Esame 1

Cosa significa la linea tratteggiata (in generale e nei class diagram)? Differenza tra <<include>> ed <<extends>>? Perché cambia la direzione della dipendenza?

Cosa viene incluso? Il flusso di eventi

La freccia tratteggiata rappresenta la dipendenza, negli use case model viene utilizzata per le Dipendenze, che si dividono in:

a. Inclusioni dove un caso d'uso usa un altro caso d'uso
   <<include>> viene utilizzato sia per:
   - realizzare Decomposizioni funzionali, quindi dove un caso d'uso complesso viene decomposto in caso d'uso piu semplici.
   - Per ottenere il riuso di funzionalità esistenti.
     In entrambi i casi la freccia indica il caso d'uso base, che non può esistere da solo.
b. Estensioni dove un caso d'uso estende un altro caso d'uso.
   <<extends>> viene utilizzato quando una funzionalità ha bisogno di essere estesa.
   In questo caso la freccia indica il caso d'uso esteso, mentre il caso d'uso base puo essere eseguito da solo.

Esame 2

Cosa sono i sequence diagram?

Un sequence Diagram è una descrizione grafica degli oggetti che partecipano al caso d'uso,
gli oggetti quindi possono essere individuati oltre che durante la modellazione statica anche durante la modellazione dinamica.
Un attore manda un messaggio ad un boundary object
Come euristica:
un evento ha sempre un trasmettitore e un ricevitore, la rappresentazione dell'evento è chiamata messaggio.

Come sono strutturati?

- **Layout:**
  la prima colonna corrisponde con l'attore che ha iniziato il caso d'uso
  la seconda ai boundary Object (rappresentano l'interazione tra l'utente e il sistema)
  la terza ai control object (rappresentano le operazioni eseguite dal sistema)
- **Creazione:**
  i control object sono creati nella fase di inizializzazione di un caso d'uso e i boundary object sono creati dai control object.
- **Accesso:**
  I Controller e i boundary accedo agli entity (che rappresentano l'informazione persistente) ma non accade il viceversa.

  Distinguiamo 2 strutture: Fork e Stair diagram
- Fork diagram: i comportamenti dinamici si trovano nel singolo oggetto, di solito nel control
- Stair diagram: Il comportamento dinamico è distribuito, ogni oggetto delega alcune responsabilità ad altri.

Com'è è fatta la barra di attivazione di un attore rispetto a quella di un oggetto?

La barra di attivazione rappresenta il periodo di tempo durante il quale gli oggetti sono attivi o stanno eseguendo un'azione o partecipando a una sequenza di eventi, la barra di attivazione di un attore è sempre attiva.

Messaggi sincroni e asincroni?

In sintesi, la differenza fondamentale tra messaggi sincroni e asincroni sta nel fatto che i messaggi sincroni richiedono che il mittente attenda una risposta prima di continuare, mentre i messaggi asincroni consentono al mittente di continuare senza attendere una risposta immediata.

Esame 3

Class diagram?

-che relazioni possono esistere tra le classi?
-cos'è l'associazione?

Rappresenta la struttura del sistema e viene usato per modellare sia i concetti del dominio del problema, che del dominio delle soluzioni.
Una classe incapsula attributi e operazioni. Ogni attributo ha un tipo, ogni operazione ha una firma.
Per indicare relazioni tra le classi si usano le Associazioni.
La molteplicità indica quanti riferimenti ad altri oggetti può avere l'oggetto sorgente.
Un'aggregazione è un caso speciale di associazione che denota una gerarchia di "consiste di" ed è indicato con un diamante vuoto.
Mentre con un diamante pieno viene indicata la composizione, che è una forma di aggregazione dove i componenti non possono esistere senza l'aggregato.

Cos'è la requirement elicitation? Legame con la matrice degli accessi?

