M5: testing

Angelo Gargantini

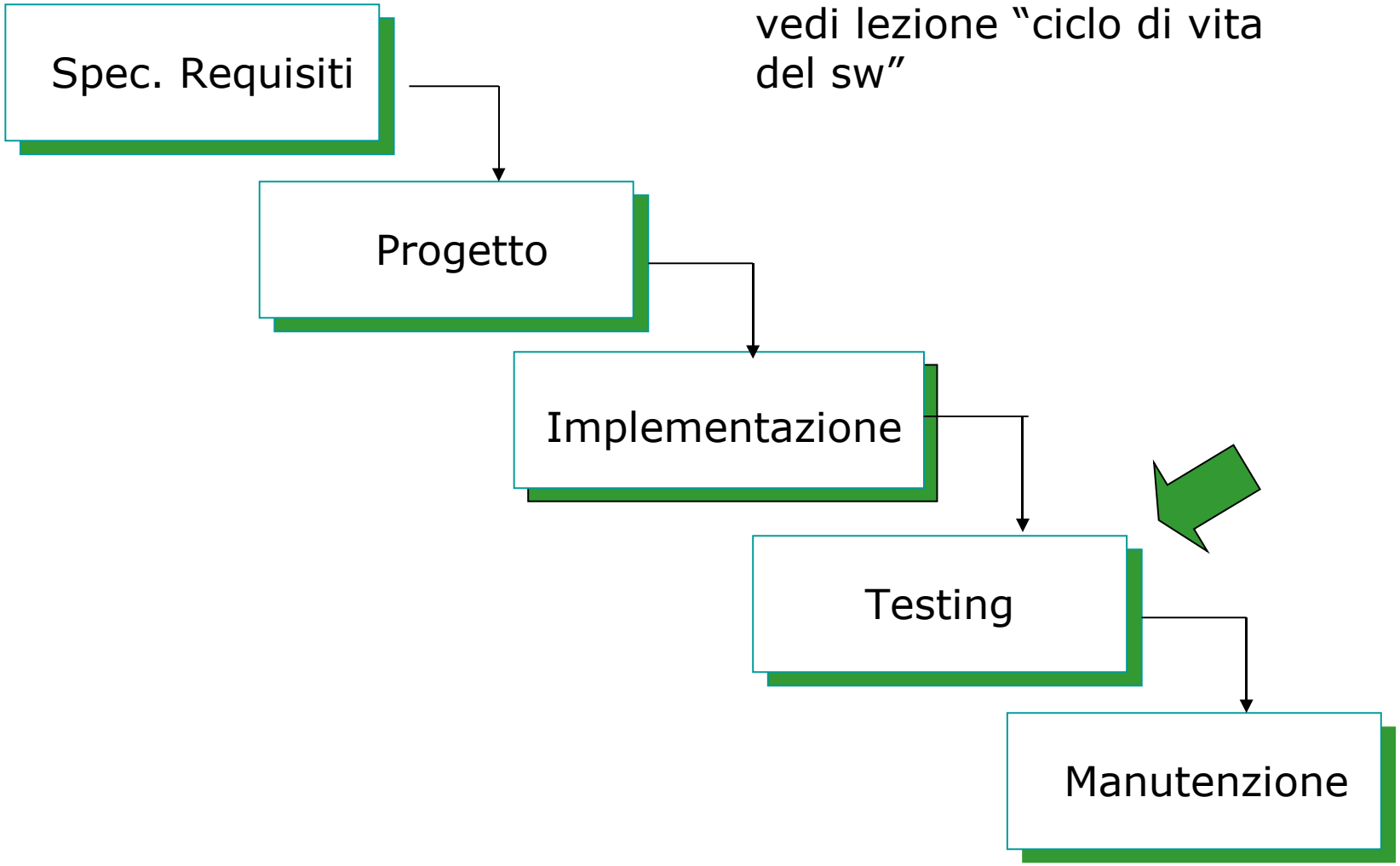
Testing in breve

- Il test di un programma o di un sistema consiste (in breve) nell'eseguirlo con alcuni casi di test e controllare che il comportamento sia corretto

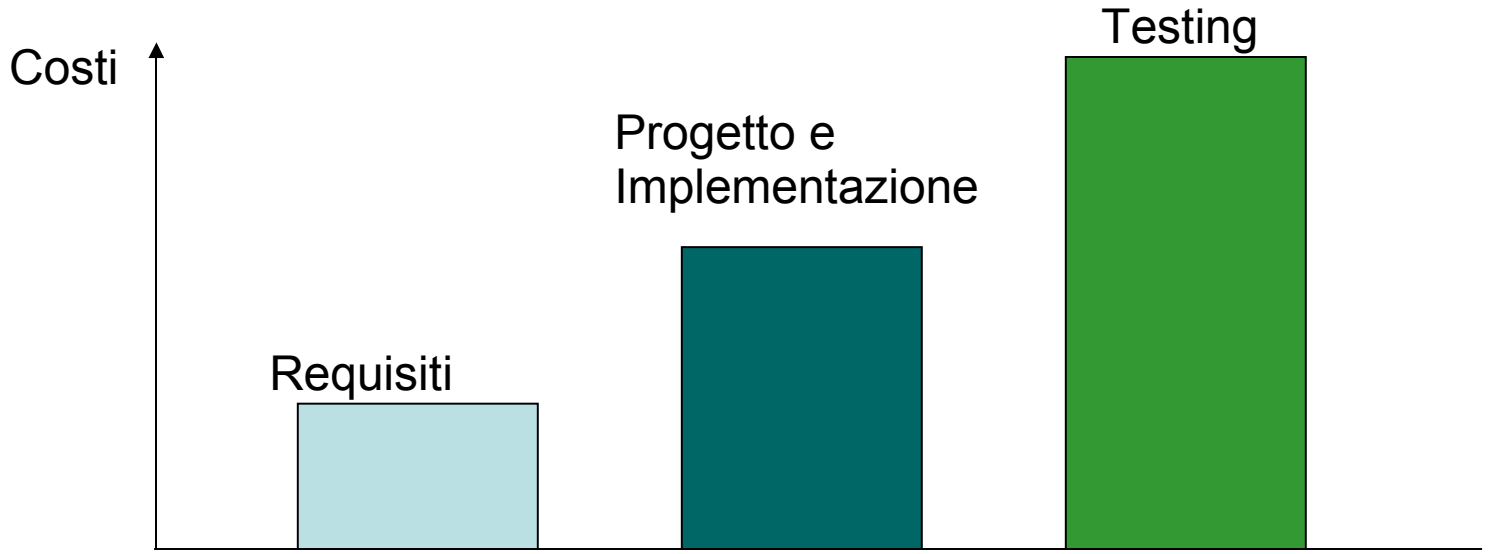
- In italiano si dice anche collaudo del software

