Università degli Studi della Campania Vanvitelli
Laurea Magistrale in Ingegneria Informatica

Ingegneria del Sofware e Sistemi Informativi

Introduzione al Testing

a.a. 2019-2020

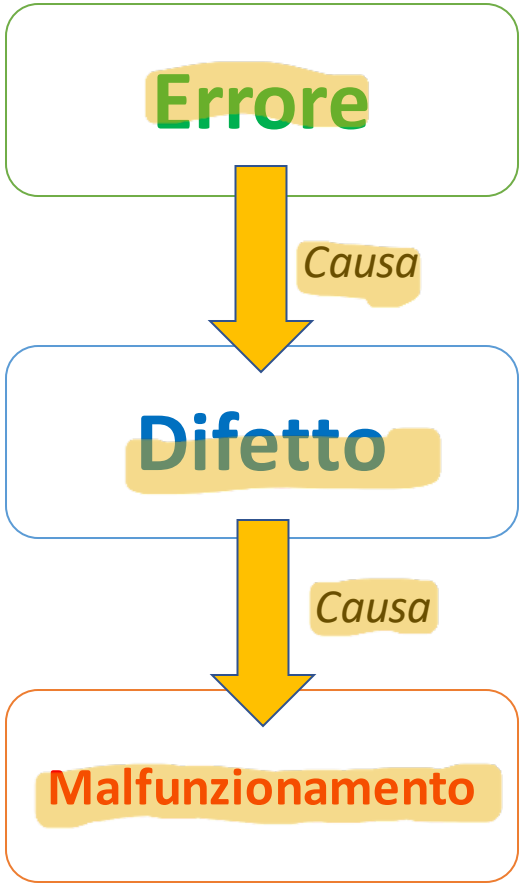
Docente: Prof. Antonio Esposito

Sommario

- Terminologia generale
  - Definizioni
  - Concetti di verifica e validazione
  - Controlli statici e dinamici
- I livelli di test:
  - Test di Unità
  - Test Integrazione
  - Test di sistema
  - Controlli di accettazione

Malfunzionamento, difetto ed errore

- Errore
  - Causa (anche umana) e origine del difetto
  - Può non essere rilevante
  - Puntuale, lascia una traccia
- Difetto (bug)
  - Introdotto da un errore
  - Magari latente, da eliminare
  - Può essere causa di un malfunzionamento
  - Permanente, ma può causare il funzionamento solo in specifiche situazioni
- Malfunzionamento
  - o guasto (failure) è un funzionamento di un programma diverso da quanto previsto dalla sua specifica
  - Prodotto da un difetto
  - Di solito è\arreca un danno



Controlli Interni ed Esterni

Pro: so bene dove andare a misurare
Contro: non è detto che se scrivo un test che metta sotto stress il codice

- **Interno** Io che ho scritto codice creo i test da eseguire.
  - Ogni attività di controllo che proviene dalla stessa organizzazione impegnata nel processo di sviluppo software
  - Estremamente mirati e con obiettivi precisi
  - Ha sicuramente a disposizione tutte le informazioni possibili
  - α-test
- **Esterno**
  - condotto da personale estraneo all'organizzazione che ha in carico lo sviluppo software.
  - Tipici delle situazioni in cui il prodotto è realizzato su commissione
  - Si sovrappongono ai controlli interni
  - Gestiti da Committente o terza parte incarica
  - β-test
    - di solito persone che hanno esperienza, ma anche persone qualunque.

Verifica e Validazione

- Verifica Vedo se processo di sviluppo è andato bene rispetto a quello che ho capito di fare.
  - Stiamo realizzando correttamente il prodotto?
  - Controllo di qualità delle attività svolte durante una fase dello sviluppo
  - Il prodotto ottenuto al termine di una fase è congruente con il semilavorato avuto come punto di partenza di quella fase?
- Validazione
  - Stiamo realizzando il prodotto corretto?
  - Controllo di qualità del prodotto rispetto ai requisiti del committente
  - Confrontare il risultato di una fase del processo di sviluppo con i requisiti del prodotto

