In breve

- Cascata: DPIII
  - Definizione dei requisiti > Progettazione del sistema e software > Implementazione e testing dei singole unità > Integrazione e testing di sottosistema > Installazione e manutenzione
- RAD: CP(MC)D
  - Comunicazione > Progettazione > Modellazione > Costruzione > Deployment
- Evolutivo: D-S-S-V
  - Definizione di massima > Specifica > Sviluppo > Validazione
- Formale: DSTI
  - Definizione dei requisiti > specifica formale > trasformazione formale > integrazione e test
- Riuso: AAPI
  - Analisi dei Componenti > Adattamento dei requisiti > Progettazione del sistema > Integrazione
- Spirale: CPASCrV
  - Comunicazione con il cliente > Pianificazione > Analisi dei rischi > Strutturazione > Costruzione e release > Valutazione del cliente
- RUP: IECT
  - Inception > Elaboration > Construction > Transition
- Agile: PI>PS, SF>D, C, C-RISP>A
  - Persone e interazione > processi e strumenti, Software funzionante > Documentazione, Collaborazione al di là del contratto, Capacità di risposta ai cambiamenti > Aderire al progetto

Domande note

1-Descrivere il modello di **sviluppo a cascata** illustrandone la struttura, descrivendo sinteticamente le attività associate alle varie fasi e inoltre indicando quali sono i principali problemi legati al modello e le sue caratteristiche di visibilità. [punti 5]

Il modello a cascata si compone delle seguenti 5 fasi:

1. Definizione dei requisiti: si lavora alla specifica dei requisiti dopo aver avuto un dialogo con il committente
2. Progettazione del sistema: si effettua la progettazione dell'architettura del sistema e delle sue componenti
3. Implementazione e testing delle singole componenti
4. Integrazione e testing delle singole componenti
5. Installazione e manutenzione: si installa il sistema e se ne si prende cura

È possibile passare da una certa fase a quella precedente, tuttavia il cambiamento dei requisiti in corso d'opera è estremamente oneroso.

Dal punto di vista della visibilità, il modello di sviluppo a cascata è buono poiché produce deliverables ad ogni fase.
Il rischio di questo modello di sviluppo dipende dalla familiarità del team nei confronti del problema da affrontare.

4-Descrivere il modello di **sviluppo a spirale** illustrandone la struttura, descrivendo sinteticamente le attività associate alle varie fasi e inoltre indicando quali sono i principali problemi legati al modello e le sue caratteristiche di visibilità. [punti 5]

Il modello di sviluppo a spirale prevede i seguenti step, che si reiterano numerose volte fino alla release finale del software:

1. Comunicazione con il cliente
2. Progettazione del sistema
3. Analisi dei rischi
4. Strutturazione
5. Costruzione e release
6. Valutazione del cliente

Questo modello è indicato per progetti di grandi dimensioni. La visibilità è buona perché ogni spicchio della spirale produce documentazione.
I rischi sono minimizzati dalle numerose iterazioni, però la erronea valutazione dei rischi può ripercuotersi successivamente.

5-Descrivere il modello di **sviluppo evolutivo** illustrandone la struttura, descrivendo sinteticamente le attività associate alle varie fasi e inoltre indicando quali sono i principali problemi legati al modello e le sue caratteristiche di visibilità. [punti 5]

Lo sviluppo evolutivo dà importanza alla produzione di prototipi usa-e-getta a partire da definizioni dei requisiti di massima, ossia si dà molta importanza alla scrittura del codice, il quale evolve secondo le direttive del cliente. Le fasi sono:

1. Definizione di massima
2. Specifica, che produce la versione iniziale
3. Sviluppo, che produce la versione intermedia
4. Validazione, che produce la versione finale

La visibilità è scarsa a causa della produzione di documenti carente, il ché può elevare il livello di rischio (che però allo stesso tempo è ridotto grazie allo sviluppo di numerosi prototipi). È fondamentale che il prodotto finale non sia un prototipo: una volta raggiunti gli obiettivi di sviluppo è opportuno riscrivere il software per la release. Questo modello si presta particolarmente per progetti di piccola dimensione o con una vita attesa corta.

6-Descrivere il modello **di sviluppo trasformazionale** (o formale) illustrandone la struttura, descrivendo sinteticamente le attività associate alle varie fasi e inoltre indicando quali sono i principali problemi legati al modello e le sue caratteristiche di visibilità. [punti 5]

Il modello di sviluppo formale è adeguato nel momento in cui ci si prefigge, ad esempio, di codificare linguaggio matematico.