Ciclo di vita del Software

Ciclo di vita a cascata – vedi lezione "ciclo di vita del sw"



Costi dello sviluppo del Software



I costi del testing sono normalmente molto maggiori che delle altre fasi

Possibili concause sono:

- risparmio sulle fasi iniziali
- frettolosità nel rilascio del software (time to market)

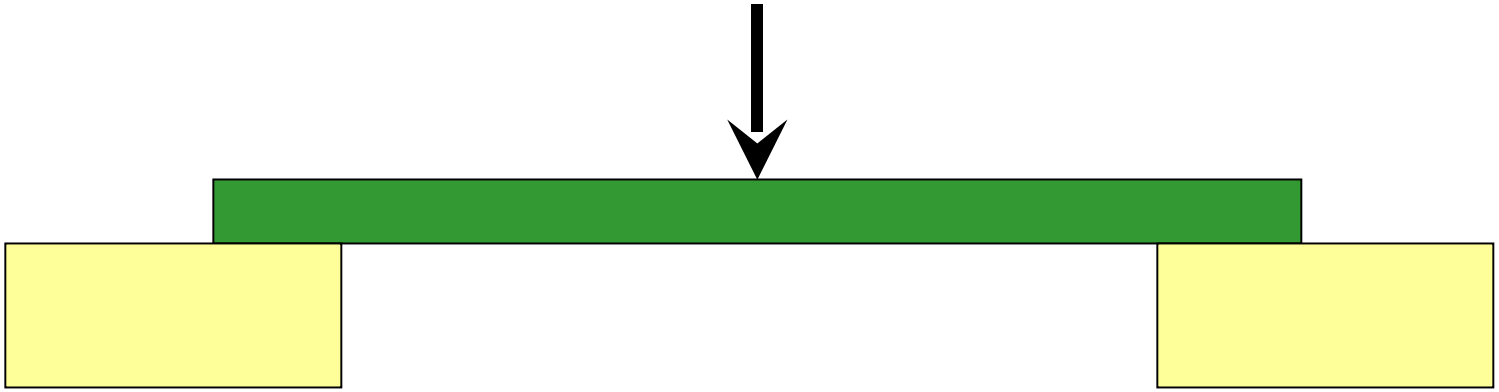
Alcune domande di base

- **E' possibile avere software senza difetti?**
- **Fino a che punto possiamo fidarci del testing o di altre tecniche di verifica?**
  - Ogni parte andrebbe verificata (anche la verifica)
- Tutti i sw devono essere corretti al 100%?
  - sistemi critici
  - sistemi per la produttività individuale

"software industry, the: unique industry where selling substandard goods is legal and you can charge extra for fixing the problems."

Test in Ingegneria

- Il testing è una pratica diffusa in tutti i campi dell'ingegneria
- Però in ingegneria classica gli oggetti hanno un comportamento "continuo":
  - un ponte si testa in un punto particolare e se funziona lì funziona tutt'attorno (o addirittura tutto)



Test nell'ingegneria del software

- **1. Il software ha un comportamento molto discontinuo quindi la selezione dei "punti" in cui effettuare il test e' molto critica**
- Esempio
- `a := x / ( x+20)`

  Quale valore prendo di x per vedere se funziona?
- Problema dei requisiti:
- **2. un ponte basta che resista, un software basta che non smetta di funzionare o richiedo qualcosa di più?**