↳ Nell'agile ogni iterazione

Controlli Statici e Dinamici

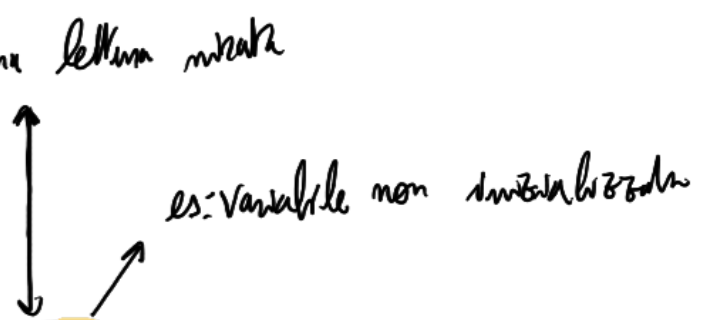
- Controlli statici non prevedono l'esecuzione del codice
  - Usati specificamente per attività di verifica
  - Eseguiti sulle componenti
- Controlli dinamici (prove/testing) prevedono l'esecuzione del codice
  - Usati per attività di verifica o di validazione
  - Eseguiti sulle componenti o sul sistema

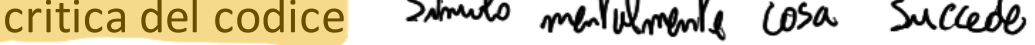
Controlli Statici

- Verifica senza esecuzione del codice
- Convenienza economica
  - Costi dipendenti dalle dimensioni del codice
  - Bassi costi di infrastruttura (desk-check)
  - Buona prevedibilità dei risultati
- Metodi manuali
  - Basati sulla lettura del codice (desk-check)
  - Più comunemente usati
  - Più o meno formalmente documentati
- Metodi formali o Statici supportati da strumenti
  - Model checking
  - Esecuzione simbolica

Desk-Check

- Inspection
  - Obiettivi ristretti e prestabiliti
    - Rivelare la presenza di difetti
    - Eseguire una lettura mirata del codice
  - Strategia
    - Focalizzare la ricerca su aspetti ben definiti (error guessing)
      - È plausibile che un controllo ottenga migliori risultati se concentrato su un solo tipo di difetti
  - Più rapido
- Walkthrough
  - Obiettivo:
    - Rivelare la presenza di difetti
    - Eseguire una lettura critica del codice
  - Strategia
    - Percorrere il codice simulandone l'esecuzione
    - Porta generalmente a scoprire difetti originati da errori algoritmici
  - Più collaborativo





Metodi Formali

- Tecniche per descrivere e analizzare sistemi
  - Basati su un formalismo matematico (logica, automi, teoria dei grafi)
- Una tecnica di verifica formale include:

1. Una metodologia per modellare sistemi:
   - rappresentare un sistema in termini di oggetti matematici, astraendo e semplificandone la descrizione

2. Un linguaggio di specifica, per descrivere le proprietà del sistema modellato.
   - Ex: logica temporale

3. Un metodo di verifica:
   - tecniche di analisi formale per verificare se un sistema soddisfa la sua specifica.

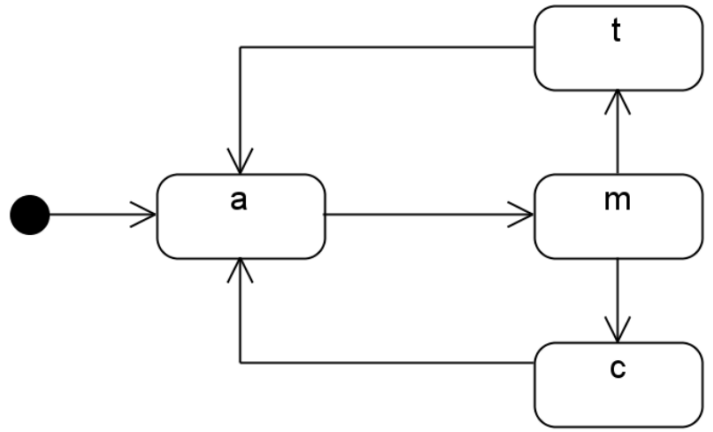
Model Checking

descrivo software come una macchina che fa cose corrette

- Rappresentazione della specifica del comportamento del sistema come un grafo
- Sistema di transizioni, formato da nodi e archi (per esempio una macchina a stati finiti).
- I nodi rappresentano gli stati di un sistema, gli archi rappresentano le transizioni da uno stato ad un altro.
- Un insieme di proposizioni atomiche è associato ad ogni nodo per descrivere le proprietà fondamentali che caratterizzano un punto di esecuzione.