La requirement elicitation è la fase di raccolta dei requisiti è consiste nella definizione del sistema in termini comprensibili dal cliente.
La matrice degli accessi è uno strumento utilizzano in ambienti multi-utente per definire a quali operazioni può accedere un determinato attore.
Le righe rappresentano gli attori mentre le colonne rappresentano le classi (GuestService)
Un diritto di accesso è un'entrata nella matrice degli accessi ed elenca le operazioni che possono essere effettuate.
Il legame tra raccolta dei requisiti e la matrice degli accessi può essere visto nel contesto della definizione dei requisiti di accesso per un sistema. Durante raccolta dei requisiti, si potrebbe chiedere al cliente, quali operazioni devono essere in grado di eseguire gli utenti, e queste informazioni potrebbero poi essere utilizzate per popolare la matrice degli accessi. Inoltre, si potrebbe anche discutere di come i requisiti di accesso possono cambiare nel tempo o in risposta a determinati eventi, il che potrebbe implicare modifiche alla matrice degli accessi

Cosa c'è nella matrice degli accessi?

La matrice degli accessi è uno strumento utilizzano in ambienti multi-utente per definire a quali operazioni può accedere un determinato attore.
Le righe rappresentano gli attori mentre le colonne rappresentano le classi (GuestService)
Un diritto di accesso è un'entrata nella matrice degli accessi ed elenca le operazioni che possono essere effettuate.

Come la rappresentiamo in un db? Rappresentazione e memorizzazione di una matrice degli accessi?

Si possono avere varie implementazioni della Matrice degli Accessi:

a. **Tabella globale degli accessi:** rappresenta esplicitamente ogni cella nella matrice come una tupla (attore,classe,operazione)
b. **Lista di controllo degli accessi:** associa una lista di coppie (attore,operazione) a ogni classe che può essere acceduta.

c. **Capability:** associa una coppia (classe,operazione) a un attore.

Come si fa a sapere se l'attore può accedere al metodo?

Dalla sessione

Decomposizione sistema in sottosistemi?

-quali sono i principi della decomposizione?

La **decomposizione** consiste nel ridurre la complessità andando a decomporre il sistema in parti piu piccole.
Un **sottosistema** è una collezione di classi,associazioni,operazioni ecc.
I **servizi** sono gruppi di operazioni fornite dai sottosistemi, che trovano origine dai casi d'uso dei sottosistemi e sono specificati dalle interfacce dei Sottosistemi.
Per ridurre la complessità della soluzione, si decompone il sistema in parti piu piccole, in Sottosistemi.
Per fare ciò, i criteri si basano sul fatto che molte interazioni dovrebbero essere interne ai sottosistemi piuttosto che attraverso i confini

-definizione coesione?

La **Coesione** misura le dipendenze tra le classi, Alta Coesione significa che le classi nel sottosistema eseguono task simili e sono in relazione con ogni altra classe, Bassa Coesione è il viceversa.
oh no!!! L'**Accoppiamento** invece misura le dipendenze tra i sottosistemi. Alto Accoppiamento significa che i cambiamenti su un sottosistema avranno alto impatto sugli altri, Basso Accoppiamento è il viceversa.
I sottosistemi dovrebbero avere la massima coesione e il minimo accoppiamento.

-come organizzo la decomposizione?
-tecnicamente come ottengo il layering? Come fa ad andare contro la coesione? Qual è la struttura di ogni layer?
-coesione e partitioning?

Il partizionamento e la stratificazione sono tecniche usate per ottenere un basso accoppiamento.
Le **partizioni** dividono verticalmente un sistema in svariati sottosistemi indipendenti
Uno **strato** è un raggruppamento di sottosistemi che forniscono servizi correlati, eventualmente realizzati utilizzando servizi di un altro livello.
I livelli sono ordinati in quanto ogni livello può dipendere solo dai livelli di livello inferiore e non ha conoscenza degli strati sovrastanti. In un'architettura **chiusa**, ogni strato può accedere solo allo strato immediatamente sottostante. In un'architettura **aperta**, uno strato può anche accedere a strati a livelli più profondi.

Component e deployment diagram?
Le componenti del component sono fisiche o logiche?

