

Il Testing

Ing. Emilio Spinicci

07/04/2004

1

Introduzione

- Cenni sulla teoria del testing
- Unit Testing
 - test funzionale
 - test strutturale
 - misure di copertura
- IPL Cantata 3.3
 - Descrizione e esempi di test

07/04/2004

2

Attività di Verifica sul codice

- Verifica che il codice soddisfi le sue specifiche
- Testing = Attività mirante la scoperta di eventuali malfunzionamenti del prodotto software dovuti a “guasti” (anomalie – difetti – faults – bugs) tramite l’esecuzione del codice stesso, fornendogli opportuni dati in ingresso (analisi dinamica)
- Debugging: attività di localizzazione e rimozione dei guasti.

07/04/2004

3

Teoria del testing

Sia dato un programma P , $P: D \rightarrow R$

Dove D è lo spazio dei dati in ingresso al programma e R l’insieme dei risultati

Sia inoltre $ok()$ una funzione binaria applicata al programma sui dati d in ingresso

$ok(P,d) \Leftrightarrow P$ sul dato di ingresso $d \in D$ dà il risultato atteso

$ok(P) \Leftrightarrow \forall d \in D \ ok(P,d)$ (P è corretto)

07/04/2004

4

Teoria del testing

L'insieme dei Test T può essere definito come un sottoinsieme di D ; (*test suite*)

In particolare,

t *caso di test* se $t \in T$

P è *corretto* per un test $T \Leftrightarrow \forall t \in T \text{ ok}(P,t)$

07/04/2004

5

Teoria del testing

Scopo dell'attività di testing

- Il testing può essere visto come attività rivolta alla caccia di guasti → bug finding
- Il testing può essere visto anche come attività mirata ad aumentare la confidenza nella bontà del prodotto
→ assenza di guasti

07/04/2004

6

Teoria del testing

Nel primo caso (*bug finding*),

un test T ha *successo* su un programma P se rileva uno o più malfunzionamenti in P:

$$\text{successo}(T,P) = \neg \text{ok}(P,T)$$

un test che non rileva malfunzionamenti manifesta non la correttezza del programma, ma la sua inadeguatezza

un test T per P e' detto *ideale* se l'insuccesso di T implica la correttezza di P : $\text{ok}(P,T) \Rightarrow \text{ok}(P)$

07/04/2004

7

Teoria del testing

Nel secondo caso (*assenza di guasti*),

un test risulta *passato* se nessun malfunzionamento è stato rilevato; altrimenti, il test risulta *fallito*:

$$\text{Passed}(T,P) = \text{ok}(P,T);$$

$$\text{Failed}(T,P) = \neg \text{ok}(P,T);$$

Il Testing è un'attività che deve risultare svolta da un "ente" indipendente da quello di sviluppo.

07/04/2004

8

Criteri di selezione di test

Un **criterio di selezione** C per un programma P è un insieme di predicati (*condizioni di selezione*) sul dominio dei dati in ingresso D

Un test T è **selezionato** da C se

- $\forall t \in T$ esiste $c \in C$ tale che $c(t)$ è vero;
(ogni test case selezionato soddisfa almeno una condizione del criterio di selezione)
- $\forall c \in C$ esiste $t \in T$ tale che $c(t)$ è vero;
(ogni condizione seleziona almeno un test case)

07/04/2004

9

Criteri di selezione di test

Esempio:

$$C = \{ (\exists t \in T: t < 0), \\ (0 \in T), \\ (\exists t \in T: t > 0) \}$$

test selezionati da C : $\{-4, 0, 5\}$, $\{-2, 9, 0, 8, -12, 6\}$

Un test è **esaustivo** se è selezionato secondo il seguente criterio:

$$C = \{ (t \in D \Rightarrow t \in T) \}$$

(ogni possibile ingresso al programma è scelto come test case)

07/04/2004

10

Criteri di selezione di test

Un criterio di selezione C è *affidabile* per P se,
 \forall coppia di test $(T1, T2)$ selezionati da C ,

$$\text{successo}(T1, P) \Leftrightarrow \text{successo}(T2, P)$$

Un criterio di selezione C è *valido* per P se, qualora P non sia corretto,

\exists un test T selezionato da C tale che $\text{successo}(T, P)$

07/04/2004

11

Criteri di selezione di test

Teorema di Goodenough e Gerhart

$\text{affidabile}(C, P)$ and $\text{valido}(C, P)$
and $\text{selezionato}(C, T)$ and $\neg \text{successo}(T, P) \Rightarrow \text{ok}(P)$

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

07/04/2004

12



Livelli di test

- **Test di unità** - mirato alla correttezza degli algoritmi.
- **Test di integrazione** - mirato alla correttezza delle interfacce.
- **Test di sistema** - affidabilità, sicurezza e prestazioni.
- **Test di accettazione** - imposto dal cliente, verifica che il programma soddisfi le richieste del cliente stesso.
- **Test di regressione** - verifica che non si siano introdotti errori in versioni successive.