Limiti del testing

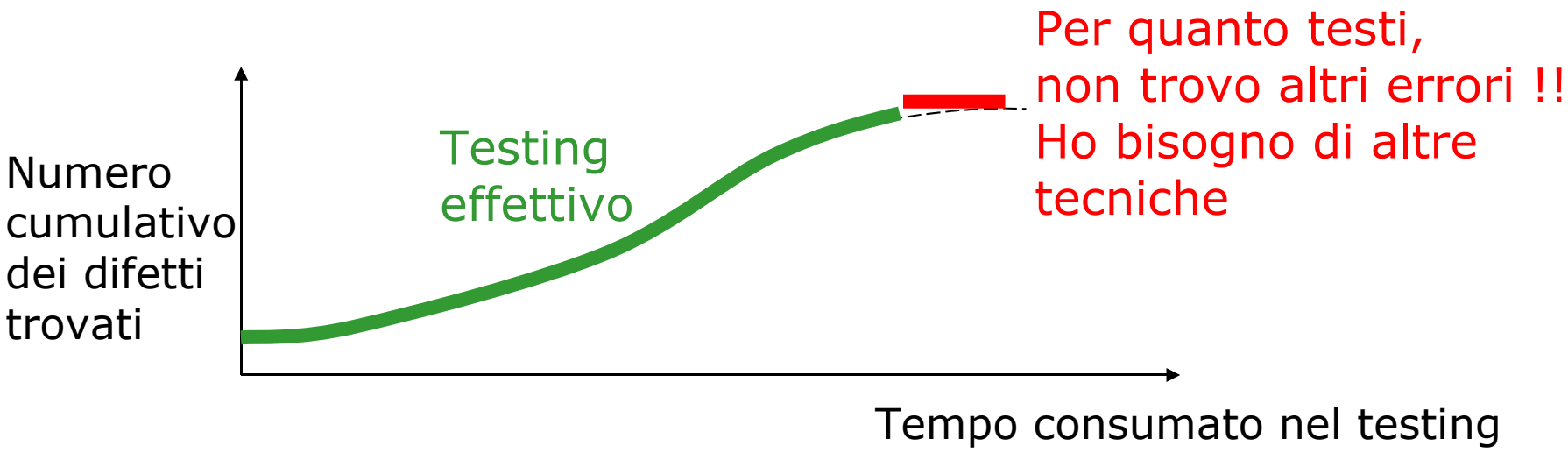
- Il testing è efficace a trovare bug
- ma è inadeguato a provare l’assenza di bug
  - Posso eseguire il programma 1000 volte ed essere (s)fortunato e non scoprire mai un certo bug
- Non può sostituire una buona pratica di progettazione e implementazione

Alcune linee guida generali

- Il testing dovrebbe:
  - essere automatizzato
    - Uso di strumenti automatici che assistano i programmatori nella scrittura di test, nell'esecuzione, nella raccolta dei dati di output e nell'analisi della loro correttezza
  - riguardare ogni fase di sviluppo
    - Non solo sul codice finale, ma anche sui requisiti, sui prototipi, … → prossima lezione
  - essere esteso a tutti i componenti di un sistema
  - essere pianificato (**test plan**)
  - seguire standard e metodologie dove possibile

