Software Testing

Introduzione

1

1

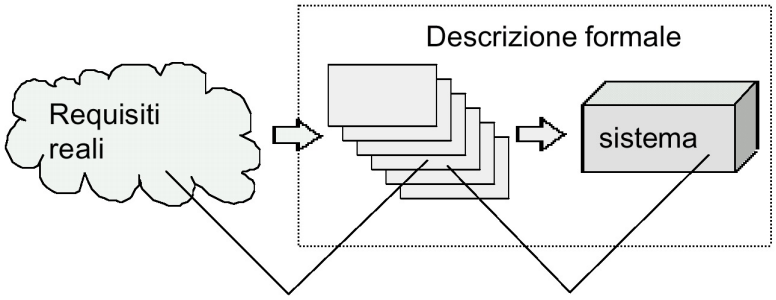
Testing e qualità

- Anni ’90 ed era della qualità
  - ✓ .. introduzione e stabilizzazione di metodi quantitativi per la qualità del software e collocazione della qualità al centro dei processi di sviluppo manutenzione ed evoluzione del software ....
- Il TESTING come una della più importanti e concrete vie di attacco al problema della qualità del software
  - ✓ ... analizzare e valutare la (e quindi promuovere il miglioramento della) correttezza di una implementazione con riferimento alle caratteristiche definite nei requisiti nelle specifiche e nel progetto....

2

2

Verifica e Convalida



Convalida
include test di usabilità, feedback dell'utente

Verifica
include test, ispezione, analisi statica

Are we building the right product ?

Are we building the product right? 3

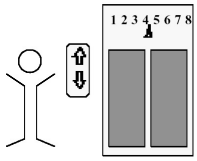
3

Caratteristiche di Qualità e V&V

- Alcune caratteristiche sono convalidabili altre sono verificabili
- Esempi
  - ✓ Verificabili
    - » Correttezza, Affidabilità, Sicurezza, Robustezza, Interoperabilità
  - ✓ Convalidabili
    - » Usabilità
  - ✓ Efficienza ?
    - » La risposta entro un tempo definito è verificabile
    - » Altrimenti l'efficienza è convalidabile (percezione dell'utente …)
- Requisiti funzionali e non funzionali … 4

4

V&V dipendono dalle specifiche



Esempio: risposta di un ascensore

- Specifica non verificabile (ma convalidabile): ... Se un utente preme un pulsante di richiesta al piano i, un ascensore disponibile deve arrivare al piano presto
- Specifica verificabile: ... Se un utente preme un pulsante di richiesta al piano i, un ascensore disponibile deve arrivare entro 30 secondi...

5

5

Enfasi diversa alla stessa proprietà

- Dependability: Correctness, reliability, safety, robustness
  - ✓ Safety-critical applications
    - » Sistemi di controllo avionico hanno requisiti di sicurezza (safety) molto forti
    - » Sistemi di telecomunicazione hanno requisiti di robustezza molto forti
  - ✓ Mass-market products
    - » dependability è meno importante del time to market
  - ✓ Possono variare anche nella stessa classe di prodotti:
    - » Affidabilità (reliability) e robustezza sono requisiti chiave per un sistema oprativo multi utente (es: UNIX), ma sono molto meno importanti per un sistema operativo a singolo utente (es: Windows o MacOS)

6

6

Verifica statica e dinamica

- *Software inspections* Analisi statica delle rappresentazioni del sistema per scoprire problemi (static verification)
  - ✓ L'analisi del codice e dei documenti può essere supportata da tool
- *Software testing* Esecuzione e osservazione del comportamento del prodotto (dynamic verification)
  - ✓ Il sistema è eseguito con dati di test

7

7

Verifica statica

- E' basata su tecniche di analisi statica del software senza ricorso alla esecuzione del codice
  - ✓ ***analisi statica***: è un processo di valutazione di un sistema o di un suo componente basato sulla sua forma, struttura, contenuto, documentazione
- Tecniche: ispezione, tecniche tipo compilatore
- Tool come lint, metriche di size/complexity, analisi del flusso dati