Il **Component Diagram** è un grafo di componenti connesse mediante relazioni di dipendenza, e mostra le dipendenze tra componenti **fisiche** come: codice sorgente, librerie, eseguibili.
Le dipendenze sono mostrare con frecce tratteggiate dal componente client al componente fornitore.
Il **Deployment Diagram** è un grafo di nodi connessi mediante associazioni di comunicazione.
I nodi vengono mostrati come box 3D e possono contenere istanze di componenti

Design Pattern

Composite Pattern

Compone gli oggetti in una struttura ad albero, per rappresentare intere parti di gerarchie con profondità e altezza arbitrarie. Permette ai client di trattare oggetti individuali e composizioni di questi in maniera uniforme.

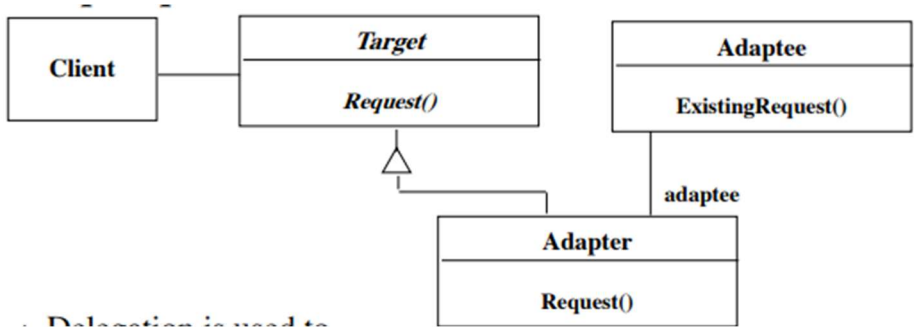
Esempio- Modellazione di un sistema
Un sistema software viene visto come un insieme di sottosistemi che sono o sottosistemi o collezioni di classi. Le foglie di questa struttura sono le classi.

Facade Pattern

Forniscono un'interfaccia unificata per un insieme di oggetti in un sottosistema. Rende il sottosistema piu semplice da usare e permette di costruire architetture chiuse. Il sottosistema decide esattamente come deve essere acceduto.

Adapter Pattern

È un pattern strutturale.
Nome : Adapter
Converte l'interfaccia di una classe in altre interfacce. Adapter permette di far lavorare insieme classi che altrimenti non potrebbero, a causa di incompatibilità di interfacce.
Viene usata la delegazione per legare un Adapter a un Adaptee (classe esistente) e viene usata l'ereditarieta dell'interfaccia per specificare l'interfaccia della classe Adapter.



Bridge Pattern

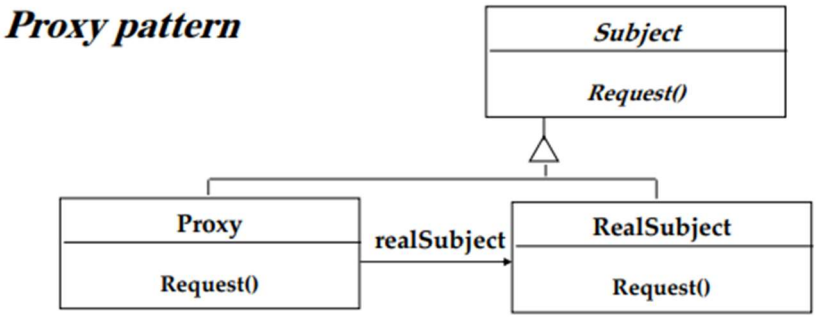
Si usa un Bridge per separare un'astrazione dalla sua implementazione. Permette a differenti implementazioni di un'interfaccia di essere decise dinamicamente.

Adapter vs Bridge

Similitudini:
vengono usati per nascondere i dettagli dell'implementazione sottostante.
Differenze:
l'adapter è attrezzato per la creazione di componenti non correlate che lavorano insieme
un bridge vien usato prima del design per permettere di poter variare astrazioni e implementazioni indipendentemente.

Proxy Pattern

Riduce il costo dell'accesso agli oggetti, usando un altro oggetto che fa le veci dell'oggetto reale. Il proxy crea l'oggetto reale solo quando l'utente lo richiede.

