18. Testing del software (II)

Paola Barra
2022/2023

Cosa studieremo del testing

- Terminologia
  - Guasto, errore, difetto
- Attività di Test
  - Testing delle unità
  - **Testing di integrazione**
  - **Testing del sistema**

Panoramica

- Testing di integrazione
  - Big bang
  - Bottom up
  - Top down
  - Sandwich
- Testing del sistema
  - Funzionale
  - Prestazioni
- Testing di accettazione

Test di integrazione

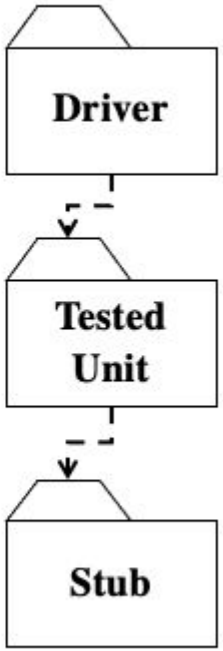
- L'intero sistema è visto come una collezione di sottosistemi (insiemi di classi) determinati durante la progettazione del sistema e degli oggetti
- Obiettivo: testare tutte le interfacce tra i sottosistemi e l'interazione dei sottosistemi
- La strategia di integrazione del testing determina l'ordine in cui i sottosistemi sono selezionati per il testing e l'integrazione

Perchè facciamo il test di integrazione?

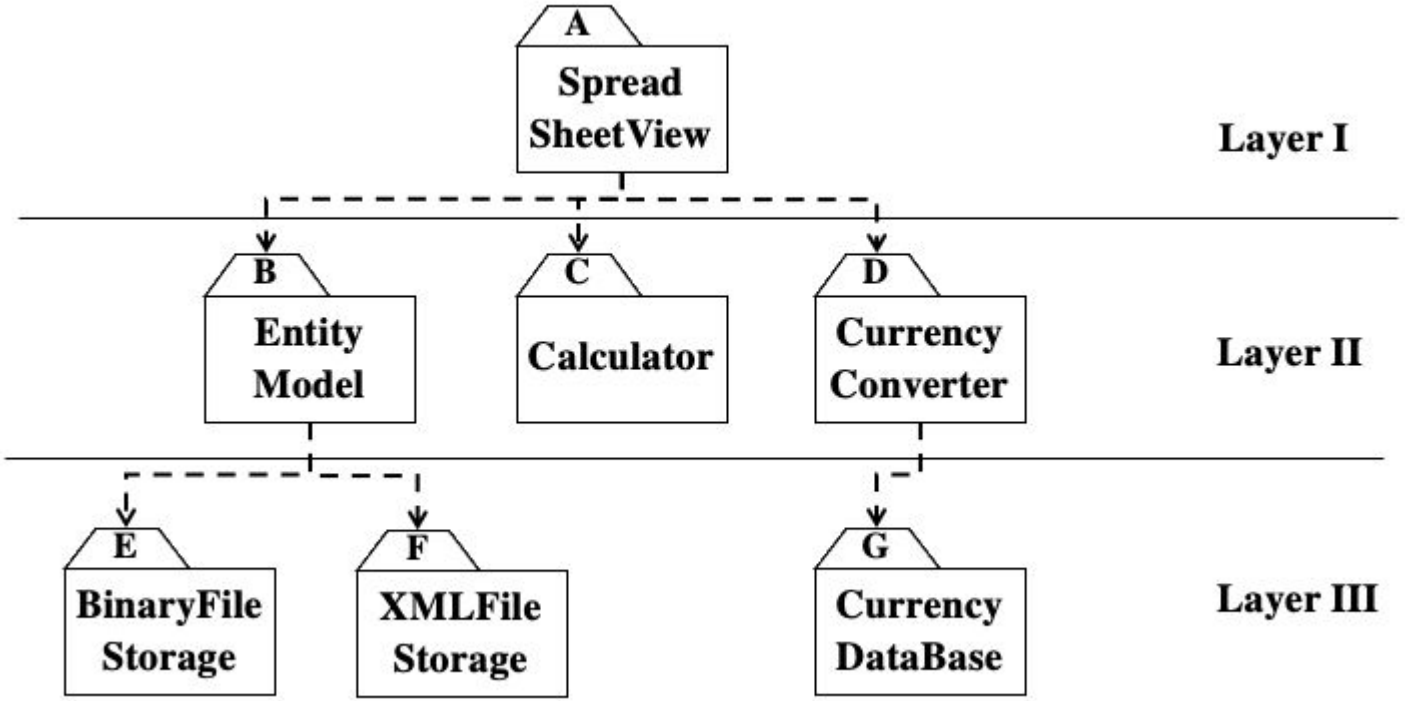
- I test delle unità valutano la singola unità in isolamento
- Molti guasti derivano da difetti nell'interazione tra i sottosistemi
- Spesso sono usati molti componenti off-the-shelf che non possono essere testati come unità
- Senza il testing d'integrazione il test del sistema sarebbe troppo costoso in termini di risorse di tempo
- I guasti che non sono scoperti nel testing di integrazione saranno scoperti dopo che il sistema è consegnato e ciò può essere molto costoso