8

8

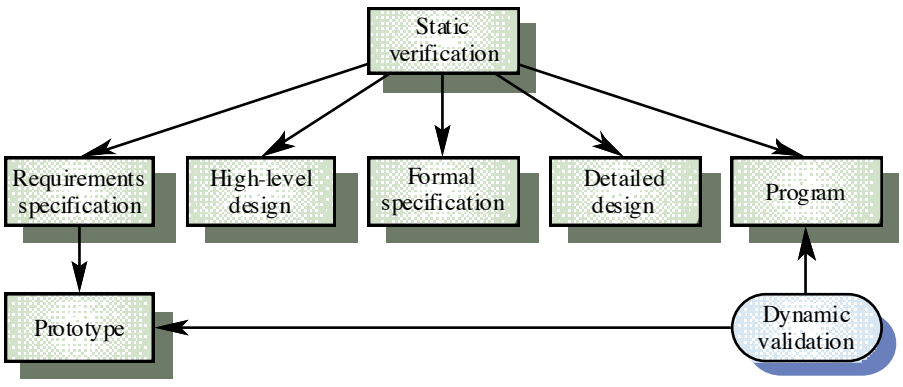
Verifica dinamica

- E' fondata sulla analisi dinamica del codice associato al software e quindi sulla esecuzione dello stesso attraverso dati di ingresso
  - ✓***analisi dinamica***: il processo di valutazione di un sistema software o di un suo componente basato sulla osservazione del suo comportamento in esecuzione
- selezione di casi di test e dati associati, strumentazione del codice (es. inserzione di probe) e monitoraggio, necessità di un oracolo

9

9

Static and dynamic V&V



10

10

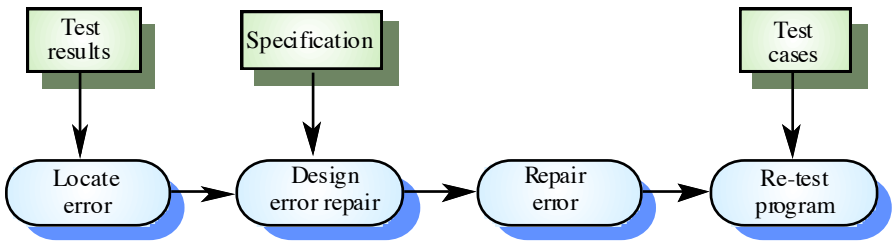
Testing e debugging

- Due processi distinti
- Verification & validation riguarda la ricerca di difetti nel softare
- Debugging riguarda l'individuazione e l'eliminazione dei difetti
- Debugging implica formulare ipotesi riguardo il comportamento del programma e quindi verificare queste ipotesi per localizzare gli errori

11

11

Il processo di debugging



12

12

Un po' di storia …

- Tracce di testing nella preistoria della IS
  - ✓con la prima linea di codice è nato anche il testing ...
  - ✓referenze bibliografiche risalgono al 1949
  - ✓una definizione degli albori: ... il testing è ciò che un programmatore fa per individuare i 'bugs' nei propri programmi...
  - ✓ è già nota la separazione concettuale fra TESTING e DEBUGGING ovvero fra attività di ricerca dei malfunzionamenti ed attività di eliminazione dei bug.

13

13

Albori

- Negli anni '70 i primi seri tentativi di fornire dei fondamenti teorici ed approcci sistematici
  - ✓il sogno del testing esaustivo ... e le relative impraticabili definizioni ad esempio:
  - ✓l'obiettivo del testing è quello di dimostrare la correttezza dei programmi ... quindi un test perfetto si ha quando il programma non contiene errori

14

14

Le Speranze degli Anni ’70

- Esemplare per comprendere l'epoca il lavoro di Goodhough and Gerhard "Toward a theory of test data selection" IEEE TSE SE-1(2) pp 156-173, 1975
- Notevoli conquiste non solo sul piano teorico, ricca produzione di metodi e tecniche classificazioni e toponomastica, modelli per la rappresentazione dei programmi, classificazione di errori, etc

15

15

Anni ’80

- Gli anni ’80 eliminano i sogni, consolidano un patrimonio di riferimento; i risultati acquisiti aprono ad una visione: il testing come processo complesso dai costi estremamente elevati.
  - ✓ il clima della fine anni ’70 e' riflesso in G. Mayer "The art of software testing" John Wiley and Sons N.Y. 1979
  - ✓ esempio di consolidamento B. Beizer "Software testing techniques" Van Nostrand Reinhold N.Y. I ed. 1983 II ed. 1990
  - ✓ due standard IEEE std 829-1983, std 1008-1987; importante per le definizioni anche lo standard 610.12-1990 (già 729-1983)