Model Checking

- Macchina a stati di una macchinetta che eroga caffè.
- Inizialmente in stato di attesa (a)
- Dopo l'inserimento di una moneta si passa a uno stato "moneta inserita" (m)
- Infine è consegnato un caffè (c) o un tè (t)



Model Checking

- Quantificatori sui percorsi
  - **A** φ – **A**ll: φ deve essere valido su tutti I percorsi a partire dallo stato corrente.
  - **E** φ – **E**xists: Esiste almeno un percorso a partire dallo stato corrente per cui φ è valido.
- Quantificatori sui percorsi
  - **G** φ – **G**lobally: φ deve essere valido su tutto il sottopercorso successivo.
  - **F** φ – **F**inally: φ deve essere valido in almeno un punto del percorso successivo.

- È sempre vero che se è stata inserita una moneta, allora in tutti i cammini del modello si incontra prima o poi uno stato nel quale viene consegnato un tè o uno stato nel quale viene consegnato un caffè
  - AG [m →AF (c OR t)]
- Non vengono mai consegnati allo stesso tempo sia il tè che il caffè
  - AG ~ (c AND t)
- È sempre vero che se è stata inserita una moneta, allora in tutti i cammini del modello si incontra prima o poi uno stato nel quale viene consegnato un tè
  - AG [m →AF t]

Algoritmo di Clarke, Emerson e Sistla

- Model checking (per la logica **CTL** Computation tree logic)
- Data una specifica M e una formula f, si verifica se M soddisfa f ($M \models f$), cioè, se M è un modello per f
- Se la risposta è negativa, di solito viene prodotto un contro-esempio, ovvero, un cammino nel modello che non verifica la (sotto)formula
  - Nel nostro caso, il tentativo di verifica della proprietà 3 fallisce restituendo il controesempio a.m.c.
- L'algoritmo attraversa lo spazio degli stati etichettandoli con sotto-formule, ed è quadratico rispetto al numero degli stati (ma può essere reso lineare)
- **Limitazioni**
  - Occorre avere un modello finito ma preciso
  - Esplosione degli stati, in particolare per sistemi paralleli

Descrivo il modello, verifico se si verificano
mai situazioni non previste, altrimenti c'è
un problema di codice.

Esecuzione Simbolica

- L'esecuzione di un programma viene simulata
  - Si rappresentano insiemi di possibili dati d'ingresso con simboli algebrici, piuttosto che con valori numerici reali

```
public static int abs (int i) {
 int r;                  //L1
 if (i<0)
       r = -i;           //L2
 else
       r = i;            //L3
 return r;               //L4
 }
```

passi	stato	note
L1	i = l	è un intero
L2	i = l & (l < 0 & r = - l)	
L3	i = l & (0 ≤ l & r = l)	
L4	i = l & ((l < 0 & r = - l) | (0 ≤ l & r = l))	I rami del condizionale si ricongiungono