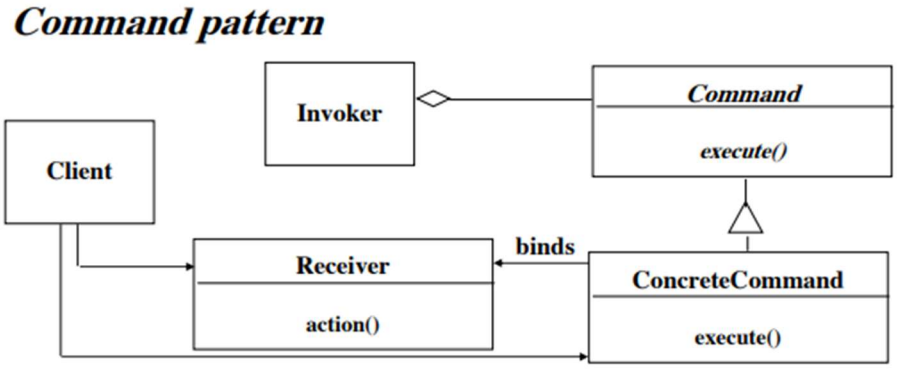


Viene usata l'ereditarietà di interfaccia per specificare un interfaccia condivisa tra le classi Proxy e RealSubject e, viene usata la delegazione per catturare e rimandare ogni accesso a RealSubject
Possono essere utilizzati in 3 maniere differenti:
Proxy Remoti: i proxy sono rappresentativi in locale di oggetti in differenti spazi.
Proxy Virtuali: gli oggetti sono troppo costosi da creare o da scaricare
Proxy di Protezione: forniscono accessi controllati a oggetti reali.

Command Pattern

Le motivazioni dietro l'utilizzo di un Command Pattern sono:

- Creare un'interfaccia utente
- Creare i menu
- Rendere l'interfaccia utente riutilizzabile su più applicazioni



Il client crea un ConcreteCommand e lo associa a un file Receiver.
Il client consegna ConcreteCommand all'Invoker che lo memorizza.
L'Invoker ha la responsabilità di eseguire il comando ("esegui" o "annulla").

Incapsula una richiesta come oggetto, permettendoti così:

- parametrizzare client con richieste diverse,
- accodare o registrare le richieste e supportiamo operazioni annullabili.

Usi:

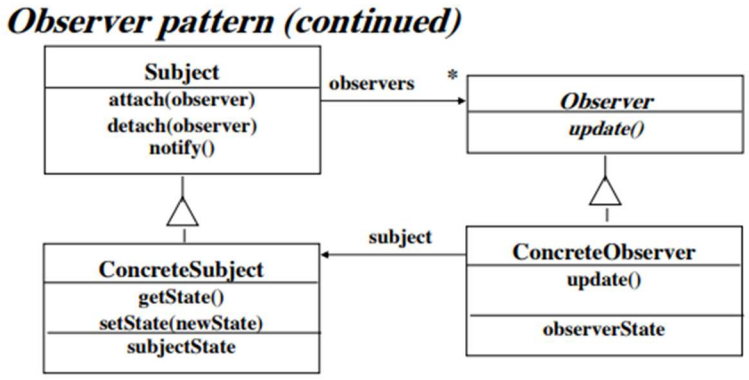
- Annulla le code
- Buffer delle transazioni del database

Observer pattern

Definire una dipendenza uno-a-molti tra gli oggetti in modo che quando un oggetto cambia stato, tutti i suoi dipendenti vengono avvisati e aggiornati automaticamente."

Usi:

- Mantenere la coerenza nello stato ridondante
- Ottimizzazione delle modifiche batch per mantenere la coerenza



Il Subject rappresenta lo stato attuale, gli Observers rappresentano visioni diverse dello Stato.

Observer può essere implementato come interfaccia Java.