1. Definizione dei requisiti
2. Specifica formale
3. Trasformazione formale
4. Integrazione e test

Il rischio è legato all'esperienza degli sviluppatori e dalla necessita di tecnologie avanzate. La visibilità è buona, ogni fase dovrebbe produrre documentazione affinché il processo di sviluppo possa svolgersi.

7-Descrivere il **modello di sviluppo basato sul riutilizzo** illustrandone la struttura, descrivendo sinteticamente le attività associate alle varie fasi e inoltre indicando quali sono i principali problemi legati al modello e le sue caratteristiche di visibilità. [punti 5]

Il modello di sviluppo basato sul riutilizzo si basa sull'uso di componenti off-the-shelf già scritte e testate per la produzione del software.
Le fasi di questo modello sono:

1. Analisi dei componenti
2. Adattamento dei requisiti
3. Progettazione del sistema
4. Integrazione

La visibilità è buona, però può essere macchinoso scrivere documentazione per componenti off-the-shelf. Il rischio è ridotto, dato che si presuppone la correttezza dei componenti riutilizzati.
Questo modello si presta molto allo sviluppo di software ad oggetti.

27-Descrivere il modello di sviluppo **RAD** (Rapid Application Development) illustrandone la struttura e inoltre indicando vantaggi e svantaggi.

Il modello RAD è particolarmente indicato nei casi in cui si deve sviluppare un progetto facilmente partizionabile in cui non vi sia la necessità di utilizzare interfacce appositamente definite. Inoltre non è il modello indicato qualora si utilizzassero tecnologie innovative con alto rischio di incontrare problemi in corso d'opera.
Le fasi che compongono questo modello sono:

1. Comunicazione
2. Pianificazione
3. Modellazione
4. Costruzione
5. Deployment

In particolare, la modellazione e la costruzione sono ad opera di ciascun team incaricato di

lavorare su una partizione del progetto.

30-RUP

Il Rational Unified Process è un modello che si rifà a UML per la progettazione di sistemi che prende elementi da vari modelli generici e fornisce buone prassi in fatto di progettazione e specifica. È particolarmente indicato per sistemi che fanno uso di cicli. Le sue fasi sono:

1. Inception: il progetto ha inizio con il dialogo con il cliente, il quale fornisce i requisiti del sistema; si propone un'architettura di base e un piano per la natura iterativa e incrementale del progetto
2. Elaboration: comunicazione con il cliente, modellazione, si ampliano gli use cases, e si espande la rappresentazione dell'architettura
3. Construction: si sviluppa il software
4. Transition: il software è pronto per essere installato nell'ambiente reale

Queste milestones, se non producono risultati soddisfacenti, devono essere reiterate oppure si deve abortire il progetto.

31-Agile

Il modello di progettazione Agile, o Extreme Programming, è un modello molto recente e si basa su quattro punti:

1. Le persone e le interazioni sono più importanti di software e processi
2. Il software funzionante è più importante rispetto alla documentazione
3. La comunicazione è fondamentale
4. La capacità di risposta ai cambiamenti è più importante rispetto ad aderire al progetto gli sviluppatori dovrebbero suggerire modifiche

Questo modello ha la particolarità di prevedere che lo sviluppo sia condotto da due sviluppatori, uno che scrive codice e l'altro che assiste, in maniera tale da favorire la concentrazione e la buona programmazione. Inoltre presenta costi dovuti al cambiamento dei requisiti molto inferiori rispetto a quelli del modello a cascata.

2-Che cosa si intende con i termini "verifica" e "validazione" di sistemi software? Cosa vuole dire effettuarle in modo "statico" o "dinamico"? [punti 4]

Per validazione si intende il processo che risponde alla domanda "si sta costruendo il prodotto giusto?", ossia ci si chiede se il software è implementato in maniera tale da soddisfare le richieste dell'utente.

Per verifica si intende il processo che risponde alla domanda "si sta costruendo il prodotto nel modo giusto?", ossia ci si assicura che il software soddisfi le specifiche imposte dal cliente.

Quando si effettuano test statici si analizza il codice in maniera statica per ricercare problemi (la si può effettuare ancor prima che il sistema sia implementato).

I test dinamici invece prevedono il collaudo del software; È possibile effettuare la validazione solo in maniera dinamica.