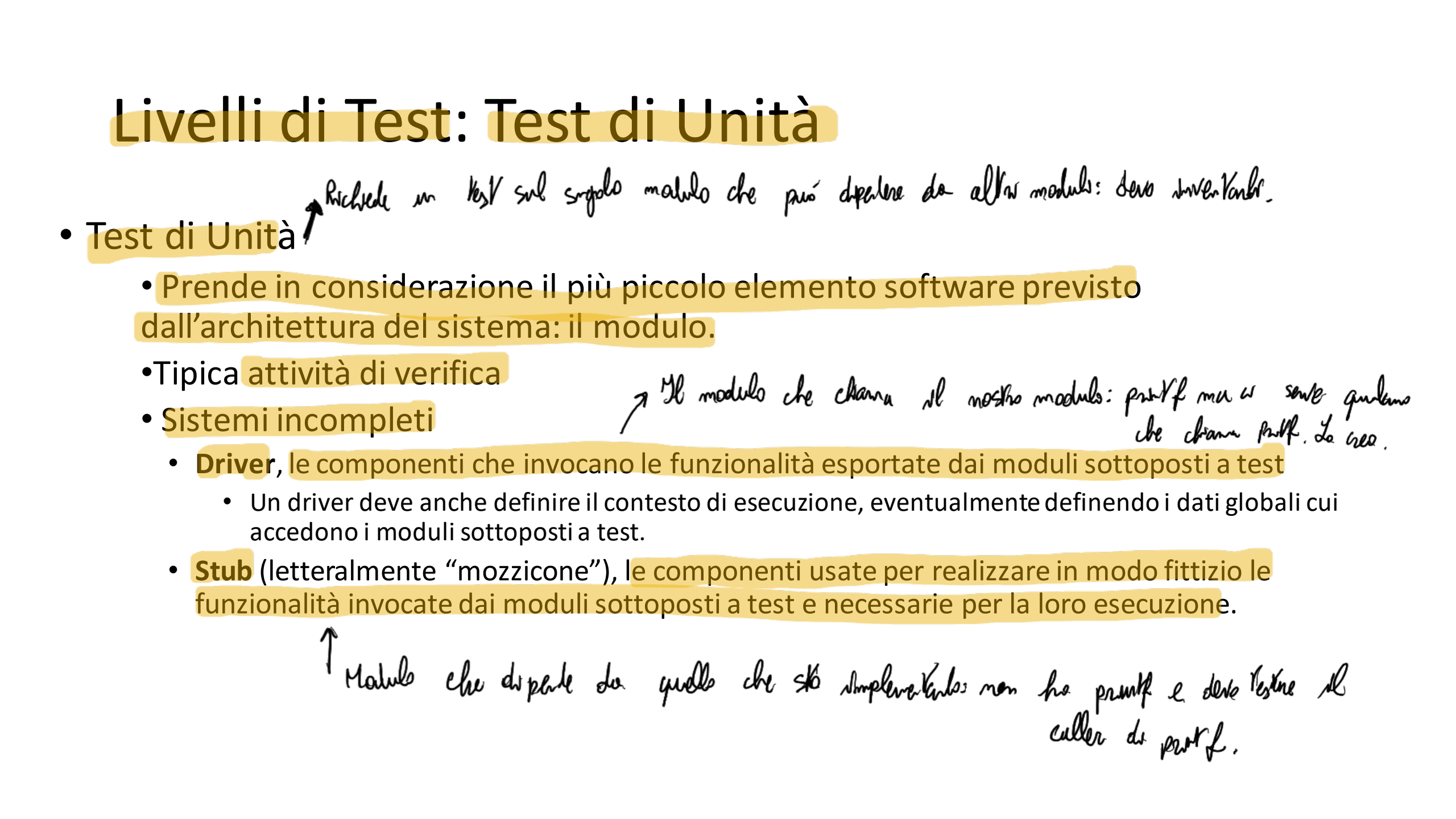
- L'esecuzione simbolica simula l'esecuzione, associando a ogni stato un'espressione che ne descrive le proprietà, in funzione dei valori assunti dalle variabili
- In generale, la valutazione dei predicati di questo genere è un problema **non decidibile**. In tanti casi però le espressioni simboliche possono essere semplificate, ed i risultati come quello appena visto possono essere ottenuti da un theorem prover → dimostrano l'esattezza dei teoremi: se sono una serie di regole logiche funziona di Kulik?

