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, ambiente di esecuzione …)
  - 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>

Progettazione casi di test

Adeguatezza dei casi di test

- Un modo per definire l'adeguatezza di un insieme di casi di test (test suite)

Se il sistema supera un'adeguato insieme di test allora deve essere corretto

E' imposibile!
L'adeguatezza come e' definita sopra e' indecidibile!

Criteri Pratici di (in)Adeguatezza

- Criteri che identificano inadeguatezza nei casi di test

Esempi:

- nella specifica: ho due casi, il casi di test non testano che i due casi siano trattati differentemente
- nel codice: ci sono n istruzioni , i casi di test ne testano k<n

- Il fatto che il nostro insieme di test non verifichi un criterio ci suggerisce come modificarlo

- Un criterio di adeguatezza =insieme di test obligations

Test obligation

- una specifica (descrizione) di casi di test che richiede proprieta' ritenute importanti per il testing
- Esempio:
  - specifica del test case: un input formato da due parole e un input formato da tre
  - i casi di test con i valori di input
    - "alpha beta"
    - "Milano Pisa Roma"
  - sono due tra i tanti test che soddisfano la specifica

Come definire le test obligations

- dalle funzionalita’ (a scatola chiusa, black box): dalla specifica sw
  - basati sulla conoscenza delle funzionalità
  - mirati a evidenziare malfunzionamenti relativi a funzionalità
  - Es: se la specifica richiede una procedura di recovery nel caso di mancanza di corrente, le test obligations dovranno includere la simulazione del fenomeno
- dalla struttura(a scatola aperta, white box): dal codice
  - basati sulla conoscenza del codice
  - mirati a esercitare il codice indipendentemente dalle funzionalità
  - Es: passare da ogni loop almeno una volta
- dal modello del programma:dal modello del sistema
  - Modelli utilizzati nella specifica o nella progettazione, o derivati dal codice
  - Es: Esercizio di tutte le transizioni nel modello di protocollo di comunicazione
- da fault ipotetici
  - Cercano difetti ipotizzati (bug comuni)
  - Es: check per la gestione del buffer overflow testando con input molto grandi

Criteri di adeguatezza

- Un criterio di adeguatezza = insieme di test obligations

Un insieme di test soddisfa un criterio di adeguatezza se:

•Tutti i test hanno successo

• Ogni test obligation e' soddisfatta da almeno un caso di test (nell'insieme di casi di test scelto)

Esempio: il criterio di adeguatezza della copertura delle istruzioni e' soddisfatto da un insieme di test S per il programma P se ogni istruzione eseguibile in P e' esercitata da almeno un test in S, e il risultato dell'esecuzione corrisponde a quello atteso.

Uno sguardo alle altre fasi del testing

Gli elementi di una prova: batteria e procedura

- Batteria di prove (o test suite)
  - un insieme (una sequenza) di casi di prova

- Procedura di prova
  - le procedure (automatiche e non) per eseguire, registrare analizzare e valutare i risultati di una batteria di prove

