Testing

Fondamenti teorici

Riferimenti

- Ian Sommerville, Ingegneria del Software, capitoli 22-23-24 (più dettagliato sui processi)
- Pressman, Principi di Ingegneria del Software, 5° edizione, Capitoli 15-16
- Ghezzi, Jazazeri, Mandrioli, Ingegneria del Software, 2° edizione, Capitolo 6 (più dettagliato sulle tecniche)

Verifica e Validazione del Software

- L'attività di Verifica e Validazione (V & V) punta a mostrare che il software è conforme alle sue specifiche (Verifica) e soddisfa le aspettative del cliente (Validazione).
  - *Verifica: Are we making the right product?*
  - *Validazione: Are we making the product right?*
- Due approcci complementari alla verifica:
  - Approccio **sperimentale** (test, o analisi dinamica, del prodotto)
  - Approccio **analitico** (analisi statica del prodotto e della sua documentazione)

Gli approcci per le attività di Verifica e Validazione

- **Analisi dinamica**: processo di valutazione di un sistema software o di un suo componente basato sulla osservazione del suo comportamento in esecuzione.
  - Di solito solo queste tecniche si identificano come **testing**.
- **Analisi statica**: processo di valutazione di un sistema o di un suo componente basato sulla sua forma, struttura, contenuto, documentazione senza che esso sia eseguito. Esempi sono: revisioni, ispezioni, recensioni, analisi data flow
- **Analisi formale**: uso di rigorose tecniche matematiche per l'analisi di algoritmi.
  - E' usata soprattutto per la verifica del codice e dei requisiti, specie quando questi sono specificati con linguaggi formali (es. Z, VDM).

- Richiami sul Software Testing

Definizioni IEEE per il Testing

Standard IEEE 610.12-1990 :

- (1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.

IEEE Std.729-1983

- (2) The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.

- See also: acceptance testing; benchmark; checkout; component testing; development testing; dynamic analysis; formal testing; functional testing; informal testing; integration testing; interface testing; loopback testing; mutation testing; operational testing; performance testing; qualification testing; regression testing; stress testing; structural testing; system testing; unit testing.

Definizioni

- **Errore** (umano)
  - incomprensione umana nel tentativo di comprendere o risolvere un problema, o nell'uso di strumenti
- **Difetto** (fault o bug)
  - Manifestazione nel software di un errore umano, e causa del fallimento del sistema nell'eseguire la funzione richiesta
- **Malfunzionamento** (failure)
  - incapacità del software di comportarsi secondo le aspettative o le specifiche
  - un malfunzionamento ha una natura dinamica: accade in un certo istante di tempo e può essere osservato solo mediante esecuzione

Un esempio

```
void raddoppia()
{
cin>>x;
y := x*x;
cout<<y;
}
```

ERRORE di editing/digitazione

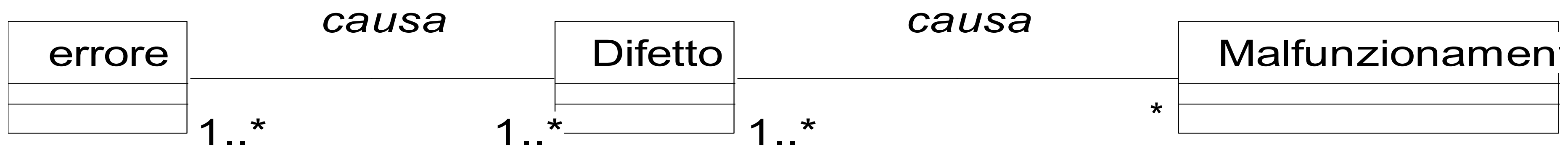
ERRORE
siamo convinti che il raddoppio si calcoli come x*x

DIFETTO
“*” invece di “+”

MALFUNZIONAMENTO
=> il valore visualizzato è errato

Possibile MALFUNZIONAMENTO in esecuzione…
(può verificarsi o meno: dipende dall’input)