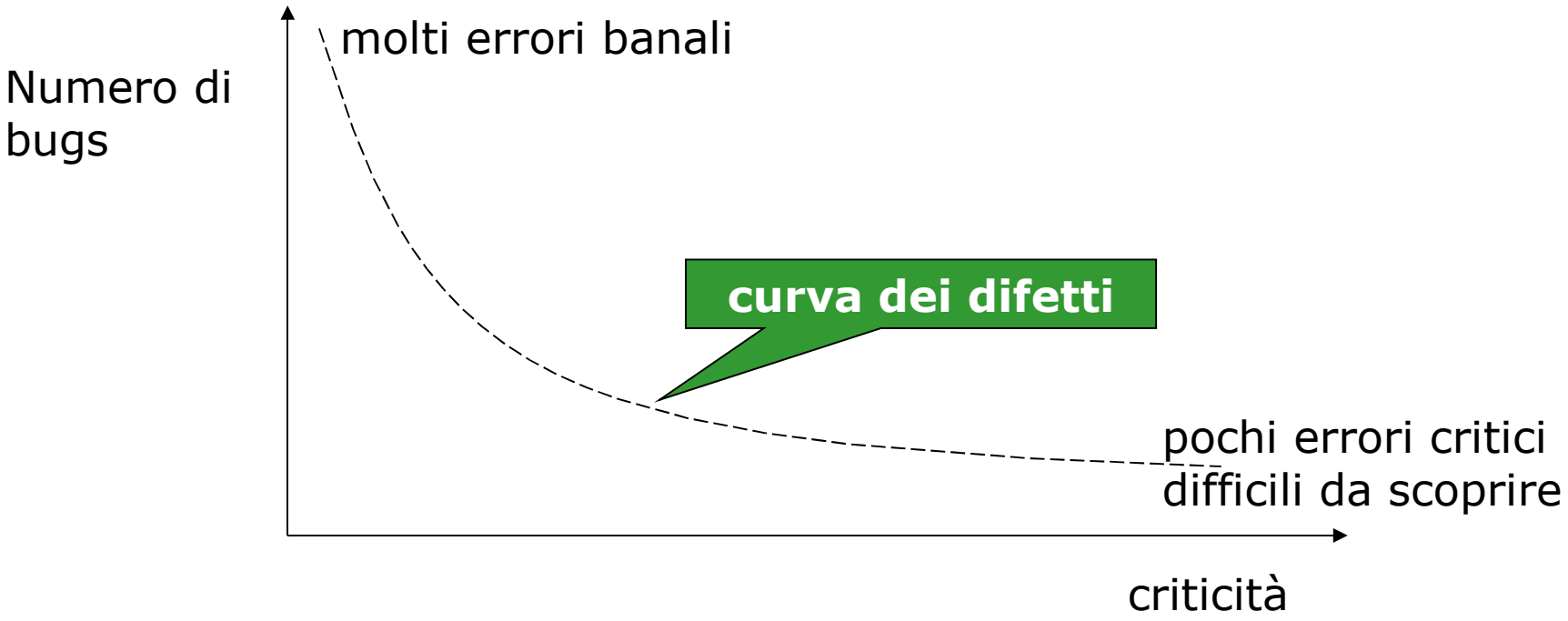
Economia del Testing **(1)**

- Curva di rimozione degli errori



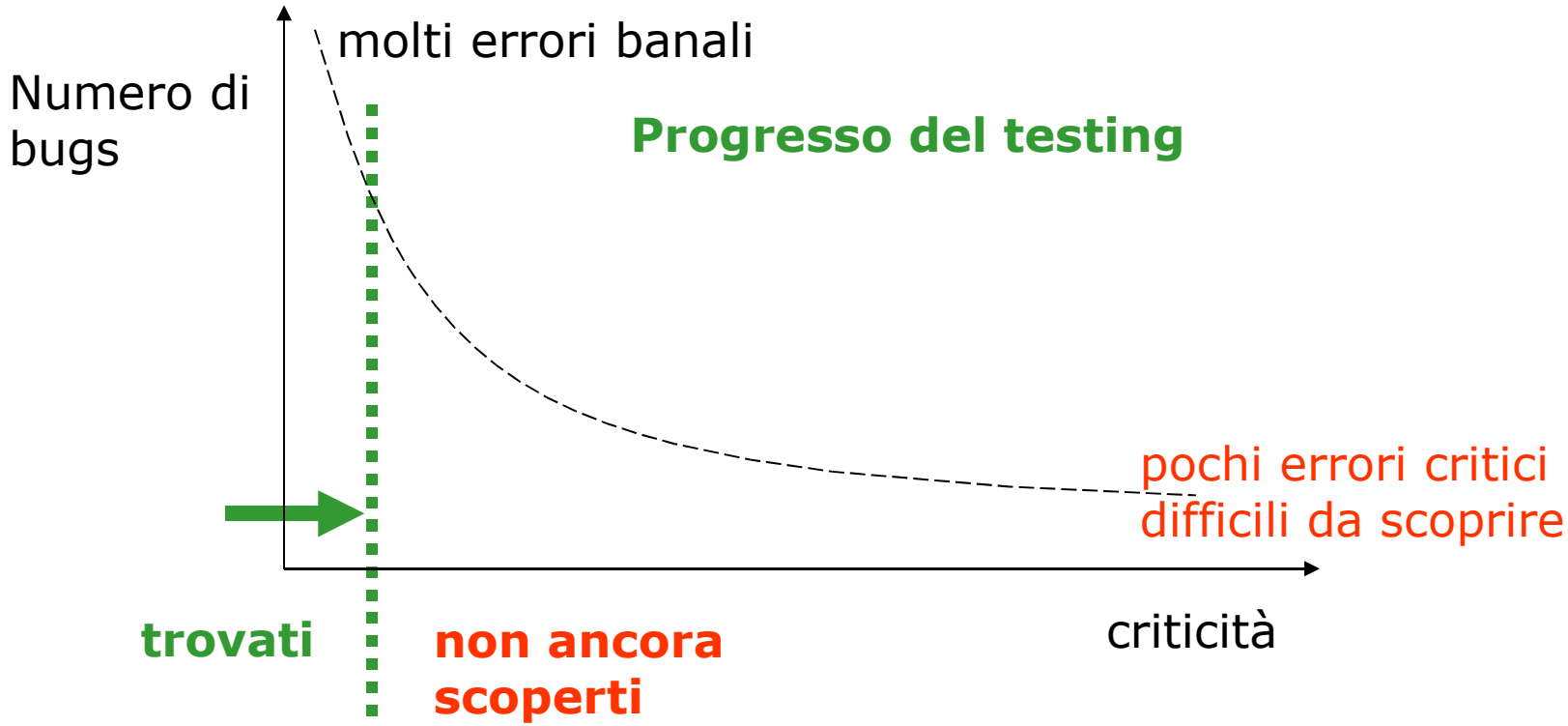
Curva dei difetti

ipotesi: i difetti più critici sono di meno e più difficili da scoprire



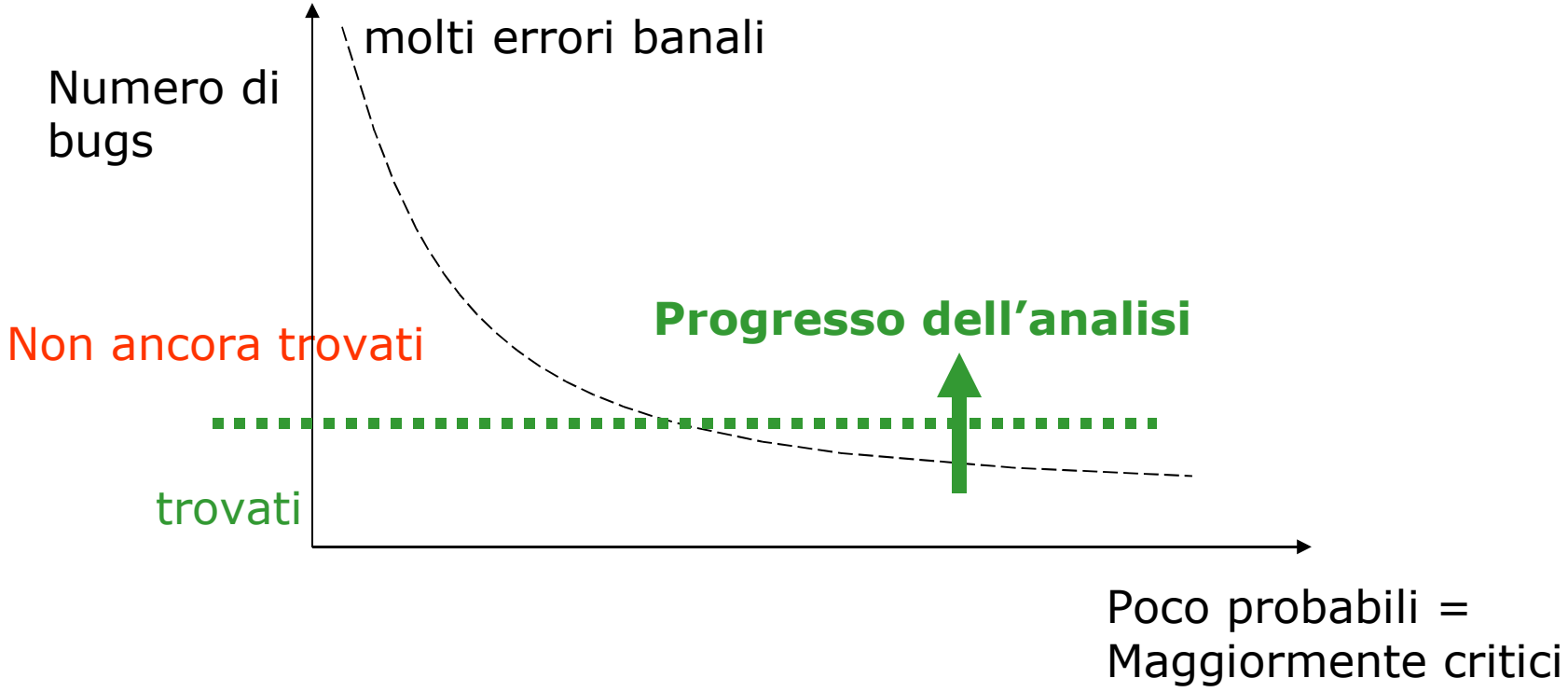
Economia del testing **(2)**

- Il testing tende a scoprire i difetti con probabilità proporzionale alla loro densità:



Economia del testing **(3)**

- **Tecniche (più costose) di analisi statica invece cercano di scoprire difetti con probabilità proporzionale alla loro criticità**



Economia del testing **(4)**

- Attenti però:
- **non sempre si ha questa distribuzione**
- **Potrei avere difetti difficili da scoprire ma non critici**
- Fattori da considerare:

5. quanto costa il testing?
6. quanto costa avere un sw con un difetto?
   - costo del danno che provoca x
     probabilità che accada
   - quindi è “inutile” spendere troppo tempo a

In sintesi

- Abbiamo visto che il testing:
  - consiste nel collaudare il sw mediante prova
  - si applica normalmente alla fine del ciclo di vita del sw
  - ha costi considerevoli
- Ricordate che il testing del sw:
  - è molto sensibile ai punti di testing
    - il successo di una prova non è molto significativo
  - è utile per trovare errori ma difficilmente garantisce l'assenza
    - alcuni errori più critici potrebbero essere molto difficili da trovare mediante testing
  - dovrebbe essere applicato durante tutto il ciclo di vita

Terminologia base del testing

- Dobbiamo concordare alcune parole che useremo da qui in avanti:
  - alcuni termini hanno un significato chiaro:
    - testing, debugging, ...
  - altri sono definiti in modo non ambiguo da standard