Verifica Dinamica o Testing

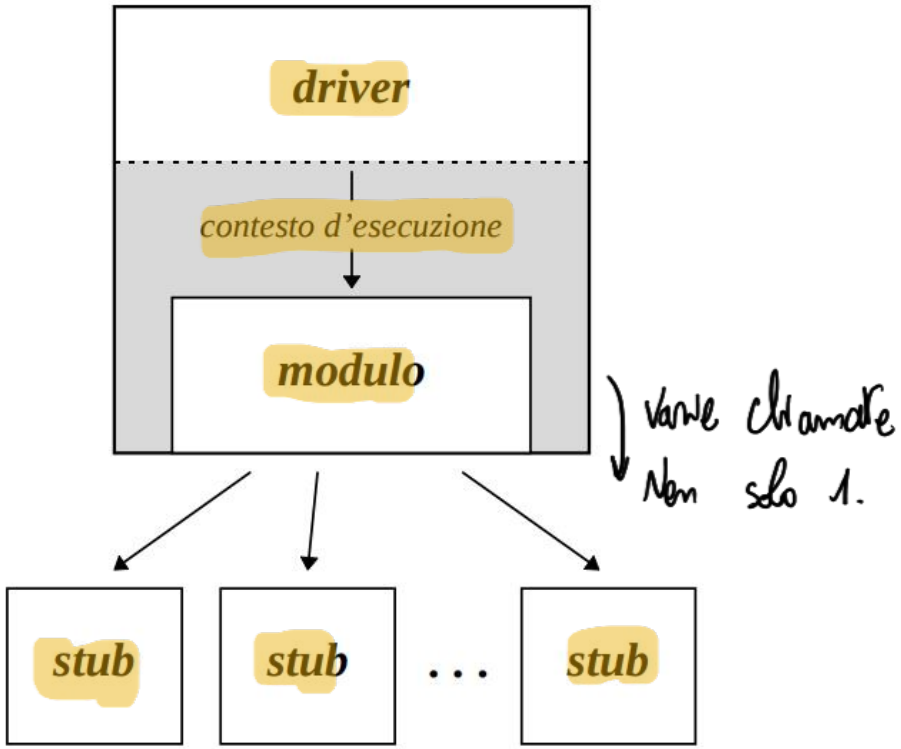
- Richiede l'esecuzione di un programma in un ambiente e con dati di ingresso controllati
  - Comprende la registrazione di tutti i dati riguardanti l'esecuzione e interessanti ai fini dei fattori di qualità che si vogliono valutare
  - Analisi dei dati e il loro confronto con i requisiti.
- Tecnicamente, il test consiste nell'eseguire un programma al fine di scoprire malfunzionamenti che denuncino l'esistenza di difetti.
- Ripetibile
- Si compone di più fasi:
  - Progettazione (input, ma non solo)
  - Definizione ambiente di test
  - Esecuzione del codice
  - Analisi dei risultati (output ottenuto con l'esecuzione vs output atteso)
  - Debugging

Livelli di Test: Test di Unità

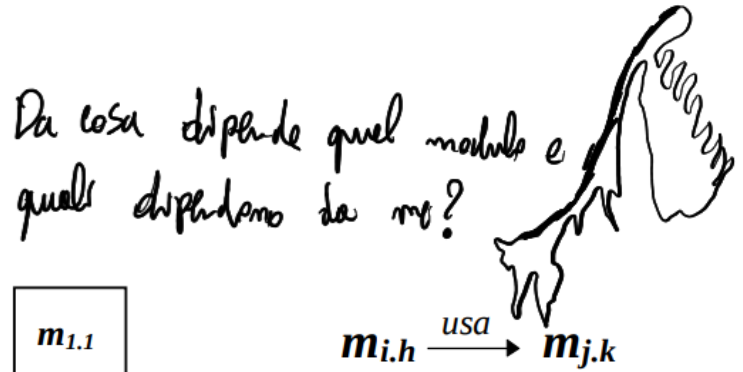
- Test di Unità
  - Prende in considerazione il più piccolo elemento software previsto dall'architettura del sistema: il modulo.
  - Tipica attività di verifica
  - Sistemi incompleti
    - **Driver**, le componenti che invocano le funzionalità esportate dai moduli sottoposti a test
      - Un driver deve anche definire il contesto di esecuzione, eventualmente definendo i dati globali cui accedono i moduli sottoposti a test.
    - **Stub** (letteralmente "mozzicone"), le componenti usate per realizzare in modo fittizio le funzionalità invocate dai moduli sottoposti a test e necessarie per la loro esecuzione.