Relazione fra Errore, Difetto e Malfunzionamento



9/9

0800 Anctan started

1000 " stopped - anctan ✓ {1.2700 9.037 847 025 / 9.037 846 795 correct

13" uc (032) MP - MC ~~1.982147000 / 2.130476415 (-3)~~ 4.615925059(-2)

(033) PRO 2 2.130476415

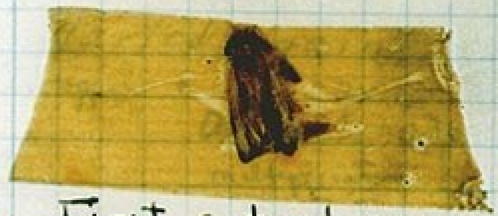
correct 2.130676415

Relays 6-2 in 033 failed special speed test
In relay " 10,000 test.
Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 Relay #70 Panel F
(moth) in relay.



First actual case of bug being found.

1630 Antangent started.

1700 closed down.

Test dei Difetti

- Ha lo scopo di scoprire i difetti di un programma.
- Approccio usato: scoprire la presenza di difetti osservando i malfunzionamenti!
- Un test dei difetti *ha successo* se porta il programma a comportarsi in maniera scorretta (cioè esibisce malfunzionamenti)

- Il testing può dimostrare la presenza dei difetti.
  - Potrà dimostrare anche la loro assenza?

Ulteriori Definizioni

- Test case: caso di prova di un software
  - Corrispondente all’esecuzione del software con una combinazione d dei propri valori in ingresso
- Test suite: insieme di casi di prova di un software

Fondamenti teorici del testing

- Consideriamo un programma P come una funzione da un insieme di dati D (dominio) in un insieme di dati R (codominio).
  - Il risultato P(d) ottenuto eseguendo P sul dato di ingresso d, con $d \in D$, è corretto se soddisfa le specifiche, non corretto se diverso dal risultato previsto dalle specifiche.
- La correttezza di un programma P rispetto ad un ingresso d è indicata con ok(P,d)
- Un programma è corretto $ok(P) \Leftrightarrow \forall\ d \in D,\ ok(P,d)$

Adeguatezza dei test

- Scopo dell'attività di testing è la rilevazione di malfunzionamenti.

- Una test suite T rileva un malfunzionamento se il programma P non è corretto per almeno un test case di T.
  - T ha successo per un programma P se rileva uno o più malfunzionamenti presenti in P,
  - viceversa è *inadeguata* una test suite T per la quale esistano dei malfunzionamenti in P che nessun test case è in grado di rilevare

- Una test suite T è detta *ideale* se l'assenza di malfunzionamenti rilevati implica l'assenza di malfunzionamenti
  - *Cioè se* $ok(P,T) \Rightarrow ok(P)$
    - *Ovvero, se qualsiasi test ulteriore non potrà scoprire malfunzionamenti che non siano stati già scoperti da T*

Problemi indecidibili

- Il settore del testing e' tormentato da problemi indecidibili
  - un problema e' detto indecidibile (irrisolvibile) se e` possibile dimostrare che non esistono algoritmi che lo risolvono
    - *Problema della terminazione della macchina di turing, teorema della esistenza di funzioni non calcolabili, …*

- Stabilire se l'esecuzione di un programma termina a fronte di un input arbitrario e' un problema indecidibile

Altri problemi indecidibili

- Braierd Landerweber 1974
  - dati due programmi, il problema di stabilire se essi calcolano la stessa funzione e' indecidibile

- Enormi conseguenze per il testing:
  - dato un programma e supposto noto e disponibile l'archetipo idealmente corretto di tale programma non possiamo comunque dimostrare l'equivalenza dei due
  - Non esiste un algoritmo in grado di stabilire se due generici cammini del grafo di flusso di controllo di un programma calcolino la stessa funzione o meno (teorema di equivalenza dei cammini)

Teorema di Weyuker

- Dato un generico programma P i seguenti problemi risultano indecidibili:
  - Esiste almeno un dato di ingresso che causa l'esecuzione di un particolare comando?
  - Esiste un particolare dato di ingresso che causa l'esecuzione di una particolare condizione (branch)?
  - Esiste un dato di ingresso che causa l'esecuzione di un particolare cammino?
  - E' possibile trovare almeno un dato di ingresso che causi l'esecuzione di ogni comando di P?
  - E' possibile trovare almeno un dato di ingresso che causi l'esecuzione di ogni condizione (branch) di P?
  - E' possibile trovare almeno un dato di ingresso che causi l'esecuzione di ogni cammino di P?