    - Ad esempio dalla IEEE Standard Glossary of Software Engineering Terminology
      - difetto, bug, ...
  - altri sono usati in modo ambiguo
    - Proponiamo un uso e cerchiamo di essere consistenti
      - errore, ...

Malfunzionamento

- **Un** failure **o** guasto **o** malfunzionamento **è il funzionamento non corretto del programma**
  - legato quindi al comportamento che si osserva durante l'esecuzione

Esempio:

```
//programma che dovrebbe restituire il doppio
//del valore passato come parametro
     int raddoppia(int x) {
        return x*x ;
     }
```

- **se chiamo raddoppia(3) noto un malfunzionamento**
  se chiamo raddoppia(2) non vi e' malfunzionamento

Difetto

- Un difetto o anomalia o fault o bug è un elemento del programma sorgente non corrispondente alle aspettative
  - riguarda quindi la parte statica.
  - uno o più difetti possono causare malfunzionamenti
  - Nell'esempio:

```
int raddoppia(int x) {
    return x*x ;}
```

il difetto e' *x invece che *2.

- Nota:
  - un programma può avere molti difetti e non presentare alcun malfunzionamento.
  - scopo del testing e' quello di evidenziare difetti mediante malfunzionamenti.

Errore

- Errore: **fattore (umano) che causa una deviazione tra il software prodotto e il programma ideale (uno o più errori possono produrre uno o più difetti nel codice)**
  - **Esempio**: errore di analisi dei requisiti, progetto, battitura,...
  - Quando il programmatore commette un errore, il programma ha un difetto che può generare un malfunzionamento

Testing e debugging

- Testing di programmi:
  - eseguire il programma con dei casi di test e analizzare i risultati per trovare i difetti (bug)
- Debugging:
  - correggere i difetti ed eventualmente scoprire gli errori

