Testing e Debugging

La verifica del software

Perché? Che cosa? Quando?

- Goal: software con zero difetti …

  ma impossibile da ottenere e garantire
- Necessaria una attenta e continua **verifica**
- Tutto deve essere verificato: documenti di specifica, di progetto, dati di collaudo, ….programmi
- Si fa lungo tutto il processo di sviluppo, **non** solo alla fine!

Terminologia

- **Verifica**:
  - Insieme delle attivita volte a stabilire se il programma costruito soddisfa le specifiche (non solo funzionali)
  - "…did we build the program right?"
    - Si assume che le specifiche esprimano in modo esauriente i desiderata del committente
- **Testing**:
  - Particolare tipo di verifica sperimentale fatta mediante esecuzione del programma, selezionando alcuni dati di ingresso e valutando risultati
  - Dà un riscontro **parziale**: programma provato solo per quei dati

Terminologia

- **Prova di correttezza**:
  - Argomentare sistematicamente (in modo formale o informale) che il programma funziona correttamente per **tutti** i possibili dati di ingresso
- **Validazione o convalida**:
  - Stabilire che le specifiche sono corrette, cioé descrivono i veri requisiti dell'utente
  - *"..did we build the right program?"*
    - Può essere svolta sulla specifica (meglio!) e/o sul sistema finale

Terminologia

- **Debugging**: localizzare errori (difetti) nel codice
  - Il testing ne rivela la presenza ma non li localizza

- **Programmazione difensiva**: insieme di tecniche di programmazione che cercano di evitare di introdurre errori, aumentano probabilità di correttezza e facilitano verifica e debugging

Terminologia (IEEE)

- **Errore** (error)
  - Fattore (umano) che causa una deviazione tra il software prodotto e il programma ideale
    - Uno o più errori possono produrre uno o più difetti nel codice
  - Esempio: errore di analisi dei requisiti, progetto, battitura,...
- **Difetto** (fault)
  - Elemento del programma non corrispondente alle aspettative
    - Uno o più difetti possono causare malfunzionamenti del software
  - Esempio: il programma somma contiene un operatore di prodotto anziché un operatore di somma
- **Malfunzionamento** (failure)
  - Comportamento del codice non conforme alle specifiche
  - Esempio: il programma somma usa i dati 4 e 3 produce 12

Verifica dei programmi

- Scopo: controllo che programmi sviluppati siano conformi alla loro specifica
- Lo strumento principale che vedremo è il testing
- Per essere efficace, il testing deve essere reso sistematico

Testing

- Si fanno esperimenti con il comportamento del programma allo scopo di scoprire eventuali errori
  - Si campionano comportamenti
  - Come ogni risultato sperimentale, fornisce indicazioni **parziali** relative al particolare esperimenti
    - Programma provato solo per quei dati
- Tecnica **dinamica** rispetto alle verifiche statiche fatte dal compilatore

Testing

- Testing **esaustivo** (esecuzione per **tutti** i possibili ingressi) dimostra la correttezza
  - Esempio: se programma calcola un valore in base a un valore di ingresso nel range 1..10, il testing esaustivo consiste nel provare tutti i valori: per le 10 esecuzioni diverse si verifica se il risultato è quello atteso
- Impossibile da realizzare in generale:
  - Se programma legge 3 ingressi interi nel range 1...10000 e calcola un valore, un testing esaustivo richiede 10^{12} esecuzioni
    - Per programmi banali si arriva a tempi di esecuzione superiori al tempo passato dal big-bang

Testing

- “Program testing can be used to show the **presence** of bugs, but never to show their absence” (Dijkstra 1972)
- Quindi obiettivo del testing è di **trovare controesempi**
- Si cerca quindi di trovare dati di test che massimizzino la probabilità di scoprire errori durante l’esecuzione

Specificità del software

- Caratteristiche che rendono il test difficile
  - Molti diversi requisiti di qualità
    - Funzionali e non funzionali
  - Continua evoluzione, che richiede di ri-effettuare il test
  - Inerente non linearità e non continuità
  - Distribuzione degli errori difficile da prevedere
- Esempio (non linearità/continuità)
  - Se verifico che un ascensore riesce a trasportare un carico di 1000 kg, trasporta anche un carico inferiore
  - Se un metodo effettua correttamente il **sort** di un insieme di 256 elementi, nessuno mi assicura che funzioni anche con un insieme di 255 o 53 o 12 elementi

