Corso di Laurea in Ingegneria Informatica

Corso di Ingegneria del Software

Test Funzionale

Sommario

- Test Funzionale
- Random testing
- Test funzionale vs random test
- Test funzionale vs test strutturale
- Partition testing

Riferimenti

M.Pezzè, M. Young; *Software Testing and Analysis. Process, principles, and techniques*. **Cap**. 10

Test Funzionale

Derivazione dei casi di test dalle specifiche

- Funzionale si riferisce alla sorgente di informazione usata nella progettazione dei casi di test, non a cosa è testato

- Anche noto come
  - *Specification-based testing* (dalle specifiche)
  - *Black-box testing* (non c'è visibilità del codice)

- **Specifiche funzionali** = descrizione del **comportamento atteso** del programma, sia essa formale o informale

Perché il test funzionale?

- E' la tecnica base per progettare i casi di test
  - Spesso è utile nel raffinare le specifiche e stimare la testabilità prima della scrittura del codice
- Efficiente
  - Trova delle classi di fault che possono eludere altri approcci (e.g., *missing logic*)
- Ampiamente applicabile
  - A ogni descrizione del comportamento del programma che serve da specifica
  - Ad ogni livello di granularità, dal test di unità al test di sistema
- Economico
  - Tipicamente meno costoso da progettare ed eseguire rispetto ai casi di test strutturali (basati sul codice)

Progettazione dei test preliminare

- Il codice del programma non è necessario
  - Serve solo una descrizione del comportamento atteso
  - Possono essere usate anche specifiche informali ed incomplete
- Specifiche più accurate e complete portano a migliori test suite
- La progettazione preliminare (già nelle prima fasi del ciclo di sviluppo) comporta benefici
  - Spesso rileva ambiguità e inconsistenza nelle specifiche
  - Utile per stimare la testabilità
  - Per migliorare il piano di test ed il budget grazie al miglioramento delle specifiche
  - Utile nel supportare la descrizione delle specifiche
    - O nel caso estremo (come nella metodologia XP) i test case *sono* le specifiche

Random testing

- Stimare la proporzione di “aghi”, campionarli in modo random
  - Stime di *reliability* richiedono campioni casuali, non influenzati da chi li sceglie, per essere statisticamente validi.
- Trovare “aghi” e rimuoverli, cercarli sistematicamente
  - A meno che non ci sia un gran numero di aghi, un campione *random* non sarà sufficientemente efficace/efficiente nel trovarli
  - Bisogna usare tutto ciò che si sa di questi “aghi”

Systematic vs Random Testing

- *Random*
  - Si scelgono gli input campionandoli dallo spazio degli input secondo una distribuzione di probabilità (uniforme o non uniforme)
  - Evita la possibile influenza del progettista nella scelta degli input
  - Ma tratta tutti gli input allo stesso modo (non usa altra conoscenza sugli input se non la distribuzione di probabilità)
- Test sistematico
  - Prova a selezionare input *particolarmente rilevanti*
  - Di solito scegliendo valori rappresentativi di classi che tendono a *fallire spesso o a non fallire mai*
- Il *Test Funzionale* è un Test sistematico

Perché non il Random Testing?

- Distribuzione dei fault non è uniforme
- Esempio: una classe Java "radici" usata per la soluzione di un' equazione di II grado
  - Problema: logica di implementazione incompleta. Il programma non gestisce bene il caso in cui delta=0 e a=0
  - I valori causanti il fallimento sono *sparsi* nello spazio degli input, come aghi in un pagliaio. Un *test random* è improbabile che scelga a=0 e b=0 come input

Test Funzionale vs. Strutturale

- Strategie diverse sono più o meno efficienti per classi diverse di fault
- Il test funzionale è migliore per fault di tipo *missing logic*
  - Un problema comune: alcune parti della logica del programma vengono semplicemente dimenticate
  - Il test strutturale (basato sul codice) non si focalizzerà mai su codice che non esiste!