Casi di test e test suite

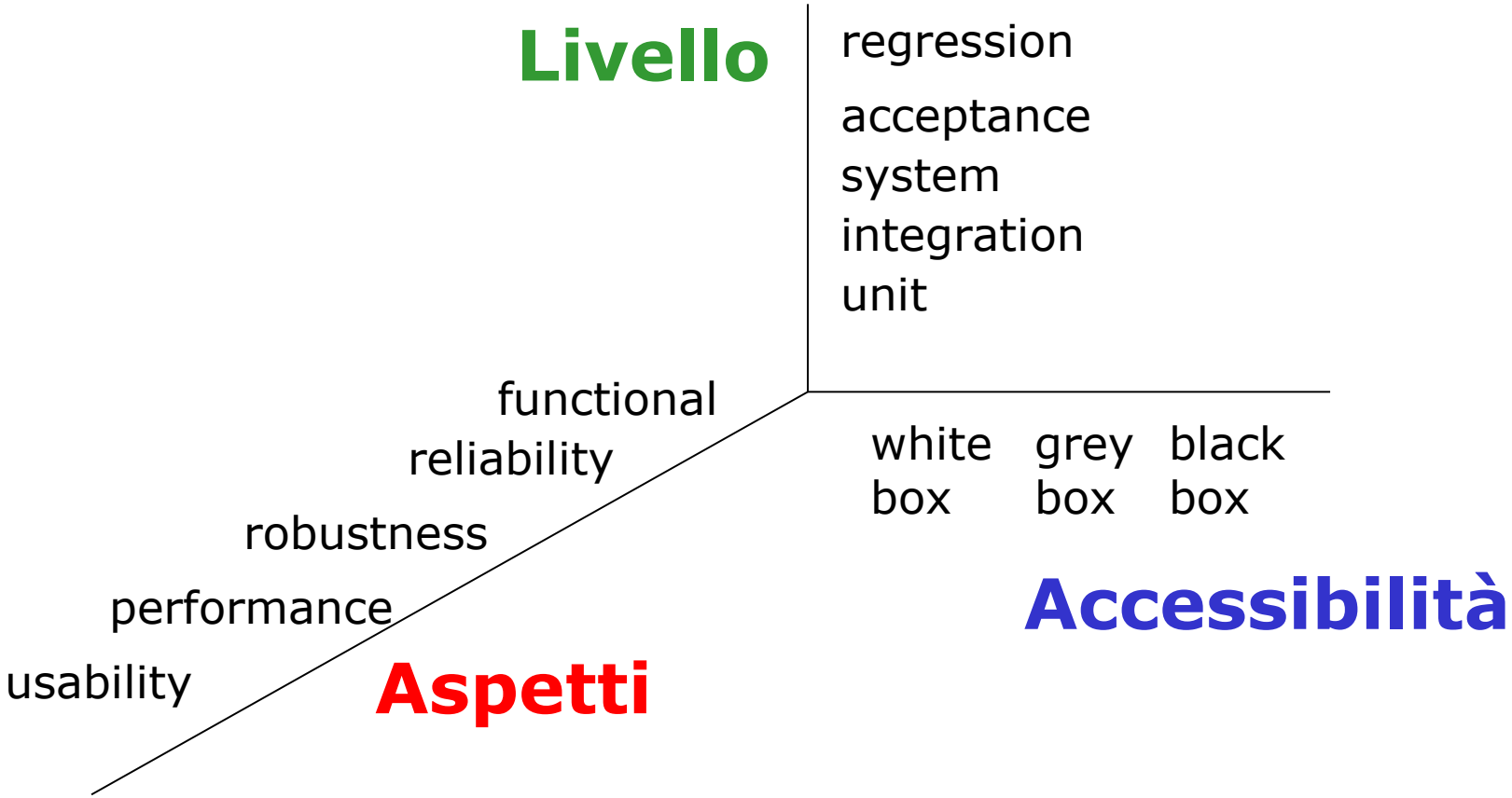
- Se un programma P ha come input un dominio D
- Es: raddoppia(int x) ha D = int
  - somma(int a, int b) ha D = int x int
- Un caso di test t è un elemento di D, cioè t $\in$ **D**
  - **Es per raddoppia un t può essere 3**
  - **Per somma un t è (2,4)**
- **Un test suite T è un insieme di casi di test, cioè T $\subseteq$ D**

Scopi del testing

- Testing di programmi ha due scopi principali
  - mettere in evidenza i difetti (bug) mediante i malfunzionamenti, per scoprire eventuali errori:
    - cercare quei comportamenti che mettono in evidenza difetti
  - poter valutare l'affidabilità di un sw (reliability) e fornire confidenza (test di accettazione)
    - dell'affidabilità del prodotto e della (probabile) correttezza
    - del aver rilevato l'assenza di particolari tipi di malfunzionamenti
    - cercare quei comportamenti più critici o più frequenti per controllare che causino o meno malfunzionamenti

Tipi di testing

Esistono tanti tipi di testing a seconda del livello a cui si effettuano, del tipo di accesso al sistema da testare e degli aspetti che si vogliono testare



Unit Testing

- Testa il codice a livello di singola unità
- In OO (come JAVA)
  - la singola unità è una class
  - UT testa quindi i singoli metodi delle classi
- Per ogni metodo testato - detto test unit - si introducono
  - test driver: metodo che chiama il test unit con opportuni parametri
  - test stub: (opzionale) metodo che sostituisce eventuali metodi usati dal test unit per testare in modo isolato e controllato
    - La scrittura di stub può essere onerosa e viene spesso evitata

Unit Testing - Esempio

- Esempio

```
foo(int x2, int y2) {

   ……
   gig(x2+2);
   ……
   }
testFoo() {
   ……
   foo(x1+1, y1-1);
   // controllo
}
gig(int x3) {
        ……
}
```

→ `foo`: test unit

→ **Metodo da testare**

→ `testFoo`: test driver

→ **Metodo che testa foo**

→ **Simula una unità chiamante**

→ `gig`: test stub **(opzionale)**

→ **Simula un metodo chiamato da foo in modo di isolare il caso di test dal resto del sistema**

In sintesi

- Abbiamo visto le definizioni di:
  - difetto o bug: elemento sintattico nel sorgente sbagliato;
  - malfunzionamento: comportamento sbagliato
  - errore e debugging
- Visto diversi tipi di testing a seconda del livello:
  - di accettazione, di conformità, di sistema
  - di integrazione
  - di unità
  - di regressione

