complesse, mentre i database object-oriented quando si ha un intenso uso di associazioni per recuperare i dati.

Descrivere i diversi approcci per la specifica del controllo degli accessi agli oggetti di un sistema software
Il controllo degli accessi si effettua attraverso la matrice degli accessi. La matrice sulle righe ha gli attori del sistema e sulle colonne le classi del sistema. Un'entry della matrice, chiamata "diritto d'accesso" specifica le operazioni che possono essere eseguite su un'istanza della classe dall'attore. La matrice si può rappresentare attraverso tre approcci:
tabella d'accesso globale: rappresenta esplicitamente ogni cella nella matrice come una tripla (attore, classe, operazione). Se tale tripla non è presente l'accesso è negato.
Lista di controllo degli accessi: associa una coppia (attore, operazione) a ciascuna classe.
Capability: associa una coppia (classe, operazione) a ciascun attore.

Descrivere i tre meccanismi principali per il flusso di controllo
Procedure-driven: le operazioni attendono l'input ogni volta che hanno bisogno di dati da un attore.
Event-driven control: un ciclo principale aspetta eventi esterni. Ogniqualvolta si verifica un evento, questo viene smistato all'oggetto appropriato.
Threads: variazione concorrente di procedure-driven.

Spiegare la differenza tra ereditarietà di specifica ed ereditarietà di implementazione e fornire un esempio per ciascun tipo di relazione
L'ereditarietà di specifica (chiamata anche Interface Inheritance) eredita da una classe astratta i metodi non implementati(implementa una interfaccia) mentre l'ereditarietà di implementazione eredita tutti i metodi dalla superclasse, ma è sconsigliato usarla, si usa la delegazione, altrimenti qualunque utilizzatore della classe potrebbe chiamare un metodo della superclasse.

Spiegare l'importanza delle interfacce. Per quale motivo si dovrebbe programmare sempre riferendosi ad un' interfaccia, piuttosto che ad un'implementazione?
Le interfacce forniscono uno strumento per realizzare astrazioni di funzionalità offerte dai sottosistemi. Riferirsi ad un'interfaccia permette allo sviluppatore di non preoccuparsi

dell'implementazione dei metodi ma di riferirsi ad essi a scatola chiusa (conosce l'input, si aspetta un output ma non conosce com'è fatto il metodo).

Descrivere il Bridge Design Pattern e fornire un problema di progettazione in cui potrebbe essere adottato

È un pattern strutturale, ovvero si occupa delle modalità di composizione di classi e oggetti per formare strutture complesse.
Problema : sviluppare, testare e integrare sottosistemi realizzati da differenti sviluppatori in maniera incrementale. Per risolvere questo problema utilizziamo il design pattern bridge.
Nome : Bridge
Descrizione del problema : separare un'astrazione da un'implementazione così che una diversa implementazione possa essere sostituita eventualmente a runtime
Soluzione : una classe Abstraction definisce l'interfaccia visibile al codice client.
Implementor è una classe astratta che definisce i metodi di basso livello disponibili ad Abstraction.
Un'istanza di Abstraction mantiene un riferimento alla sua istanza corrispondente di Implementor.
Abstraction ed Implementor possono essere raffinate indipendentemente.
Conseguenze : disaccoppiamento tra interfaccia ed implementazione (un'implementazione non è più legata in modo permanente ad un'implementazione. L'implementazione di un'astrazione può essere configurata durante l'esecuzione. La parte di alto livello di un sistema dovrà conoscere soltanto le classi Abstraction ed Implementor); maggiore estendibilità (le gerarchie Abstraction ed Implementor possono essere estese indipendentemente); mascheramento dei dettagli dell'implementazione ai client (i client non devono preoccuparsi dei dettagli implementativi).

Descrivere l' Adapter Design Pattern e fornire un problema di progettazione in cui potrebbe essere adottato

È un pattern strutturale.
Nome : Adapter
Descrizione del problema : convertire l'interfaccia utente di una classe legacy in un'interfaccia diversa che il cliente si aspetta, in maniera tale che classi diverse possano operare insieme nonostante abbiano interfacce incompatibili.
Soluzione : ogni metodo dell'interfaccia verso il client è implementato in termini di richieste alla classe legacy. Ogni conversione tra strutture dati o variazioni nel comportamento sono realizzate dalla classe Adapter. Viene usata per fornire una nuova interfaccia a componenti legacy esistenti.
Conseguenze : se il Client utilizza ClientInterface allora può utilizzare qualsiasi istanza dell'Adapter in maniera trasparente senza dover essere modificato. L'Adapter lavora con la LegacyClass e con tutte le sue sottoclassi. L'adapter pattern viene utilizzato quando l'interfaccia e la sua implementazione esistono già e non possono essere modificate.