Stub e Driver

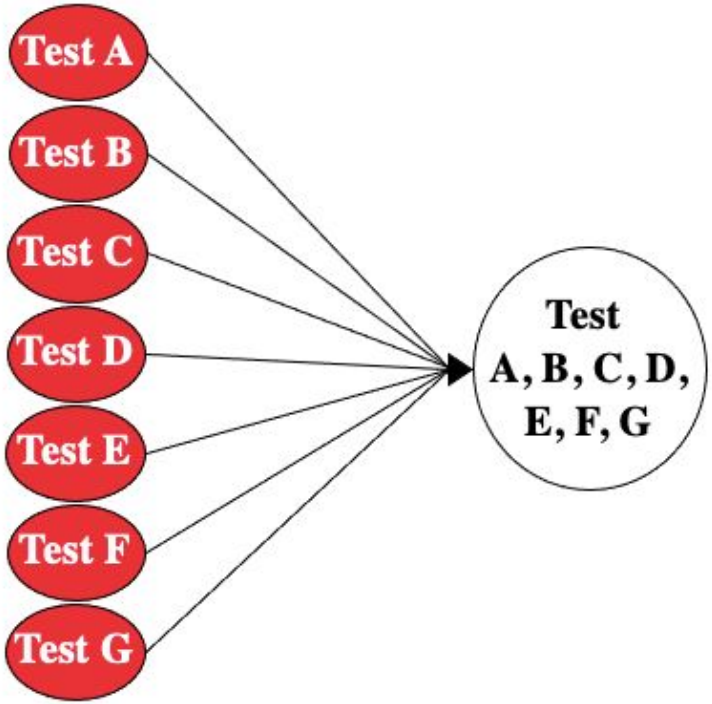
- Driver
  - Un componente che invoca la `TestedUnit`
  - Controlla i casi di test
- Stub
  - Un componente da cui `TestedUnit` dipende
  - Implementazione parziale
  - Restituisce valori fittizi

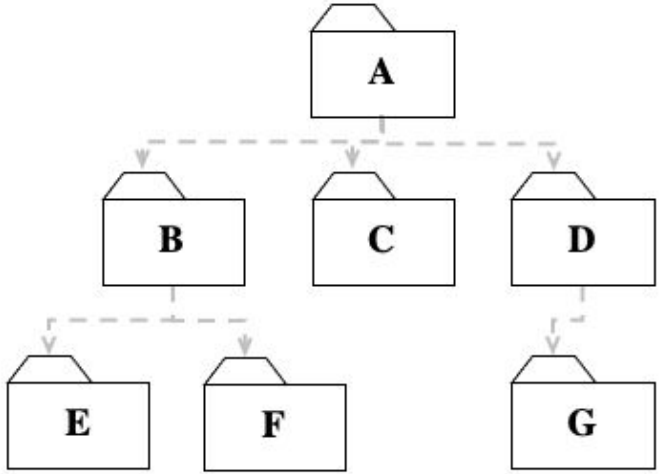


Esempio: un progetto a tre strati (spreadsheet)