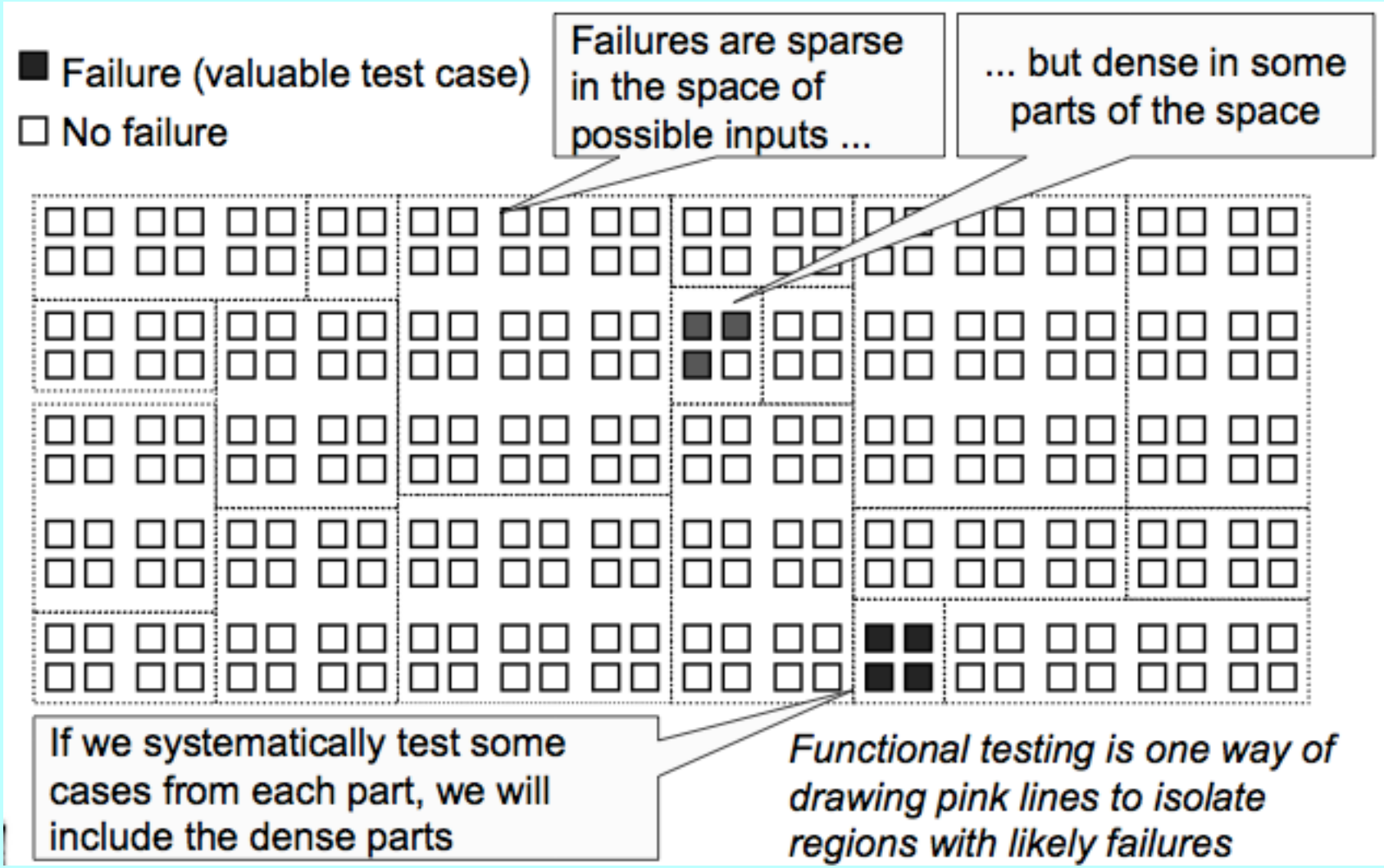
Test Funzionale vs. Strutturale

- Cerca di rilevare difetti compresi nelle seguenti categorie:
  - *Funzioni incomplete o mancanti*
  - *Errori di interfaccia*
  - *Errori nelle strutture dati o negli accessi a data base esterni*
  - *Errori di presentazione*
  - *Errori di inizializzazione e terminazione*
- Non è alternativo, ma complementare al test strutturale

Test Funzionale vs. Strutturale: livelli di granularità

- Il test funzionale si applica a tutti i livelli di granularità
  - Unità (dalle specifiche delle interfacce dei moduli)
  - Integrazione (dalle API o dalle specifiche dei sottosistemi)
  - Sistema (dalle specifiche dei requisiti di sistema)
  - Regressione (dai requisiti di sistema + la storia dei bug)
- Il test strutturale si applica a parti relativamente piccole del sistema:
  - Unità
  - Integrazione

