

Ingegneria del Software e Sistemi Informativi

Progettazione delle Prove

a.a. 2019-2020

Docente: Prof. Antonio Esposito

Prova, Collaudo, Test

- Termini Equivalenti
 - Si parla propriamente di collaudo quando la prova è prevista da contratto
- Verifiche (o validazioni) dinamiche
 - attività che prevedono l'esecuzione del software in un ambiente controllato
 - Input e output definiti sui moduli o sul sistema
- Progettazione complessa
 - Come e dove eseguo i test?
 - Che risultati sono attesi?
 - Quando fermo le prove?
 - Sono definitive?

Il Test Ideale?

- Prova formale di correttezza: corrisponderebbe all'esecuzione del sistema con tutti i possibili input
- Teorema di Howden:
 - Non esiste un algoritmo che, dato P qualsiasi, generi un test ideale finito (test definito da un criterio affidabile e valido)
- Tesi di Dijkstra:
 - Il test di un programma può rilevare la presenza di malfunzionamenti, ma non dimostrarne l'assenza
- Una prova non è sempre definitiva
 - è definita e ripetibile
 - i suoi risultati non sono estendibili: i risultati di una prova valgono solo per le condizioni di quella prova

*Controllando le condizioni nel confronto
può fornire Verif*

Il Caso di Prova

- Caso di prova (o test case), è una tripla
 $\langle \text{INPUT}, \text{OUTPUT}, \text{AMBIENTE} \rangle$
↑ ambiente in cui sta funzionando sistema
- Input: Insieme degli ingressi considerati per la prova
- Output: Insieme delle risposte attese dal sistema esaminato
- Ambiente: Le condizioni in cui il test viene eseguito
es: ambiente Windows, Linux...

Test Suite e procedure

- Batteria di prove (o test suite)
 - un insieme (una sequenza) di casi di prova
 - una batteria può servire:
 - per la creazione di uno stato
 - per la copertura
- Procedura di prova
 - le procedure per eseguire, registrare analizzare e valutare i risultati di una batteria di prove
- La probabilità di rilevare il maggior numero di malfunzionamenti aumenta proporzionalmente al numero dei test eseguiti
- I test non devono essere condotti artigianalmente: si perde la possibilità di valutare oggettivamente il valore dei controlli.

Criteri di Test

- Criteri funzionali
 - A scatola chiusa (black box)
 - Basati sulla conoscenza delle funzionalità
 - Mirati a evidenziare malfunzionamenti sospettati o comunque relativi a funzionalità identificate
- Criteri strutturali
 - A scatola aperta (white box)
 - Basati sulla conoscenza del codice
 - Mirati a esercitare il codice indipendentemente dalle funzionalità
- Gray box
 - una strategia più che un criterio

Criteri funzionali: Si basano sulla conoscenza delle fzn. del mio sistema:
so quali sono gli input che mi aspetta e gli output che
mi aspetto

- I casi di input sono individuati sulla base delle specifiche software
- Statistico
- Partizionamento dei dati
- Valori di frontiera
- Grafo causa-effetto

Criterio Statistico

Prende i dati e considera n'analisi con cui che più riferiscono all'ingresso

- I casi di test sono selezionati in base alla distribuzione di probabilità dei dati di ingresso del programma
- Il test è quindi progettato per esercitare il programma sui valori di ingresso più probabili per il suo utilizzo a regime -> **Profilo Operazionale**
- Pro:
 - Nota la distribuzione di probabilità, la generazione dei dati di test è facilmente automatizzabile
- Contro
 - Non sempre corrisponde alle effettive condizioni d'utilizzo del software
 - È oneroso calcolare il risultato atteso

Criterio di Partizionamento

- Il dominio dei dati di ingresso è ripartito in classi di equivalenza
 - Due valori d'ingresso appartengono alla stessa classe di equivalenza se, in base ai requisiti, dovrebbero produrre lo stesso comportamento del programma
- Valido solo se le classi sono in numero ridotto
 - Input di classi diverse danno comportamento diverso
- Pro
 - I risultati attesi dal test sono noti e quindi non si pone il problema dell'oracolo
- Contro
 - Il criterio è basato su un'affermazione generalmente plausibile, ma non vera in assoluto
 - Problema di dover capire con attenzione per un input elementare cosa mi aspetto. Per esempio: Non per forza vero che tutti gli input è alla stessa classe di equivalenza.

es: magiare m. da 1K a 2K e da 2K a 3K
dovrebbero diverse
Pendo n dati di ingresso e ho
diverse nr

input di classi diverse danno
comportamento diverso

→ Problema di dover capire con attenzione per un input elementare

cosa mi aspetto. Per
esempio: Non

aver dati

Non per forza vero che tutti gli input è alla stessa classe di equivalenza.

