Progettazione delle prove

Roberta Gori, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

Verifica dinamica o testing

- Si compone di più fasi:
  - Progettazione (input, output atteso …)
  - Definizione ambiente di test
  - Esecuzione del codice
  - Analisi dei risultati (output ottenuto con l'esecuzione vs output atteso)
  - Debugging

Proprietà e aspetti del testing

- Ripetibilità
- Verifica di componenti vs verifica di sistema
- Test di integrazione
- Vari tipi di test sul Sistema
- Test di accettazione (o collaudo)

Ripetibilità

- Ripetibilità della prova
  - Ambiente definito (hardware, condizioni, …)
  - casi di prova definiti (ingressi e comportamenti attesi)
  - procedure definite
- Registrazione e analisi dei dati di prova

Gli elementi di una prova: Caso di prova (o test case)

- Caso di prova (o test case), è una tripla

<input, output, ambiente>

Gli elementi di una prova: batteria e procedura

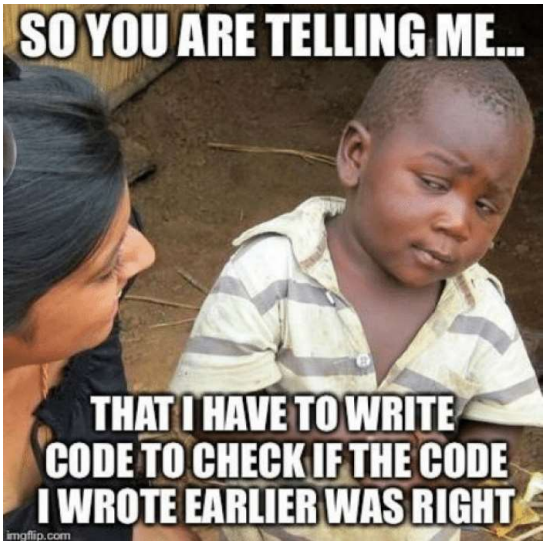
- Batteria di prove (o test suite)
  - un insieme (una sequenza) di casi di prova
  - una batteria può servire:
    - per la creazione di uno stato
    - per la copertura (concetto di copertura spiegato nel seguito della lezione)
- Procedura di prova
  - le procedure (automatiche e non) per eseguire, registrare analizzare e valutare i risultati di una batteria di prove

Conduzione di una prova

- Definizione dell'obiettivo della prova
  - è importante definire l'obiettivo
- Progettazione della prova
  - la progettazione consiste soprattutto nella scelta e nella definizione dei casi di prova (della batteria di prove)
- Realizzazione dell'ambiente di prova
  - ci sono driver e stub da realizzare, ambienti da controllare, strumenti per la registrazione dei dati da realizzare

Test scaffolding

- Codice aggiuntivo necessario per eseguire un test.
- Si chiama scaffolding (impalcatura), per analogia alle strutture temporanee erette attorno a un edificio durante la costruzione o la manutenzione.



Test scaffolding

- Lo scaffolding può includere:
  - **driver** di test (sostituiscono un programma principale o di chiamata),
  - **test harness** (sostituiscono parti dell'ambiente di distribuzione) (ATTENZIONE: per altri autori harness è sinonimo di scaffolding)
  - **stub** (sostituiscono funzionalità chiamate o utilizzate dal software in prova) (**mock**),
  - tool per gestire l'esecuzione del test
  - tool per registrare i risultati

Scelta dei casi di input

- Per progettare i casi di test si comincia col definire un "buon" insieme di casi di input.

- Le strategie usate sono:
  - Criteri funzionali (o black box)
  - Criteri strutturali (o white box)
  - Gray box
  - ...

