

Integrazione e Test di Sistemi Software

Test strutturali e copertura del codice

Azzurra Ragone

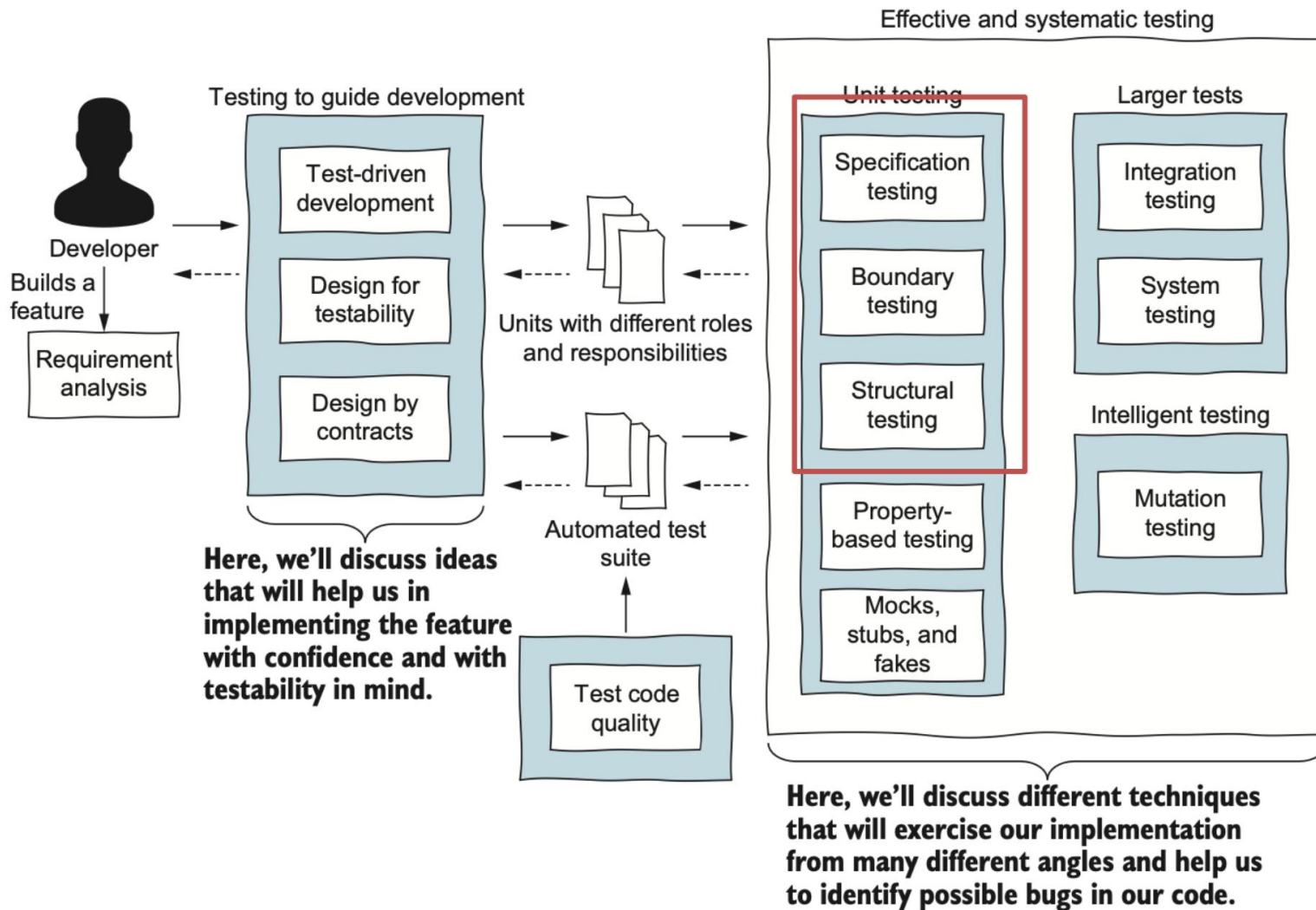
Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 | Fax: +39.080.5442536
serlab.di.uniba.it



Che cosa sono i test strutturali?