Systematic Partition Testing



Il principio di partizionamento

- Sfruttare conoscenza per scegliere campioni che più probabilmente includono regioni dello spazio degli input "speciali" o propense a causare *failures* (*failure-causing inputs*)
  - I *failure-causing inputs* sono sparsi nell'intero spazio degli input
  - … ma possiamo trovare regioni in cui sono densi
- Si chiama (Quasi-)**Partition Testing** un metodo di testing che divide l'insieme ("infinito") di casi di test in un insieme finito di classi, la cui unione è l'intero spazio
  - "Quasi-" perché le classi possono sovrapporsi

Specification-based partition testing

- Quando si adopera un partition testing nel contesto di un test funzionale, cioè quando le partizioni sono scelte sulla base delle specifiche funzionali, si parla di **specification-based partition testing** o functional partition testing o spesso solo di functional testing

- Caso desiderabile: ogni *fault* porta a *failures* che sono densi (facili da trovare) in alcune classi di input
  - Campionare ogni classe nel quasi-partition seleziona almeno un input che porta a fallimento, rivelando il guasto
  - Raramente è garantito; ci si basa su euristiche basate sull'esperienza

Test funzionale: sfruttare le specifiche

- Il test funzionale usa la specifica (formale o informale) per partizionare lo spazio degli input
  - E.g., la specifica del programma "radici" suggerisce la divisione tra casi con zero, una e due radici reali
- Test di ogni categoria, e dei limiti tra le categorie
  - Non c'è alcuna garanzia, ma l'esperienza suggerisce che i *failure-causing inputs* spesso si trovano ai limiti tra le categorie (come nel programma "radice quadrata")

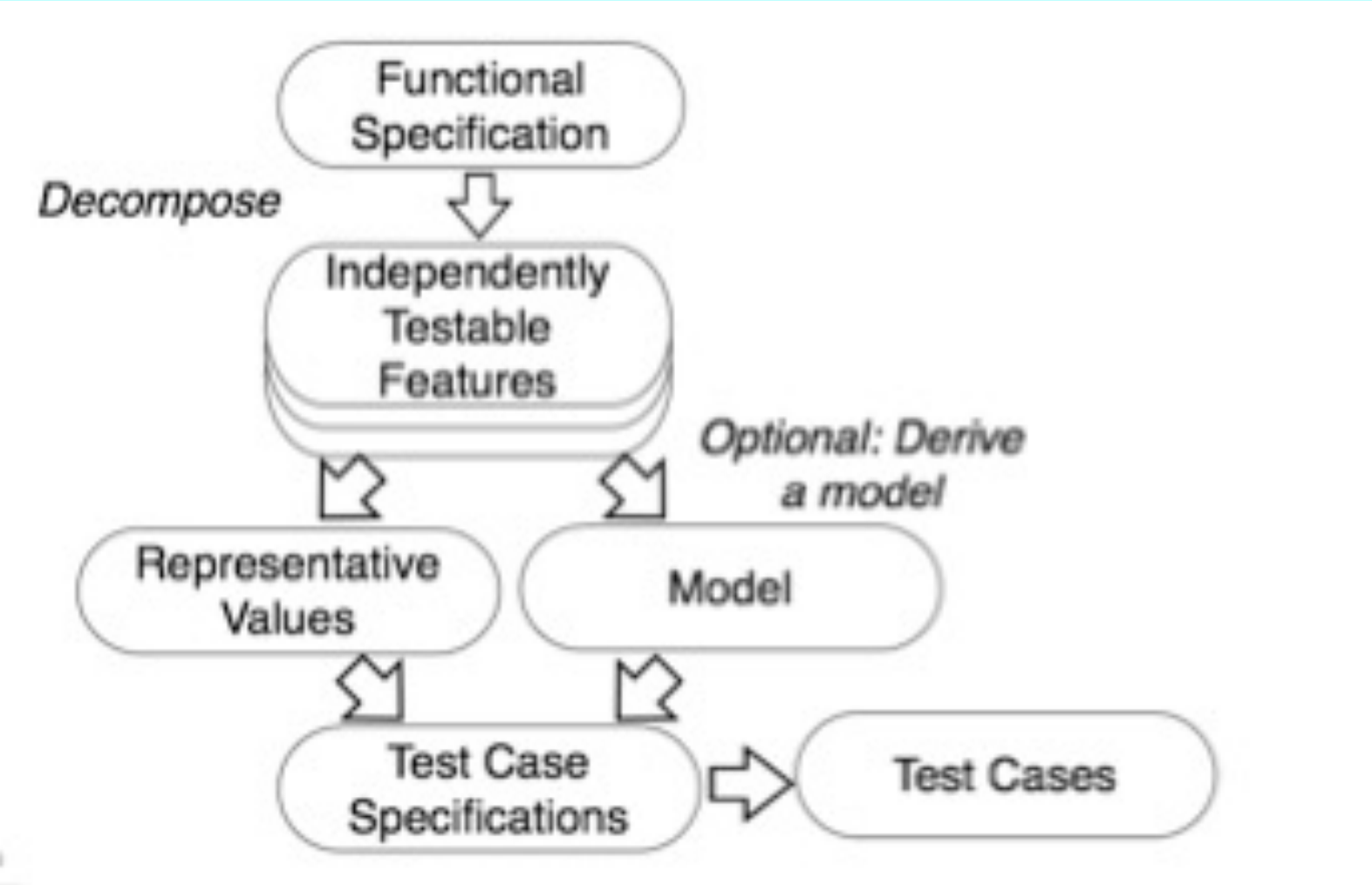
Test funzionale: sfruttare le specifiche