16

16

Il Punto di Vista degli Anni ’80

- **il testing è il processo di esecuzione del software con l'obiettivo di trovare malfunzionamenti**
  - ✓ ***Un test che non rivela malfunzionamenti è un test fallito***
  - ✓ ***Il testing non può dimostrare l’assenza di difetti, ma può solo dimostrare la presenza di difetti (tesi di Dijkstra)***
- una nuova conquista è anche quella che svincola il testing dal suo sodalizio con il programma
- la progettazione, pianificazione ed implementazione del processo di testing inizia con i primi passi di un qualsiasi processo di produzione, manutenzione evoluzione del software

17

17

Anni ’90

- consolidamento del rapporto fra testing e qualità
- più profonda conoscenza e miglioramento dei processi di testing
- nascono nuove sfide legate ai nuovi paradigmi di programmazione, produzione manutenzione ed evoluzione del software.

18

18

Slogan degli Anni ’90

- il testing è l'analisi e l'esecuzione sotto condizioni controllate dei vari componenti di un progetto software con l'obiettivo di mettere in evidenza difetti di qualità
- esemplare l'affermazione di Hetzel: "il testing è pianificazione, progettazione, costruzione, manutenzione e realizzazione di test ed ambienti di test"

19

19

Sfide degli Anni ’90

- Tra le molte sfide :
  - ✓ come abbattere i costi del testing (40-60% del costo totale di produzione)
  - ✓ come approcciare il testing di sistemi object oriented
  - ✓ come risolvere il testing in manutenzione ed evoluzione
  - ✓ come aumentare i livelli di automazione dei processi di testing
- Rilevante esigenza di approcci ingegneristici, quantitativi, di esperimenti controllati di feedback da esperienze ed applicazioni in ambienti reali:
  - ✓ tutto ciò costituisce anche il presupposto per nuovi significativi avanzamenti nella composizione del puzzle teorico e modellistico del testing.

20

20

Definizioni di Testing (Standard IEEE)

- (IEEE std 829-1983) 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
- (IEEE std 610.12-1990) 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 829-2008) an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.

21

21

Altri Termini

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

22

22

Definizioni

- ***Failure*** (fallimento): evento osservabile percepito dall'utente come una mancata prestazione di un servizio atteso
- ***Fault*** (difetto): la causa di una failure, insieme di informazioni (dati di ingresso, valori di variabili, istruzioni, etc) che quando processate producono un fallimento
- In sostanza failure è un comportamento anomalo, inatteso o errato del sistema mentre il fault è la sua causa identificata o ipotizzata

23

23

Fault e Failure

```
   void raddoppia()
   {
     int x, y;
1.   cin >> x;
2.   y = x*x;
3.   cout << y;
   }
```

Per il valore di ingresso x = 3 si ha il valore di uscita y = 9. La causa di questa ***failure*** è il ***fault*** di linea 2, in cui anziché l'operatore + è usato l'operatore *

NB: Se il valore di ingresso è x = 2, il valore di uscita è y = 4 (***nessuna failure***)

24

24

Definizioni

- Non tutti i fault generano failure, una failure può essere generata da più fault, un fault può generare diverse failure
- ***Defect*** (difetto) quando non è importante distinguere fra fault e failure si può usare il termine defect per riferirsi sia alla causa (fault) che all'effetto (failure)
- ***Error*** (errore) è usato con 2 significati diversi:
  - ✓ una discrepanza fra un valore calcolato, osservato o misurato e il valore corretto
  - ✓ l'azione di una persona che causa la presenza di un fault in un software

25

25

Test e Casi di test

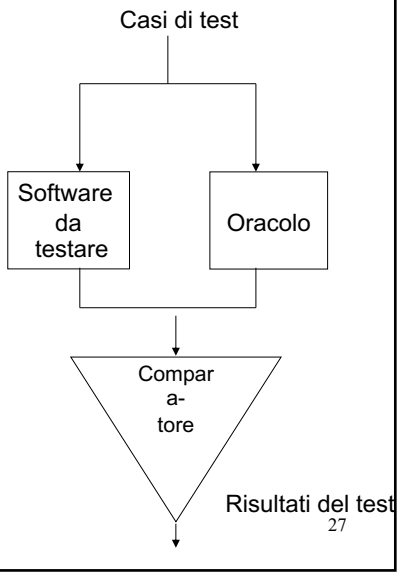
- Un programma è esercitato da un caso di test (insieme di dati di input)
- Un ***test*** è formato da un insieme di casi di test
- L'esecuzione del test consiste nell'esecuzione del programma per tutti i casi di test
- Un test ha **successo** se rileva uno o più malfunzionamenti del programma