Teorema di Weyuker

- **Tesi di Dijkstra**
  - Il testing non può dimostrare l'assenza di difetti, ma solo la loro presenza

- Non vi è garanzia che se alla n-esima prova un modulo od un sistema abbia risposto correttamente (ovvero non sono stati più riscontrati difetti), altrettanto possa fare alla (n+1)-esima

- Impossibilità di produrre tutte le possibili configurazioni di valori di input (test case) in corrispondenza di tutti i possibili stati interni di un sistema software

- **"We did about 10,000 tests on it, and it was working fine until Monday."**
  - Anonymous - Spokesperson for 7-11 after Y2K-related failure of their credit card processing on 2001-01-01

Problemi e Limitazioni

- **La correttezza di un programma è un problema indecidibile!**

- **Problemi:**
  - non vi è garanzia che se alla n-esima prova un modulo od un sistema abbia risposto correttamente (ovvero non sono stati più riscontrati difetti), altrettanto possa fare alla (n+1)-esima
  - Impossibilità di produrre tutte le possibili configurazioni di valori di input (test case) in corrispondenza di tutti i possibili stati interni di un sistema software

Ulteriori Problemi

- In molti campi dell’ingegneria, il testing è semplificato dall’esistenza di proprietà di continuità
  - Se un ponte resiste ad un carico di 1000 tonnellate, allora resisterà anche a carichi più leggeri
- Nel campo del software si ha a che fare con sistemi discreti, per i quali piccole variazioni nei valori d’ingresso possono portare a risultati scorretti
  - Il testing esaustivo (ideale) è condizione necessaria per poter valutare la correttezza di un programma a partire dal testing

Amenità sul Testing

da "The Zen of Programming" - Geoffrey James

Thus spoke the master: "Any program, no matter how small, contains bugs"

The novice did not believe the master's words. "What if the program were so small that it performed a single function?" he asked.

"Such a program would have no meaning," said the master, "but if such a one existed, the operating system would fail eventually, producing a bug".

But the novice was not satisfied. "What if the operating system did not fail?" he asked.

"There is no operating system that does not fail," said the master, "but if such a one existed, the hardware would fail eventually, producing a bug".

The novice still was not satisfied. "What if the hardware did not fail?" he asked.

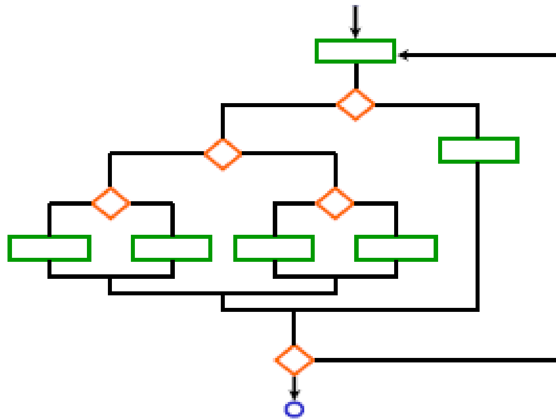
The master gave a great sigh. "There is no hardware that does not fail," said the master, "but if such a one existed, the user would want to do something different, and this too is a bug."

A program without bugs would be an absurdity, a nonesuch. If there were a program without any bugs then the world would cease to exist.

L'impossibilità del Testing Esaustivo!

Se il ciclo fosse eseguito al più 20 volte, ci possono essere oltre 100.000 miliardi di possibili esecuzioni diverse!!!

Se ogni test fosse elaborato in 1 ms, sarebbero necessari 3170 anni!!!



```
......
repeat
  B0
  if R1 then
        if R2 then
              if R3 then
                    B1
                   else
                    B2
              endif
              if R4 then
                    B3
                   else
                    B4
              endif
             else
              B5
        endif
  endif
until R6
..........
```

Esempio

```
char x;
do {
cin>>x;
} while (x!='0')
```

In quanti modi diversi può essere eseguito questo semplice codice?