- I test sono condotti sulle interfacce software per verificare che
  - *I dati di input sono accettati in modo appropriato*
  - *I dati di output sono corretti*
  - *Mantenuta integrità delle informazioni esterne (es.: archivi di dati)*
- Sono esaminati alcuni aspetti del modello fondamentale del sistema, senza preoccuparsi della struttura logica interna del software

Suddivisione in classi di equivalenza

- Il dominio dei dati di input è suddiviso in **classi**
- Una classe rappresenta un insieme di stati validi o non validi per le condizioni d'ingresso
- Un test-case ideale rileva da solo una classe di fault (es. una scorretta elaborazione di tutti i dati di tipo char), che altrimenti richiederebbe l'esecuzione di molti test, prima di intuire il fault generico
- In generale una condizione di input può essere un valore specifico, un intervallo di valori, un insieme di valori, una condizione booleana

Dalla specifica ai casi di test



Pezzè & Young, Software Testing And Analysis: Process, Principles And Techniques, 2008

Dalla specifica ai casi di test

1. Decomporre la specifica
   - Se è grande, decomporla in *caratteristiche testabili indipendentemente*
2. Selezionare dei *rappresentanti*
   - Valori rappresentativi di ogni input oppure
   - Comportamenti rappresentativi di un modello
     - Spesso, semplici trasformazioni input-output non descrivono un sistema. Vengono usati modelli nella specifica del programma, nella progettazione e, nella progettazione dei casi di test
3. Formare le specifiche dei test
   - Tipicamente: combinazioni di valori di input, o comportamenti del modello
4. Produrre ed eseguire i test effettivi

Esempio: ricerca del codice postale

Pezzè & Young, Software Testing And Analysis: Process, Principles And Techniques, 2008

Input:

ZIP code (codice postale US a 5 cifre)

Output:

Lista di città

Quali sono dei valori rappresentativi (o classi di valori) da testare?

Esempio: valori rappresentativi

- ZIP code corretto:
  - Con 0, 1 o più città
- ZIP code sbagliato nel formato
  - Vuoto; 1-4 caratteri; 6 caratteri; molto lungo
  - Caratteri non cifre

Approcci al test funzionale

- Data una specifica ci possono essere più approcci per derivare i casi di test
- Ad es., la presenza di molti vincoli nel domino di input suggerisce un metodo di partizionamento con vincoli (**category-partition** testing)
  - Al contrario, valori di input con pochi o nessun vincolo suggeriscono un approccio combinatoriale (**pairwise testing**)
  - Se le transizioni tra un insieme finito di stati sono facilmente identificabili, un approccio basato su FSA può essere opportuno
  - Specifiche descritte in un dato formato suggeriscono approcci corrispondenti (basate sullo stesso formalismo)

Sommario

- Il test funzionale, i.e., la generazione dei casi di test dalle specifiche, è un valido e flessibile approccio per il testing del software
  - Applicabile dalle specifiche iniziali del sistema fino alle specifiche di modulo
- Il (quasi-)Partition testing suggerisce di dividere lo spazio degli input in classi (quasi-)equivalenti
  - Il test sistematico è intenzionalmente non uniforme per testare casi speciali, condizioni di errore, ed altri parti mirate
  - Divide un grande "pagliaio" in piccoli mucchi uniformi dove gli "aghi" possono essere concentrati