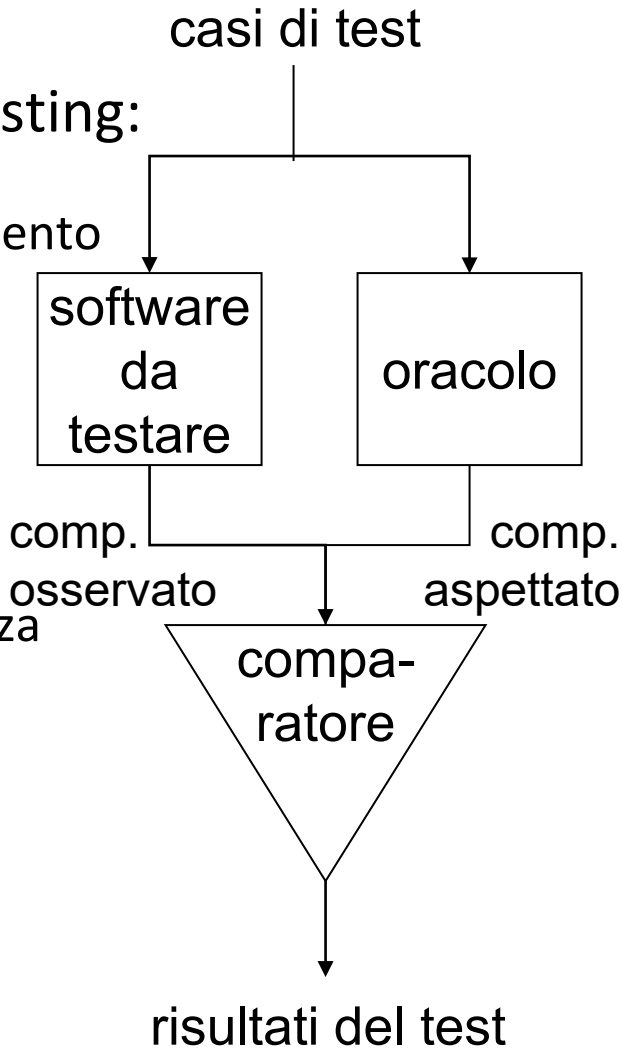
Testing

Prof. Filippo Lanubile

Testing e oracolo

- Condizione necessaria per iniziare il testing:
  - conoscere il comportamento aspettato per poterlo confrontare con il comportamento osservato
- L'**oracolo** è un'entità che conosce il comportamento aspettato per ogni caso di test
- Oracolo umano (**test manuale**)
  - si basa su specifiche scritte o su conoscenza tacita
- Oracolo automatico
  - specifiche formali (**test automatico**)
  - stesso software ma sviluppato da altri
  - versione precedente del software (**test di regressione**)



Concetti base

- Un **test** è l'esecuzione di un programma guidata da uno o più casi di test
  - Un **test ha successo** se non provoca un malfunzionamento
  - Un **test fallisce** se provoca un malfunzionamento
- Un **caso di test** è una descrizione delle attività necessarie per confrontare un comportamento aspettato con quello osservato
  - Comprende un sottoinsieme dei possibili dati di ingresso e l'output aspettato
- Una **test suite** è formata dall'insieme dei casi di test

Problema della selezione dei casi di test

- Un programma è **corretto** se è corretto per ogni dato di ingresso
- Un **test è ideale** se il successo del test implica la correttezza del programma
- Un **test esaustivo** è un test che contiene tutti i dati di ingresso al programma
- Un test esaustivo è un test ideale

  ma

  un test esaustivo non è pratico e quasi sempre non è fattibile
  - **Tesi di Dijkstra**: «Il test di un programma può essere usato per mostrare la presenza di bug, ma mai per mostrare la loro assenza»
- Obiettivo realistico: **selezionare casi di test che approssimano un test ideale**

  ovvero trovare un ragionevole compromesso tra
  - numero di malfunzionamenti scoperti
  - numero di casi di test (dimensione della test suite)