Approccio Big Bang





Pro e contro del Big Bang

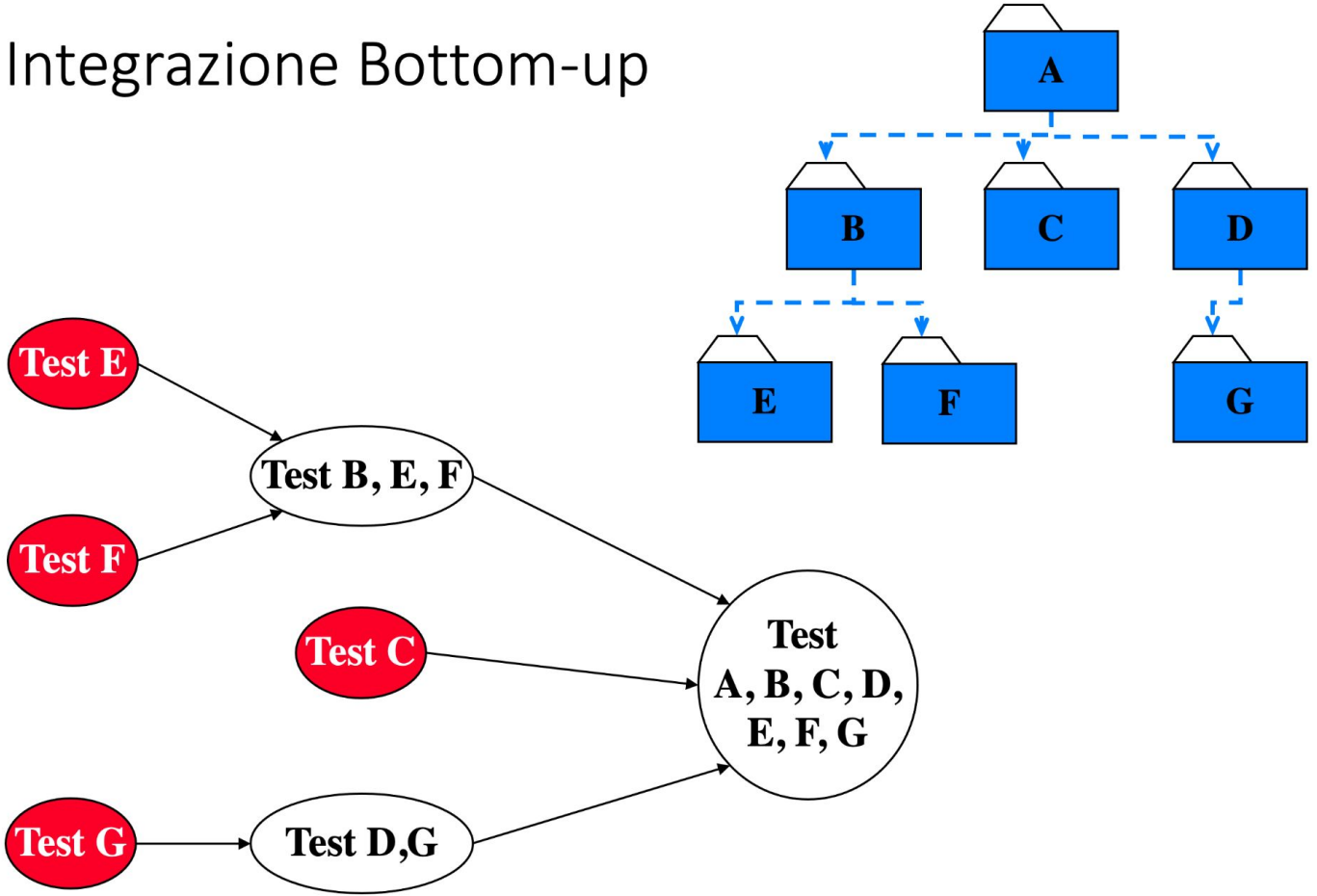
- Pro
  - Non sono necessari stub e drivers aggiuntivi

- Contro
  - Costoso:
    - se un test scopre un difetto, impossibile indicare con esattezza il componente(o combinazione di componenti) specifico che lo ha originato