Il Subject è una super classe (deve memorizzare il vettore degli osservatori) e non un'interfaccia

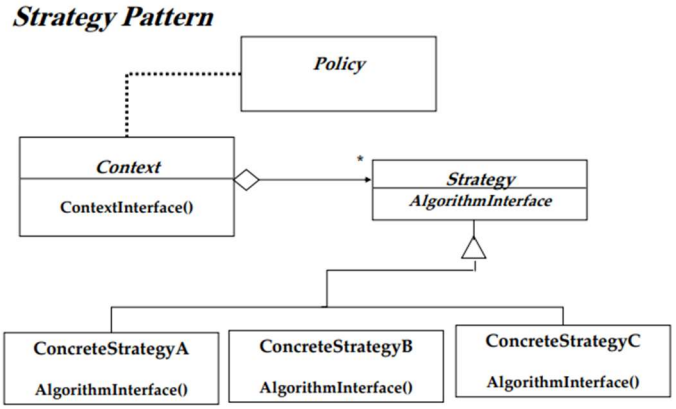
Strategy Pattern

Esistono molti algoritmi diversi per lo stesso compito

I diversi algoritmi saranno appropriati in momenti diversi

- Prototipazione rapida vs consegna del prodotto finale

Non vogliamo supportare tutti gli algoritmi se non ne abbiamo bisogno e nel caso avessimo bisogno di un nuovo algoritmo, vogliamo aggiungerlo facilmente senza disturbare l'applicazione che utilizza l'algoritmo.



`Policy` decides which `Strategy` is best given the current `Context`

Abstract Factory

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.

Builder Pattern

La realizzazione di un prodotto complesso deve essere indipendente dalle parti particolari che compongono il prodotto.

In particolare, il processo di creazione non dovrebbe conoscere il processo di assemblaggio (come le parti vengono assemblate per comporre il prodotto).

Il processo di creazione deve consentire diverse rappresentazioni dell'oggetto che viene costruito.

Abstract Factory vs Builder

Abstract Factory

- Si concentra sulla famiglia di prodotti io !!!!! :3
  - I prodotti possono essere semplici (“lampadina”) o complessi
- La fabbrica astratta non nasconde il processo di creazione
  - Il prodotto viene immediatamente restituito

Builder

- Il prodotto sottostante deve essere costruito come parte del sistema ma è molto complesso
- La costruzione del prodotto complesso cambia di volta in volta
- I modelli di creazione nascondono all'utente il complesso processo di creazione:
  - Il prodotto viene restituito dopo la creazione come passaggio finale

Abstract Factory e Builder funzionano bene insieme per una famiglia di piu prodotti complessi

OCL (Object Constraint Language)

• OCL consente di specificare formalmente i vincoli su singoli elementi del modello o gruppi di elementi del modello

• L'OCL non è un linguaggio procedurale

» Non può essere utilizzato per denotare il flusso di controllo

l OCL e UML

• OCL è un linguaggio che ci permette di esprimere vincoli sui modelli UML

» Un vincolo viene espresso come un'espressione OCL che restituisce il valore true o false

» Può essere rappresentato come una nota allegata ai modelli UML vincolati da una relazione di dipendenza

Cosa sono i contratti?

I contratti su una classe consentono al chiamante e al chiamato di condividere gli stessi presupposti sulla classe.

I contratti prevedono tre tipi di vincoli:

Invariante: un predicato che è sempre vero per tutte le istanze di una classe.

Gli invarianti sono vincoli associati a classi o interfacce.

Gli invarianti vengono utilizzati per specificare vincoli di coerenza tra gli attributi della classe.

Precondizione: un predicato che deve essere vero prima che venga richiamata un'operazione.

Le precondizioni sono associate a un'operazione specifica.

Le precondizioni vengono utilizzate per specificare i vincoli che un chiamante deve soddisfare prima di chiamare un'operazione.

Postcondizione: un predicato che deve essere vero dopo che è stata invocata un'operazione.

Le postcondizioni sono associate a un'operazione specifica.

Le postcondizioni vengono utilizzate per specificare i vincoli che l'oggetto deve garantire dopo l'invocazione dell'operazione

Testing random e testing sistematico?
Qual è l'ipotesi dietro al primo test e al secondo?
Perche devo scegliere il primo e non il secondo?