26

26

L'oracolo …

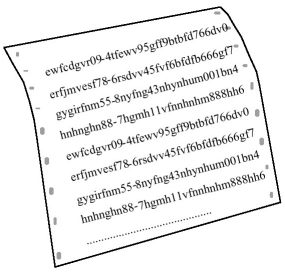
- Condizione necessaria per effettuare un test:
  - ✓ conoscere il comportamento atteso per poterlo confrontare con quello osservato
- L'oracolo conosce il comportamento atteso per ogni caso di prova
- Oracolo umano
  - ✓ si basa sulle specifiche o sul giudizio
- Oracolo automatico
  - ✓ generato dalle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



27

27

Un buon oracolo

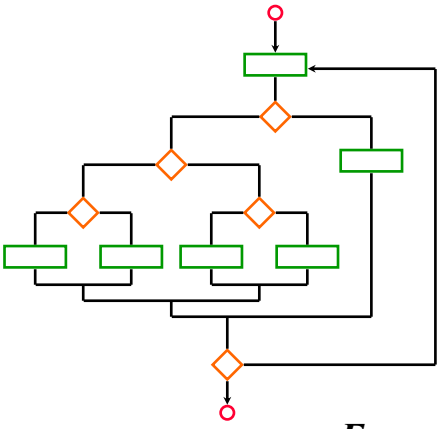


- Il test di applicazioni grandi e complesse può richiedere milioni di casi di test
- La dimensione dello spazio di uscita può eccedere le capacità umane
- L'occhio umano è lento e poco affidabile anche per uscite di piccole dimensioni

ORACOLI AUTOMATICI SONO ESSENZIALI ! 28

28

Problemi del testing: Test Esaustivo???



10^{14} cammini possibili

1 test / millisecondo

Ciclo da 1 a 20

3170 anni per testare questo programma

E non e' tutto ....

29

29

Problemi indecidibili

- Il settore del testing è tormentato da problemi indecidibili
  - ✓ un problema è detto ***indecidibile*** (irresolubile) se è possibile dimostrare che non esistono algoritmi che lo risolvono ... (macchina di Turing, halting problem, teorema della esistenza di funzioni non calcolabili, etc...)
- es. stabilire se l'esecuzione di un programma termina a fronte di un input arbitrario è un problema indecidibile
- ma se stiamo costruendo una società basata sul software ...

30

30

V&V goals and confidence

- Verification and validation dovrebbero stabilire un certo livello di fiducia che il software è adatto ai suoi scopi (fa quello che deve fare).
  - ✓ Questo non significa completa assenza di difetti
  - ✓ Significa invece che deve essere abbastanza buono per l'uso per il quale è stato pensato e proprio il tipo di uso determinerà il livello di fiducia richiesto
- Il livello di fiducia dipende dagli scopi e funzioni del sistema, dalle attese degli utenti e dal time-to-market

31

31

Terminazione del testing

- Quando il programma si può ritenere analizzato a sufficienza
  - ✓ 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

32

32

Equivalenza di Funzioni

- Braierd Landerweber 1974
  - ✓ dati due programmi il problema di stabilire se essi calcolano la stessa funzione è 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
  - ✓ altre indecidibilità derivano direttamente da quelle enunciate

33

33

Test selection

- Un test è **ideale** se l'insuccesso del test implica la correttezza del programma
  - ✓ se il programma fornisce dati di uscita corretti per tale test il programma è corretto per qualsiasi altro test
- *Problema*: selezionare un test ideale per un programma
- Un test **esaustivo** è un test che contiene tutti i dati di ingresso al programma
  - ✓ un test esaustivo è un test ideale
  - ✓ un test esaustivo non è pratico e quasi sempre non è fattibile

34

34

Criterio di Selezione di Test

- Specifica le condizioni che devono essere soddisfatte da un test
  - ✓ Consente di selezionare più test per uno stesso programma
- Un criterio di selezione di test è **affidabile** per un programma se per ogni coppia di test selezionati, T1 e T2, se T1 ha successo anche T2 ha successo e viceversa (*ossia ogni insieme di casi di test che sddisfa il criterio rileva gli stessi errori*)
- Un criterio di selezione di test è **valido** per un programma se, qualora il programma non è corretto, esiste almeno un test selezionato che ha successo