07/04/2004

13



Unit Testing

- **Test funzionale (black box)**
- **Test strutturale (white box)**

Variano per il principio su cui si basa il criterio di selezione dei test.

07/04/2004

14



Test funzionale (black box)

Il testing funzionale è eseguito sulla base delle specifiche dei requisiti funzionali, che stabiliscono quali funzionalità devono essere espletate dal codice sotto test.

I test cases cui il codice viene sottoposto sono progettati secondo i seguenti criteri:

- **Equivalence Class Partitioning**
- **Boundary Value Analysis**
- **Test Case by Error Guessing**

07/04/2004

15



Test funzionale (black box)

EQUIVALENCE CLASS PARTITIONING (ECPT)

Si divide il dominio di input del programma in classi con l'ipotesi che un caso di test per ciascuna classe sia rappresentativo di tutti i valori della classe. Per ciascuna condizione di input si associano almeno due classi di equivalenza, una valida ed una non valida.

BOUNDARY VALUE ANALYSIS (BVAN)

Si scelgono i casi di test in prossimità della frontiera delle classi.

TEST CASE by ERROR GUESSING (TCEG)

Tecnica ad-hoc basata sull'intuito e sull'esperienza di chi esegue il test.

07/04/2004

16

Esempio di ECPT e BVAN

Ipotesi: l'unità di codice sotto test si può comportare diversamente per valori negativi o positivi dei dati in ingresso, ma il valore effettivo non è particolarmente interessante.

Un criterio di test basato sul partizionamento può essere:

$$C = \{ (\exists t \in T: t < 0), (\exists t \in T: t \geq 0) \}$$

Cioè si testa l'unità con almeno un valore negativo e uno positivo.

→

07/04/2004

17

Esempio di ECPT e BVAN

Se vogliamo prestare attenzione ai valori di frontiera (boundary analysis), possiamo includere lo zero nel criterio di selezione:

$$C = \{ (\text{esiste } t \in T \mid t < 0), (0 \in T), (\text{esiste } t \in T: t > 0) \}$$

07/04/2004

18

Test strutturale (white box)

Il criterio di selezione dei test si deriva dalla struttura stessa del programma → (in particolare, dal grafo di flusso)

Si selezionano quei test che esercitano *tutte* le strutture del programma

Occorre decidere quali sono le strutture di riferimento: → vari tipi di testing a seconda della struttura scelta

Occorre una misura che permetta di decidere se sono state esercitate tutte le strutture → misura di copertura (**coverage**)

07/04/2004

19

Statement coverage

- I test sono selezionati in base alla loro capacità di coprire gli statement del codice (**nodi del grafo di flusso**)

- $\text{STATEMENT_COVERAGE} = \frac{\text{numero comandi eseguiti}}{\text{numero totale comandi}}$

Un test è selezionato se garantisce una copertura dei comandi uguale a 1; in questo modo si garantisce che i comandi siano eseguiti tutti almeno una volta.

07/04/2004

20

Statement coverage

La copertura degli statement non garantisce di aver percorso tutte le “strade” (rami) almeno una volta:

```
if (c)
{
    comando1;
}
comando2;
```

Un test che rende la condizione c vera esegue tutti i comandi, ma non tutti i rami

07/04/2004

21

Branch coverage

- Si richiede che in questo caso vengano percorsi dai test tutti i rami del programma almeno una volta. (ogni **arco del grafo di flusso** deve appartenere almeno ad un cammino di esecuzione esercitato dal test).

- $BRANCH_COVERAGE = \frac{\text{numero rami percorsi}}{\text{numero totale rami}}$

Un test e` selezionato se garantisce una copertura dei rami uguale a 1

07/04/2004

22

Decision coverage

- I test sono selezionati in base alla loro capacità di coprire i risultati di ciascuna decisione (True / False) nel codice; un decisione risulta coperta se entrambi i risultati booleani sono stati esercitati dai casi di test selezionati

- $DECISION_COVERAGE = \frac{\text{numero di decisioni coperte}}{\text{numero totale di decisioni}}$

La copertura delle decisioni implica la copertura dei rami.