Perche i dati sono uniformi e quindi scelgo randomicamente

Il test random si basa:

- Distribuzione uniforme dello spazio di input
- Considerare tutti gli input ugualmente importanti
- Non va bene siccome la distribuzione dei difetti non è uniforme

Il test sistematico si basa:

- Seleziona l'input piu importante
- Di solito si selezionano rappresentanti di classi di input
- Lo spazio di input è suddiviso in classi

Testing BlackBox e WhiteBox? In cosa differiscono le tecniche del primo dal secondo?

Entrambi sono tipi di testing ottenuti mediante analisi dinamica:

- Black-box (funzionale), la selezione dei test è basata sulla specifica
  (scalabile a qualsiasi livello di test)
- White-box (strutturale), la selezione dei test è basata sulla logica interna del programma
  (NON scalabile, solo a livello di unità)

  BLACK-BOX:
- Il test funzionale utilizza la specifica per suddividere lo spazio di input in classi.
- Dividere i dati di input in classi di equivalenza e scegliere test case per ciascuna classe di equivalenza
- Testare ogni classe e i confini tra le classi.

  WHITE-BOX
  Si focalizza sulla completezza (copertura).
  Ogni stato nel modello dinamico dell'oggetto e ogni interazione tra gli oggetti viene testata.

  Test white-box
- Statement testing (test algebrico): testare singoli statement
- Loop Testing:
  Causa di saltare completamente l'esecuzione del ciclo. (Eccezione:Ripeti i cicli)
  Ciclo da eseguire esattamente una volta
  Ciclo da eseguire più di una volta
- Path Testing:
  Assicurarsi che tutti i percorsi nel programma vengano eseguiti
- Branch Testing (Test condizionale): assicurarsi che ogni possibile risultato di una condizione venga testato almeno una volta

Una volta decisi i cammini da eseguire, cosa devo decidere?

Gli input per percorrere i cammini.

Black-Box cosa utilizza per selezionare i casi di test?

La specifica del software siccome non si conosce il codice.

Cosa mi da l'oracolo nel BB e nel WB?

Nel BB lo da la specifica del software mentre nel WB la specifica del codice

Cosa mi permette di ottenere WB che non mi dà il BB?

Il White-Box Testing spesso testa ciò che è stato fatto, invece di ciò che dovrebbe essere fatto. Tuttavia, non individua i casi d'uso mancanti. Il Black-Box Testing ha la peculiarità di essere una potenziale esplosione combinatoria di casi di test. Spesso, però, non è chiaro se i casi di test selezionati scoprono particolari errori o meno e non scopre casi d'uso estranei.

Equivalence class testing?

Test di equivalenza

Questa tecnica di test blackbox riduce al minimo il numero di casi di test. I possibili input sono suddivisi in classi di equivalenza e per ciascuna classe viene selezionato un caso di test.
Il presupposto del test di equivalenza è che i sistemi solitamente si comportano in modo simile per tutti i membri di una classe. Per testare il comportamento associato a una classe di equivalenza, dobbiamo testarne solo un membro della classe. Il test di equivalenza consiste di due fasi:
identificazione delle classi di equivalenza e selezione degli input del test.

I seguenti criteri vengono utilizzati per determinare le classi di equivalenza.
• Copertura.
Ogni possibile input appartiene a una delle classi di equivalenza.
• Disgiunzione.
Nessun input appartiene a più di una classe di equivalenza.
• Rappresentanza.
Se l'esecuzione dimostra uno stato errato quando un particolare membro di una classe di equivalenza viene utilizzato come input, lo stesso stato errato può essere rilevato utilizzando qualsiasi altro membro della classe come input.

Per ciascuna classe di equivalenza vengono selezionati almeno due dati: un input tipico, che esercita il caso comune e un input non valido, che esercita la capacità di gestione delle eccezioni del componente. Dopo che tutte le classi di equivalenza sono state identificate, è necessario identificare un input di test per ciascuna classe che copra la classe di equivalenza.
Se c'è la possibilità che non tutti gli elementi della classe di equivalenza sono coperti dall'input del test, la classe di equivalenza deve essere suddivisi in classi di equivalenza più piccole e gli input di test devono essere identificati per ciascuno delle nuove classi.