35

35

Esempio

```
   void raddoppia()
   {
     int x, y;
1.   cin >> x;
2.   y = x*x;
3.   cout << y;
   }
```

Criterio affidabile ma non valido:

- T deve contenere sottoinsiemi di {0, 2}

Criterio valido ma non affidabile:

- T deve contenere sottoinsiemi di {0,1, 2, 3, 4}

Criterio valido e affidabile:

- T deve contenere almeno un valore maggiore di 2

36

36

Test selection e indecidibilità

- Goodenough and Gerhard 1975
  - ✓ esiste un test ideale …
  - ✓ il fallimento di un test T per un programma P, selezionato da un criterio C affidabile e valido, permette di dedurre la correttezza del programma P
- Howden 1976
  - ✓ il problema di costruire un test ideale è indecidibile: 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

37

37

In pratica …

- Obiettivo realistico: selezionare casi di test che approssimano un test ideale
  - ✓ 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 …

38

38

Tecniche di testing

- Testing dei difetti (sistematico)
  - ✓ Test progettati per scoprire in maniera sistematica i difetti del sistema
  - ✓ *A successful defect test is one which reveals the presence of defects in a system.*
  - ✓ Black box testing (testing funzionale)
  - ✓ White box testing (testing strutturale)
- Testing statistico
  - ✓ Test progettati per riflettere la frequenza di input degli utenti.
  - ✓ Usato per la stima dell'affidabilità
- Analisi mutazionale
  - ✓ Consente di valutare la bontà del test

39

39

Systematic vs Random Testing

- Random (uniforme):
  - ✓ Prende tutti i possibili input in maniera uniforme
  - ✓ Evita "bias" da parte del test designer
    - » Un problema reale: Il test designer può commettere gli stesi errori logici e cattive assunzioni del program designer (specialmente se sono la stessa persona)
  - ✓ Ma tratta tutti gli input con la stessa importanza
- Systematic (non uniforme):
  - ✓ Cerca di selezionare input che hanno maggiore importanza
  - ✓ Di solito scegliendo rapprentanti di classi

40

40

Perché non Random?

- Distribuzione dei fault non uniforme
- *Esempio:* la classe Java "roots" applica l'equazione

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

Logica implementativa incompleta: il programma non gestisce adeguatamente il caso in cui b^2 - 4ac =0 e/o a=0

I valori che causano una failure sono sparsi nello spazio di input - aghi in un enorme pagliaio. E' improbabile che in maniera random si scelgano a=0.0 and b=0.0

41

41

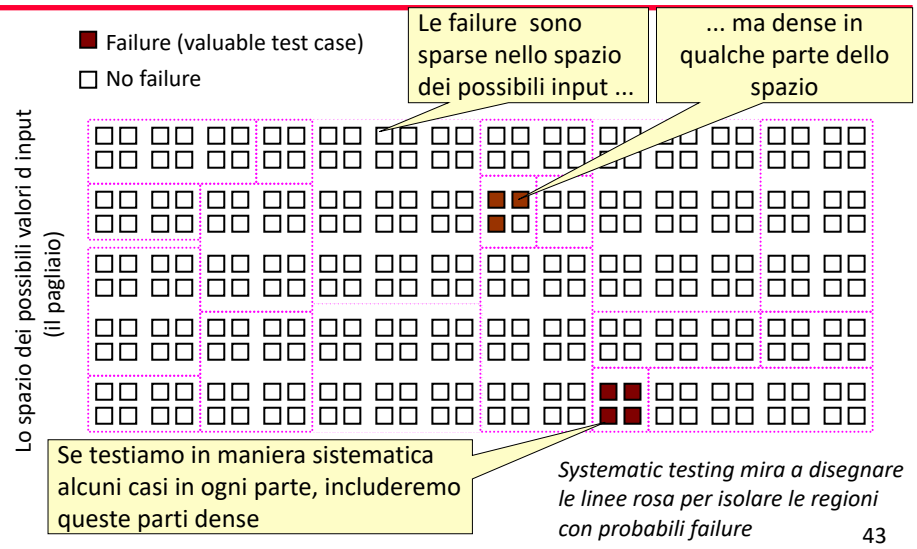
Consideriamo lo scopo del testing ...