07/04/2004

23

Condition testing

Si consideri il seguente codice:

```
if (x>1 && y==0)
{comando1}
else
{comando2}
```

ed il seguente test:

{x = 2, y =0} e { x=2, y=1}

Il test garantisce la piena copertura delle decisioni, quindi dei rami e degli statements, ma non esercita tutte le combinazioni delle due condizioni in and.

07/04/2004

24

Condition testing

**Cfr. con
programma equivalente:**

```
if (x>1)
  if (y==0) {comando1}
  else {comando2}
else {comando2}
```

In cui lo stesso test da` una copertura rami e comandi del 60%

Test precedente:

{x = 2, y =0} e { x=2, y=1}

07/04/2004

25

Condition Testing

- Si potrebbe allora pensare di coprire entrambi i risultati True/False di ogni singola condizione booleana (basic condition coverage), ma senza considerare tutte le possibili combinazioni dei due valori di verita` su tutte le condizioni basiche.

```
if (x>1 && y==0)
{comando1}
else
{comando2}
```

Nel caso precedente, i due casi di test {x = 0, y =0} e { x=2, y=1}, permettono di avere una copertura del 100% delle condizioni basiche.

Come e` evidente in questo caso, la piena copertura delle condizioni basiche non garantisce la copertura dei rami.

07/04/2004

26

Condition Testing

D'altra parte, la copertura di tutte le possibili combinazioni dei valori di verità di tutte le condizioni basiche (compound condition coverage), è pari a 2 elevato al numero di condizioni basiche, e diventa un numero eccessivo per condizioni appena complesse.

Vengono perciò utilizzati altri criteri, che considerano non solo i valori di verità delle condizioni basiche, ma anche la **decisione** finale (MCDC: Modified Condition Decision Coverage)

→ Boolean Expression Coverage

07/04/2004

27

Boolean Expression coverage

$$\text{BOOLEAN_COVERAGE} = \frac{\text{numero di espressioni booleane coperte}}{\text{numero totale di espressioni booleane}}$$

Considerando i possibili risultati di ogni singola espressione booleana costituente una decisione

OR (||)

T || ?

F || T

F || F

AND (&&)

F && ?

T && F

T && T

07/04/2004

28

Boolean Expression coverage

Si consideri il seguente codice:

```
if ((a > 10) || (a < 1))  
    x++;
```

Testando con {a = 11} (T || ?) e {a = 3} (F || F) si raggiunge una Decision Coverage del 100%, ma solo il 66% della Boolean Coverage; è necessario allora un terzo caso di test, con {a = 0} (F || T) per ottenere la copertura completa.

La Boolean Coverage è una misura di copertura più forte della Decision Coverage, e garantisce la copertura dei rami.

07/04/2004

29

Path coverage

I criteri di copertura finora visti non sempre consentono di rilevare quei guasti che vengono attivati solo da particolari cammini di esecuzione nel programma.

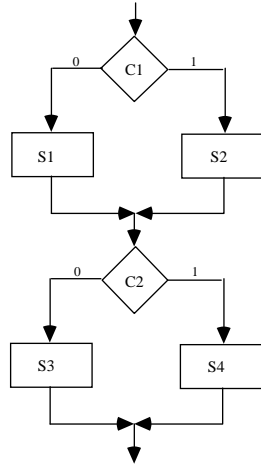
Copertura dei cammini:

$$\text{PATH_COVERAGE} = \frac{\text{numero dei cammini eseguiti}}{\text{numero totale dei cammini}}$$

07/04/2004

30

Path testing



<i>test</i>	<i>C1</i>	<i>C2</i>	<i>cammino</i>
T1	0	0	S1 S3
T2	0	1	S1 S4
T3	1	0	S2 S3
T4	1	1	S2 S4

07/04/2004

31

Path testing

- Sono sufficienti i due casi di test T1 e T4 per ottenere una copertura del 100% dei rami. Alternativamente, sono sufficienti i due casi di test T2 e T3.
- Per una copertura del 100% dei cammini sarebbero necessari tutti i casi di test T1, T2, T3, T4.

07/04/2004

32

Path testing

- Il caso precedente è il solo caso, in assenza di cicli, in cui la copertura dei cammini e la copertura dei rami differiscono; nel caso in cui si hanno due **if then else** annidati, infatti, sia i rami che i cammini sono 3.
- Si noti che il numero dei cammini è esponenziale con il numero degli if.

07/04/2004

33

Unfeasible paths

- Caso di condizioni C1 e C2 dipendenti tra loro (es. $C1=C2$). Si potranno esercitare solo due cammini, S1-S3 e S2-S4. La copertura dei cammini totali si riduce quindi al 50%, considerando i cammini sul grafo.
- I cammini possibili del programma (*feasible paths* o *achievable paths*) sono coperti al 100%.
- La possibile presenza di cammini *unfeasible* (o *Infeasible*) rende difficile la misura della path coverage;
- in generale, non può essere rilevata da un'analisi statica; anche la completa rilevazione dei cammini unfeasible attraverso un'analisi dinamica è stata dimostrata essere teoricamente impossibile.