Test efficaci e sistematici (flusso di lavoro)



Test strutturali: creare casi di test basati sulla struttura del codice



Test basati sulle specifiche vs test strutturali

Test basati sulle specifiche :

Test della scatola nera

I requisiti guidano i test

contro

Test strutturali :

Test della scatola bianca

Il codice sorgente guida i test (e i criteri di copertura del codice)



Copertura del codice: un esempio

Specifica: *Data una frase, il programma deve contare il numero di parole che terminano con "s" o "r". Una parola termina quando compare una lettera diversa dalla lettera "s". Il programma restituisce il numero di parole.*

Esempio: *“I gatti e i cani sono innamorati”*



Implementazione di CountWords

```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' '  
  
        for (int i = 0; i < str.length(); i++) {  
  
            if (!isLetter(str.charAt(i)) &&  
                (last == 's' || last == 'r')) {  
                words++;  
            }  
  
            last = str.charAt(i);  
        }  
  
        if (last == 'r' || last == 's') {  
            words++;  
        }  
  
        return words;  
    }  
}
```

Loops through
each character
in the string

If the current character is a non-
letter and the previous character
was “s” or “r”, we have a word!

Stores the current
character as the
“last” one

Counts one more
word if the string
ends in “r” or “s”



Implementazione di CountWords

```
@Test
void twoWordsEndingWithS() {
    int words = new CountLetters().count("dogs cats");
    assertThat(words).isEqualTo(2);
}
```

Two words ending in “s”
(dogs and cats): we expect
the program to return 2.

```
@Test
void noWordsAtAll() {
    int words = new CountLetters().count("dog cat");
    assertThat(words).isEqualTo(0);
}
```

No words ending in “s”
or “r” in the string: the
program returns 0.

Ci sfugge
qualcosa?

Nei test strutturali possiamo
identificare quale parte del codice la nostra
suite di test non esercita... e scrivere
nuovi casi di test utilizzando uno strumento
di copertura del codice



Strumento di copertura del codice

Verde: linea *completamente* coperta dalla suite di test

Giallo: la linea è *parzialmente* coperta

Rosso: la linea *non* è coperta

Diamonds indicate that this is a branching instruction and there may be many cases to cover.

```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' ';  
        for (int i = 0; i < str.length(); i++) {  
            if (!Character.isLetter(str.charAt(i)) && (last == 's' || last == 'r')) {  
                words++;  
            }  
            last = str.charAt(i);  
        }  
        if (last == 'r' || last == 's')  
            words++;  
        return words;  
    }  
}
```

The color indicates whether the line is covered.



Strumento di copertura del codice

Riga 8 (se): 1 di 6 rami mancanti

Riga 13 (se): 1 di 4 rami mancanti

CountWords.java

```
1. package ch3;
2.
3. public class CountWords {
4.     public int count(String str) {
5.         int words = 0;
6.         char last = ' ';
7.         for (int i = 0; i < str.length(); i++) {
8.             if (!Character.isLetter(str.charAt(i)) && (last == 's' || last == 'r')) {
9.                 words++;
10.            }
11.            last = str.charAt(i);
12.        }
13.        if (last == 'r' || last == 's')
14.            words++;
15.        ds;
16.    }
17. }
```

1 of 4 branches missed. ds;



Test delle parole che terminano con 'r'

```
@Test
void wordsThatEndInR() {
    int words = new CountWords().count("car bar");
    assertThat(words).isEqualTo(2);
}
```

Words that end in “r”
should be counted.