Strategia di testing Bottom-up

- I sottosistemi nello strato più basso della gerarchia di chiamate sono testati individualmente
- Successivamente sono integrati i successivi sottosistemi che chiamano i sottosistemi testati precedentemente
- Ciò viene ripetuto fino a che sono inclusi tutti i sottosistemi
- Sono necessari i driver

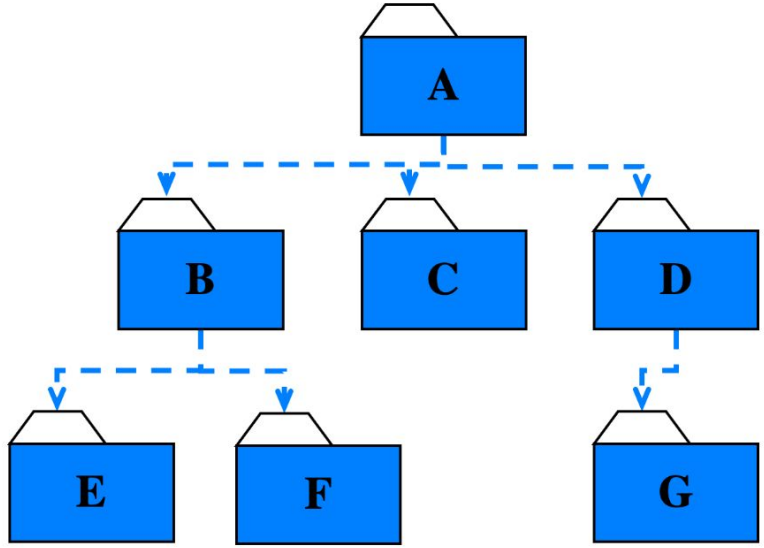
Integrazione Bottom-up

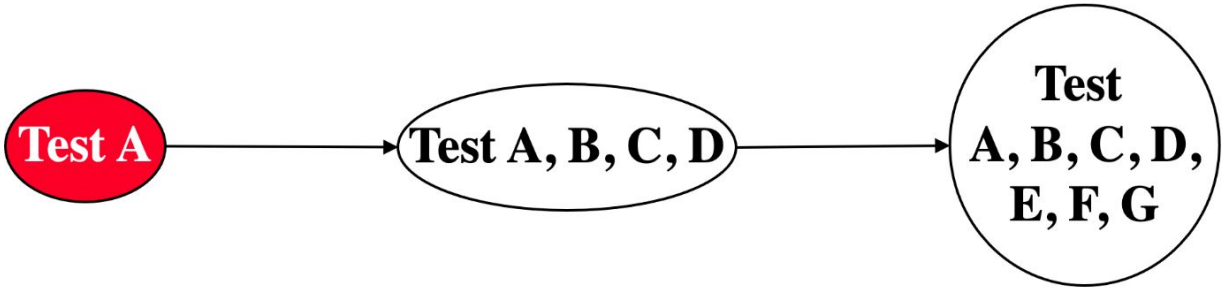


Strategia di testing Top-down

- Testa prima lo strato in cima alla gerarchia o il sottosistema controllore
- Successivamente combina tutti i sottosistemi che sono chiamati dai sottosistemi testati e testa la collezione risultante di sottosistemi
- Ripete il tutto fino a che tutti i sottosistemi sono incorporati nel test
- Sono necessari gli stub per fare il testing