07/04/2004

34

Unfeasible paths

- In alcuni casi, la presenza di cammini *Unfeasible* e` voluta: **Defensive Programming**: condizioni normalmente vere, possono risultare false solo in presenza di guasti hardware/software: nel caso risultino false si effettuano opportune azioni di trattamento del guasto.
- In altri casi, la presenza di cammini *Unfeasible* e` indice di disattenzione nella stesura (o nel riutilizzo) del codice.
- Copertura minore del 100% dovra` essere *giustificata* in modo da distinguere i casi di Defensive Programming da quelli di codice non sufficientemente curato.

07/04/2004

35

Path coverage in presenza di cicli

Copertura $C_t k$, con k numero di iterazioni eseguite.

Ad esempio, $k=2 \Rightarrow$ il ciclo eseguito con 0, 1, 2 iterazioni.

Copertura dei cammini arrestata al primo ciclo $C_t(k=1)$, considerando sia il cammino che non esegue le azioni interne al ciclo sia quello che le segue (almeno) una volta (è garantita la copertura dei rami).

Il ciclo **for** ha un numero di iterazioni fissato e sarà sempre eseguito esattamente quel numero di volte.

Si parla anche di criterio di **N-copertura**: copertura legata al numero di iterazioni del loop.

07/04/2004

36

Cyclomatic testing

- Si noti comunque che il numero dei cammini, anche limitando il numero delle iterazioni, rimane esponenziale con il numero di indipendenti tra loro.
- Si può considerare allora il testing cicломatico, basato sul numero di McCabe (numero di cammini indipendenti nel grafo di flusso del codice).
Il criterio risulta soddisfatto se il numero di cammini indipendenti testati raggiunge il numero cicломatico della unità sotto test.

07/04/2004

37

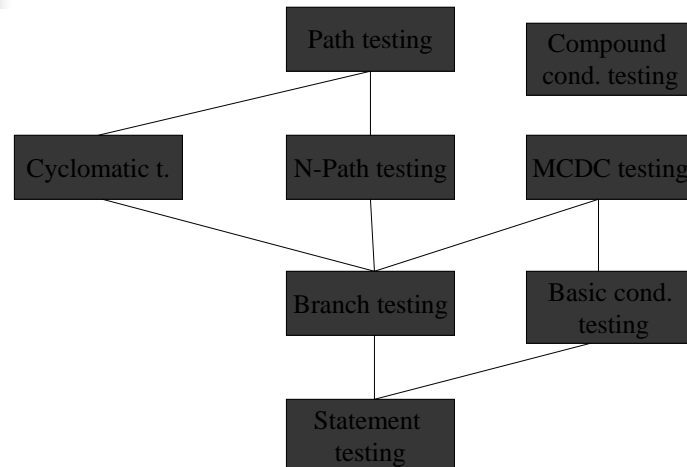
Misure di copertura

- Il concetto di copertura può essere utilizzato come metrica per valutare l'efficacia dei test effettuati: ad esempio, i casi di test T1 e T4 forniscono il 100% di copertura dei rami e il 50% di copertura dei cammini.
- Un criterio quantitativo di copertura viene spesso utilizzato come soglia, (es. 90% dei cammini) raggiunta la quale l'attività di testing viene considerata troppo costosa e viene quindi interrotta.

07/04/2004

38

Ordinamento dei criteri di copertura



07/04/2004

39

Unit testing

- **Stub**
Modulo fittizio che presenta la stessa interfaccia di un modulo invocato dall'unità sotto test.
- **Drivers**
Modulo che ha la funzione di invocare l'unità sotto test passandole i casi di test individuati.
- **Oracolo**
Può essere un programma, ma anche l'utente stesso, e rappresenta l'entità che decide se il test è passato o fallito.
- **Automazione del testing: strumenti di supporto alle attività di costruzione di Drivers, Stubs, e Oracoli**
→ Cantata, Adatest, Logiscope,.....

07/04/2004

40

Altre tecniche di verifica

- Analisi statica
 - Analisi lessicale
 - Analisi sintattica
 - Controllo dei tipi
- Ispezione/ Review

07/04/2004

41

IPL Cantata

- Cantata è un tool per l'analisi statica / dinamica di codice C, sviluppato da IPL sulla base di un tool analogo per codice ADA (AdaTest);
- Si tratta di un tool eseguibile dal prompt dei comandi, ed è costituito da più componenti.