Descrivere il Abstract Factory Design Pattern e fornire un problema di progettazione in cui potrebbe essere adottato

È un pattern creazionale ovvero fornisce un'astrazione del processo di istanziazione degli oggetti e aiuta a rendere un sistema indipendente dalle modalità di creazione,

composizione e rappresentazione degli oggetti utilizzati
Problema : Consideriamo un'applicazione per un'abitazione intelligente: l'applicazione riceve eventi da sensori ed attiva comandi per dispositivi. L'interoperabilità in questo dominio è debole, di conseguenza è difficile sviluppare una singola soluzione SW per tutte le aziende.
Nome: Abstract Factory (Kit).
Descrizione del problema: Fornire un'interfaccia per la creazione di famiglie di oggetti correlati o dipendenti senza specificare quali siano le loro classi concrete.
Soluzione: Una piattaforma (es., un sistema per la gestione di finestre) è rappresentato con un insieme di AbstractProducts, ciascuno dei quali rappresenta un concetto (es., un bottone).
Una classe AbstractFactory dichiara le operazioni per creare ogni singolo prodotto. Una piattaforma specifica è poi realizzata da un ConcreteFactory ed un insieme di ConcreteProducts.
Conseguenze : Isola le classi concrete (Poiché Abstract Factory incapsula la responsabilità e il processo di creazione di oggetti prodotto, rende i client indipendenti dalle classi effettivamente utilizzate per l'implementazione degli oggetti). E' possibile sostituire facilmente famiglie di prodotti a runtime (una factory concreta compare solo quando deve essere istanziata). Promuove la coerenza nell'utilizzo dei prodotti. Aggiungere nuovi prodotti è difficile poiché nuove realizzazioni devono essere create per ogni factory.

Descrivere lo Strategy Design Pattern e fornire un problema di progettazione in cui potrebbe essere adottato
È un pattern comportamentale ovvero si occupa di algoritmi e dell'assegnamento di responsabilità tra oggetti collaboranti.
Problema : Consideriamo un'applicazione mobile che si deve connettere a diversi tipi di rete, facendo lo switch in base alla posizione ed al costo della rete disponibile. Si vuole che l'applicazione possa essere adattata a futuri protocolli di rete senza dover ricompilare l'applicazione.
Nome: Strategy (Policy)
Descrizione del problema: Definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Permette agli algoritmi di variare indipendentemente dai client che ne fanno uso.
Soluzione: Un Client accede ai servizi forniti da un Context. I servizi del Context sono realizzati utilizzando uno dei diversi meccanismi, come deciso da un oggetto Policy. La classe astratta Strategy descrive l'interfaccia che è comune a tutti i meccanismi che Context può usare. La classe Policy configura Context per usare un oggetto ConcreteStrategy.
Conseguenze : Un'alternativa all'ereditarietà (L'ereditarietà legherebbe staticamente il comportamento nel Context e mescolerebbe l'implementazione del Context con l'implementazione dell'algoritmo. Inoltre sarebbe impossibile modificare l'algoritmo dinamicamente). Policy decide quale Strategy è la migliore, in base alle circostanze correnti Scelta dell'implementazione (Attraverso Strategy è possibile fornire diverse implementazioni dello stesso comportamento). Le strategie eliminano i blocchi condizionali

Descrivere il Proxy Design Pattern e fornire un problema di progettazione in cui potrebbe essere adottato

Un proxy, nella sua forma più generale è una classe che funziona come interfaccia per qualcos'altro. L'altro potrebbe essere qualunque cosa: una connessione di rete, un grosso oggetto in memoria, un file e altre risorse che sono costose o impossibili da duplicare.
Un esempio ben conosciuto di proxy pattern è l'oggetto reference dei puntatori.
Nelle situazioni in cui molte copie di un oggetto complesso devono esistere, il proxy pattern può essere adottato per incorporare il Flyweight pattern per ridurre l'occupazione di memoria dell'oggetto. Tipicamente viene creata un'istanza di oggetto complesso, e molteplici oggetti proxy, ognuno dei quali contiene un riferimento al singolo oggetto complesso. Ogni operazione svolta sui proxy viene trasmessa all'oggetto originale. Una volta che tutte le istanze del proxy sono distrutte, l'oggetto in memoria può essere deallocato.

