
 UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## A bug


```
void count(int *counter)
{
    *counter++;
}
```

*Where is the bug?*



13/10/22

9


 UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## A bug

Correctness of a program is always related to its intended function.

**Correctness w.r.t. a specification**

*What is a bug?*  
*How many bugs are in a program?*  
*Which are the effects of a bug?*  
*Can I write a program with no bugs?*  
*How much this costs?*  
*Can I verify that there are no bugs in a program?*  
*How much this costs?*



13/10/22

10

## Correctness with respect to a Specification

### Specification

- suppose your program  $T$  has to compute the function  $f(x)$ , with  $x \in D$ :
- the specification  $S_T$  of  $T$  is:  $T(x) = f(x) \forall x \in D$

### Correctness

- We can say that  $T$  is **correct** with respect to  $S_T$  iff it actually occurs that  $T(x) = f(x) \forall x \in D$ .

### Fault (Bug)

- We have a **Fault** if  $\exists \bar{x} \in D : T(\bar{x}) \neq f(\bar{x})$

2 / 4



## Correction of a Fault

### Correction of the Program

- suppose you know a fault occurs when  $T$  is applied to  $\bar{x}$ :
- find/build a new Program  $T'$  such that:
  - $T'(x) = T(x) \forall x \in D - \{\bar{x}\}$
  - $T'(\bar{x}) = f(\bar{x})$
- restrict  $T$  to  $T'$ , defined only on  $D - \{\bar{x}\} : T'(x) = f(x) \forall x \in D - \{\bar{x}\}$

3 / 4

## Multiple Faults

### Multiple faults and iterative corrections

- We cannot exclude that there is more than one bug in  $T$ : a second, not corrected, bug remains in  $T'$  as well;
- The set  $F_T$  of faults of  $T$ , can be defined as  

$$F_T = \{x \in D : T(x) \neq f(x)\}$$
- The number  $N_T$  of bugs in  $T$  is defined as  $N_T = |F_T|$
- A correction  $T'$  of  $T$  therefore satisfies:  $N_{T'} = N_T - 1$
- $N_T$  decreases to zero if enough corrections are made
- $N_T$  is not known a priori...

### Probabilistic correctness

- $T$  is correct with probability  $p_T$  if  $p_T = |F_T|/|D|$

4 / 4

11

UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Il Testing

- Tecnica di **verifica dinamica**, basata sulla sperimentazione:

la correttezza di una **Implementation Under Test** (IUT) viene accertata esercitandola e comparandone il funzionamento rispetto alle attese.


Verifica se, per ogni  $x \in C$ , con  $C \subseteq D$ ,  $T(x) = f(x)$

- *in opposizione a una tecnica statica nella quale le proprietà dell'implementazione sono derivate attraverso un processo di analisi (**analisi statica**)*

- *Tecnica di verifica più comune e più usata industrialmente*

13/10/22

12





UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Il Testing

Scopi:


- prospettiva formativa: identificare e rimuovere difetti
- prospettiva di valutazione: acquisire una ragionevole confidenza circa l'assenza di difetti residui non riscontrati nel corso dello sviluppo,

→ senza però mai poterne garantire l'assenza  
(concetto anche noto come **Tesi di Dijkstra**).

13/10/22

13



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE


## Il Testing

Test case selection

attività di selezione dei **casi di test** ( $C$ ) che si ritengono adeguati a individuare difetti o che in assenza di fallimenti riscontrati si ritengono adeguati a garantire con accettabile confidenza l'assenza di difetti residui (in genere in riferimento a qualche **criterio di copertura**)


L'insieme dei casi di test costituisce una **test-suite**, che è tipicamente costruita attraverso un **criterio di selezione**.

(Criterio di selezione **ideale**: **seleziona C**: se per ogni  $x \in C$ ,  $T(x) = f(x)$   
allora per ogni  $x \in D$ ,  $T(x) = f(x)$ )



Test-suite, (o criterio di selezione) caratterizzata da:

- **fault-detection-capability** (definisce quali sono i faults **che generano una failure osservabile nell'esercizio della suite**);
- **complessità** (definisce quanti casi di test compongono la suite)



13/10/22

14



### Input selection (sensitization)

identifica gli input che fanno sì che il sistema esegua il test case che abbiamo selezionato;  
(in un sistema in tempo reale consiste anche nel decidere a che istante far arrivare l'input).