Ora possiamo rieseguire il
Strumento di copertura del codice, ogni
riga dovrebbe ora essere coperta,
altrimenti ripeteremo il processo



Strumento di copertura del codice

La suite di test ora raggiunge la copertura completa di rami e condizioni

All lines are green, which means all lines and branches of the method are covered by at least one test case.



```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' ';  
        for (int i = 0; i < str.length(); i++) {  
            if (!Character.isLetter(str.charAt(i)) && (last == 's' || last == 'r')) {  
                words++;  
            }  
            last = str.charAt(i);  
        }  
        if (last == 'r' || last == 's')  
            words++;  
        return words;  
    }  
}
```



Flusso di lavoro di test per test strutturali

1. Eseguire test basati sulle specifiche (approccio in 7 fasi)
2. Leggi l'implementazione: comprendi il codice (se non l'hai codificato tu)
3. Eseguire la suite di test con uno strumento *di copertura del codice* (per identificare in modo automatizzato parti non coperte)
4. Per ogni pezzo di codice “non coperto”:
 - a. Perché non è stato coperto?
 - b. Decidi se quel pezzo di codice necessita di un test (in caso affermativo vai a c.)
 - c. Implementare un caso di test automatizzato
5. Torna al punto 3

~~La suite di test strutturali integra la suite di test ideata tramite le specifiche-
test basati~~



Criteri di copertura del codice

Cosa significa esattamente "coprire una riga di codice"?

```
se (!Character.isLetter(str.charAt(i)) && (ultimo == 's' || ultimo == 'r'))
```

Uno sviluppatore può applicare criteri diversi:

- 1 – **Copertura della linea**: la linea è considerata “coperta” anche se un singolo test passa attraverso la linea if (1 caso di test)
- 2 – **Copertura dei rami**: l'istruzione if può essere valutata come vera o falsa (2 casi di test)
- 3 – **Condizione + copertura dei rami**: esplora ogni condizione nell'istruzione if: qui abbiamo 3 condizioni che richiedono ciascuna 2 test ($3 \times 2 = 6$ test)
- 4 – **Copertura del percorso**: copre ogni possibile percorso di esecuzione di questa istruzione ($2^3 = 8$ casi di test)



Copertura della linea

Uno sviluppatore ottiene questo risultato se almeno un caso di test copre la linea sottoposta a test

Se quella riga contiene un'istruzione if complessa con più condizioni, non importa: un singolo test è sufficiente per considerare quella riga come coperta.

$$\text{line coverage} = \frac{\text{lines covered}}{\text{total number of lines}} \times 100\%$$



Copertura delle filiali

Quando abbiamo **istruzioni di diramazione** (if, for, while, ecc.) che fanno sì che il programma si comporti in modi diversi, a seconda di come viene valutata l'istruzione

$$\text{branch coverage} = \frac{\text{branches covered}}{\text{total number of branches}} \times 100\%$$

Esempio: `if(a && (b || c))`

Quanti test sono necessari per raggiungere la copertura delle filiali?



Condizione + Copertura della filiale

Considera *non solo le possibili diramazioni* , ma anche **ogni condizione** di ogni istruzione di diramazione.

La suite di test dovrebbe esercitare: -

ciascuna di queste condizioni individuali deve essere valutata come vera e falsa almeno una volta

- l'intera istruzione di diramazione è vera e falsa almeno una volta.

$$c+b \text{ coverage} = \frac{\text{branches covered} + \text{conditions covered}}{\text{number of branches} + \text{number of conditions}} \times 100\%$$



Condizione + Copertura della filiale

Considera *non solo le possibili diramazioni* , ma anche **ogni condizione** di ogni istruzione di diramazione.

La suite di test dovrebbe esercitare: -

ciascuna di queste condizioni individuali deve essere valutata come vera e falsa
almeno una volta

- l'intera istruzione di diramazione è vera e falsa almeno una volta.

Esempio: `if(A || B)`

T1: A = vero, B = falso

T2: A = falso, B = vero

Questi due test sono sufficienti?