11

Generazione di casi di test

- È cruciale la scelta di opportuni dati di test (chiamati semplicemente **test** o **test case**) "sufficienti a convincerci" che il programma è corretto
- Devono esercitare il programma in maniera significativa
- Definiti in base a **criteri di testi**

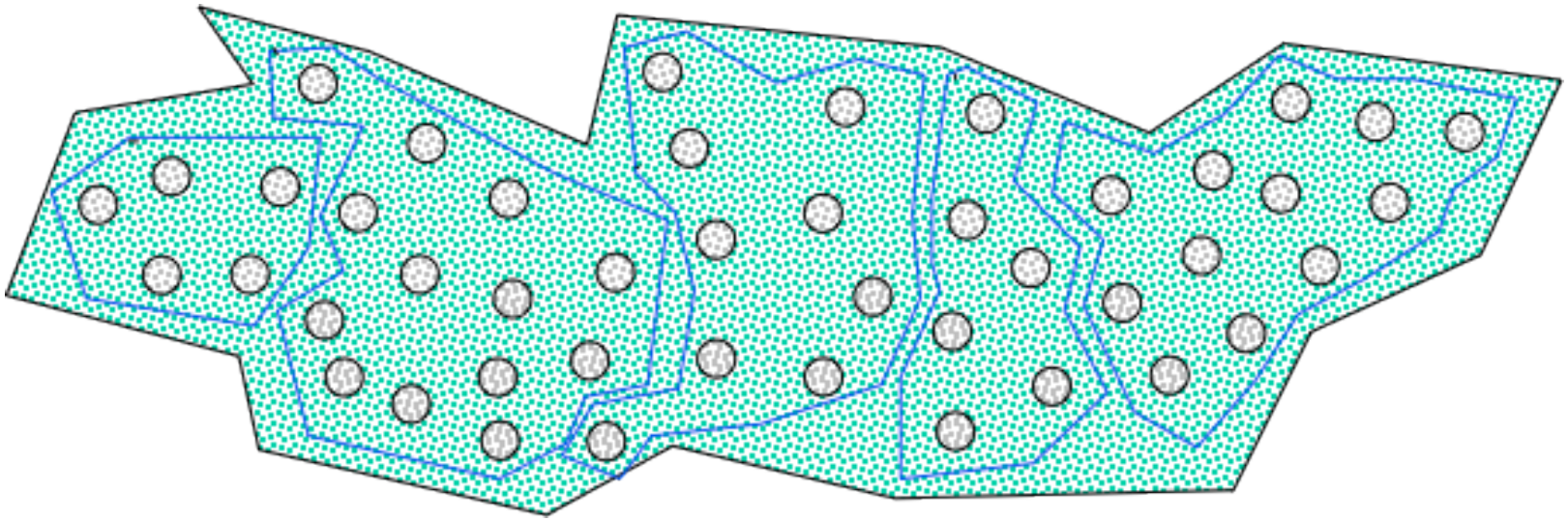
Criteri sistematici e test random

- Random
  - Casi di test generati in maniera casuale
  - Possibile pro
    - Evita le polarizzazioni del progettista
  - Contro
    - Non esplora necessariamente valori che potrebbero rilevare errori!
- Criteri sistematici
  - Effettuano esplorazioni mirate del dominio di input
    - Esempio: il metodo che calcola le radici un'equazione quadrata
      - Difficilmente il test random genererebbe dati per i casi "critici" in cui a=0, b2 - 4ac =0

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

13

Partizionamento sistematico



Si carca di partizionare il dominio di input in modo tale che da tutti i punti del dominio ci si attende lo stesso comportamento (e quindi si possa prendere come rappresentativo un punto qualuque in esso)

L'esperienza dimostra poi che è anche opportuno prendere punti sui confini delle regioni