Integrazione top-down





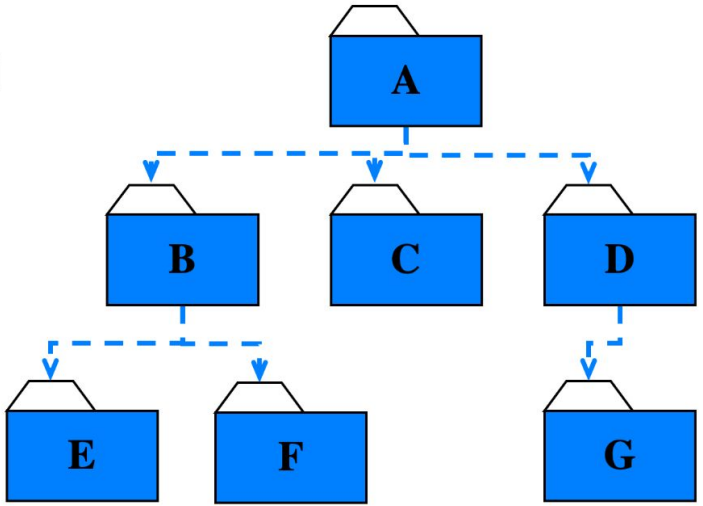
Pro e contro del testing Top-down

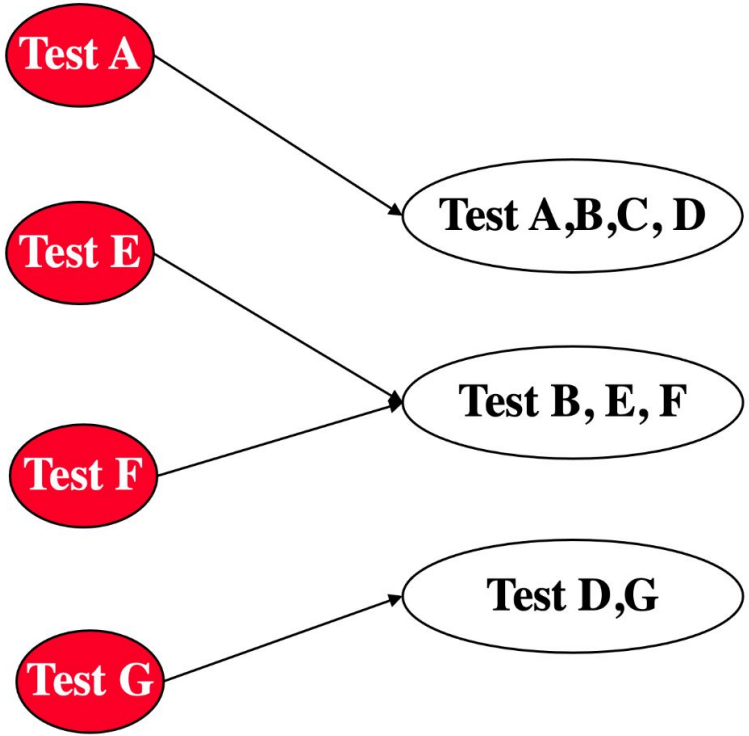
- Pro
  - I casi di test possono essere definiti in termini delle funzionalità del sistema (requisiti funzionali)
  - Non sono necessari driver
- Contro
  - Scrivere gli stub è difficile: gli stub devono permettere di testare tutte le possibili condizioni
  - Possono essere richiesti molti stub, specialmente se il livello più inferiore del sistema contiene molti metodi
  - Alcune interfacce non sono testate separatamente

Strategia di testing a Sandwich

- Combina la strategia top-down con la strategia bottom-up
- Il sistema è visto come un sistema a tre strati
  - Uno strato target nel mezzo
  - Uno sotto al di sopra del target
  - Uno strato al di sotto del target
- Il testing converge nello strato target