Condizioni + Copertura di Filiale: esempi

Esempio: `if(A || B)`

T1: A= vero, B = vero

T2: A = falso, B = vero

Calcola il valore della copertura c+b (2 minuti)

$$c+b \text{ coverage} = \frac{\text{branches covered} + \text{conditions covered}}{\text{number of branches} + \text{number of conditions}} \times 100\%$$



Condizioni + Copertura di Filiale: esempi

Esempio: `if(A || B)`

T1: A= vero, B = vero

T2: A = falso, B = vero

Calcola il valore della copertura c+b (2 minuti)

Ramo = 1/2

Condizione = 3/4

Copertura C+b = $(1+3)/(2+4) * 100 = 66,6\%$

$$c+b \text{ coverage} = \frac{\text{branches covered} + \text{conditions covered}}{\text{number of branches} + \text{number of conditions}} \times 100\%$$



Condizioni + Copertura di Filiale: esempi

Esempio:

if(A || B)

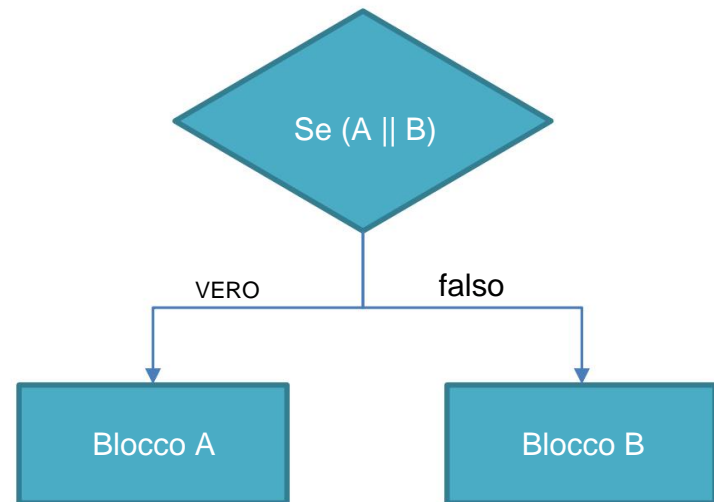
{Blocco A}

altro

{Blocco B}

T1: A = vero, B = falso

Calcola il valore della copertura di
linea, diramazione e c+b (3 minuti)



Condizioni + Copertura di Filiale: esempi

Esempio:

if(A || B)

{Blocco A}

altro

{Blocco B}

T1: A = vero, B = falso

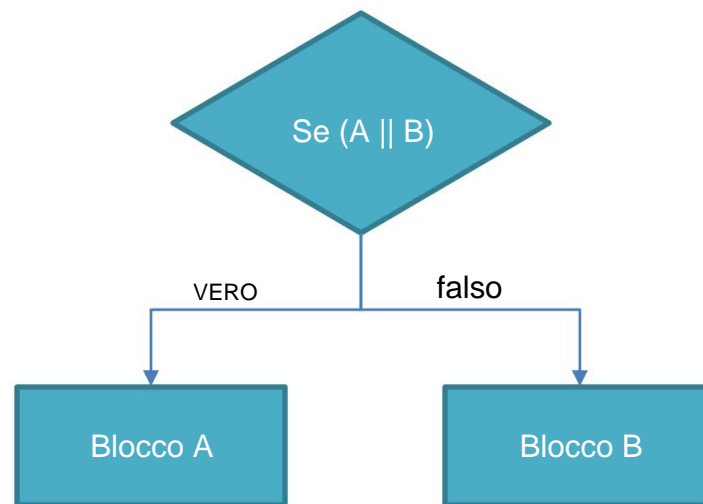
Calcola il valore della copertura di linea,
diramazione e c+b (3 minuti)

Linea = $2/3 = 66,6\%$

Ramo = $1/2 = 50\%$

Condizione = $2/4$

Copertura C+b = $(1+2)/(2+4) * 100 = 50\%$



L



Copertura del percorso

Quando si coprono tutti i possibili percorsi di esecuzione del programma.