Valori di Frontiera

- Basato su una partizione dei dati di ingresso
 - Classi identificate in base a:
 - Output atteso
 - Caratteristiche dei dati di ingresso
- Selezione degli estremi
 - I valori estremi sono in genere trattati in maniera particolare
 - Complementare ad altri test di partizione
 - I comportamenti possono non essere noti (problema dell'oracolo)

Sempre divisione in partition, ma non tutti valori a caso nella classe, ma sui limiti.

Sono quelli che possono causare ambiguità.

Test di partizionamento OK. Si fa sulla frontiera, poi gli altri
Se vuoi ci escono (caso nullo)

↑ Punto ambiguo, ma so nell'esempio cosa accade.
Fallimento non per errore ma per motivo di valutazione.

Grafi Causa Effetto:

Sembra alle macchine stessi eventi, ma solo una certa comb. di input, quali sono gli output che mi aspetto da loro?

Specifiche:

L'accesso è consentito se l'utente è registrato e la password è corretta; in ogni altro caso è negato. Se l'utente è registrato come speciale e la password è errata viene emesso un warning sulla console di sistema.

Fatti elementari:

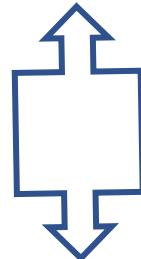
D'ingresso: Condizioni

- I_1 = utente registrato;
- I_2 = utente speciale;
- I_3 = password corretta.

D'uscita:

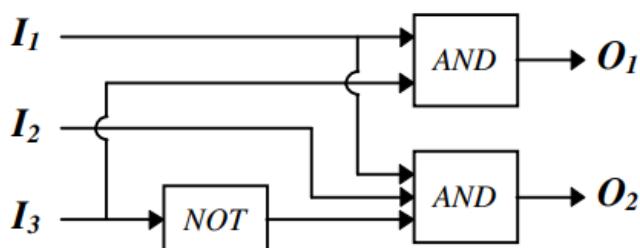
- O_1 = consentire l'accesso;
- O_2 = emettere il warning.

Ingresso



Rete Combinatoria

Problema: semplificare troppo
sistema.



Uscita

- Grafo che lega un insieme di fatti elementari di ingresso (cause) e di uscita (effetti) in una rete combinatoria che definisce relazioni di causa-effetto
- Risolve il problema dell'Oracolo

Criteri Funzionali

Base: conoscenza del codice che sto osservando

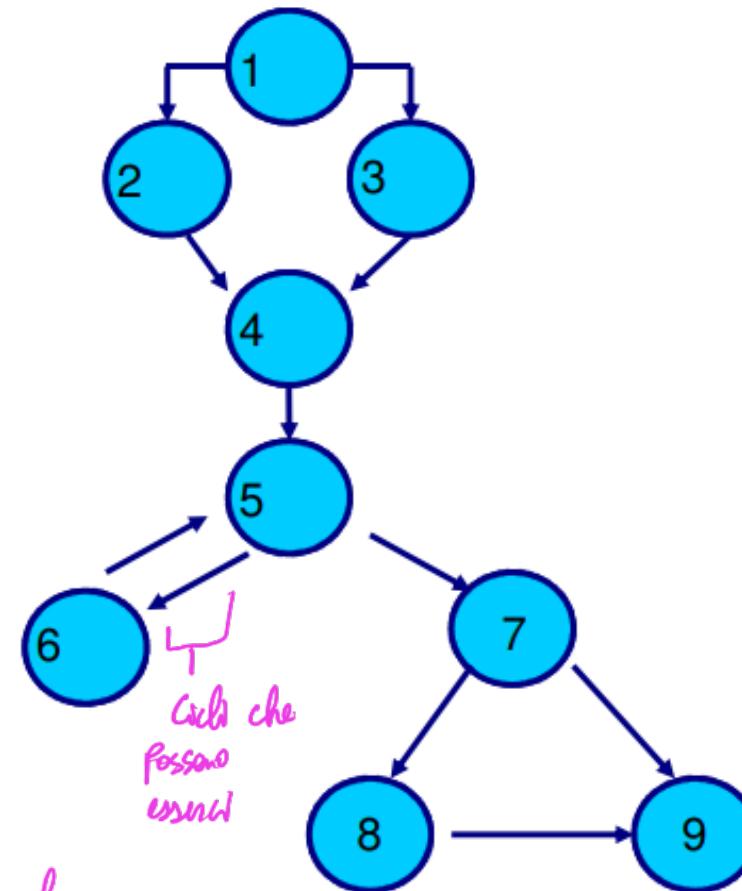
- Basati su grafi di flusso di controllo o dei dati
- Grafo di flusso
 - Definisce la struttura del codice identificandone le parti
 - È ottenuto a partire dal codice (con tool appositi)
 - Ad ogni nodo del grafo è associato un comando (statement)
 - Gli archi orientati rappresentano il legame di sequenzialità fra i comandi
 - A essi possono essere associate le condizioni che determinano le direzioni (branch) prese dal flusso del controllo in seguito all'esecuzione di comandi condizionali.