Descrivere il Facade Design Pattern e fornire un problema di progettazione in cui potrebbe essere adottato

Letteralmente façade significa "facciata", ed infatti nella programmazione ad oggetti indica un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Descrivere i concetti di design pattern, frame work e libreria di classi in riferimento all'attività di riuso

I design pattern sono dei template di soluzioni a problemi comuni, raffinate nel tempo dagli sviluppatori. Un framework è una struttura di supporto su cui un software può essere organizzato e progettato. Lo scopo di un framework è di risparmiare allo sviluppatore la riscrittura di codice già steso in precedenza per compiti simili. Le librerie di classe sono il più generico possibile e non si focalizzano su un particolare dominio di applicazione fornendo solo un riuso limitato.

Descrivere i diversi tipi di contratto che possono essere specificati su una classe

Invariante: predicato riferito ad una classe che deve essere sempre vero, sia prima che dopo l'esecuzione di un'operazione.
Pre-condizione: predicato che deve risultare vero prima dell'esecuzione di un'operazione.
Post-condizione: predicato che deve risultare vero dopo l'esecuzione di un'operazione.

Descrivere i diversi tipi di trasformazione tra i modelli (modello a oggetti e codice sorgenti)

Ci sono quattro tipi d trasformazione:

- Refactoring: trasformazione che opera sul codice sorgente
- Forward engineering: produce un template di codice sorgente che corrisponde ad un modello a oggetti.
- Reverse engineering: si applica quando conosciamo il codice del sistema e dobbiamo ricavarne il design.

Descrivere in che modo è possibile mappare un'associazione uno a molti tra due oggetti e come memorizzare tale relazione in un database relazionale. Fornire un esempio.
L'associazione viene mappata come attributo delle entità con molteplicità "molti". Tale relazione viene memorizzata sottoforma di campo di una tabella.
Es : supponiamo che ad un cliente possano corrispondere più conti. Poiché i conti non hanno un ordine specifico possiamo usare un insieme di riferimenti per modellare la parte "molti" dell'associazione.

Descrivere in che modo è possibile mappare un'associazione molti a molti tra due oggetti e come memorizzare tale relazione in un database relazionale. Fornire un esempio.
L'associazione viene mappata come una nuova tabella contenenti le chiavi primarie delle entità coinvolte.
Esempio: supponiamo di avere due entità "persona" e "autobus" e la relazione "viaggio". Si crea una nuova tabella che contiene come chiavi primarie le chiavi esterne delle due entità.

Descrivere in che modo è possibile mappare una relazione di ereditarietà su di uno schema relazionale (mapping verticale e mapping orizzontale).
I database relazionali non supportano l'ereditarietà. Esistono due opzioni per mappare l'ereditarietà in uno schema di un database :
Mapping verticale : la superclasse e la sottoclasse sono mappate in tabelle distinte. Quella della superclasse mantiene una colonna per ogni attributo definito nella superclasse e una colonna che indica quale tipo di sottoclasse rappresenta quell'istanza. La tabella della sottoclasse include solo gli attributi aggiuntivi e una chiave esterna che la collega alla tabella della superclasse. Mapping orizzontale : non esiste una tabella per la superclasse ma una tabella per ciascuna sottoclasse.

Descrivere i concetti di fallimento, stato erroneo e fault(difetto)
Il fallimento si verifica quando il comportamento osservato del sistema è diverso da quello atteso.
Uno stato erroneo è uno stato in cui se si continua ad utilizzare il sistema si potrebbe andare incontro ad un fallimento.
Un fault è un errore hardware/software.

Descrivere la differenza tra fallimento meccanico ed algoritmico
Il fallimento meccanico è causato da un problema hardware. Il fallimento algoritmico dipende da problemi di implementazione o progettazione del sistema.

Descrivere le differenze tra testing black-box e white-box
il testing black-box si effettua senza conoscere l'implementazione del sistema. Fornisci un input e lo confronti con il risultato atteso (oracolo).
Il testing white-box si effettua,invece, indipendentemente dall'input, testando gli statement del codice.