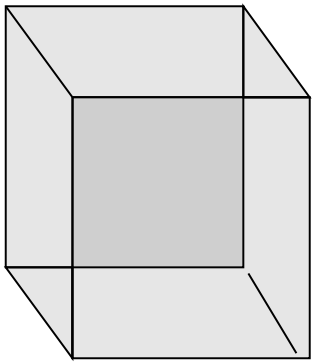
Accessibilità del Testing

- **A seconda del tipo di “accesso” all’unità testata si ha:**
- white box testing, o structural testing, o program-based testing
  - si assume che il programma sorgente sia disponibile
- black box testing, o functional testing, o specification-based testing
  - si assume che non si guardi il programma sorgente ma solo quello che dovrebbe fare
- grey box testing
  - un mix tra i due

White box testing

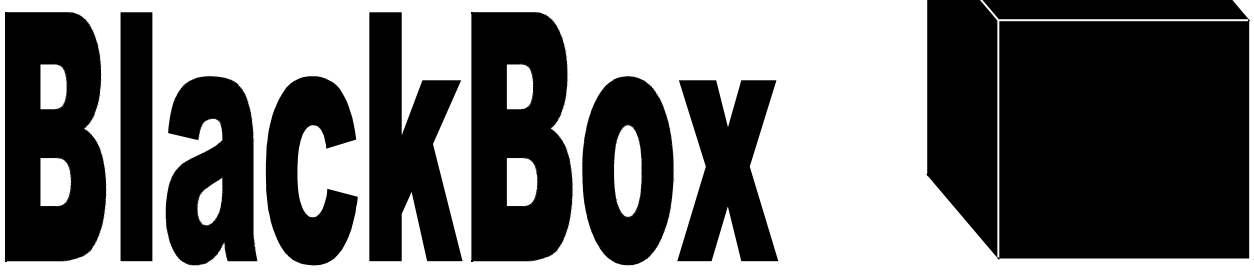
- E’ basato sulla struttura interna del programma
  - deriva i casi di test dal programma
  - controlla e osserva i programmi durante l’esecuzione
- In genere analizza “quanto programma è stato eseguito” o coperto
  - non garantisce che il programma faccia quello effettivamente richiesto
  - ci sono diverse alternative che vedremo

White Box



Black-Box testing

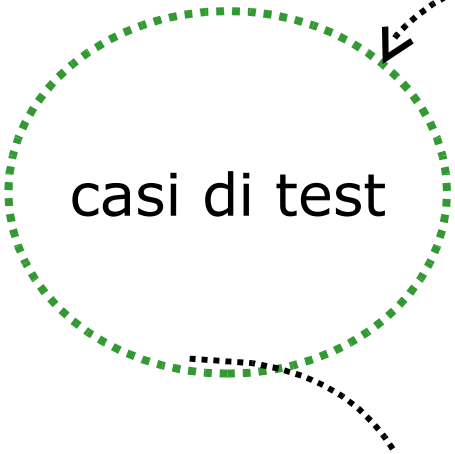
- Ignora la struttura del programma e considera solamente i suoi requisiti
  - deriva i casi di test dai requisiti
  - controlla e osserva il programma solo attraverso la sua interfaccia esterna (input/output)
- In genere misura quanti input/output sono stati utilizzati
  - non tutti gli input vengono utilizzati
  - cerca di scoprire il maggior numero di difetti e di escludere quelli più critici



Program-Based Testing **(1)**

- Passi principali
  1. esamina la struttura del programma
     - quali i punti critici, le decisioni importanti,...
  2. trova i casi di test (cioè input) che soddisfano un certo criterio di copertura
  3. applica gli input (uno alla volta) e osserva il programma (output)
  4. controlla che non si verifichino errori e che gli output siano quelli attesi

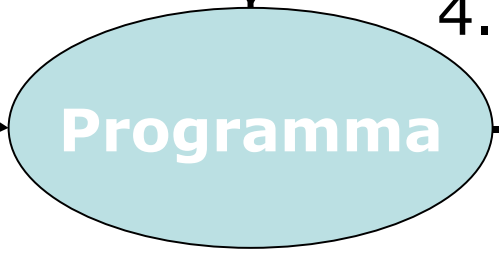
Program-Based Testing **(2)**



2. trova i casi di test

1. analizza il programma

3. seleziona e chiama il programma con input scelti



4. Osserva gli output

Valutazione del program-based testing

- Vantaggi:
  - il codice è una importante fonte di informazione disponibile e usabile
- Limitazioni:
  - non riesce a trovare errori di omissione
    - cosa succede se un programma dimentica di gestire un caso particolare? Guardando solo la struttura tale caso non verrà mai selezionato come caso di test
  - non fornisce “test oracles”
    - come faccio sapere se l’output ottenuto è quello atteso?

Test oracle: modo per stabilire se il test ha evidenziato un malfunzionamento oppure no

Specification-Based Testing **(1)**

- Passi principali:
  - esamina la specifica dei requisiti del programma
  - seleziona un insieme di casi di test che soddisfano qualche criterio
  - applica questi input alla specifica e colleziona gli output **A** (attesi)
  - applica gli stessi input al programma e colleziona gli output **B** (osservati)
  - confronta gli output **A** con gli output **B** e controlla che siano uguali

Specification-Based Testing **(2)**

casi di test

2. trova i casi di test

1. analizza la specifica

3. seleziona e applica gli input scelti alla specifica

Specifica

A

5. confronta gli output **A** e **B**

4. applica gli stessi input al programma

Programma

B

Valutazione del spec.-based testing

- Principali vantaggi:
  - la specifica funziona da **oracolo**
- Principali limiti:
  - le specifiche potrebbero non essere disponibili
    - potrebbe esserci solo il codice
  - le specifiche devono essere “formali”
    - per generare i casi di test
    - per eseguire i casi di test selezionati per ottenere gli output
  - maggiore sforzo rispetto al program-based test