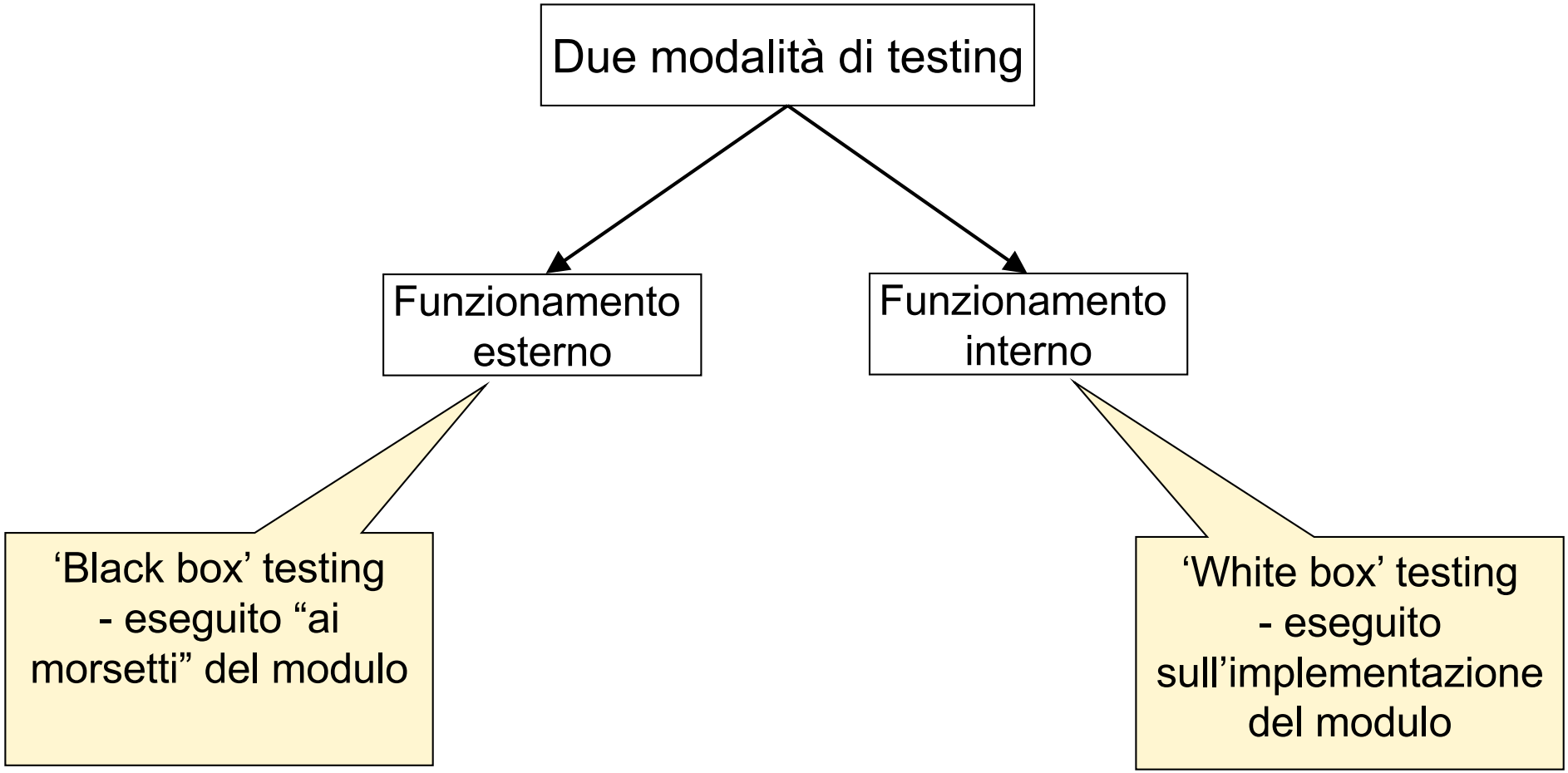
Talvolta non è una partizione in senso proprio (le classi di valori hanno intersezione non vuota)

Test black-box e white-box testing

- Black-box **funzionale**
  - Casi di test determinati in base a ciò che il componente deve fare
    - La sua **specifica**
- White-box **strutturale**
  - Casi di test determinati in base a come il componente è implementato
    - Il **codice**

15

Black box e white box testing



TEST FUNZIONALE

Test black-box

- Test funzionale usa la specifica per partizionare il dominio di input
- Esempio: la specifica del metodo per calcolare le radici di un equazione quadratica suggerisce di considerare 3 diversi casi in cui ci sono zero, una e due radici reali
  - Testare ogni “categoria”
  - Testare i confini tra le categorie
  - Nessuna garanzia, ma l’esperienza dimostra che spesso i malfunzionamenti sorgono ai “confini”