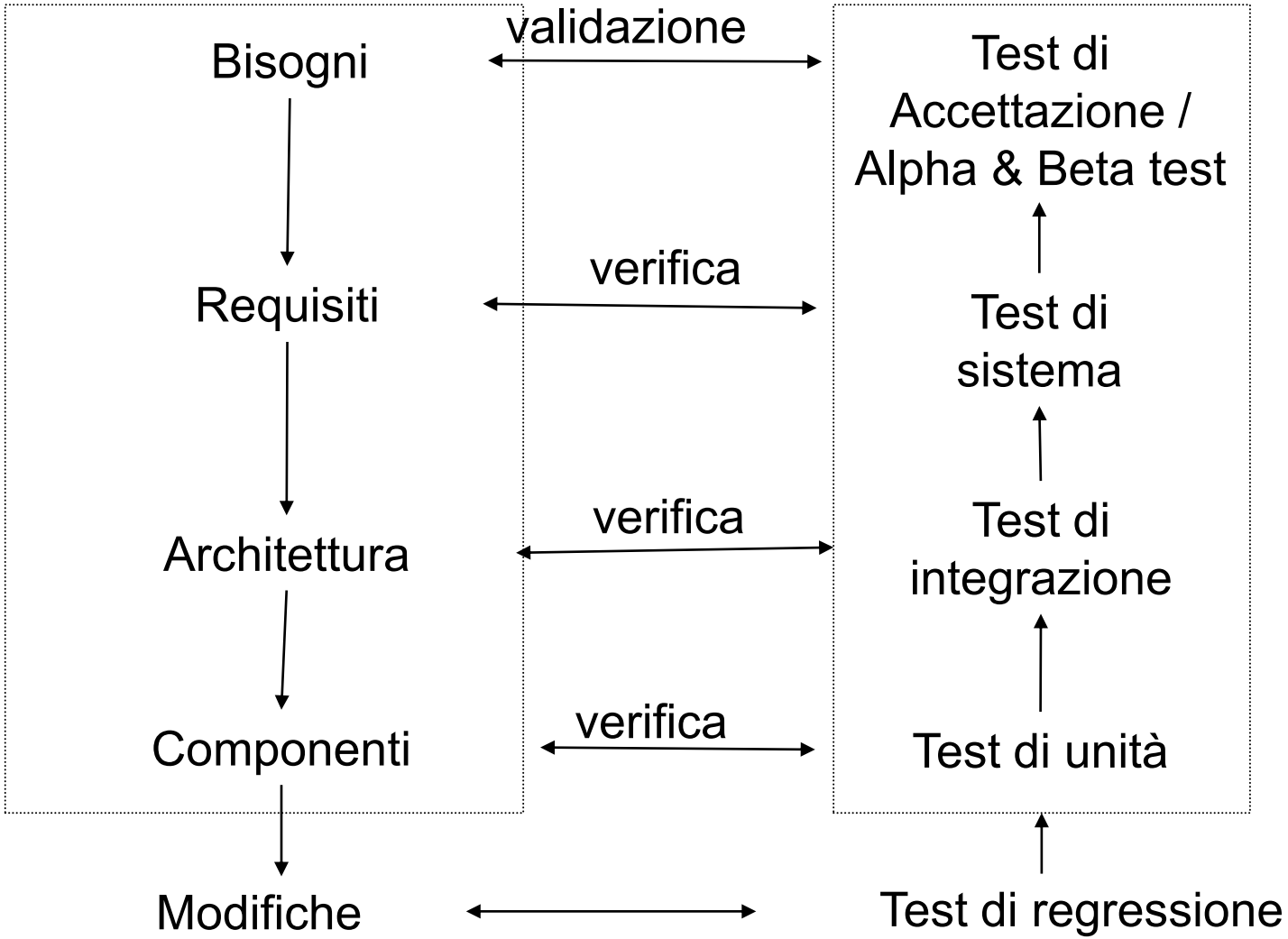
Terminazione del testing

Quando un programma si può considerare testato a sufficienza?

- Criterio temporale
  - Periodo di tempo predefinito
- Criterio di costo
  - Sforzo allocato predefinito
- Criterio di copertura
  - Percentuale predefinita degli elementi di un modello di programma

    Es: 100% di copertura delle linee di codice
- Criterio statistico
  - MTBF (mean time between failure) predefinito e confrontato con un modello di affidabilità esistente

Granularità dei test



Test di accettazione

- Il sistema è collaudato dal committente nel proprio ambiente di esecuzione
- Due casi:
  - Le caratteristiche del sistema sono conformi alle specifiche
    - Il sistema è accettato
  - Si rileva una deviazione dalle specifiche
    - Viene stilato un elenco dei problemi
    - Si contratta con il committente la risoluzione dei problemi

Alpha e Beta test

- Tipi di test eseguiti quando il software sviluppato è un prodotto che sarà utilizzato da molti utenti
- Alpha test
  - Una rosa ristretta di utenti utilizza il software e segnala i problemi riscontrati
  - Si concentra sui requisiti funzionali
- Beta test
  - Un platea allargata di utenti utilizza il software e segnala i problemi riscontrati
  - Si concentra sui requisiti non funzionali