Per cosa uso le precondizioni?

Individuo le classi valide, le non valide comunque le uso per il test di robustezza

Cosa uso per partizionare il dominio delle classi di equivalenza?

La postcondizione

Cosa partizioni in classe di equivalenza?

Per ogni parametro seleziono degli elementi

Weak e Strong equivalence testing?

Weak

Scegliere un valore variabile da ciascuna classe di equivalenza

Strong

Prodotto cartesiano dei sottoinsiemi di partizione

Da cosa mi è dato il numero di casi di test minimo?

In weak è uguale al numero di classi di test, in strong è uguale alla combinazione delle classi di test

Problemi equivalence class testing?

ECT è appropriato quando i dati di input sono definiti in termini di intervalli e insiemi di valori discreti, e siccome parte dal presupposto che il comportamento del programma sia “simile”, non si concentra su alcuni tipici errori che si trovano al limite tra classi diversi

Boundary Testing

Questo caso speciale di test di equivalenza si concentra sulle condizioni al confine delle classi di equivalenza. Invece di selezionare qualsiasi elemento nella classe di equivalenza, il testing dei confini richiede che gli elementi siano selezionati dai “margini” della classe di equivalenza.

Quindi il minimo, un po piu del minimo, un valore nominale, un po meno del massimo, il massimo.

Il test set si ottiene fissando una variabile al suo valore nominale e facendo variare le altre per tutti i valori possibili. Questo per ogni variabile. Per una funzione con n variabili richiede 4n+1 test case.

Worst Case Testing

Prodotto cartesiano di {min,min+,nom,max-.max}, è una tecnica piu completa dell’analisi dei Boundary Value, ma molto piu costosa, 5^n casi di test.

Category Partititon: Steps

Il sistema è suddiviso in singole funzioni che possono essere testate in modo indipendente

Il metodo individua i **parametri** di ciascuna “funzione” e, per ciascun parametro, identifica **categorie** distinte

Oltre ai parametri si possono considerare anche gli **oggetti dell’ambiente**

Le categorie sono proprietà o caratteristiche principali

Le categorie vengono ulteriormente suddivise in **scelte** analogamente a come viene applicato il partizionamento per equivalenza (possibili “valori”)

Vengono poi individuati i **vincoli** che operano tra le scelte, cioè, come il verificarsi di una scelta può influenzare l'esistenza di un'altra

Vengono generati i **Test Frame** che consistono nelle combinazioni consentite di scelte nelle categorie

I **Test Frame** vengono quindi convertiti in dati di prova

Testing d’integrazione e strategie?

io (bug)

Il **test di integrazione** rileva bug che non sono stati determinati durante il Test di Unità, focalizzando l’attenzione su un insieme di componenti che vengono integrate. Le strategie di test di integrazione si dividono in:
-orizzontale: in cui i componenti sono integrati in base ai livelli
-verticale: in cui i componenti sono integrati in base alle funzioni.

-testing orizzontale?
Sono stati ideati diversi approcci per implementare una strategia di test di integrazione **orizzontale**: test big bang, test bottom-up, test top-down, test sandwich, test sandwich modificata. Ciascuna di
queste strategie sono state originariamente ideate partendo dal presupposto che la scomposizione del sistema sia gerarchica e che ciascuno dei componenti appartenga a strati gerarchici ordinati rispetto all'associazione “Call”.

-differenza testing incrementale e big bang?
-strategie?

BIG-BANG

La strategia del test big bang presuppone che tutti i componenti vengano prima testati individualmente e poi testati insieme come un unico sistema. Il vantaggio è che non sono necessari stub o driver di test aggiuntivi. Anche se questa strategia sembra semplice, il big bang testing è costoso: se un test scopre un guasto, è impossibile distinguere i guasti nell'interfaccia dai guasti all'interno di un componente.