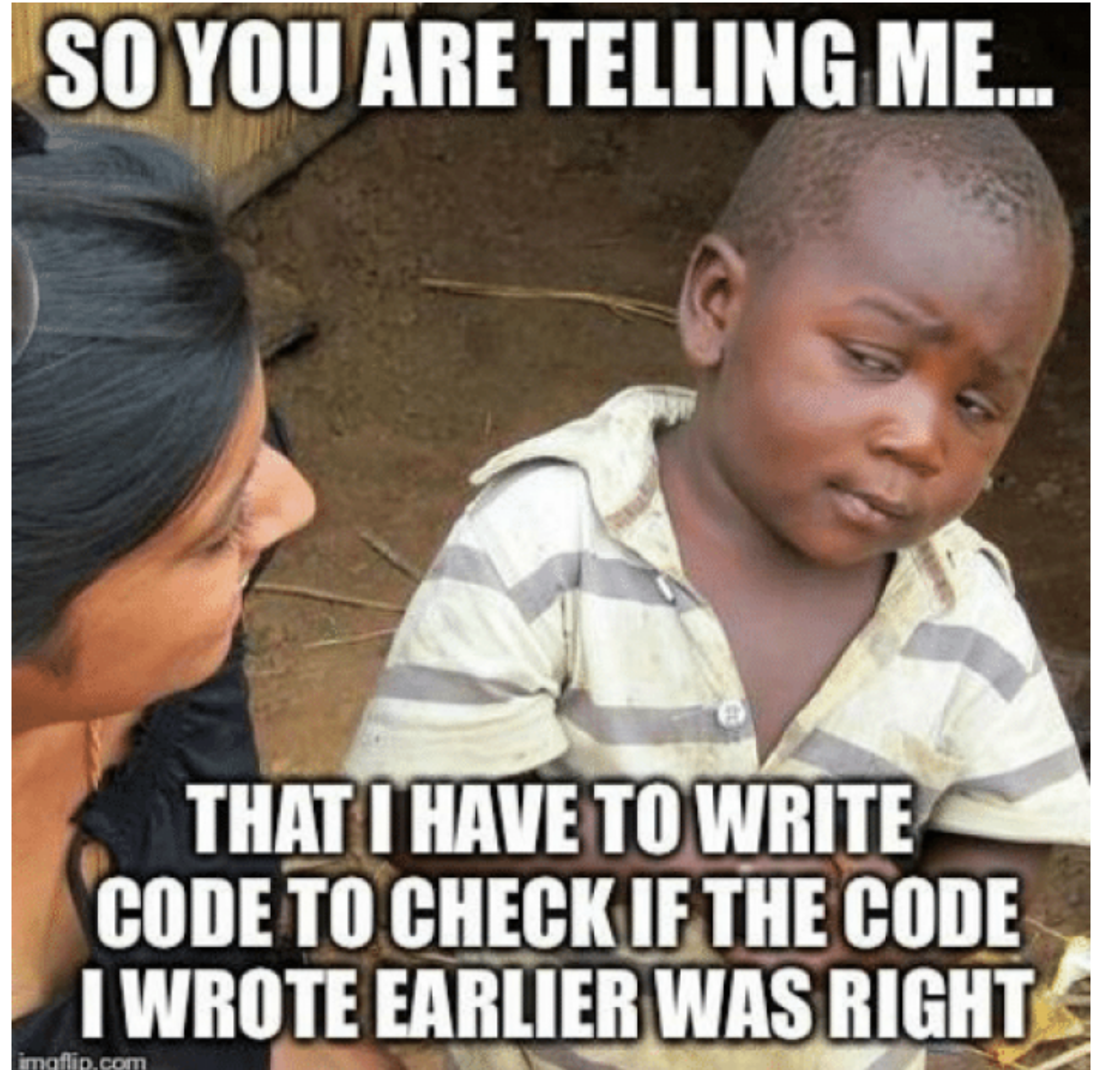
Conduzione di una prova

- 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

- Esecuzione della prova
  - l'esecuzione può richiedere tempo
- Analisi dei risultati
  - l'esame dei risultati alla ricerca di evidenza di malfunzionamenti
- Valutazione della prova

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

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
      - Questi possono essere scelti in modo random
    - Il 5% dei valori tra 36 e 100
      - Questi possono essere scelti in modo random

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

$F(x1, x2, x3, \ldots)$

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+3x3)** x3x6x3= 5 + 1350 = 1.355
- da 7.168 a 2.651 a 1.355

Vincolo [single]

- Per uno (o più) parametri si può decidere di testare un solo valore

- per esempio x4 [single]

- 5 + (4x4+3x3) x3x1x3= 5 + 225 = 230
- da 7.168 a 2.651 a 1.355 a 230

Combinazione di test basato su coppie

- (Pairwise testing)
- La tecnica vista precedentemente permette di introdurre vincoli che limitino il numero di test ottenuti dalla generazione di tutte le le combinazioni di valori possibili.

- Funziona bene se i vincoli che imponiamo sono reali vincoli del dominio e non se li aggiungiamo al solo scopo di limitare le combinazioni

- Nel caso in cui il dominio non contenga in sé questi vincoli è preferibile optare per la generazione di tutte le combinazioni per solo k variabili, con $k<n$

Combinazione di test basato su coppie

- L'idea: generare tutte le possibili combinazioni solo per k variabili
- se k=2 genero tutte le combinazioni per tutte le coppie di variabili

- Quanto si risparmia?

Esempio

Display Mode	Language	Fonts
full-graphics	English	Minimal
text-only	French	Standard
limited-bandwidth	Spanish	Document-loaded
	Portuguese	

Color	Screen size
Monochrome	Hand-held
Color-map	Laptop
16-bit	Full-size
True-color	

Esempio

- Se volessimo generare tutte le combinazioni per *Display mode*, *Screen size* e *Fonts* avremmo

$$3^3 = 27$$

- Se invece generiamo tutte le combinazioni solo per la coppia <*Display mode*, *Screen size*> abbiamo

$$3^2 = 9$$

- il valore del terzo parametro può essere aggiunto in modo da coprire tutte le combinazioni di *FontsxScreen size* e *FontsxDispay mode*

Esempio

Display mode ×	*Screen size*	*Fonts*
Full-graphics	Hand-held	Minimal
Full-graphics	Laptop	Standard
Full-graphics	Full-size	Document-loaded
Text-only	Hand-held	Standard
Text-only	Laptop	Document-loaded
Text-only	Full-size	Minimal
Limited-bandwidth	Hand-held	Document-loaded
Limited-bandwidth	Laptop	Minimal
Limited-bandwidth	Full-size	Standard

- La generazione di combinazioni che in maniera efficiente coprano tutte le coppie è impossibile da fare a mano per molti parametri con molti valori ma può essere fatta con euristiche.

Test basato su catalogo

- Nel tempo un organizzazione può essersi costruita un’esperienza nel definire tests
- Collezionare questa esperienza in un catalogo può rendere più veloce il processo e automatizzare alcune decisioni riducendo l’errore umano
- I cataloghi catturano l’esperienza di coloro che definiscono i tests elencando tutti i casi che devono essere considerati per ciascun possibile tipo di variabile

Esempio di voce nel catalogo

- Assumiamo che una funzione usi una variabile il cui valore deve appartenere ad un intervallo di interi, il catalogo potrebbe indicare i casi seguenti come rilevanti:

1. The element immediately preceding the lower bound of the interval
2. The lower bound of the interval
3. A non-boundary element within the interval
4. The upper bound of the interval
5. The element immediately following the upper bound

Syllabus

- Cap 9-10-11

  Software Testing and Analysis: Process, Principles and Techniques-

  Mauro Pezzè e Michal Young