La **sensitization** è in generale **un problema non decidibile** essendo evidentemente riducibile al problema della terminazione.

*Nonostante siano due momenti diversi, test-case selection e sensitization sono fortemente correlate:*

- la test case selection è in generale organizzata come la selezione di una test-suite che realizza un qualche **criterio di copertura**;
- lo stesso criterio può essere soddisfatto con test-suites diverse;
- può avvenire che i casi di una test-suite siano più difficili da sensitizzare rispetto a quelli di un'altra;
- per questo è utile che la sensitization possa produrre un feedback sulla test-case selection.

13/10/22

15



Per vari aspetti **test case selection** e **coverage analysis** si riducono ad uno stesso problema, ma uno opera **a priori** e l'altro **a-posteriori**.

Entrambi fanno riferimento ad una **astrazione dell'unità sotto test**

**Prospettiva funzionale (black box):** astrazione = **specificazione dei requisiti**,

**Prospettiva strutturale (white box):** astrazione = **struttura del sistema**:


- Control Flow Graph del codice sorgente,
- Control Flow Graph del codice compilato,
- Diagramma UML delle classi dell'implementazione, ...

**Prospettiva architetturale (grey box):** astrazione = struttura della combinazione di componenti software, ciascuno visto come black box.

Approccio tipicamente ritenuto più virtuoso:  
**test case selection in prospettiva funzionale e coverage analysis in prospettiva strutturale.**

13/10/22

16



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Requirement-based testing

**TEST FUNZIONALE (BLACK BOX)**  
*Requirement based testing*


Il criterio di selezione si basa sulle specifiche funzionali dell'unità sotto test, ad esempio attraverso:

*EQUIVALENCE PARTITIONING*

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 invalida.


*BOUNDARY VALUE ANALYSIS*

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



Testing - Informatica  
Industriale - pag. 17

17



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Requirement-based testing

Ogni requisito deve essere testato:


Esiste almeno un test per ogni requisito

- ⇒ **tracciabilità** dei requisiti ai test.
- ⇒ **100% copertura** dei requisiti

*Unfortunately, a test set that meets requirements coverage is not necessarily a thorough test of the software, for several reasons:*

- *the software requirements and the design description (used as the basis for the test set) may not contain a complete and accurate specification of all the behavior represented in the executable code;*
- *the software requirements may not be written with sufficient granularity to assure that all the functional behaviors implemented in the source code are tested; and,*
- *requirements-based testing alone cannot confirm that the code does not include unintended functionality.*

(NASA/TM-2001-210876A Practical Tutorial on Modified Condition/ Decision Coverage)



Testing - Informatica  
Industriale - pag. 18

18



## Testing strutturale: control flow testing

- Il control flow testing mira a esercitare la IUT in modo da coprire i diversi percorsi del suo flusso di controllo.
- Per questo la IUT è **astratta** in un **control flow graph** (CFG) che rappresenta un insieme di locazioni logiche della IUT e la relazione di transizione tra le locazioni.
- Tipicamente: un nodo astrae uno statement, o un blocco di statement consecutivi:
- La struttura del grafo è data principalmente dai punti di decisione
- L'approccio si presta in modo naturale ad una interpretazione in chiave **strutturale**



13/10/22

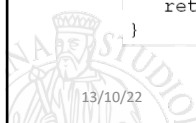
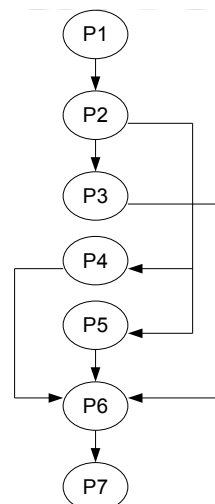
19



## Testing strutturale: control flow testing


```

int myfunct(int,int);
int myfunct(int a, int b){
    int result;
    /* P1 */
    /* P2 */
    if(a>b){
        result=a;
    /* P3 */
    }
    else if(a<b){
        result=b;
    /* P4 */
    }
    else{
        result=a*b;
    /* P5 */
    }
    /* P6 */
    /* P7 */
    return result;
}
  
```



13/10/22

20



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Criteri di selezione dei test.

