Domande Orale APS

Elia Ronchetti
@ulerich

2022/2023

Indice

Capitolo 1

Domande Capitolo 1

Che cosa sono l'analisi e la Progettazione

- Analisi - Enfatizza l'investigazione di un problema e dei suoi requisiti, anzichè di una soluzione
- La progettazione enfatizza una soluzione concettuale che soddisfa i requisiti del problema

Che cosa sono l'analisi e la progettazione Object Oriented

L'analisi OO (Object Oriented) definisce le classi concettuali (di dominio), servono a descrivere i concetti o oggetti (del mondo reale) relativi al problema.
La progettazione OO definisce le classi software che servono a modellare i concetti del mondo reale in classi e oggetti software utilizzabili all'interno del codice scritto in un linguaggio OO.

UML è una metodologia?

No è un linguaggio visuale

Cos'è un processo software

É un approccio disciplinato per la costruzione, il rilascio e la manutenzione del codice. Definisce chi (ruoli) fa cosa (attività), quando (organizzazione temporale) e come (metodologie) per raggiungere un certo obiettivo. Vi sono vari tipi di processi per lo sviluppo, ma la maggior parte hanno (o rielaborano) le attività fondamentali.

Come ricordarsi la risposta Parallelo con la costruzione di una casa, costruzione-rilascio(vendita)-manutenzione.
Il chi sono le varie aziende che operano (ruoli), il cosa sono le attività (caldaia, pannelli solare, cappotto, ecc.), il quando è (quando l'azienda arriva, organizzazione tmeporale) e come il metodo che utilizzano per il lavoro.

Quali sono le attività fondamentali di processo?

Le seguenti attività sono solitamente incluse in ogni processo software (che tuttavia può cambiarle e gestirle come vuole):

- Requisiti
- Analisi
- Progettazione
- Implementazione
- Validazione
- Rilascio e installazione
- Manutenzione ed evoluzione
- Gestione del progetto

Come ricordarsi la risposta Parallelo con richiesta PC assemblato dove, chiedo quali sono le componenti, analizzo se sono ok, produco un preventivo (progettazione), costruisco la macchina (implementazione) Testo se è tutto oK (Validazione), installo il sistema operativo (rilascio e installazione), il cliente poi dovrà mantenere la macchina (manutenzione ed evoluzione) ed eventualmente effettuare upgrade in futuro, quindi gestirla (gestione del progetto).
Altro esempio valido è la costruzione di una casa.

Che cos'è il processo a cascata

Il processo a cascata è un processo software sequenziale che prevede le seguenti fasi:

- Definizione dei requisiti
- Design del sistema e del software

- Implementazione e testing unitario
- Integrazione e testing di sistema
- Rilascio e manutenzione (che fa ripartire la cascata da uno degli step precedenti)

Si è rivelato essere un approccio fallimentare dato che il suo presupposto è che i requisiti siano stabili nel tempo, assunzione falsa dato che possono variare e anche di molto nel tempo (a seconda per esempio delle esigenze del cliente).

Che cos'è UP e quali sono le sue fasi?

UP (Unified Process) è un processo per lo sviluppo software di tipo iterativo, incrementale dato che si basa su iterazioni time-boxed alla fine delle quali si verifica se i requsiti definiti sono corretti e se ciò che è stato prodotto durante l'iterazione (software eseguibile) è ok. Incrementale dato che nella successiva iterazione si riparte dai risultati prodotti dalla precedente. Le sue fasi sono le seguenti

- Ideazione - Analisi e stime iniziali per avviare il progetto (stime sia temporali che economiche)
- Elaborazione - Realizzazione del nucleo dell'architettura, visione raffinata, identificazione di gran parte dei requisiti
- Costruzione - Implementazione delle capacità operative iniziali e successiva preparazione al rilascio
- Transizione - Completamento del progetto