Test di sistema

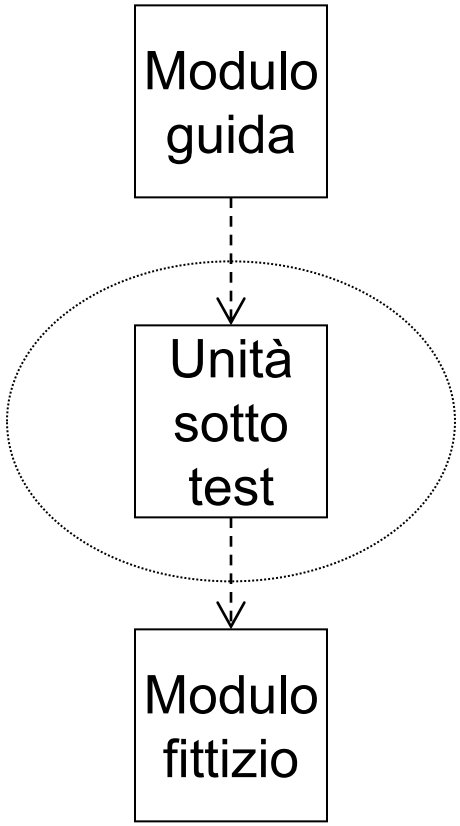
- Il sistema è collaudato nel suo insieme
- Basato sulle specifiche dei requisiti
  - Requisiti funzionali:
    - **Test funzionale**
  - Requisiti non funzionali:
    - **Test di performance**
    - **Test di usabilità**
    - **Test di sicurezza**
- Spesso effettuato da personale specializzato

Test di integrazione

- Lo scopo del test di integrazione è verificare se unità differenti,
  sviluppate e testate indipendentemente,
  lavorino insieme in modo appropriato
  - Si cerca di scoprire malfunzionamenti dovuti alla interazione tra le unità
- Approcci
  - Big-bang
    - Prima si collauda ogni unità singolarmente
    - poi si collauda il sistema integrato
  - Incrementale
    - Le unità sono progressivamente integrate nel sistema e collaudate

Test di unità

- Le unità sono il risultato di un lavoro individuale, isolato dal resto del sistema
  - per programmi OO: singole classi
- Costruzione di
  - moduli “guida” (**driver**):
    - attivano l’unità sotto test
  - moduli “fittizi” (**stub**):
    - sono utilizzati dall’unità sotto test
- Disponibilità di software per l’automazione del test di unità
  - per programmi Java: **JUnit**



Unit Test: criteri di selezione dei casi di test

Criteri black-box (o funzionali)

I casi di test sono selezionati esclusivamente a partire dalle specifiche

- Suddivisione in classi di equivalenza
- Analisi dei valori limite
- *State-based testing*
- …

Criteri white-box (o strutturali)

I casi di test sono selezionati conoscendo la struttura interna del software

- **Criteri basati sulle istruzioni**
  - **% di copertura**
- Criteri basati sul flusso di controllo
- Criteri basati sul flusso dei dati
- Criteri basati sul binding (*polymorphism testing*)
- *Mutation testing*
- …

Sono spesso usati in modo complementare

Suddivisione in classi di equivalenza

- Il dominio dei dati di ingresso è suddiviso in classi di casi test in modo tale che se il programma è corretto per un caso di test si può dedurre ragionevolmente che è corretto per ogni caso di test in quella classe
- Una **classe di equivalenza** rappresenta un insieme di stati validi o non validi per una condizione di ingresso
  - **intervallo di valori**
    una classe valida per valori nell'intervallo, una non valida per valori inferiori al minimo, e una non valida per valori superiori al massimo
  - **valore specifico**
    una classe valida per il valori specificato, una non valida per valori inferiori, e una non valida per valori superiori
  - **elemento di un insieme discreto**
    una classe valida per ogni elemento dell'insieme, una non valida per un elemento non appartenente
  - **valore booleano**
    una classe valida per gli elementi per il valore TRUE, una classe non valida per il valore FALSE

Selezione dei casi di test dalle classi di equivalenza

- Ogni classe di equivalenza deve essere coperta da almeno un caso di test
  - Un caso di test per ogni classe non valida
  - Un caso di test per ogni classe valida, possibilmente condiviso da più classi valide