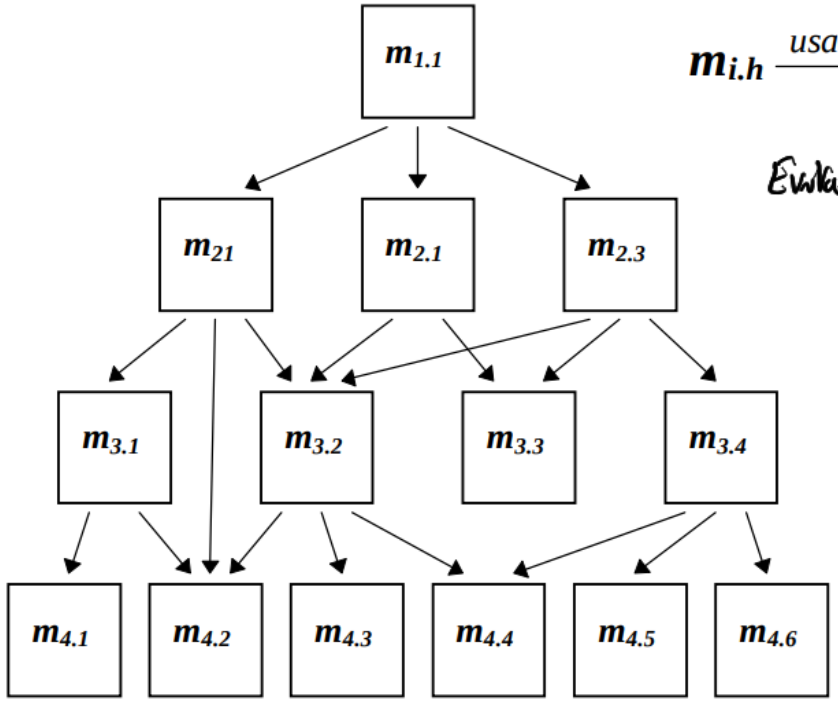
Livelli di Test: Test di Unità



Varie chiamate
Non solo 1.



Evita dipendenza circolare



- Grafo delle dipendenze: necessario per I test di Integrazione

Livelli di Test: Test di Integrazione

- L'eliminazione di un difetto a partire da malfunzionamento rilevato da controlli sul sistema onerosa
- Circa il 40% dei malfunzionamenti riscontrati sono riconducibili a difetti di mutuo interfacciamento tra i moduli.
- Testiamo il sistema ma mano che si integrano I moduli
  - Ancora sistema incompleto, ancora Driver e Stub
  - Strategie di integrazione diverse, guidate grafo delle dipendenze tra moduli

Livelli di Test: Test di Integrazione

- **Big-bang**
  - Assenza di una strategia
  - Tutti i moduli sono controllati singolarmente
  - L'integrazione avviene in un solo passo e si procede direttamente ai controlli di sistema.
  - Approccio costoso
- **Bottom-up**
  - Tutti i moduli che non dipendono da altri moduli sono controllati singolarmente.
  - Quindi si integrano i moduli controllati e si sale nell'albero eseguendo un nuovo livello di controlli.
  - Il procedimento si ripete fino a raggiungere il modulo radice.
  - Sono necessari solamente driver.
- **Top-down**
  - Il modulo radice dell'albero viene controllato singolarmente
  - Si integrano alla radice i moduli figli e si scende nell'albero eseguendo un nuovo livello di controlli.
  - Il procedimento si ripete fino all'integrazione completa del sistema.
  - Ad ogni livello si ripete sempre la stessa serie di controlli, dato che l'interfaccia del modulo radice è sempre la stessa.
  - Sono necessari solamente stub
- **Sandwich**
  - La strategia più usata
  - Combino le strategie bottom-up e top-down con l'obiettivo di minimizzare i costi per la realizzazione di stub e driver.
  - La linea di demarcazione, corrispondente al livello di simulazione eliminato, non necessariamente coincide con un livello dell'albero: in generale sarà una spezzata

Test di Sistema

- **Attività di Validazione**
  - Spesso guidata da criteri funzionali
- **Facility test** (test delle funzionalità).
  - È il più intuitivo dei controlli, quello cioè che mira a controllare che ogni funzionalità del prodotto stabilita nei requisiti sia stata realizzata correttamente.
- **Security test**
  - Cercando di accedere a dati o a funzionalità che dovrebbero essere riservate, si controlla l'efficacia dei meccanismi di sicurezza del sistema.
- **Usability test**
  - Valuta la facilità d'uso del prodotto da parte dell'utente finale
    - valutazione fra le più soggettive
    - considera prodotto + documentazione + livello di competenza dell'utenza + caratteristiche operative dell'ambiente d'uso del prodotto