Ogni nodo collegato con due branch che corrispondono alle varie esecuzioni

Venne usato fin da subito "impostare code"

Un Grafo di Flusso

```
double eleva(int x, int y) {  
    1. if (y<0)  
    2.     pow = 0-y;  
    3.     else pow = y;  
    4. z = 1.0;  
    5. while (pow!=0)  
    6.     { z = z*x; pow = pow-1 }  
    7. if (y<0)  
    8.     z = 1.0 / z;  
    9. return(z);  
}
```



Ogni riga con 1 statement \Rightarrow modo specifico. Ogni tool lo fa da solo.

Differenza: grafi di flusso di controllo, le frecce indicano controlli, es. if, else ecc.

Grafi di flusso di dati mostrano come un dato può cambiare.

Criteri di Copertura

Il test: cerco a individuare percorsi che mi fanno percorrere tutto il grafo? Generalmente no. Vedrò l'ut.

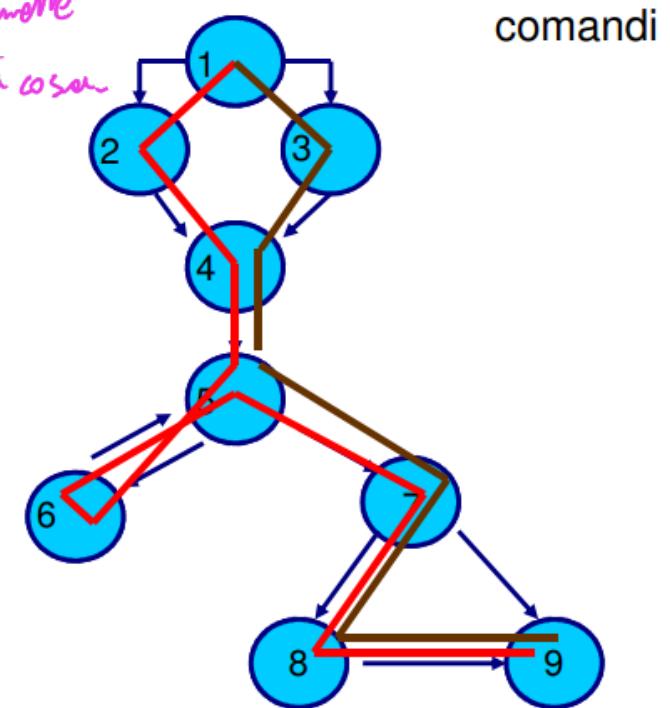
Test sui comandi (statement test)

- Ha l'obiettivo di eseguire i comandi del programma
- Test di copertura delle istruzioni
- Ogni nodo deve essere percorso almeno una volta



Voglio coprire TUTTE le istruzioni. Trovo il numero più piccolo che mi permette di fare questa cosa:

```
double eleva(int x, int y) {  
    1. if (y<0)  
    2.     pow = 0-y;  
    3.     else pow = y;  
    4.     z = 1.0;  
    5.     while (pow!=0)  
    6.         { z = z*x; pow = pow-1  
    }  
    7.     if (y<0)  
    8.         z = 1.0 / z;  
    9.     return(z);  
}
```



Criterio di copertura: Cosa deve essere eseguito almeno 1 volta nel sistema?