07/04/2004

42

IPL Cantata

- CTP (Cantata Preprocessor)

Risolve le direttive #include ed espande le macro

- CTI (Cantata Instrumenter)

Inserisce nel codice le direttive per l'analisi statica e dinamica

- CTS (Cantata Test Script Generator)

Costruisce un sorgente .C relativo ai casi di test selezionati

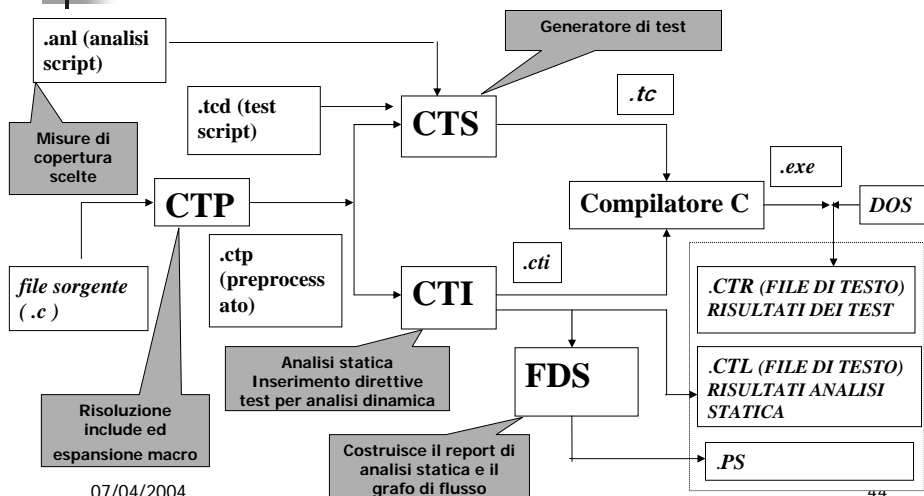
- FDS (FlowGraph Drawing System)

Produce il report finale relativo al test in formato PostScript

07/04/2004

43

Struttura del tool



07/04/2004

44

IPL Cantata

- E' necessario costruire un ambiente batch che esegua in sequenza:

CTP → CTI → CTS →
COMPILAZIONE → ESEGUIBILE
DI TEST → FDS

In modo da ottenere automaticamente i risultati finali.

07/04/2004

45

Unit Testing con Cantata

- Test di ogni singola funzione (isolata tramite compilazione condizionale `#ifdef`);
- Testing funzionale
Sulla base dei documenti di specifica dei requisiti funzionali, si progettano opportuni casi di test, verificando che il codice restituisca gli output attesi per ciascuna funzionalità (Test Black Box);
- Testing strutturale
Tramite la scelta delle misure di copertura;

07/04/2004

46

Cantata: misure di copertura

- STATEMENT_COVERAGE;
- DECISION_COVERAGE; (misure ottenute automaticamente)
- BOOLEAN_COVERAGE;
- PATH_COVERAGE
I cammini devono essere individuati dal tester etichettando i rami che li costituiscono; i cammini da verificare sono quindi sequenze di etichette che devono essere percorse nei casi di test.

07/04/2004

47

Realizzazione dei casi di test

- Costruzione del file .TCD
Il file TCD contiene la codifica, tramite direttive di Cantata, dei casi di test per l'analisi funzionale;
- Principali direttive
 - %%CASE < n > → individua il caso di test <n>
 - %%CALL <nome_funzione> < stub_sequence >
→ indica la chiamata alla funzione da testare, con l'indicazione delle STUB delle funzioni esterne che verranno invocate

07/04/2004

48

Realizzazione dei casi di test

%%INPUTS

< nome parametro ... input >

→ Indica i parametri in ingresso alla funzione o alla stub ed valori assegnati;

%%OUTPUTS

< nome parametro ... output >

→ Indica per ciascun parametro, compreso il valore di ritorno della funzione, quale deve essere il valore atteso in uscita alla funzione

%%STUB

→ Definisce la STUB di una funzione esterne che viene richiamata

%%ACTION < n >

→ Definisce l'azione della STUB che deve essere eseguita, indicata in %%CALL

07/04/2004

49

Esempi di casi di test

- Software C per la gestione di strumentazione in ambito ferroviario
- Test richiesto per conformità alla normativa CENELEC EN 50128

Si definiscono 5 livelli di Software Safety Integrity Level (S.I.L.), da 0 a 4;

Maggiore è il SIL, più forti e restrittive sono le misure di copertura richieste.

07/04/2004

50