Che cos'è un'iterazione

Si tratta di una finestra temporale (2-6 settimane) all'interno della quale si produce un mini progetto (tranne nella fase dell'ideazione), ogni iterazione include:

- Pianificazione
- Analisi e Progettazione
- Costruzione
- Integrazione e Test
- Release

Che cos'è il Metodo Agile

Lo sviluppo Agile è una forma di sviluppo che incoraggia l'agilità, ovvero una risposta rapida e flessibile ai cambiamenti, adottabile da qualsiasi processo iterativo. UP può essere reso agile aggiungendo:

- Un piccoli insieme di attività ed elaborati
- I requisiti e la progettazione non vengono completati prima dell'implementazione, ma emergono in modo adattivo durante una serie di iterazioni, anche sulla base di feedback
- Applicazione di UML in stile agile (fare solo ciò che serve)

Capitolo 2

Capitolo 2 - Analisi dei requisiti e Casi d'uso

Che cosa sono i requisiti e di che tipo possono essere?

Un requisito è una capacità o condizione a cui il software deve essere conforme. Ogni software deve possedere delle funzionalità o caratteristiche di qualità.
I requisiti possono essere:

- Funzionali - Definiscono servizi o funzionalità che soddisfano le richieste dal cliente. Sono descritti dai **Casi d'uso**
- Non funzionali - Requisiti legati alle proprietà del sistema (velocità, sicurezza, affidabilità, ecc.)

I requisiti funzionali sono spesso più vincolanti dei funzionali, un requisito non funzionale per essere rispettato può generare la creazione di una serie di requisiti funzionali.
I requisiti non funzionali possono entrare in contrasto fra di loro e sono divisi nelle seguente categorie:

- Prodotto - Affidabilità, velocità, sicurezza, ecc.
- Organizzativi - Processo - Che modello di sviluppo usiamo, linguaggio di programmazione
- Esterni - Leggi da rispettare e normative

I requisiti sono estremamente importanti in un progetto, considerando che la loro definizione incompleta è una delle cause principali del fallimento di progetti.

In UP vengono definiti inizialmente durante l'ideazione (insieme al cliente) e successivamente nella fase di elaborazione. Non è da escludere che possano essere modificati anche più tardi, ma con UP tendono a stabilizzarsi nel tempo. Consideriamo che il 25 % dei requisiti cambia dopo l'ideazione. UP incoraggia un'acquisizione dei requisiti agile, attraverso la scrittura dei **Casi d'uso** con i clienti, anche tramite interviste, workshop dei requisiti con gli sviluppatori e clienti, e feedback dai clienti dopo ogni iterazione.

Cosa sono i casi d'uso?

I casi d'uso sono storie scritte in formato testuale (preferibilmente in forma ridotta) e costituiscono un dialogo fra uno o più attori e un sistema che svolge un compito. Sono utilizzati per scoprire e registrare i requisiti funzionali (possono anche descrivere requisiti non funzionali).
Se presenti il caso d'uso prevede anche il SSD (System Sequence Diagram - Diagrammi di Sequenza di Sistema).

Perchè scriviamo i casi d'uso

- Perchè ci aiutano a identificare e descrivere i requisiti funzionali
- Sono comprensibili dal cliente dato che sono i formato testuale e sono privi di gergo informatico
- Mettono in risalto gli obiettivi dell'utente e il loro punto di vista
- Sono utili per produrre test e la guida utente

Che cos'è un attore e quali sono i vari tipi

Un attore è qualcosa dotato di un comportamento, anche il Sistema in discussione è considerato un attore quando ricorre ai servizi di altri sistemi.
I vari tipi di attori sono:

- Attore primario - Utilizza direttamente i servizi del sistema in discussione affinchè vengano raggiunti gli obiettivi dell'utente
- Attore finale - Vuole che il sistema venga utilizzato per raggiungere i propri obiettivi (spesso coincide con il primario)
- Attore di supporto - Offre un servizio al Sistema (es. Servizi esterni di pagamento)

- Attore fuori scena - Attori interessati al comportamento del Caso d'uso, ma non è nessuno dei 3 sopra elencati, e non interviene nel caso d'uso (es. Stato per quanto riguarda il rispetto di una normativa).