Nonostante x sia un char (quindi solo 256 valori possibili) il numero di possibili sequenze di inserimento è illimitato.

Esempi:

abcde0

qiunonooiiouuoiounoinoinoinonoiioio0

0

...

Caratteristiche di qualità del testing

- Deve aiutare a localizzare i difetti
  - Per facilitare il debugging
    - *Tanti test semplici consentono una più semplice localizzazione rispetto a pochi test complessi*
- Dovrebbe essere ripetibile
  - Potrebbe non essere possibile se l'esecuzione del caso di test influenza l'ambiente di esecuzione senza la possibilità di ripristinarlo
  - Potrebbe non essere possibile se nel software ci sono degli elementi indeterministici
    - *Ovvero dipendenti da input non controllabili*
- Dovrebbe essere preciso
  - La valutazione del risultato deve essere univoca

Efficacia ed Efficienza

- Il Testing deve essere:
  - **Efficace**
    - *Condotto attraverso strategie che consentano la scoperta di quanti più difetti possibili;*
  - **Efficiente**
    - *In grado di trovare difetti provando il minor numero possibile di casi di test*
- Necessità di uso di metodi e tecniche per ridurre lo sforzo (inteso come impiego di risorse umane, tecnologiche e di tempo) per poter individuare il massimo numero possibile di malfunzionamenti (con il minimo numero possibile di test case) prima che il prodottto sia rilasciato
    - *circa il 40% dei costi di produzione del software per il raggiungimento di ragionevoli livelli di qualità sono richiesti dal testing*

Misura dell’efficacia e dell’efficienza

- L’**Efficacia** di un’attività di testing può essere definita come:
  Numero di difetti scoperti / numero di difetti esistenti
- L’**Efficienza** di un’attività di testing può essere definita come:
  Numero di casi di test rilevanti difetti / numero di casi di test eseguiti
- Per aumentare l’efficacia spesso si aumenta il numero di casi di test eseguiti (diminuendo l’efficienza)
- Per aumentare l’efficienza si eseguono soltanto i casi di test corrispondenti ad esecuzioni “critiche” del software (diminuendo spesso l’efficacia)

Misura dell'efficacia e dell'efficienza: problemi

- L'efficienza del test può essere misurata in termini di numero di casi di test eseguiti, tempo o sforzo necessario per eseguirli
  - Il numero di test è la grandezza più semplice da misurare. Se il testing è automatico, la misura più utile è il tempo, altrimenti la più utile è lo sforzo
- L'efficacia dei casi di test in generale non può essere misurata
  - Non sappiamo quanti difetti sono presenti nel software!
    - *Ciò è possibile solo in ambiente sperimentale, nel caso in cui iniettiamo appositamente difetti nell'applicazione allo scopo di valutare l'efficacia della test suite nel rivelare malfunzionamenti*
  - Possiamo, però, sempre misurare l'efficacia relativa di una test suite rispetto ad un'altra in base al rapporto tra i difetti scoperti

Misura indiretta dell’efficacia

- In mancanza di misure dirette, l’efficacia si può misurare in termini di copertura. Ad esempio
  - CovLOC=LOC coperte da almeno un caso di test / Totale LOC
  - CovMetodi=Metodi coperti da almeno un caso di test / Totale Metodi
  - CovClassi=Classi coperte da almeno un caso di test / Totale Classi

- La copertura può essere interpretata come una «speranza» di verifica
  - se non si esegue la riga difettosa, sicuramente non si riscontrerà il fallimento; in caso contrario c’è la possibilità (non certezza) di riscontrarlo

Valutazione della copertura

- Per valutare la copertura è necessario inserire (possibilmente in maniera automatica) sonde nel codice dell'applicazione, che ci permettano di monitorare i percorsi di esecuzione