Questo è il criterio **più forte** , ma spesso **impossibile** o troppo **costoso** da raggiungere

In un programma con n condizioni, dove ciascuna potrebbe essere valutata vera o falsa, abbiamo 2^n percorsi da percorrere

Esempio: se un programma ha 10 condizioni, il numero totale di combinazioni sarà $2^{10} = 1024$. Ciò significa più di mille test!



Quali criteri di copertura scegliere

Dipende...

Compromesso tra: -

massimizzare il numero di **bug** trovati -

minimizzare lo **sforzo/costo** di creazione della suite di test

Esiste un buon compromesso tra la copertura del percorso (troppo costosa) e la copertura delle condizioni+rami?



Criterio di copertura MC/DC

La copertura delle condizioni/decisioni modificate (MC/DC) è una buona risposta.

Il criterio MC/DC esamina **le combinazioni** di condizioni, come fa la copertura del percorso.

Invece di testare *tutte* le possibili combinazioni, MC/DC identifica le combinazioni *importanti* che devono essere testate.

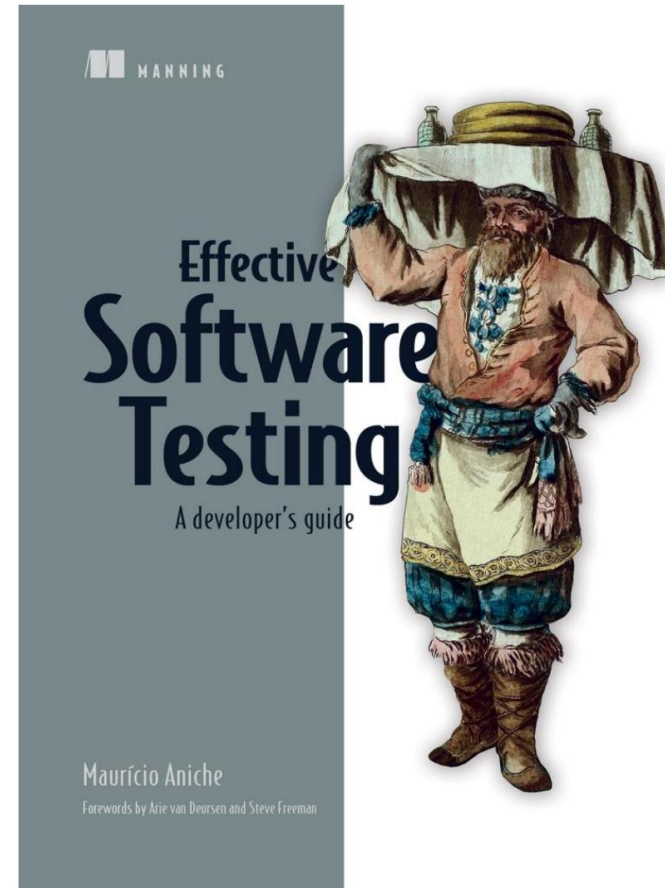
Se le condizioni hanno solo esiti **binari** (ovvero, *vero* o *falso*), il numero di test necessari per raggiungere una copertura MC/DC del 100% è $N + 1$, dove N è il numero di condizioni nella decisione.

Nota che $(N+1) \ll 2^N$



Libro di riferimento:

Test software efficaci. Guida
per sviluppatori. Mauricio Aniche.
Ed. Manning. **(Capitolo 3)**



Riferimenti

- Documentazione sulla copertura del codice JetBrains IntelliJ IDEA:
<https://www.jetbrains.com/help/idea/code-coverage.html>

L





Azzurra Ragone

Dipartimento di Informatica - Piano VI - Stanza 616 Email:
azzurra.ragone@uniba.it