Capitolo 3

Elaborazione

Cosa succede nella fase dell'elaborazione?

Si stabilizzano la maggior parte dei requisiti (circa l'80 %) viene programmato, definito e testato il nucleo dell'architettura e si attenuano i rischi maggiori. L'intera fase dura alcuni mesi (sicuramente più di 1 iterazione). Qua si producono elaborati come il Modello di Dominio, il Modello di Progetto e un Documento sull'architettura.

Cos'è un Modello di Dominio

Il modello di Dominio è un diagramma delle classi concettuali che mostrano i concetti del mondo reale e come sono in relazione fra di loro nel dominio di interesse. É il documento principale dell'analisi a oggetti. In esso troviamo:

- Classi concettuali - Rappresentano un'idea un oggetto del mondo reale, è un descrittore di oggetti che possiedono le stesse caratteristiche
- Attributi - Proprietà elementare degli oggetti (visibilità di default private)
- Associazioni - Relazione semantica fra le classi

Viene utilizzato per:

- Mell'analisi per comprendere il dominio del sistema e per definire un linguaggio comune sulle parti interessate
- Nella progettazione come fonte di ispirazione per lo strato di dominio

Che cos'è un SSD

Un SSD è un elaborato che descrive eventi di input e output relativi al Sistema in discussione, illustra inoltre come gli attori (o più sistemi) interagiscono con esso.
Si adotta un approccio a scatola nera, quindi tratteremo il sistema come un'unica entità che interagisce con gli attori. Il testo dei Casi d'uso sarà la base per costruire questo diagramma. É buona norma produrre un SSD per ogni scenario principale di ciascun caso d'uso.

Cosa sono le operazioni di sistema

Le operazioni di sistema sono funzionalità pubbliche che il sistema, a scatola nera, mette a disposizione tramite la sua interfaccia. Vengono eseguite quando l'utente genera un evento di sistema. Ricordiamoci che siamo nel livello di dominio quindi non dobbiamo ragionare come se stessimo creando metodi. Un metodo software è l'implementazione di un'operazione.

Cosa sono i Contratti

I contratti delle operazioni descrivono nel dettaglio il cambiamento degli oggetti di dominio dopo aver eseguito un'operazione di sistema. La descrizione del cambiamento post operazione viene effettuata nelle Post Condizioni.

Che cos'è l'architettura logica

L'architettura logica si occupa di dividere le classi software in starti/pacakges/sottosistemi in base alla loro semantica e appartenenza a un determinato strato/package/sottositema a livello software.

Con quali criteri possiamo dividere l'Architettura Logica

- Livello
- Strato
- Partizione

Per ricordarlo LiSP (Livello Strato Partizione) "Anagrammi che evocano forti emozioni sono utili per ricordare"

Quali sono gli strati dell'architettura logica

- UI (Presentazione o Interfaccia Utente)
- Logica Applicativa
- Servizi Tecnici

Che cos'è la logica applicativa

Lo stato che riguarda gli oggetti di dominio, utile suddividere questo strato in:

- Strato del dominio - Oggetti di dominio
- Strato application - Oggetti che gestiscono il workflow degli oggetti di dominio

La logica applicativa non va messa direttamente nell'interfaccia UI.