- La copertura si riferisce tecnicamente alle righe di codice eseguibile: esistono delle difficoltà nel riportarlo al codice sorgente o al modello
  - Ad esempio la riga:
    - *If (a>b && x>y && f(&x)==5)*
  - È spesso tradotta con cicli innestati in assembler
  - Se in un caso di test a<=b, x>y non verrà effettuata e f(&z) non verrà eseguita
    - *Attenzione: l'esecuzione di f(&z) può modificare il valore di z, per cui l'ordine con cui vengono eseguite le condizioni è anch'esso rilevante sull'esito del confronto)*
  - In questi casi, molti programmi convengono in misurare la copertura delle LOC del codice sorgente in termini frazionari

Problema della generazione di codice a run-time

- In molti linguaggi è possibile generare codice a tempo di esecuzione
  - una pagina server può generare codice lato client (anche Javascript eseguibile) diverso in seguito ad ogni diversa esecuzione
  - Una shell «esegue» esattamente ciò che gli viene immesso come input (che sia un eseguibile o uno script batch)
  - L'istruzione eval (o evaluate) disponibile in molti linguaggi di programmazione (specialmente in quelli interpretati) consente di eseguire codice «arbitrario» formato a tempo di esecuzione

- In questi casi non è possibile definire il quantitativo esatto di codice eseguibile da coprire!

Efficacia ed efficienza

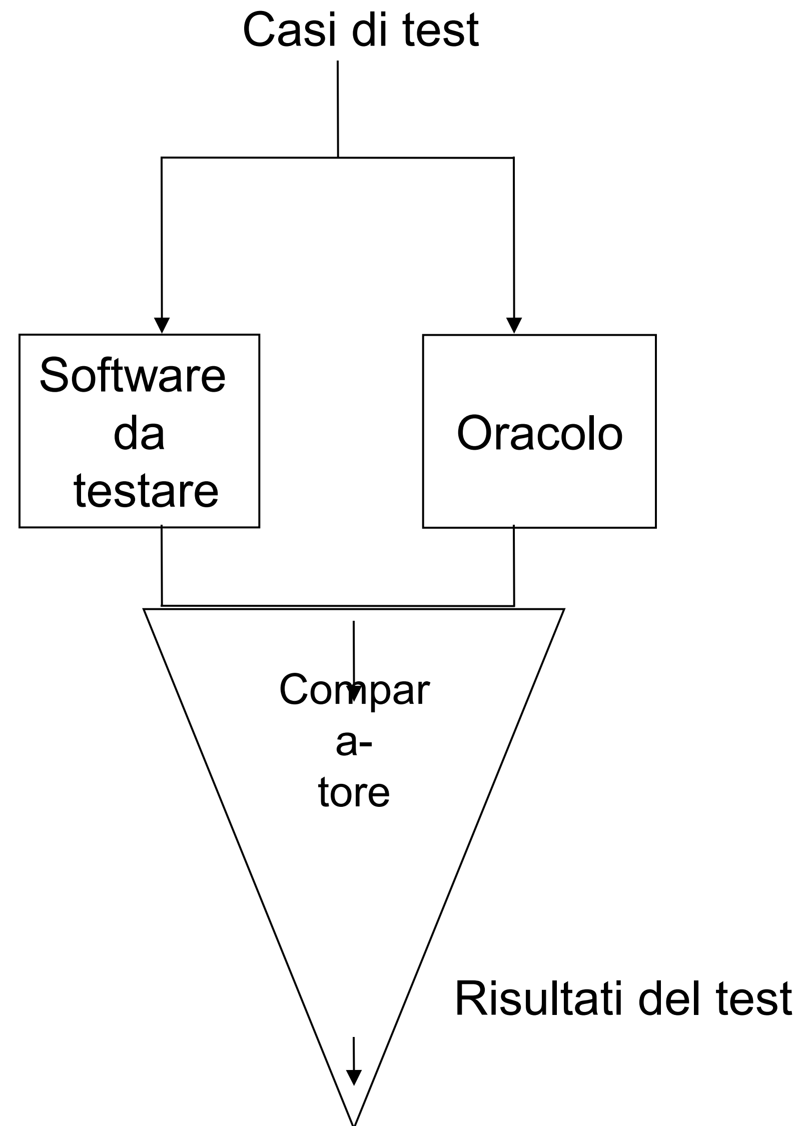
- Le varie tecniche abbinano livelli crescenti di efficacia a livelli decrescenti di efficienza
  - Non è facile trovare il giusto compromesso