- Per stimare la proporzione degli aghi nella paglia, dobbiamo campionare uniformemente (random)
  - ✓ Le stime di affidabilità richiedono un campionamento imparziale per produrre delle statistiche valide. *Ma questo non è il nostro obiettivo!*
- Per trovare gli aghi e toglierli dalla paglia, dobbiamo cercarli in maniera sistematica e non uniformemente
  - ✓ A meno che non ci siano moltissimi aghi nel pagliaio, un campionamento uniforme non è efficace nel trovarli
  - ✓ Abbiamo bisogno di usare tutto ciò che conosciamo degli aghi, ad esempio, sono più pesanti della paglia? Tendono ad andare verso il fondo?

42

42

Systematic Partition Testing



43

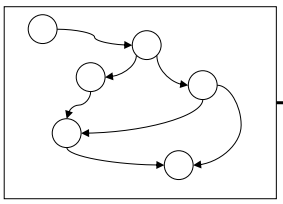
Principio del partizionamento

- Scegliere campioni che con maggiore probabilità includono regioni speciali o potenzialmente difettose dello spazio di input (sulla base della conoscenza …)
  - ✓ Le failure sono sparse nell'intero spazio di input ...
  - ✓ ... ma possiamo trovare regioni in cui sono dense
- (Quasi*-)Partition testing: separare lo spazio di input in classi la cui unione è l'intero spazio
  - ✓ *Quasi perché le classi potrebbero non essere disgiunte
- Caso desiderabile: ogni fault porta a failure che sono dense (facili da trovare) in qualche classe di input
  - ✓ Campionando ogni classe nella quasi-partition selezioneremo almeno un input che porta a una failure (che rivela un fault)
  - ✓ Raramente garantito: dipende da euristiche basate sull'esperienza

44

Test Coverage

Rappresentazione del Software
(Modello)



Criterio Associato

I test cases devono coprire
tutti i … nel modello

Test Data

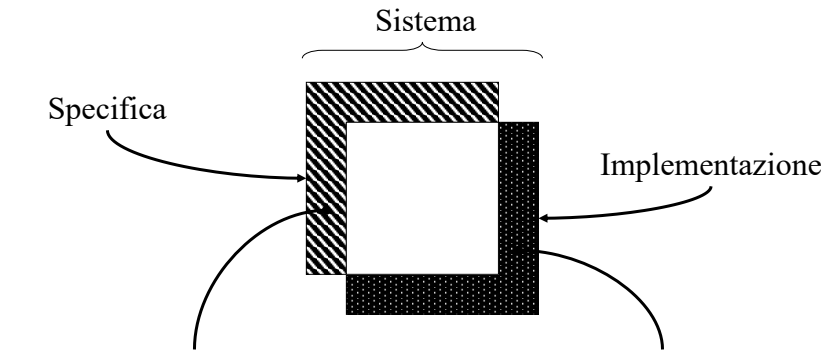
Rappresentazione di

- specifica ⇒ Black-Box Testing
- implementazione ⇒ White-Box Testing

45

45

Black vs. White Box Testing



Missing functionality:
Non può essere rivelata da
tecniche white-box

Unexpected functionality:
Non può essere rivelata da
tecniche black-box

46

46

Analisi Mutazionale

- Tecnica derivante dal test hardware
  - ✓ Identificare un insieme di locazioni del programma (legate a un particolare difetto)
  - ✓ Generare programmi alternativi (*mutanti*) inseminando difetti nel programma originale nelle locazioni identificate
  - ✓ Generare casi di test stimando l'adeguatezza nello scoprire difetti reali dalla capacità di rilevare i difetti inseminati
- Tecnica ancora poco efficace per il test del software
  - ✓ Mancano buoni modelli dei difetti (insieme di possibili deviazioni dal programma corretto)
  - ✓ Molti errori derivano da errori di progetto (è un problema anche per il test hardware)

47

47

Testing Statistico

- Usato per il testing dell'affidabilità del software
- Misurando il numero di errori permette di predire l'affidabilità del software.
- Dovrebbe essere prima specificato un livello accettabile di affidabilità e poi il software dovrebbe essere testato e corretto finché non si raggiunge il livello di affidabilità desiderato
- Bisogna definire il profilo operativo del sistema e costruire casi di test che riflettono tale profilo

48

48