Criteri per l'individuazione dei casi di input

- Criteri funzionali (a scatola chiusa, black box)
  - basati sulla conoscenza delle funzionalità
  - mirati a evidenziare malfunzionamenti relativi a funzionalità
- Criteri strutturali(a scatola aperta, white box)
  - basati sulla conoscenza del codice
  - mirati a esercitare il codice indipendentemente dalle funzionalità
- Criteri basati sul modello del programma
  - Modelli utilizzati nella specifica o nella progettazione, o derivati dal codice
  - Esempio: Esercizio di tutte le transizioni nel modello di protocollo di comunicazione
- Criteri basati su fault
  - Cercano difetti ipotizzati (bug comuni)
  - Ex: check per la gestione del buffer overflow testando con input molto grandi

Casi di test (istanze) e specifica di casi di test (descrizioni)

- Test case specification: descrizione di uno o più casi di test
- Esempio:
  - specifica del test case: input formato da due o più parole
  - i casi di test con i valori di input
    - "alpha beta"
    - "Milano Pisa Roma"
  - sono due tra i tanti test che soddisfano la specifica

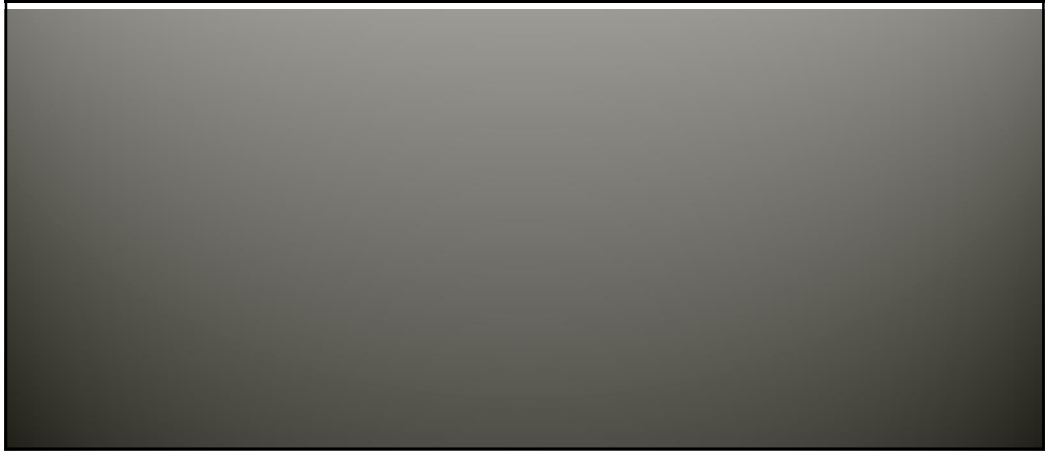
Test obligation e criterio di adeguatezza

- **Adequacy criterion** (criterio di adeguatezza) : predicato che può essere vero o falso per una coppia

  <programma, test suite>
- **Test obligation**: una test case specification (richiesta)
- Diciamo che una suite di test soddisfa un criterio di adeguatezza se tutti i test hanno successo e se ogni test obligation è soddisfatto da almeno uno dei test case nella suite di test.

Criteri funzionali

Sono criteri per l'individuazione dei casi di input che si basano sulle specifiche