- L'efficacia va privilegiata quando si vuole un software affidabile (ad esempio per applicazioni critiche)

- L'efficienza va privilegiata se si vuole ridurre lo sforzo allocato alle attività di testing
  - in particolare se non può essere eseguito automaticamente

Valutazione dei risultati del test

- Condizione necessaria per effettuare un test:
  - conoscere il comportamento atteso per poterlo confrontare con quello osservato
- L'oracolo conosce il comportamento atteso per ogni caso di prova
- Oracolo umano
  - si basa sulle specifiche o sul giudizio
- Oracolo automatico
  - generato dalle specifiche (formali)
  - stesso software ma sviluppato da altri
  - versione precedente (test di regressione)



Specifica dei casi di test

- Ogni caso di test, indipendentemente dalla sua tipologia, dovrebbe essere descritto quanto meno dai seguenti campi
  - Numero Identificativo
  - Descrizione
    - *Può indicare anche la funzionalità che si va testando*
  - Precondizioni
    - *Asserzioni che devono essere verificate affinchè il test possa essere eseguito*
  - Valori di Input
  - Asserzioni di Output Attese
    - *Indicati dall'oracolo*
    - *Nei casi più semplici consistono di valori di output attesi*
  - Postcondizioni Attese
    - *Asserzioni assimiliabili agli output ma verificabili su risorse (db/file/altri device/etc)*

TC	Precond	Input	Output Attesi	Output Riscontrato	PostCond	Esito	Priorità

Specifica dei casi di test

- All'atto dell'esecuzione del test, verranno aggiunti i seguenti campi:
  - Output riscontrati
  - Postcondizioni riscontrate
  - Esito
    - *Positivo - successo (cioè malfunzionamento - fail rilevato) se almeno un'asserzione relativa ad output o postcondizioni non è riscontrato*
    - *Negativo altrimenti*

TC	Precond	Input	Output Attesi	Output Riscontrato	PostCond	Esito	Priorità

Input e Output

- Gli input possono essere caratterizzati da coppie attributo/valore (ad esempio i campi di un form), oppure da uno stream di dati in input (ad esempio un file in input)
  - Il testing di programmi interattivi potrebbe essere definito redirigendo l'input utente verso un file di ingresso contenente i dati del test
- Gli output possono essere caratterizzati come gli input
  - Analogamente, redirigendo l'output verso un file può essere più semplice automatizzare la valutazione dell'esito dei test

Precondizioni e postcondizioni

- Le precondizioni e le postcondizioni possono riguardare
  - lo stato di fonti di dati persistenti (ad esempio file esterni, database, risorse remote, variabili globali)
  - Lo stato dell'applicazione prima e dopo un'esecuzione
    - *Esempi: avvenuta autenticazione, presenza/assenza di dati sul database, …*

Terminazione del testing

- Dal momento che il testing esaustivo è, in generale, irraggiungibile, altri criteri sono proposti per valutare quando il testing possa essere terminato
  - Criterio temporale: periodo di tempo predefinito
  - Criterio di costo: sforzo allocato predefinito
  - Criterio di copertura:
    - *percentuale predefinita degli elementi di un modello di programma*
    - *legato ad un criterio di selezione dei casi di test*
  - Criterio statistico
    - *MTBF (mean time between failures) predefinito e confronto con un modello di affidabilità esistente*

Problema della selezione dei casi di test

- Test case: caso di prova di un software
- Test suite: insieme di casi di prova di un software
- L'esecuzione di un test consiste ha successo se rileva uno o più malfunzionamenti del programma