3-Uno dei principi fondamentali della progettazione in interfacce utente richiede di ridurre il ricorso alla "memoria a breve termine". Che cosa si intende e come è possibile realizzare in pratica questo principio? [punti 3]

Affinché si possa raggiungere questo scopo, è importante che l'interfaccia grafica a cui è sottoposto l'utente sia di facile lettura e conforme a standard assodati in termini di estetica e usabilità, in maniera tale che non si tratti di un processo di apprendimento, bensì di riconoscimento. Inoltre l'UI dovrebbe sempre fornire all'utente informazioni circa le operazioni svolte in precedenza e in maniera progressiva. Un'altra ragione per ridurre la mode di informazioni che l'utente deve ricordare è che ad una maggiore memorizzazione corrisponde una maggiore propensione a commettere errori nelle interazioni con il sistema.

8-Cosa si intende con "analisi dei cammini critici" durante la pianificazione di un progetto? Cosa implica il cammino critico e che strumento si può usare per studiare la situazione? [punti 4]

Il cammino critico è la successione di attività legate allo sviluppo del software per cui un ritardo su una qualsiasi delle attività interessate risulta in un ritardo nella consegna.
Per analisi dei cammini critici si intende lo studio della disposizione temporale dei compiti all'interno di un progetto volta ad ottimizzarla per mezzo di una rischedulazione dei compiti che tenga conto della disposizione del personale, delle dipendenze tra le attività (parallelismo) e dei tempi di consegna. Attraverso l'analisi dei cammini critici si può ridurre la durata di sviluppo del progetto.
Uno strumento utile per lo studio dei cammini critici è il diagramma di Pert.
È molto oneroso compiere quest'analisi quando il progetto è già in esecuzione.

9-Durante la fase di progettazione architetturale, cosa si intende con "scomposizione modulare"? Potete fare qualche esempio?

Per scomposizione modulare si intende un raffinamento a livello strutturale per cui i sottosistemi sono scomposti in moduli: ciò comporta una riduzione del tempo di sviluppo, ma richiede che il sistema sia effettivamente partizionabile. Alcuni esempi di scomposizione modulare sono il modello ad oggetti, che fa uso di interfacce ben definite e modelli di controllo, e il modello data-flow, il quale è riconducibile all'architettura "pipe-and-filter" e si presta molto alla gestione dei dati (ma non a sistemi interattivi).

10-Come valuto se un rischio è grave o meno?

Per ogni rischio bisogna valutare l'impatto, le conseguenze e la probabilità che si verifichi una situazione problematica che impedisca al programma di funzionare adeguatamente. Non tutti i rischi, e i fallimenti che ne derivano sono uguali, per esempio il rischio di un fallimento corruttivo è molto più grave del rischio di un fallimento transiente recuperabile a parità di probabilità di occorrenza. In altre parole bisogna chiedersi "cosa si perde qualora il rischio si verificasse?"
Il fattore di rischio si misura come prodotto tra probabilità che il rischio che si verifichi e la

la sua gravità.

11-Raccolta requisiti: i problemi (scope eccetera)

Durante la raccolta dei requisiti i problemi che possono verificarsi e che impattano le fasi successive del processo di ingegneria dei requisiti sono:

- problemi di scope, per esempio limiti del sistema mal definiti e presenza di requisiti non necessari che provocano confusione
- problemi di comprensione, dati dall'incapacità del cliente e dello sviluppatore di comunicare tra loro le caratteristiche del software; molto spesso il cliente non sa precisamente cosa vuole, pertanto lo sviluppatore per evitare di incorrere in problemi di questo genere deve riuscire a instaurare un dialogo con gli stakeholders e studiare l'ambiente in cui il software verrà rilasciato
- problemi di volatilità, ossia il fatto che nel tempo i requisiti possono cambiare.