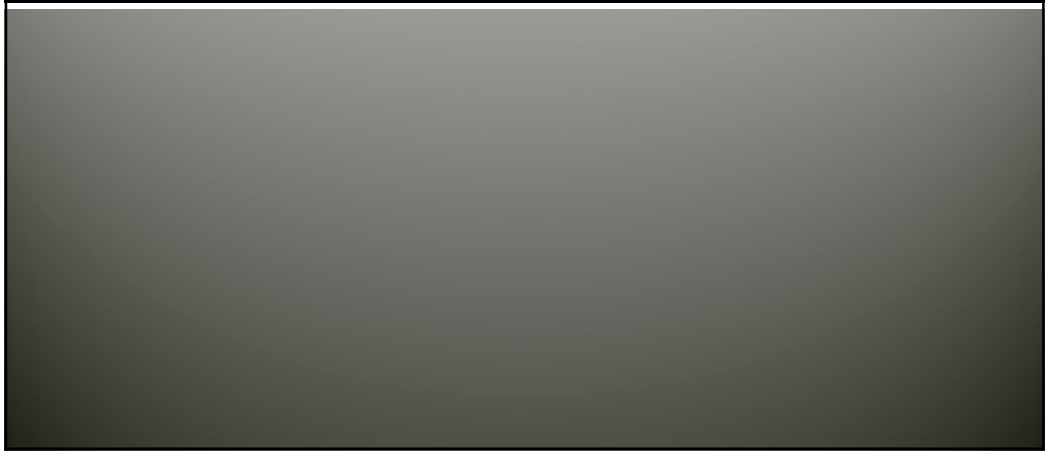


Strategia

- Separare le funzionalità da testare
  - Per esempio usando i casi d'uso
- derivare un insieme di casi di test per ogni funzionalità
  - Per fare ciò
    - per ogni (tipo di) parametro di input
      - Si individuano dei valori da testare
      - Per questo si usano alcune tecniche (metodi) che vediamo nei prossimi lucidi
    - Per l'insieme dei parametri
      - Si usano tecniche che vanno sotto il nome di testing combinatorio per ridurre le combinazioni

Metodi black box per generare valori di input

(data una funzionalità e dato un parametro)



Metodo random

- Generare in modo automatico un insieme grande a piacere di valori
  - Costo zero la generazione
  - Applicabile se costa poco l'esecuzione
  - Difficilmente considera i casi limite
    - Esempio: trovare le radici di un'equazione di secondo grado

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

    - Quasi impossibile che il caso **b=0**, **a=0** sia generato in modo casuale

Metodo statistico

- 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
- Il vantaggio è che, nota la distribuzione di probabilità, la generazione dei dati di test è facilmente automatizzabile
- Non sempre corrisponde alle effettive condizioni d'utilizzo del software
- È oneroso calcolare il risultato atteso

Esempio di selezione usando il metodo statistico

- Si consideri l'input "età il giorno della laurea":
  - Il tipo è int
  - In questo caso è ragionevole usare il metodo statistico e dare le specifica di test:
    - tutti i valori compresi tra 20 e 27
    - Il 40% dei valori tra 27 e 35
    - Il 5% dei valori tra 36 e 100

Partizione dei dati d'ingresso (in categorie)

- Il dominio dei dati di ingresso è ripartito in classi di equivalenza (categories nel libro Pezzé Young)
  - due valori d'ingresso appartengono alla stessa classe di equivalenza se, in base ai requisiti, dovrebbero produrre lo stesso comportamento del programma
- Il criterio è economicamente valido solo per quei programmi per cui il numero dei possibili comportamenti è sensibilmente inferiore alle possibili configurazioni d'ingresso
  - per come sono costruite le classi, i risultati attesi dal test sono noti e quindi non si pone il problema dell'oracolo
- Il criterio è basato su un'affermazione generalmente plausibile, ma non vera in assoluto
  - la deduzione che il corretto funzionamento sul valore rappresentante implichi la correttezza su tutta la classe di equivalenza dipende dalla realizzazione del programma e non è verificabile sulla base delle sole specifiche funzionali

Esempio: Partizione dei dati d'ingresso

Scaglioni di reddito	*Aliquote*
Fino a € 15.000	23%
Oltre a € 15.000 e fino a € 28.000	27%
Oltre a € 28.000 e fino a € 55.000	38%
Oltre a € 55.000 e fino a € 75.000	41%
Oltre a € 75.000	43%