Strategia di testing a Sandwich





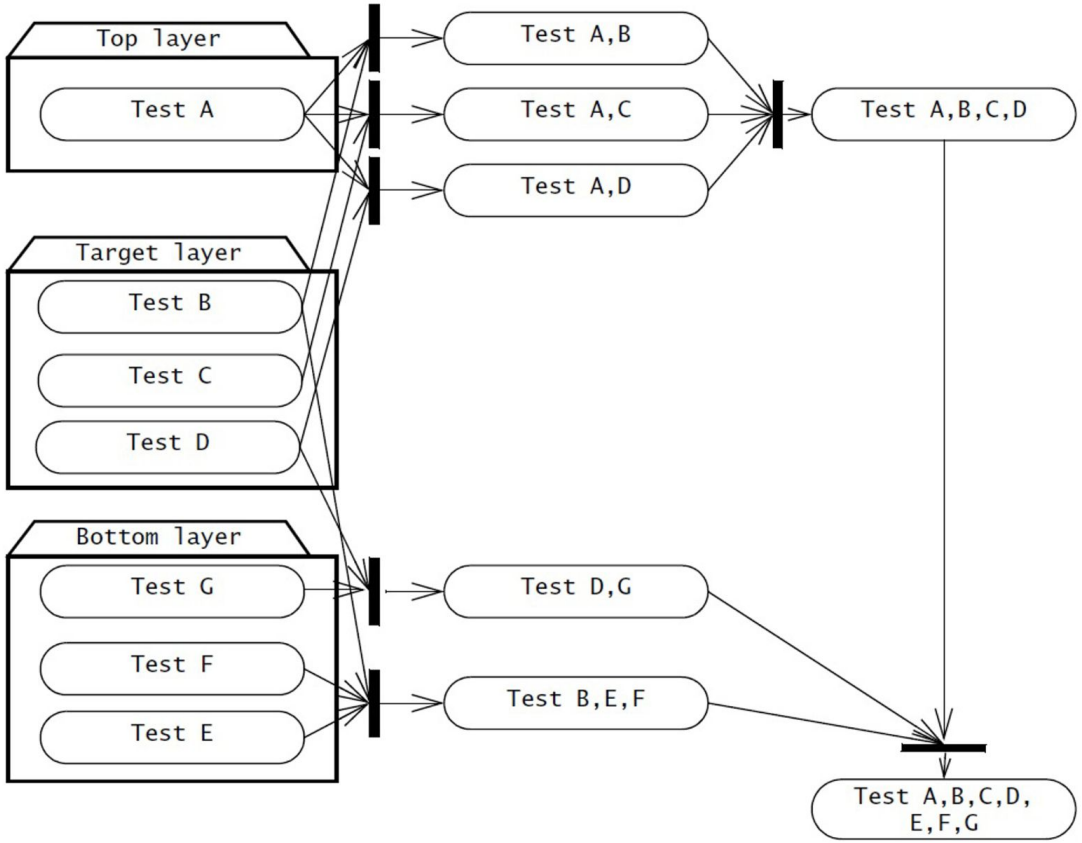
Pro e contro strategia Sandwich

- Pro
  - I test degli strati inferiore e superiore possono essere fatti in parallelo
  - Stub e driver non necessari per gli strati top e bottom poiché usano i componenti effettivi dello strato target
- Contro
  - non testa i singoli sottosistemi e le loro interfacce in modo completo prima dell'integrazione
- Soluzione: strategia di testing a sandwich modificata

Strategia Sandwich modificata

- Testa i tre starti singolarmente prima di combinarli in test incrementali l'uno con l'altro
- Test in parallelo
  - Strato centrale con driver e stub
  - Strato più alto con stub
  - Strato più basso con driver
- Test in parallelo
  - Lo strato più in alto accede allo strato centrale (lo strato in cima sostituisce i driver)
  - Lo strato più in basso è acceduto dallo strato centrale (lo strato più in basso sostituisce gli stub)

Testing a sandwich modificato



Passi nel testing di integrazione

1. Selezionare un componente da testare sulla base della strategia di integrazione. Test di unità di tutte le classi del componente
2. Mettere insieme il componente selezionato; eseguire qualsiasi correzione preliminare necessaria per rendere operativo il test di integrazione (driver, stub)
3. Testare i requisiti funzionali: definire i casi di test che esercitano tutti i casi d'uso con il componente selezionato
4. Testare la decomposizione del sistema: definire i casi di test che esercitano tutte le dipendenze
5. Testare i requisiti non funzionali: eseguire i test delle prestazioni
6. Tenere traccia dei casi di test e delle attività di testing
7. Ripetere i passi 1-7 fino al test dell'intero sistema

- L'obiettivo primario del test di integrazione è identificare i guasti con la configurazione (corrente) del componente

Testing del sistema

- Testing funzionale
  - Valida i requisiti funzionali
- Testing delle prestazioni
  - Valida i requisiti non funzionali
- Testing di accettazione

Testing funzionale

- Obiettivo: testare le funzionalità del sistema
- I casi di test sono progettati dal documento di analisi dei requisiti (meglio: manuale utente) e centrati intorno ai requisiti e le funzioni chiave (casi d'uso)
- Il sistema è trattato come una black-box
- I casi di test delle unità possono essere riusati ma devono anche essere sviluppati nuovi casi di test