Cosa sono i diagrammi di interazione

Sono i diagrammi che descrivono il modo in cui interagiscono gli oggetti attraverso lo scambio di messaggi. Un'interazione è una specifica di come gli oggetti si scambiano messaggi nel tempo per eseguire un compito.
Ci sono 4 tipi di diagrammi di interazione:

- Diagrammi di sequenza SD - Mostrano le interazioni tra linee vita verticali
- Diagrammi di comunicazione CD - Mostrano le interazioni tra oggetti in un formato a grafo o rete
- Diagrammi di interazione generale
- Diagrammi di temporizzazione

Noi abbiamo visto i primi 2

Cosa sono i Sequence Diagram

Simili agli SSD per quanto riguarda la struttura, in questo caso però siamo a livello software e non più di dominio. negli SD infatti le linee vita rappresentano spesso gli oggetti, iniziano con un messaggio trovato. I messaggi sono il modo di comunicare tra gli oggetti.

Capitolo 4

Classi

Che cos'è un diagramma delle Classi Software

Si tratta di un diagramma che rappresenta le classi, le interfacce e le relative associazioni, sono utilizzati per la modellazione statica delle classi (già introdotti nel modello di dominio).

Che cos'è un oggetto

Un oggetto è un'istanza di una classe che definisce l'insieme comune di caratteristiche (operazioni e attributi) condivisi da tutte le istanze.
Tutti gli oggetti hanno:

- Identificativo univoco
- Valore degli Attributi - la parte dei dati (detta stato)
- Operazioni - la parte del comportamento

Che cos'è un'interfaccia

Un'interfaccia è un insieme di funzionalità pubbliche identificate da un nome, separa le specifiche di una funzionalità dall'implementazione stessa, definendo un contratto con la classe che andrà a estendere l'interfaccia stessa, la classe dovrà rispettare questo contratto.

Che cosa sono le macchine a stati

Le macchine a stati sono diagrammi utili per descrivere il comportamento dinamico di classificatori come casi d'uso e classi, attraverso stati, transizioni

ed eventi. UP non li indica come elaborati, per questo vanno prodotti solo se necessario.

Che cos'è un diagramma di attività

Un diagramma di attività si occupa di descrivere le attività di un processo attraverso reti di nodi connessi da archi, si utilizzano per visualizzare i flussi. É composto dai seguenti oggetti:

- Token Game
- Nodo azione
- Nodo controllo
- Nodo oggetto
- Pin
- Partizioni

Capitolo 5

Design Pattern

5.1 GRASP

Che cos'è l'RDD e come viene svolta

La Responsability Driven Design considera un progetto OO come una comunità di oggetti ognuno con le proprie responsabilità, che collabora per fornire delle funzionalità.
Le responsabilità sono assegnate durante la progettazione ad oggetti e sono di due tipi:

- Di fare
- Di conoscere

L'RDD viene fatta nella seguente maniera:

- Identificare le responsabilità una alla volta
- Chiedersi a quale oggetto software assegnarla
- Chiedersi come l'oggetto fa a soddisfarla

Per l'assegnamento delle responsabilità di fa riferimento a GRASP

Che cos'è un pattern?

Un pattern è una coppia problema/soluzione ben conosciuta identificato con un nome. I pattern si basano su soluzioni e principi già applicati e dimostratisi corretti.

Che cosa sono i Design Pattern GRASP

I Design Pattern GRASP (General Responsability Assignment Software Patterns) descrivono i principi base per la progettazione di oggetti e l'assegnazione di responsabilità.
I Design Pattern GRASP sono 9:

- Creator - Chi crea l'oggetto A? Assegnare all'oggetto B la responsabilità di creare A
- Information Expert - Su che principio assegnare responsabilità a una classe? Fare fare le cose a chi ha le conoscenze, quindi assegnare responsabilità a una clase solo per le cose che conosce (può portare a problemi di accoppiamento e coesione)
- Low Coupling - Come ridurre l'impatto dei cambiamenti?
- Controller - Qual è il primo oggetto oltre alla UI a ricevere e coordinare un'operazione di sistema?
- High Cohesion -
- Pure Fabrication - Quale oggetto deve avere le responsabilità quando non si vogliono violare High Cohesion e Low Coupling, ma la soluzione fornita da Information Expert non è valida? Si assegna un insieme di responsabilità altamente coeso a una classe artificiale di convenienza
- Polymorfism - Come gestire alternative basate sul tipo? Operazioni che variano in base al tipo, creare per esempio un'interfaccia che racchiude le caratteristiche principali (es. forma) e poi le istanze avranno le loro specifiche (quadrato, cerchio, ecc.)
- Indirection
- Protected Variables

Che cosa sono i Design Pattern GoF?