12-Modello architetturale client-server

Il modello client-server si basa sulla richiesta di servizi da parte del client nei confronti del server.
Uno dei modelli principali è il three-tier architecture (di cui fanno parte i framework MVC), il quale si compone di tre livelli:

- Presentation layer: si tratta dell'interfaccia grafica attraverso la quale l'utente interagisce con il sistema.
- Application processing layer: si occupa di fornire le funzionalità del sistema
- Data management layer: si occupa della comunicazione con il DBMS

I vantaggi di questo modello risiedono nella sua semplicità ed efficacia in fatto di scalabilità e contenimento dei costi; gli svantaggi sono la necessità di replicare alcune componenti di gestione dei dati tra il DBMS e il Data management layer, e che la mancanza di un modello unico e condiviso può comportare un calo di prestazioni

13-Diversità tra i problemi HW e SW e come posso risolverli

Il software, al contrario dell'hardware, non si erode nel tempo, dunque non si può sostituire una componente guasta senza imbattersi nello stesso problema.
Nel caso di fallimento di una componente hardware, la ridondanza costruttiva permette di superare efficacemente malfunzionamenti; Il concetto di ridondanza è simile anche in ambiente software, ma, anziché basarsi su repliche della componentistica, sfrutta diverse implementazioni dello stesso modulo e confronta tra loro le soluzioni trovate.

14-Modularizzazione del problema (Meyer)

Una strategia di modularizzazione che trovi un buon compromesso tra costi di sviluppo e integrazione è seguire i 5 criteri di Meyer:

- scomponibilità: implica meno complessità
- componibilità: implica più produttività
- comprensibilità: ossia interfacce minime, implica più facilità di costruzione e modificabilità
- continuità: poche ripercussioni in caso di modifiche ai requisiti dei sistemi

- protezione: gli effetti anomali non si ripercuotono su altre parti del software, più manutenibilità

Altre strategie di modularizzazione del problema sono:

- top-down: si partiziona il problema a partire dai sottosistemi maggiori, scendendo verso i sottosistemi atomici e trattabili
- bottom-up (o per composizione)
- Sandwich (soluzione naturale)

15-Quando si parla di validazione del software, cosa si intende con i termini "revisione" e "walkthrough"?

La revisione di progetto consiste in riunioni in cui si riesamina il codice a seguito dello sviluppo del processo, ricorrendo alla compilazione di checklists di errori comuni.
Il walkthrough, che è un processo meno rapido rispetto alla revisione, consiste in una lettura critica del codice con l'intento di simularne l'esecuzione; si tratta di un processo collaborativo che si basa sull'esperienza degli sviluppatori.

16-Cos'è la visibilità - modello a cascata

Per visibilità si intende proprietà di un progetto di produrre documentazione durante le proprie fasi. Il modello a cascata ha una buona visibilità, infatti ogni fase produce deliverables

17-Modelli di progettazione architetturale della struttura di sistema

- Repository
- Client-Server
- flusso di dati Pipe and Filter
- Macchina Astratta

18-Tipi di modelli di progettazione

- Modelli di struttura di sistema
- Modelli di controllo centralizzato
- Modelli basato su eventi

19-Rapporto mese-uomo

Per valutare l'effort di un progetto in funzione del numero di sviluppatori è possibile ricorrere al "mese-uomo". Questa misura è efficace solamente in casi in cui il compito da svolgere è perfettamente partizionabile tra gli sviluppatori e non richiede comunicazione. Per questo motivo difficilmente la si può utilizzare per valutare il numero di sviluppatori richiesti per un task, infatti l'aggiunta di programmatore richiede che questi vengano adeguatamente formati da altri membri del personale, i quali dovranno rinunciare a parte della propria produttività.

20-Le 3 regole della UI

Le tre regole della UI sono:

1. **Il controllo deve essere nelle mani dell'utente**: l'utente deve avere un'interazione

flessibile e quindi avere sempre la possibilità di interrompere o annullare le operazioni in corso; ciò non significa che all'utente debbano essere fornite informazioni di carattere tecnico
2. **L'utente deve ricorrere il meno possibile all'uso della memoria**: (detto sopra)
3. **L'interfaccia deve essere uniforme**: il sistema deve avere un aspetto grafico consistente in ogni suo elemento; è inoltre opportuno che si rispettino canoni stilistici e di usabilità assodati, così come è importante che si mantengano modelli interattivi preesistenti.

21-Architettura a flusso di dati (pipe-and-filter)

Si tratta di un modello architetturale (tipico delle shell nei sistemi operativi tradizionali) che favorisce la manipolazione di input codice attraverso filtri. È un modello semplice ed efficace per gestire computazioni complesse attraverso la concatenazione di numerosi filtri semplici, tuttavia è poco adatto quando la struttura non è facilmente "linearizzabile"

22-White-box

Per white box si intende il collaudo del software di cui si conosce il codice e per cui si controllano tutti i vari cammini indipendenti possibili interni ai moduli. Questo tipo di collaudo è estensivo ed è ad opera degli sviluppatori. Si studiano gli aspetti procedurali del programma piuttosto che mancanze dei requisiti.