Esempio: funzione di calcolo del fattoriale di un numero

- Classi di equivalenza valide:
  - numeri interi positivi TC1. n= 4 → *n!=24*
- Classi di equivalenza non valide:
  - numeri interi negativi TC2. n= −4 → *input non valido*

Analisi dei valori limite

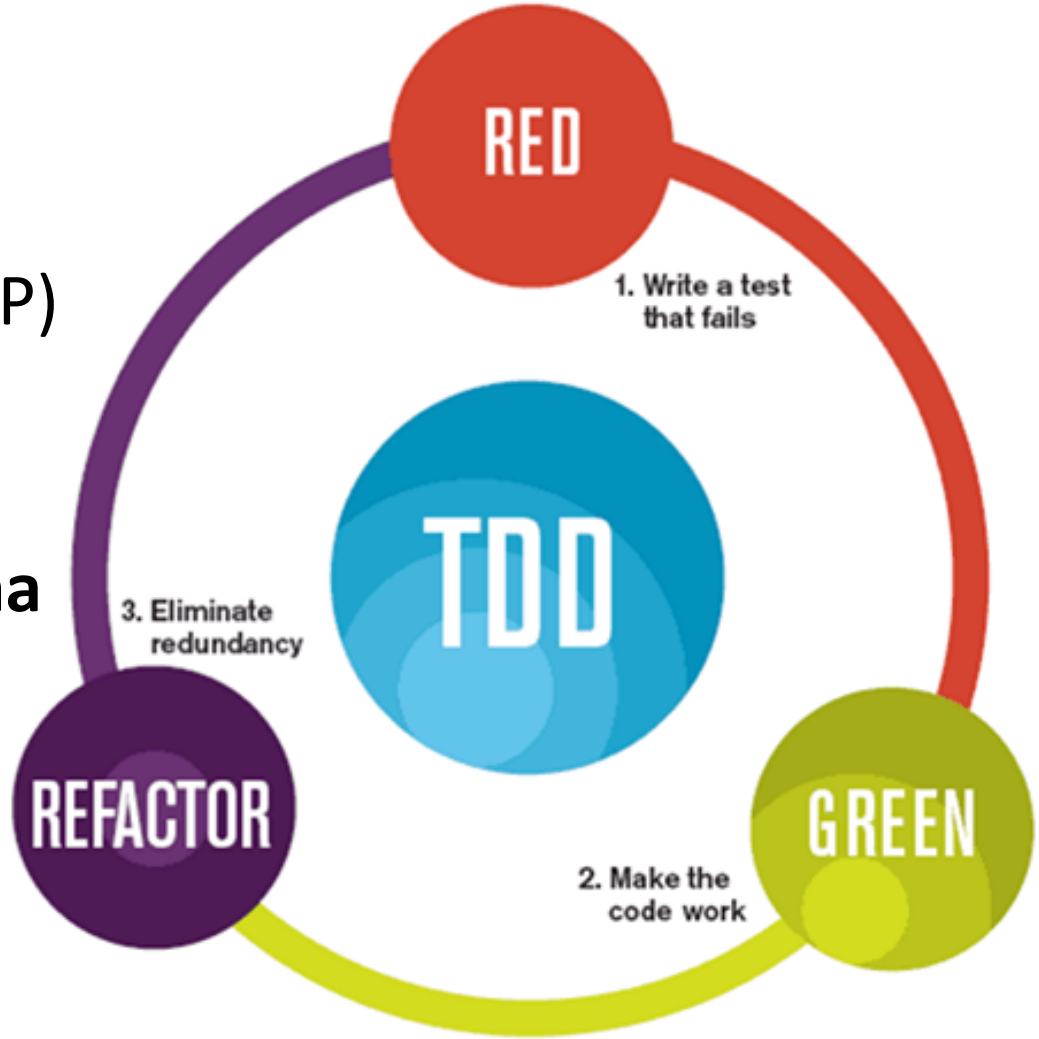
- I casi di test che esplorano condizioni limite spesso rilevano la presenza di malfunzionamenti
- Le condizioni limite
  - sono direttamente agli estremi
  - immediatamente al di sopra degli estremi
  - immediatamente al di sotto degli estremi
    - di classi di equivalenza d'ingresso
    - classi di equivalenza di uscita

Esempio: funzione di calcolo del fattoriale di un numero

- TC3. n=1 → *n!=1*
- TC4. n= 0 → *n!=1*
- TC5. n=-1 → *input non valido*

Test Driven Development

- Pratica dell'Extreme Programming (XP) per lo Unit Test
- I test automatici sono scritti **prima** del codice



The mantra of Test-Driven Development (TDD) is "red, green, refactor."