Elencare i diversi tipi di test di usabilità

Scenario test :
Viene presentato a uno o più utenti un Visionary Scenario. Gli sviluppatori determinano quanto velocemente gli utenti comprendono lo scenario, la bontà del modello e come reagiscono gli utenti alla descrizione del nuovo sistema. Lo scenario selezionato dovrebbero essere il più realistico possibile.
Vantaggi: sono economici da realizzare e da ripetere
Svantaggi: gli utenti non possono interagire direttamente con il sistema.
Test di prototipo :
Agli utenti finali viene presentato una parte del software che implementa gli aspetti chiave del sistema : Prototipo verticale. Implementa completamente uno use case.
Prototipo orizzontale. Implementa un singolo layer nel sistema (per esempio un prototipo dell'interfaccia utente).
Prototipo funzionale. Usato per valutare le richieste cruciali (es: tempo di risposta)
Vantaggi: forniscono una vista realistica del sistema all'utente ed il prototipo può essere concepito per collezionare informazioni dettagliate
Svantaggi: richiede un impegno maggiore nella costruzione rispetto agli scenari cartacei
Test di prodotto:
Simile al test di prototipo, eccetto per il fatto che viene utilizzata una versione funzionale Del sistema. Il test può essere affrontato solo dopo che una buona parte del sistema sia stata sviluppata.

Elencare i diversi tipi di test di unità e descrivere una delle tecniche
Il testing di unità si concentra sui blocchi base del sistema: gli oggetti e i sottosistemi.
Ci sono vari tipi di test di unità:
Equivalence Testing: è una tecnica black-box che minimizza il numero di casi di test. I casi di test sono divisi in classi di equivalenza e viene preso un solo caso di test da ogni classe.
Boundary Testing: è un caso specifico di test di test di equivalenza, che si focalizza sui Casi limite delle classi di equivalenza.
Path Testing: è una tecnica white-box che identifica fault nell'implementazione di una componente. L'assunzione dietro al path testing è che provando tutti i possibili percorsi del codice almeno una volta tutte le fault scateneranno delle rispettive failure.
State-based Testing: è una tecnica di testing recente per rilevare failure relative ad caratteristiche degli ambienti ad oggetti, come polimorfismo, binding dinamico e distribuzione di funzionalità su un grande numero di piccoli metodi.

Elencare i diversi tipi di test di integrazione e descrivere una delle tecniche
Big-Bang Integration: consiste nel testare tutte le componenti individualmente e poi unirle; questa tecnica è sconsigliata perché in caso di fault non si può capire facilmente in quale sottosistema risiede.
Bottom-Up Testing: Testa prima i sottosistemi risiedenti agli strati inferiori, avvalorandosi dell'uso dei Driver per simulare quelli superiori, poi vengono testati quelli che chiamano i sottosistemi.

Quest'approccio non è consigliato per sistemi decomposti funzionalmente, perché testa i sottosistemi importanti solo alla fine.
Top-Down Testing: Questa strategia testa prima i sottosistemi che risiedono nello strato superiore e poi l'insieme di quelli testati e quelli che sono chiamati dal componente testato Sandwich Testing: Combina le strategie Bottom-Up e Top-Down. Si seleziona un livello target (minimizzando il numero di stub e di driver) poi vengono eseguiti I test sui livelli superiori e sui livelli inferiori contemporaneamente e successivamente integrati con il livello target.
Sandwich Modificato: Viene testato anche il livello di target, e quindi il target con il livello superiore(che viene sostituito al driver) ed il target con il livello inferiore(che viene sostituito allo stub).

Elencare i diversi tipi di test di sistema e descrivere una delle tecniche
Testing di unità e di integrazione si focalizzano sulla ricerca di bug nelle componenti individuali e nelle interfacce tra le componenti. Il testing di sistema assicura che il sistema completo sia conforme ai requisiti funzionali e non funzionali. Attività :
Testing funzionale. Test dei requisiti funzionali
Pilot Testing. Test di funzionalità comuni, fra un gruppo selezionato di utenti finali, nell'ambiente target
Testing di prestazioni. Test dei requisiti non funzionali
Testing di accettazione. Test di usabilità, delle funzionalità e delle prestazioni effettuato dal cliente nell'ambiente di sviluppo
Testing di installazione. Test di usabilità, delle funzionalità e delle prestazioni effettuato dal cliente nell'ambiente operativo

Descrivere i diversi tipi di test di accettazione
Benchmark test. Il cliente prepara un insieme di test case che rappresentano le condizioni tipiche sotto cui il sistema dovrà operare
Competitor testing. Il nuovo sistema è testato rispetto ad un sistema esistente o un prodotto competitore
Shadow testing. Una forma di testing a confronto, il nuovo sistema e il sistema legacy sono eseguiti in parallelo ed i loro output sono confrontati. Se il cliente è soddisfatto, il sistema è accettato, eventualmente con una lista di cambiamenti da effettuare.