NOTA: eseguire comandi 1 sola volta con input specifico non ci assicura che non ci siano bug.

Criterio di copertura è sicuramente limitato. Se ne a coprire se ho almeno un modo per eseguire i comandi

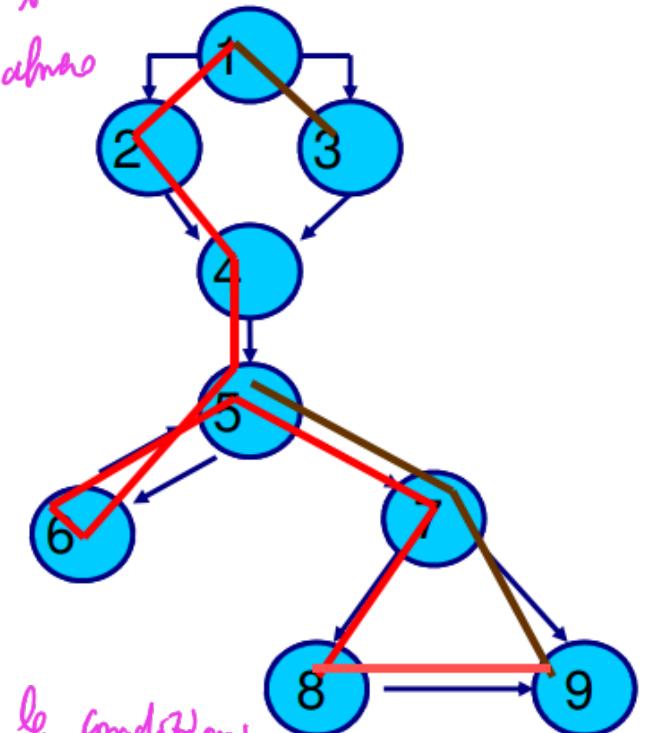
Criteri di Copertura

✓ punto di decisione, faccio assumere Vero o Falso almeno 1 volta \Rightarrow Siamo eseguiti tutti i comandi almeno 1 volta.

- **Test sulle decisioni** (branch test)
- Fa assumere I valori vero o falso a tutte le espressioni di controllo almeno una volta
- Edge test
- Comprende il test dei comandi

```
double eleva(int x, int y){  
    if (y<0)  
        pow = 0-y;  
    else pow = y;  
    z = 1.0;  
    while (pow!=0)  
        { z = z * x; pow = pow-1 }  
    if (y<0)  
        z = 1.0 / z;  
    return(z);  
}
```

Ma se ho un if che dipende da 3 variabili devo provare tutte le condizioni



Criteri di Copertura

- **Test sulle condizioni (condition test)**
 - Variante del branch test
 - Non si concentra sul risultato della condizione, ma sui parametri che l'hanno generata
 - Una semplice AND si traduce in 4 test differenti!
 - **Test sulle decisioni e sulle condizioni**
 - Combinazione di test diversi
 - I dati di test sono definiti in modo che sia tutte le decisioni presenti nel programma che tutte le condizioni di cui sono composte assumano almeno una volta entrambi i valori di vero e falso
 - **E per i cicli? Come testo il ciclo?**
 - È possibile percorrere tutti i nodi e tutti gli archi eseguendo un ciclo una sola volta
 - Si fa in modo che ciclo venga eseguito almeno una volta
 - Gli errori "di uno" nell'indicizzazione dei vettori o nelle condizioni di disegualanza (Obi-Wan error)
 - Memory leak
 - Sfondo velleire
- Mi metto nei 2 limiti: best per l'and, 0 e m-1 per eventuali problemi oltre all'esecuzione finale.