### All Nodes (All-Statements, Statement Coverage)

Il criterio è soddisfatto quando sono stati coperti tutti i nodi.


**Complessità massima  $O(N)$**  rispetto al numero  $N$  dei vertici del grafo

### All Edges (All-Decisions, Decision Coverage, Branch Coverage)


Oltre a richiedere la visita di tutti i nodi, richiede anche l'attraversamento di tutti gli archi.

**Complessità massima  $O(Nc)$** , dove  $c$  è il massimo grado di uscita (fan-out) di ciascun vertice (per IF, WHILE, ...  $c = 2$ )



13/10/22

21



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Criteri di selezione dei test.

### All Paths (Path Coverage)

Richiede la copertura di tutti i possibili cammini sul grafo.

**Complessità massima  $O(2^N)$**  in assenza di cicli,  
 $O(\infty)$  se ci sono cicli.

*Ovviamente è interessante il numero dei cammini effettivi – nel caso del classico FOR è fissato - ma in generale la determinazione di questo numero (finito o infinito) è un problema indecidibile.*

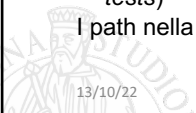
  

### Boundary Interior

Il boundary-interior testing considera due classi di path per ogni gruppo di path simili rispetto ad ogni ciclo.

I path della prima classe entrano nel loop ma non iterano su di esso (*boundary tests*)

I path nella seconda classe iterano sul ciclo almeno una volta (*interior tests*).



13/10/22

22





## Focus sulle condizioni

L'astrazione del CFG non consente di esercitare tutte le condizioni composte. Occorre astrazione più fine che prenda in carico le condizioni composte.

### All Conditions (Basic Condition Coverage)

Una **condizione** è un'espressione che non contiene operatori booleani.

Il criterio **all-conditions** richiede che tutte le condizioni di tutti i predicati siano state testate con esito TRUE e FALSE. Le condizioni sono però **testate indipendentemente**, per cui non è garantito che alla fine il predicato sia testato con esito sia TRUE che FALSE.

Es. `if(x>10 && y<3)`

In fase di sensitizzazione ciascun test case è associato ai valori di input che lo esercitano:

Test case	x>10	y<3	Input
1	T	F	<x==20,y==4>
2	F	T	<x==5,y==1>

13/10/22

23



## Focus sulle condizioni

### Condition Decisions

E' definito come l'**unione** di all conditions e all decisions, ovvero è ottenuto come unione di due test suites che soddisfano separatamente i due criteri.

Test case	x>10	y<3	x>10&& y<3	Input
3	T	T	T	<x==20,y==1>
4	F	F	F	<x==5,y==4>

Per costruzione condition-decision **include all-decisions**.

Ha complessità **O (cNk)**, dove k è il massimo numero di condizioni in una guardia.

13/10/22

24



## Focus sulle condizioni

### Multiple Conditions (MCC)

Considera **tutte le combinazioni** di tutte le condizioni.

Test case	$x > 10$	$y < 3$	$x > 10 \& y < 3$	Input
1	T	F	F	$\langle x == 20, y == 4 \rangle$
2	F	T	F	$\langle x == 5, y == 1 \rangle$
3	T	T	T	$\langle x == 20, y == 1 \rangle$
4	F	F	F	$\langle x == 5, y == 4 \rangle$

Multiple conditions implica tutti gli altri ma comprende un numero di casi che è esponenziale rispetto al numero di condizioni:  $O(cN \cdot 2^k)$



13/10/22

25



## MCDC

### Modified Condition Decision Coverage (MCDC)

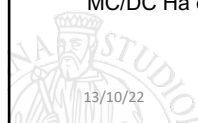
**Modified Condition/Decision Coverage (MC/DC)** è raccomandato nello standard RTCA DO178B applicato nello sviluppo di airborne-SW (SW avionico)

Il criterio stabilisce che:

- i) ciascuna decisione deve avere avuto i possibili esiti True e False;
- ii) ciascuna condizione deve avere avuto i possibili esiti True e False;
- iii) ciascun entry e exit point è stato invocato (il che esula dagli argomenti ora trattati);
- iv) ciascuna condizione ha determinato in maniera indipendente l'esito della decisione.