Testing delle prestazioni

- Obiettivo: cercare di violare i requisiti non funzionali
- Testa come il sistema si comporta quando è sovraccaricato
  - Possono essere identificati colli di bottiglia? (primi candidati per la riprogettazione nella iterazione successiva)
- Cercare ordini di esecuzione insoliti
  - Chiamare una receive() prima di una send()
- Controllare la risposta del sistema rispetto a grandi quantità di dati
  - Se si suppone che il sistema gestisca 1000 elementi, provarlo con 1001 elementi
- Quanto è la quantità di tempo impiegato in casi d'uso differenti?
  - I casi d'uso tipici sono eseguiti in tempi consoni?

Tipi di test delle prestazioni

- Testing di stress
  - Stressa i limiti del sistema
- Testing di volume
  - Testa cosa accade se sono gestite grandi quantità di dati
- Testing di configurazione
  - Testa le varie configurazioni di hardware e software
- Testing di compatibilità
  - Testa la compatibilità retroattiva con sistemi esistenti
- Testing di temporizzazione
  - Valuta i tempi di risposta e il tempo per eseguire una funzione
- Testing di sicurezza
  - Cerca di violare i requisiti di sicurezza
- Test ambientale
  - Testa la tolleranza al caldo, umidità, movimento
- Testing di qualità
  - Testa l'affidabilità, manutenibilità e la disponibilità
- Testing di recupero
  - Testa la risposta del sistema alla presenza di errori o perdite di dati
- Testing dei fattori umani
  - Test con gli utenti finali

Testing di accettazione

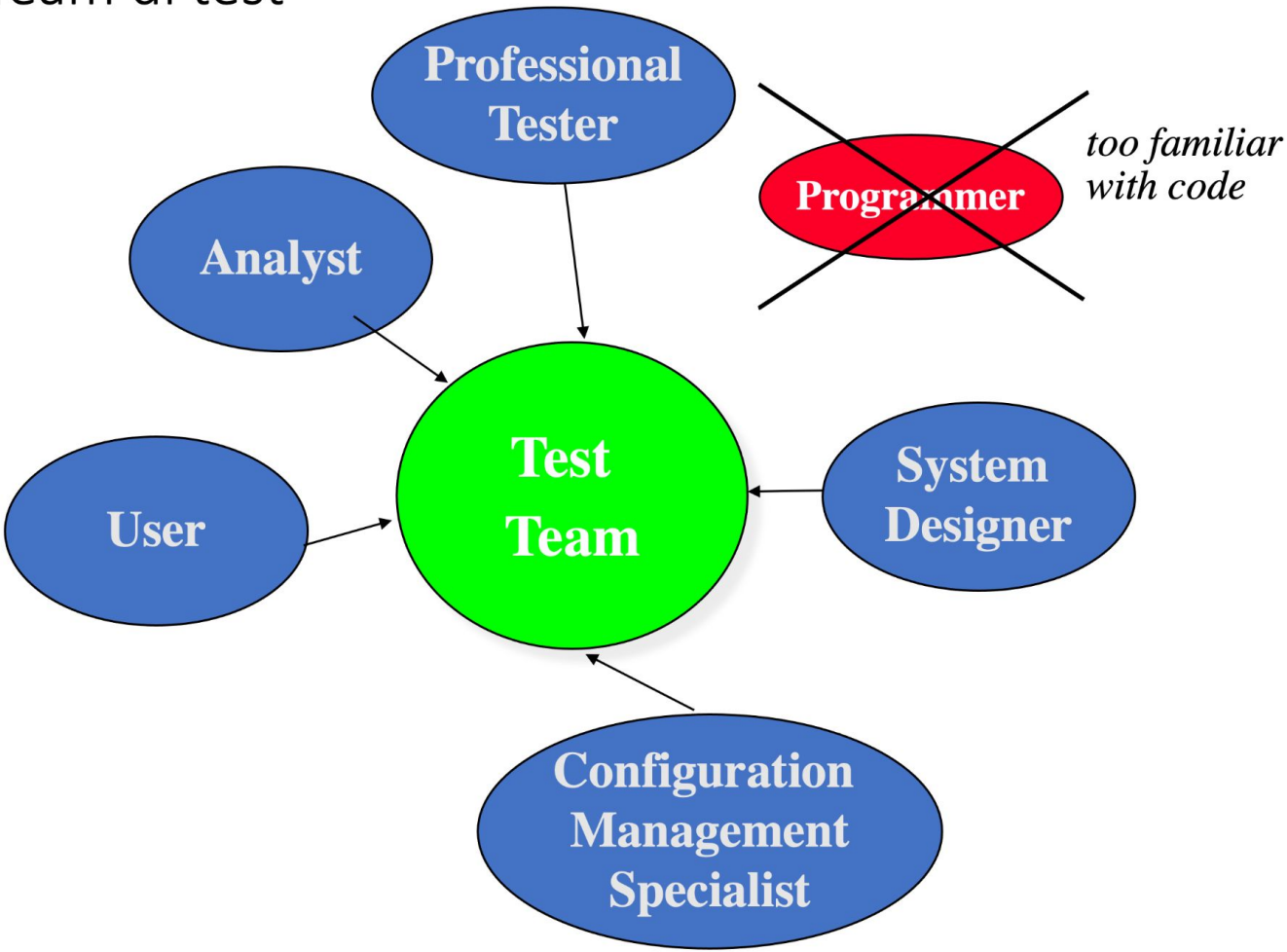
- Obiettivo: dimostrare che il sistema è pronto per un uso operativo
  - La scelta dei test è fatta dal cliente
  - Possono essere presi molti test dal testing di integrazione
  - Il test di accettazione è eseguito dal cliente e non dallo sviluppatore