Criteri di Copertura

- Test sulle condizioni (condition test)
 - Variante del branch test
 - Non si concentra sul risultato della condizione, ma sui parametri che l'hanno generata
 - Una semplice AND si traduce in 4 test differenti!
- Test sulle decisioni e sulle condizioni
 - Combinazione di test diversi
 - I dati di test sono definiti in modo che sia tutte le decisioni presenti nel programma che tutte le condizioni di cui sono composte assumano almeno una volta entrambi i valori di vero e falso
- E per I cicli?
 - È possibile percorrere tutti I nodi e tutti gli archi eseguendo un ciclo una sola volta
 - Gli errori “di uno” nell’indicizzazione dei vettori o nelle condizioni di disuguaglianza (Obi-Wan error)
 - Memory leak

Criteri di Copertura

- **Path coverage** = copertura dei cammini
 - Cerca di percorrere tutti i possibili percorsi all'interno del diagramma di flusso
 - Include il criterio di copertura per decisioni
 - Può portare all'esplosione del numero di test: in genere è solo teorico se ci sono cicli
- **n-test dei cicli**
 - Caso particolare di path coverage
 - In presenza ci cicli, si imposta un limite n al numero di volte che ogni ciclo può essere attraversato
 - N viene definito dal progettista sulla base dell'esperienza

Magari n = dimensione vettore non basta

Percentuale di Copertura

→ Il tempo ragionevole non è possibile.

- Quanto la prova “esercita” il prodotto *Così test niente sempre va raggiungere il 100% del codice?*
 - copertura funzionale rispetto alla percentuale di funzionalità esercitate
 - copertura strutturale: rispetto alla percentuale di codice esercitata
- Una misura della bontà di una prova
 - La copertura del 100% non significa assenza di difetti *ma ha un numero di errori inferiore*
 - non è detto che il 100% di copertura sia raggiungibile *non tempo ragionevole*
- I casi di test possono essere molto distanti dalla distribuzione di probabilità dei dati d'ingresso del programma.
- È infatti frequente che una parte sostanziale di un programma sia destinata al trattamento di casi particolari
 - le statistiche più pessimiste parlano di un 80% del codice usato solamente nel 20% dei casi.
 - In questo senso, i casi di test ottenuti dai criteri strutturali possono essere considerati inefficienti.

Criteri Gray box

→ Assunto: conoscenza centrale del codice. Riesce a creare gravi errori che il sistema e poi chiede delle diritte al programmatore su cosa potessere avere problemi.
Faccio delle supposizioni sui problemi eventuali. Guessing: suppongo che
tengano gruppetti causa problema

1. Testare il programma conoscendo i requisiti ed avendo una limitata conoscenza della realizzazione, per esempio conoscendo solo l'architettura.
2. Progettare il test usando criteri funzionali e quindi di usare le misure di copertura dei criteri strutturali per valutare l'adeguatezza del test
3. Approccio di Myers
 - Un primo test è progettato utilizzando il grafo causa-effetto
 - Si partizionano i dati di ingresso seguendo il criterio dei valori di frontiera
 - I progettisti sono chiamati a formulare delle ipotesi di malfunzionamento (**error guessing**) e a integrare di conseguenza i casi di test
 - La struttura del programma è usata per stabilire se i test realizzati ai passi precedenti hanno "esercitato" a sufficienza il codice
4. Test mutazionale

* Sai fin' quando il codice è codice che ho creduto
e non è niente - Error guessing

Test Mutazionale

- La tecnica si applica in congiunzione con altri criteri di test
- Insiemi di test già realizzati
- La strategia prevede di introdurre modifiche controllate nel programma originale
- Le modifiche riguardano in genere l'alterazione del valore delle variabili e la variazione delle condizioni booleane.
- I programmi così ottenuti (incorrecti rispetto alle specifiche), sono definiti **mutanti**.
- L'insieme dei test realizzati precedentemente viene quindi applicato, senza modifiche, a tutti i mutanti e i risultati confrontati con quelli degli stessi test eseguiti sul programma originale

Prende codice già testato e lo modifica cambiando dove ci sono decisioni da prendere creando codice sintatticamente corretto ma diverso per quello che fa.
Il test li prova sui codici mutanti.

Se nel mutante non mi accorgo delle differenze, allora il test fu scelto.

Scopo:

- Favorire la scoperta di malfunzionamenti ipotizzati
- Valutare l'efficacia dell'insieme di test, controllando se "si accorge" delle modifiche introdotte sul programma originale.
- Cercare indicazioni circa la localizzazione dei difetti la cui esistenza è stata denunciata dai test eseguiti sul programma originale

- Trovate la teoria e le slide originali da cui sono state estratte le presenti su:
- <http://didawiki.cli.di.unipi.it/doku.php/informatica/is-a/start>
- <http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/informatica/is-a/disponsatesting2014.pdf>