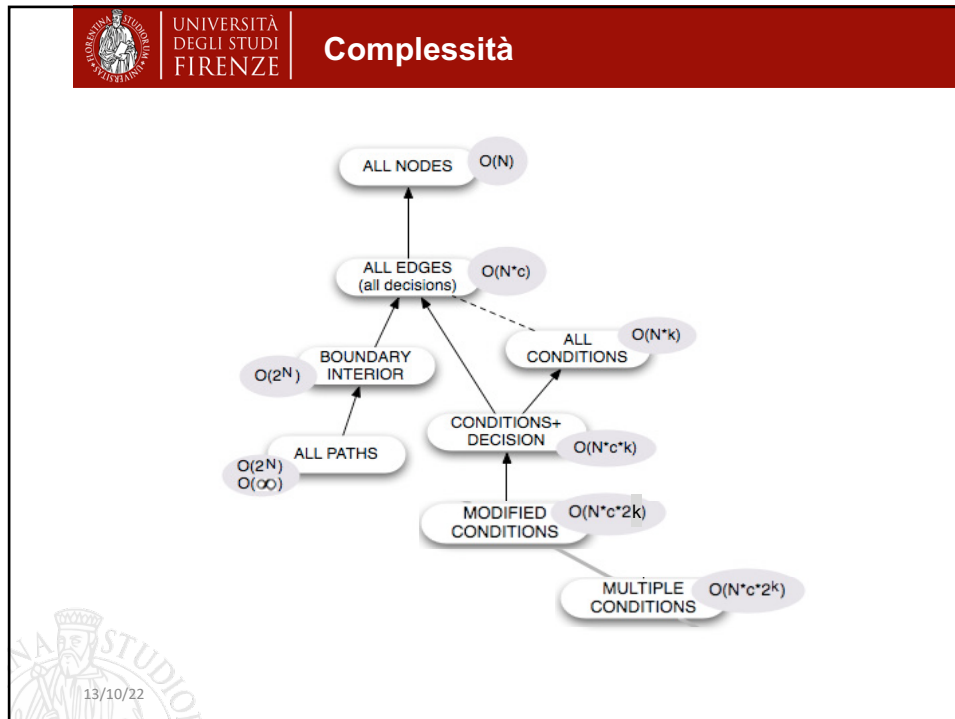
La condizione iv) in sostanza prescrive che ciascuna condizione è stata testata con esito True e False in condizioni che producono diversi esiti della decisione complessiva. Questo ovviamente implica le condizioni i) e ii).

MC/DC Ha complessità  $O(cN \cdot 2K)$



13/10/22

26



27

UNIVERSITÀ DEGLI STUDI FIRENZE

## Requisiti di copertura - DO-178C

Level	Coverage	Explanation
Level A	MCDC	Level B + 100% Modified Condition Decision Coverage
Level B	DC	Level C + 100% Decision Coverage
Level C	SC	Level D + 100% Statement (or Line) Coverage
Level D		100% Requirements Coverage Requirements
Level E		No Coverage Requirements

Testing - Informatica Industriale - pag. 28

28

UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

## Valutazione della copertura

*Sono state proposte altre astrazioni del programma sotto test che permettono di definire criteri di selezione e misure di coverage diverse (coppie uso-definizioni di variabili → data-flow analysis)*

Coverage analysis utile non solo per determinare la completa copertura, Ma anche valutare l'efficacia del testing svolto fino ad un certo punto:

Requirement based test case selection + coverage analysis

Un caso limite è l'approccio di **random testing (testing statistico)** nel quale gli inputs sono generati in maniera casuale, delegando alla fase di coverage analysis una valutazione a posteriori dell'efficacia dei test svolti.

→ Testing come attività quantificabile in termini di fault residui...



13/10/22