- Alpha test:
  - Il client usa il software nell'ambiente dello sviluppatore
  - Il software è usato con impostazioni controllate in modo che lo sviluppatore è sempre pronto a fissare i bug
- Beta test:
  - Condotto nell'ambiente del cliente (lo sviluppatore non è presente)
  - Il software ottiene un allenamento realistico nell'ambiente di destinazione

Il testing ha molte attività

- Stabilire gli obiettivi del test
- Progettare i casi di test
- Scrivere i casi di test
- Testare i casi di test
- Eseguire i test
- Valutare i risultati del test
- Cambiare il sistema
- Fare il testing di regressione

Team di test


Professional Tester
Programmer
too familiar with code
Analyst
Test Team
User
System Designer
Configuration Management Specialist

I 4 passi del testing

- Selezionare cosa deve essere testato
  - Analisi: completezza dei requisiti
  - Progettazione: coesione
  - Implementazione: codice sorgente
- Decidere come fare il testing
  - Review o ispezione del codice
  - Dimostrazioni (progetto per contratto)
  - Balck-box, white-box
  - Selezionare la strategia di testing di integrazione (big-bang, bottom-up, top-down, sandwich)
- Sviluppare i casi di test
  - Un caso di test è un insieme di dati di test o situazioni che saranno usati per esercitare l'unità (classe, sottosistema, sistema) da testare o sull'attributo da misurare
- Creare l'oracolo di test
  - Un oracolo contiene i risultati predetti per un insieme di casi di test
  - L'oracolo del test deve essere scritto prima che il testing effettivo abbia luogo

Guida per la selezione dei casi di test

- Usare la conoscenza dell'analisi *sui requisiti funzionali* (black-box testing):
  - Casi d'uso
  - Dati input attesi
  - Dati input non validi
- Usare la conoscenza della progettazione *sulla struttura del sistema*, algoritmi, strutture dati (white-box testing):
  - Strutture di controllo
    - Condizioni di test, cicli, ...
  - Strutture dati
    - Testare campi di record, array, ...
- Usare *la conoscenza dell'implementazione* su algoritmi e strutture dati:
  - Forzare divisioni per zero
  - Se il limite superiore di un array è 10, usare 11 come indice

Riepilogando

- Il testing è ancora un'arte nera, anche se sono disponibili molte regole euristiche
- Il testing consiste di
  - Testing Unità
  - Testing Integrazione
  - Testing Sistema
    - Testing accettazione
- Il testing ha il proprio ciclo di vita