18

Utilità del test funzionale

- Non è necessario che esista il codice per determinare i dati di test
  - Basta la specifica, formale o informale
  - Nel caso di **extreme programming** i test **sono** la specifica
- Questi possono dunque essere determinati in fase di progettazione
- Useremo esempi di programmi molto banali per vedere alcune tecniche

19

Test combinatorio

- Identificare attributi che possono essere variati
  - Nei dati, nell'ambiente, nella configurazione
  - Per esempio, in un programma il browser potrebbe essere "IE" o "Firefox", il sistema operativo da scegliere potrebbe essere "Vista", "XP", or "OSX"
- Si generano in maniera sistematica le combinazioni (sensate) da testare
  - Per esempio, IE con Vista, IE con XP, Firefox con Vista, Firefox con OSX, ...
- Vediamo i tre passi da seguire…

20

Passo 1: Scomporre la specifica

Occorre dapprima scomporre la specifica in funzionalità testabili indipendemente

1. Per ciascuna feature prevista, identificare parametri, elementi dell'ambiente
2. Per ciascun parametro ed elemento dell'ambiente, identificare le caratteristiche elementari (***categorie***)

Passo 2: identificare i valori

- Identificare classi rappresentative di valori per ciascuna categoria
  - Ignorare le interazioni tra i valori di diverse categorie (vedi prossimo step)
- I valori rappresentativi si identificano in base alle seguenti classi
  1. Valori normali
  2. Valori di confine/limite (*boundary values*)
  3. Valori speciali
  4. Valori errati

22

Passo 3: Introduzione di vincoli

- Una combinazione di valori per le diverse categorie corrisponde alla specifica di un caso di test
  - Spesso metodo combinatorio genera gran numero di casi di test (gran parte dei quali magari sono impossibili)
    - Esempio: valore valido, ma non nel database
- Introdurre vincoli per
  - Eliminare combinazioni impossibili
  - Ridurre la dimensione di un insieme di test, se questo è troppo grande

23

Esempio: una singola feature

Input:
CAP (ZIP Code)

Output:
Lista di città

Quali sono le classi di valori significative per il test?

Scelta di casi significativi

- Si tratta di un semplice caso
  - Singolo input, singolo optput
- Casi significativi
  - CAP ben formato
    - Con 0, 1, o molte città
  - CAP mal formato
    - Vuoto; 1-4 caratteri; 6 caratteri; molto lungo (per generare overflow?)
    - Caratteri che non siano cifre
    - Dati che non siano caratteri

25

Esempio test combinatorio

```
/*restituisce il massimo fra x, y, z */
int maxOfThree (int x, int y, int z)
```

- Metodo delle **combinazioni**: studiare ciascuna alternativa nella specifica
  - Qui ci sono tre alternative: il massimo è x, è y, o è z
  - Casi di test ricavabili dalla specifica:
    - Un caso in cui il massimo è x, p. es. (5,3,0)
    - Un caso in cui il massimo è y, p. es. (7,11,2)
    - Un caso in cui il massimo è z, p. es. (7,10,12)

Esempi Valori limite

- Se valore dell'input può stare in un intervallo, testare estremi dell'intervallo e combinare valori limite
- Esempi:
  - Valori estremi per i numeri (max. `int` ammissibile)
  - `sqrt` con radicando = 0
  - Stringa: vuota o di 1 carattere
  - Array: array vuoto o di un elemento
  - Rlaborazioni con array: considerare valori estremi degli indici

- Esempio:

```
/*restituisce il massimo fra x, y, z */
int maxOfThree (int x, int y, int z)
```

  - x = y = z: p.es. 3, 3,3
  - x=y !=z

Altri esempi

- Triangoli identificati da vertici:
  - Tre punti allineati
  - Due punti coincidenti
  - Tre punti coincidenti
  - Triangolo rettangolo
  - Un vertice nell'origine o sugli assi
  - ….
- Valori erronei, valori speciali,…

Valori limite: Errori di aliasing

- Due parametri si riferiscono a due oggetti mutabili, dello stesso tipo
- Considerare casi in cui coincidono, anche se non previsto esplicitamente dalle specifiche

```
static void appendVector(Vector v1, Vector v2){
 // EFFECT removes all elements of v2 and appends them in reverse
 // order to the end of v1
  while (v2.size() > 0) {
   v1.addElement(v2.lastElement());
   v2.removeElementAt(v2.size()-1); }
}
```

- NON è vietato che v1 e v2 siano lo stesso Vector: testando questo caso si trova un errore