Valori di frontiera

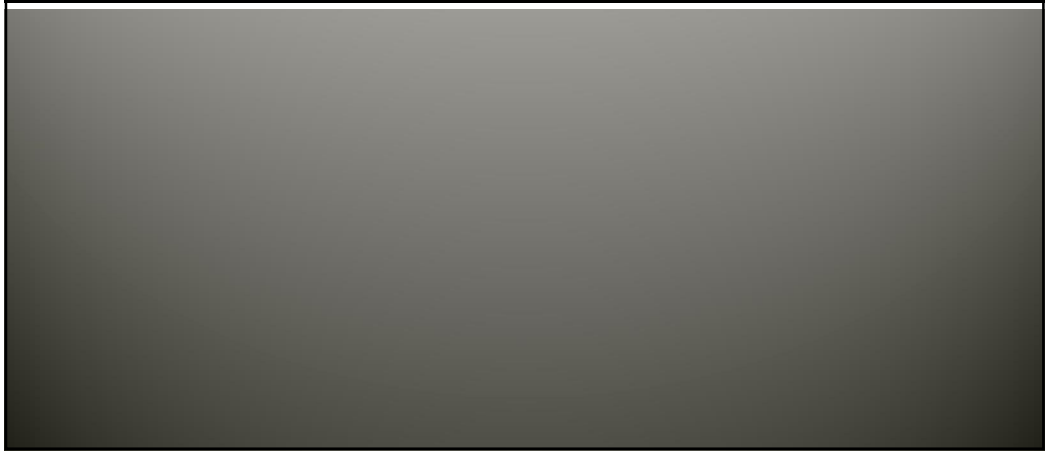
- Basato su una partizione dei dati di ingresso
  - le classi di equivalenza realizzate o in base all'eguaglianza del comportamento indotto sul programma o in base a considerazioni inerenti il tipo dei valori d'ingresso
- Dati di test: valori estremi di ogni classe di equivalenza
- È possibile che debba essere considerato il problema dell'oracolo
- Questo criterio ricorda i controlli sui valori limite tradizionali in altre discipline ingegneristiche per le quali è vera la proprietà del comportamento continuo
  - in meccanica, ad esempio, una parte provata per un certo carico resiste con certezza a tutti i carichi inferiori
- Questa proprietà non è applicabile al software: i valori limite sono frequentemente trattati in modo particolare

I casi non validi

- Per ogni input si definiscono anche i casi non validi (che devono generare un errore):
  - Età inferiori a 20 o superiori a 120 per la laurea
  - Reddito negativo per il calcolo delle aliquote
  - …

Testing combinatorio

F(x1, x2, x3, …)



Esplosione combinatoria

- In presenza di più dati di input, se si prende il prodotto cartesiano dei casi di test individuati, facilmente si ottegono numeri non gestibili
- Occorrono strategie per generare casi di test significativi in modo sistematico
- Tecniche per ridurre l'esplosione combinatoria
  - Vincoli
  - Pairwise testing
  - Testing basato su catalogo

Vincoli (constraints)

- Servono per ridurre le possibili combinazioni
  - di errore,
  - di proprietà,
  - singoletti

Vincoli di errore (Error constraints)

- <x1, x2,x3,x4,x5>
- Dominio di x1 e x2 ripartibile in 8 classi (di cui una di valori non validi → errore)
- Dominio di x3 e x5 ripartibile in 4 classi (di cui una di valori non validi → errore)
- Dominio di x4 ripartibile in 7 classi (di cui una di valori non validi → errore)
- Un rappresentante per classe: 8x8x4x7x4= 7.168 casi di test

Vincoli di errore (Error constraints)

- <x1, x2,x3,x4,x5>
- Viene preso un solo caso, per ogni posizione, con input non valido
- 5 + 7x7x3x6x3= 2.651

- Da 7.168 a 2.651

Altri vincoli (property – if property)

- <x1, x2,x3,x4,x5>
  - x1:

    classe 1, classe 2, classe 3, classe 4 [negativi]
    classe 5, classe 6, classe 7 [positivi]
    (classe8 [error])
  - x2:

    classe 1, classe 3, classe 5, classe 7 [if negativi] error
    classe 2, classe 4, classe 6 [if positivi] error
    (classe8 [error])
- 5 + **4x4**x3x6x3 + **3x5**x3x6x3= 5 + 864 + 810 = 899
- da 7.168 a 2.651 a 899

Vincolo [single]

- Per uno (o più) parametri si può decidere di testare un solo valore
- per esempio x4 [single]
- 5 + 4x4x3x**1**x3 + 3x5x3x**1**x3= 5 + 144 + 135 = 284
- da 7.168 a 2.651 a 899 a 284

Esempio (con notazione diversa)