Affidabilità e validità dei criteri di selezione dei test

- Un criterio di selezione di test C è **affidabile** per un programma P se per ogni coppia di test suite T1, T2 selezionati dal criterio C, se la T1 ha successo, allora anche T2 ha successo e viceversa
  - Quindi tutti i possibili esemplari di test suite generati dal criterio C hanno la stessa capacità di rilevamento di un malfunzionamento
- Un criterio di selezione C è **valido** per un programma P se, qualora il programma non sia corretto, esiste almeno una test suite T selezionata da C che ha successo per il programma P.
- Se un criterio è **affidabile e valido**, allora qualsiasi test suite T generata da C per un programma non corretto avrà successo
  - Quindi, la validità è un requisito atteso per criteri che mirano a ottenere test suite efficaci; la affidabilità è un requisito atteso per criteri che mirano a ottenere test suite più efficienti

Esempio

```
void raddoppia()

{int x,y;
      cin>>x;
      y:= x*x;
      cout<<y;
return;}
```

- Criterio affidabile ma non valido:
- T deve contenere sottoinsiemi di {0, 2}
  - Nessun insieme trova malfunzionamenti (quindi tra loro si equivalgono)

- Criterio valido ma non affidabile:
- T deve contenere sottoinsiemi di {0,1, 2, 3, 4}
  - Alcuni criteri trovano il malfunzionamento, altri no

- Criterio valido e affidabile:
- T deve contenere almeno un valore maggiore di 3
  - Tutti gli insiemi trovano l'unico malfunzionamento, quindi si equivalgono

Criterio di selezione di test

- Un criterio di selezione C che non esclude alcun elemento del dominio di ingresso D del programma P è valido, ma non necessariamente affidabile.
  - Infatti, se esiste almeno un malfunzionamento, questo si manifesterà in corrispondenza di un dato ingresso d, ma il criterio C seleziona almeno una test suite T che contiene d e quindi rileva il malfunzionamento
  - non tutte le test suite selezionate dal criterio C conterranno però un dato di test come d, che rileva il malfunzionamento.

- Un criterio di selezione C non soddisfatto da tutti gli elementi di D può essere affidabile, ma non valido.

- Un criterio valido e affidabile genera sempre test esaustivi

- *Teorema di Goodenough e Gerhart*

  Il fallimento di una test suite T per un programma P, selezionato da un criterio C affidabile e valido, permette di dedurre la correttezza del programma P

Selezione dei casi di test

- Teorema di Howden
  - Non esiste un algoritmo che, dato un programma arbitrario P, generi un test ideale finito, e cioè un test definito da un criterio affidabile e valido
- Al di là di casi banali, non è possibile costruire un criterio di selezione generale di test valido e affidabile che non sia il test esaustivo
- Obiettivi pratici
  - massimizzare il numero di malfunzionamenti scoperti (richiede molti casi di test)
  - minimizzare il numero di casi di test (e quindi il costo del testing)
- E' preferibile usare più di un criterio di selezione dei test

Testing vs Ispezione

- Testing e ispezione sono due approcci complementari al problema della verifica del software

- L'Ispezione mira alla verifica della **correttezza** di un programma, tramite analisi statica e lettura del codice
  - Si basa sull'analisi statica dei programmi
- Il Testing mira alla **ricerca degli errori** in un programma tramite l'osservazione del suo comportamento e il confronto con il comportamento atteso
  - Si basa sull'analisi dinamica dei processi

Testing vs Ispezione

- L'ispezione è possibile anche per algoritmi che non siano stati ancora implementati, oppure per programmi incompleti

- Il testing è possibile solo per codice che possa essere eseguito

- Sia l'ispezione che il testing possono essere svolte manualmente oppure automatizzate
- Sia l'ispezione che il testing mirano a stabilire l'esistenza di bugs, ma è durante il processo di debugging che si vanno a localizzare e correggere i difetti

Tipologie fondamentali di testing

- In base all’obiettivo:
  - Testing funzionale
  - Testing strutturale
  - Quality Assessment (Stress Testing, Security Testing, Performance Testing, …)
- In base all’oggetto del testing
  - Testing di sistema
  - Testing di integrazione
  - Testing di unità
- In base alle informazioni disponibili:
  - Testing Black Box
  - Testing Grey Box
  - Testing White Box
- …