BOTTOM-UP

La strategia di test bottom-up testa innanzitutto ciascun componente dello strato inferiore individualmente, quindi li integra con i componenti dello strato successivo. Questo viene ripetuto finché tutti i componenti di tutti gli strati non vengono combinati. I test **driver** vengono utilizzati per simulare i componenti degli strati superiori che non sono ancora stati integrati. Si noti che **non sono necessari stub** di test durante i test bottom-up.

VANTAGGI:
Il vantaggio del test bottom-up è che i guasti dell'interfaccia possono essere individuati più facilmente
SVANTAGGI:
Lo svantaggio del test bottom-up è che testa i sottosistemi più importanti, vale a dire i componenti dell'interfaccia utente, per ultimi.

TOP-DOWN

Top-down testa prima i componenti dello strato superiore, quindi integra i componenti dello strato successivo. Quando tutti i componenti del nuovo livello sono stati testati insieme, viene selezionato il livello successivo. **Test stub** vengono utilizzati per simulare i componenti degli strati inferiori che non sono ancora stati integrati. Nota che i **test driver non sono necessari** durante i test top-down.

VANTAGGI:
Il vantaggio del test top-down è che inizia con i componenti dell'interfaccia utente.
SVANTAGGI:
Lo svantaggio dei test top-down è che lo sviluppo degli stub di test richiede molto tempo ed è soggetto a errori.

SANDWICH

Combina l'uso di strategie top-down e bottom-up. Il sistema è visto come se avesse 3 strati:
un livello target nel mezzo, uno sopra il target, uno sotto il target
i test top-down e bottom-up possono ora essere eseguiti in parallelo.
Il test di integrazione top-down viene eseguito testando il livello superiore in modo incrementale con i componenti del livello target, mentre il test bottom-up viene utilizzato per testare il livello inferiore in modo incrementale con i componenti del livello target.
Consente il test anticipato dei componenti dell'interfaccia utente.

PROBLEMA

C'è un problema con il test sandwich: non testa a fondo i singoli componenti dello strato target prima dell'integrazione.

SANDWICH MODIFICATA

La strategia di test sandwich modificata testa i tre strati individualmente prima di combinarli tra loro in test incrementali.

I test dei singoli livelli consistono in un gruppo di tre prove:
• un test del livello superiore con stub per il livello target.
• un test del livello target con driver e stub che sostituiscono i livelli superiore e inferiore
• un test dello strato inferiore con un driver per lo strato target.

I test a strati combinati consistono in due test:
• Il livello superiore accede al livello target. Sostituendo i driver con componenti del livello superiore.

• Al livello inferiore si accede dal livello target. Sostituendo lo stub con i componenti del livello inferiore.

-cosa ho bisogno in entrambi per effettuare test?
-quando si usa stub e driver?
-perche il sandwich è utile?

Permette di unire i vantaggi di bottom up e top down, quindi testare facilmente le interfacce grafiche e subito.

System Testing

♦ Functional Testing, Structure Testing, Performance Testing, Acceptance Testing

Testing di struttura

♦ Essenzialmente uguale al test white box.

♦ Obiettivo: coprire tutti i percorsi nella progettazione del sistema

Test funzionali

Lo stesso del BB

♦ Obiettivo: testare la funzionalità del sistema

♦ I casi di test sono progettati dal documento di analisi dei requisiti e incentrati sui requisiti

Test di performance

♦ Spingere il sistema (integrato) ai suoi limiti.

♦ Obiettivo: provare a rompere il sottosistema

♦ Testare il comportamento del sistema in caso di sovraccarico.

Test di accettazione

♦ **Obiettivo: dimostrare che il sistema è pronto per l'uso operativo**

Viene eseguito il test di accettazione dal cliente, non dallo sviluppatore.

Due tipi di test aggiuntivi:

♦ Alfa Test:

Lo sponsor utilizza il software presso il sito dello sviluppatore.

Software utilizzato in un ambiente controllato, con lo sviluppatore sempre pronto a correggere i bug.

♦ Beta Test:

Condotto presso il sito dello sponsor (lo sviluppatore non è presente)

Il software esegue un allenamento realistico nell'ambiente di destinazione