23-Black-box

Per collaudo black box si intende il collaudo del software di cui non si conosce il codice, e dunque si valuta la conformità delle specifiche indicate nella specifica dei requisiti (disinteressandosi quindi della struttura interna). Si ricercano comportamenti erronei del programma o mancanze. È complementare al collaudo white-box

24-Descrivere cosa si intende, rispettivamente, per requisiti funzionali e requisiti non-funzionali. Potete fare qualche esempio per entrambi i tipi?

Per requisiti funzionali si intendono i servizi che il programma offre o le sue funzionalità.
Per requisiti non-funzionali si intendono gli aspetti di programmazione, come librerie di codice e sistemi di sviluppo, che se non rispettati possono portare il programma ad essere inutilizzabile. Inoltre, per proprietà non funzionali si intendono caratteristiche come reliability, performance, security.

25-Design pattern (in generale cosa sono)

I design pattern sono soluzioni consolidate e accettate per un problema ricorrente. La progettazione di software che segue design pattern favorisce il "concept reuse" e permette la scrittura di codice chiuso alle modifiche e aperto alle estensioni

26-Descrivere cosa si intende per affidabilità di un sistema software, e come conviene procedere per cercare di migliorarla.

Per affidabilità si intende la probabilità che il sistema funzioni senza errori per un dato intervallo di tempo, in un dato ambiente e per un determinato scopo. È una proprietà

molto importante di ogni sistema, ma è molto difficile e costoso ottenere un'affidabilità tale da garantire la correttezza di ogni output per ogni possibile input, pertanto è più importante nella maggior parte dei casi diminuire il più possibile la possibilità che si verifichino fallimenti gravi piuttosto e implementare funzioni di recupero da fallimenti transienti piuttosto che applicare una politica di fault avoidance. Inoltre, molto spesso gli sforzi per l'aumento dell'affidabilità comportano un costo in prestazioni ed efficienza non indifferenti, senza avere la garanzia di non aggiungere errori non previsti.
Esistono numerosi metodi per misurare l'affidabilità:

- POFOD: misura la probabilità che si verifichi un fallimento a seguito di un input
- MTTF: misura il tempo medio tra fallimenti
- AVAIL: misura il tempo necessario per ripristinare l'operatività del sistema a seguito di un fallimento

Alcuni passi per migliorare l'affidabilità del prodotto finito è applicare nelle fasi di progettazione e di sviluppo sani principi di buona programmazione, per cui evitare goto e fare un uso ragionato dell'ereditarietà e dell'allocazione in memoria.

28-A cosa serve la fase di "negoziazione" durante l'analisi dei requisiti?

La negoziazione durante l'analisi dei requisiti è molto importante ai fini della stesura della specifica dei requisiti in quanto permette agli sviluppatori di valutare i requisiti proposti dai clienti e i vincoli indicati dagli stakeholders e imposti dall'ambiente, così da poter trovare compromessi e risolvere i conflitti per accontentare tutte le parti in gioco.

29-Descrivere cosa si intende per "piano di progetto" e per "piano esecutivo".

Il piano di progetto è la sequenza potenziale dei compiti da svolgere per completare il processo che tiene conto solamente dei vincoli di precedenza tra i vari tasks, ignorando fattori come la durata di sviluppo e le risorse economiche.
Il piano esecutivo è la sequenza effettiva delle attività elementari che tiene conto dei fattori economici, delle tempistiche di sviluppo e della gestione del personale.

32-Back-to-back testing

Utilizzato quando sono disponibili più versioni diverse dello stesso sistema

- se l'output è diverso nelle diverse versioni ci sono errori potenziali
- il confronto può essere automatizzato per ridurre il costo dei test
- Si può usare quando è disponibile un prototipo, quando un sistema viene sviluppato in più versioni (magari su diverse piattaforme), oppure nel caso di upgrade o nuove release del sistema

33-TMR – Triple Modular Redundancy

Nell'hardware la tolleranza avviene attraverso la triple-modular redundancy, ossia tre componenti identiche che ricevono gli stessi input e devono generare gli stessi output: è diverso dagli altri due, viene ignorato e la componente che l'ha prodotto viene dichiarata guasta. Si assume che le componenti hw falliscano causa usura, non perché progettate male