Test di Sistema

- **Performance test**
  - Controllo mirato a valutare **l'efficienza** di un sistema soprattutto rispetto ai tempi di elaborazione e ai tempi di risposta
  - Controllo critico per esempio per i **sistemi in tempo reale**, per i quali ai requisiti funzionali si aggiungono rigorosi vincoli temporali
  - Il sistema viene testato a **diversi livelli di carico**, per valutarne le prestazioni
- **Storage use test** Test di prestaz. sulla memoria
  - Legato all'efficienza di un sistema, ma mirato alla richiesta di **risorse** (la memoria)
- **Volume test (o load test, test di carico)** Prima condiz. reali, ogni max possibile: es: 500 persone fisse
  - Il sistema è sottoposto al **carico di lavoro massimo** previsto dai requisiti e le sue funzionalità sono controllate in queste condizioni
  - Lo scopo è sia individuare malfunzionamenti che **non** si presentano **in condizioni normali**
  - Le tecniche e gli strumenti sono gli stessi usati per il performance test

I due tipi di test hanno scopi molto differenti, da un lato valutare le prestazioni a vari livelli di carico, non limite, dall'altro valutare il comportamento del sistema sui valori limite

Test di Sistema

- **Stress test**
  - Il sistema è sottoposto a carichi di lavoro superiori a quelli previsti dai requisiti o è portato in condizioni operative eccezionali – in genere sottraendogli risorse di memoria e di calcolo
  - Esplicito superamento dei limiti operativi previsti dai requisiti
  - Lo scopo è quello di controllare la capacità di "recovery" (recupero) del sistema dopo un fallimento
- **Configuration test** Posso verificare come funz. sistema su diverse condiz. di funzionamento?
  - Prova del sistema in tutte le configurazioni previste Cambiando configurazione dei parametri (es. Intel VS AMD chip).
  - Piattaforme di installazione diverse per sistema operativo o dispositivi hardware installati
  - Insiemi di requisiti funzionali leggermente diversi

↳ Parametri sia Hardware che software:
cambio cose fisiche o sistema operativo
- **Compatibility test**
  - Valutare la compatibilità del sistema con altri prodotti software
  - Versioni precedenti dello stesso prodotto
  - Sistemi diversi, ma funzionalmente equivalenti che il prodotto deve rimpiazzare
  - Altri sistemi software con i quali il prodotto deve interagire

↳ Vecchia versione SW compatibile con la nuova?

↳ Testo su Windows 10 e 10 patch 3.
Piccole modifiche, non cambio di configurazione.

Test di Sistema

- **Stress test**
  - Il sistema è sottoposto a carichi di lavoro superiori a quelli previsti dai requisiti o è portato in condizioni operative eccezionali – in genere sottraendogli risorse di memoria e di calcolo
  - Esplicito superamento dei limiti operativi previsti dai requisiti
  - Lo scopo è quello di controllare la capacità di “recovery” (recupero) del sistema dopo un fallimento
- **Configuration test**
  - Prova del sistema in tutte le configurazioni previste
  - Piattaforme di installazione diverse per sistema operativo o dispositivi hardware installati
  - Insiemi di requisiti funzionali leggermente diversi
- **Compatibility test**
  - Valutare la compatibilità del sistema con altri prodotti software
  - Versioni precedenti dello stesso prodotto
  - Sistemi diversi, ma funzionalmente equivalenti che il prodotto deve rimpiazzare
  - Altri sistemi software con i quali il prodotto deve interagire

Progettazione dei Test (cenni)

- **Criteri Funzionali (Black box)**
  - Basati solamente sui requisiti di un programma
  - Nella progettazione dei test interverranno solamente le caratteristiche esterne del programma
    - La sua interfaccia di ingresso/uscita
    - L'ambiente di esecuzione.
- **Criteri Strutturali (White box)**
  - Tengono conto della realizzazione del programma da controllare
  - Analisi del **flusso di controllo** e/o del **flusso dei dati** di un programma
  - Test sui comandi
  - Test sulle decisioni
  - Test sulle condizioni
  - Test sui cammini

So cosa deve fare il mio sistema. Conosco interfaccia ingresso uscita. Lo tratto come scatola nera.

Ok so cosa c'è nel sistema