I Design Pattern GoF offronto una soluzione a degli specifici problemi di progettazione in cui uno sviluppatore piò incappare, offronto degli esempi specifici di classi che risolvo il problema in oggetto. Sono 23 Pattern in totale suddivisi in 3 categorie:

- Creazionale - Forniscono astrazione per l'istanziazione degli oggetti

- Strutturale - Si occupano di organizzare le classi in strutture più complesse, sfruttando per esempio ereditarietà e proprietà delle classi
- Comportamentale - Si dedicano all'assegnazione di responsabilità e algoritmi

e noi ne abbiamo visti 7:

- Adapter
- Abstract Factory
- Singleton
- Strategy
- Observer
- Composite
- Facade

Spiega un Design Pattern a scelta

Il Design Pattern Singleton risponde all'esigenza di avere una sola istanza di una classe specifica, per esempio nella comunicazione con un Database non desidero avere più istanze per evitare scritture concorrenti, per questo Singleton entra in gioco. Definisco un metodo statico della classe che restituisce l'oggetto singleton.

Capitolo 6

Testing

Che cos'è il TDD

Il TDD (Test Driven Development) è una best practice applicabile ad UP, consiste nel creare prima i test relativi a una classe e poi scrivere codice per far passare il test stesso. Ci si immagina quindi che il codice esista già e si scrivano i test su di esso.
I tipi di test sono i seguenti:

- Test di unità - Test di singole classi e metodi
- Test d'integrazione - Verifica dell'integrazione di tutte le parti dell'architettura
- Test di sistema - Detto anche test End-To-End verifica il collegato complessivo di tutte le parti
- Test di accettazione - Test a scatola nera per verificare il funzionamento complessivo dal punto di vista dell'utente

Che cos'è il test di unità

Consiste nel preparare l'oggeto da testare (fixture), produrre dei relativi test, eseguire delle operazioni, validare queste operazioni (verificando che il risultato sia quello desiderato) e infine effettuare il Rilascio, rimuovendo eventuali parti utilizzate esclusivamente per il test.
Riassumendo:

- Preparazione
- Esecuzione

- Validazione
- Rilascio

Qual è il ciclo TDD

Il ciclo di lavorazione (molto breve) consiste nel:

- Produrre un test che fallisce
- Scrivere il codice più semplice per far passare il test
- Riscrivere o effettuare il Refactoring del codice per migliorarlo e passare a scrivere il prossimo test unitario

Che cos'è il Refactoring

Si tratta di una disciplina per ristruttura il codice esistente senza cambiarne il suo comportamento, ma migliorando alcuni aspetti come efficienza e chiarezza. Dato che non deve cambiarne il comportamento ad ogni refactoring vengono rieseguiti i test di unità.
I vantaggi del refactoring sono:

- Continuo miglioramento del codice
- Preparazione al cambiamento

I passi sono i seguenti:

- Assicurarsi che i test passino
- Trovare un code smell
- Determinare come migliorare il codice
- Migliorare il codice
- Rieffettuare i test
- Ripetere

Cosa sono i code smells

Un code smell indica una serie di caratteristiche nel codice che sono indice di una cattiva programmazione.
Quelli studiati in totale sono 10 e possiamo dividerli in 5 categorie

- Bloaters
  - Long Method
  - Long Class
  - Long Parameter List
- Objective-Orientation Abuse
  - Switch Statement
  - Refused Bedquest
- Change Preventers
  - Shotgun Surgery
- Dispensables
  - Duplicated Code
  - Data Class
  - Comments
- Couplers
  - Feature Envy