In sintesi

- Abbiamo visto:
  - i diversi aspetti da testare nel sw
  - i diversi tipi di test a seconda dell'accessibilità
- Ricordate che:
  - nel white box o program based testing
    - viene analizzato solo il programma
    - si selezionano alcuni input e
    - si osserva il comportamento del programma
  - nel black box testing o specification-based
    - si tiene in considerazione la specifica di un programma
    - per generare i casi di test
    - per calcolare gli output attesi
    - da confrontare con quelli ottenuti applicando gli stessi input al programma

Limite del testing

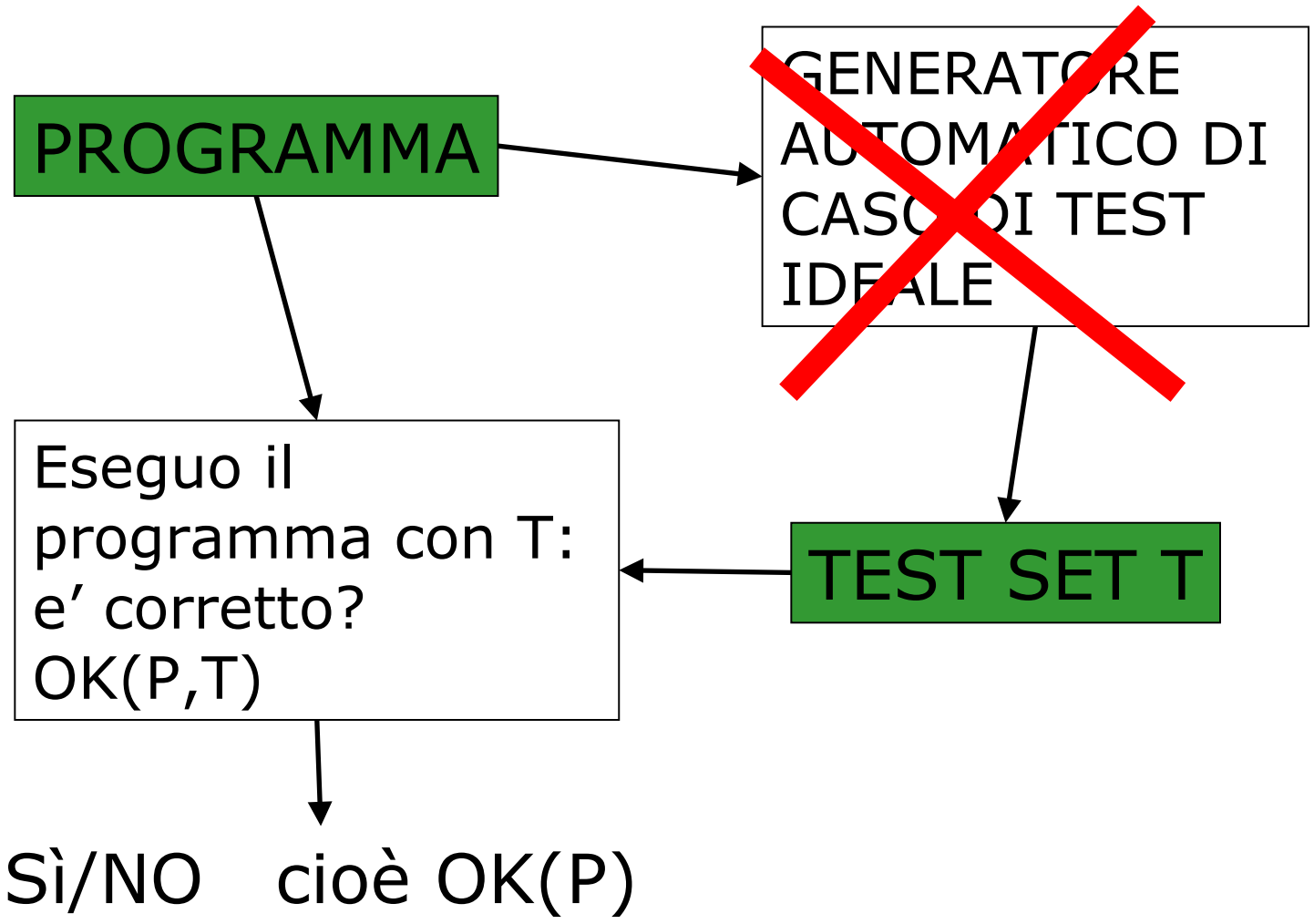
- Il testing
  - **non** consiste nel mostrare l'assenza di difetti in un programma
- ma nel mostrarne la presenza
- Un test che ha successo è uno che trova un bug!

Dijskra

- IL TESTING NON PUÒ DIMOSTRARE CORRETTEZZA DEL SW

Djiskra:
Il test di un programma può rilevare la presenza di malfunzionamenti ma mai dimostrarne l'assenza.

Correttezza mediante testing?



Test Criteria non ideali

- Se non posso definire test criteria ideali a cosa servono?
- Definizione di “empirical” test criteria
  - criteri che non sono ne’ validi ne’ affidabili ma si sono dimostrati utili
- Test criteria come “stopping rule”:
  - ho testato abbastanza (posso essere confidente – ma non certo - che P sia corretto?)
- Test adequacy non solo true o false ma e’ una misura di copertura:
  P x S x T -> [0, 1]

Teorema di Weyuker

- Anche i seguenti problemi risultano indecidibili (non risolvibili mediante algoritmo in tutti i casi):
  - esiste almeno un dato in ingresso che causa l'esecuzione di un particolare comando?
  - esiste almeno un dato in ingresso che causa l'esecuzione di un particolare cammino?
  - esiste almeno un dato in ingresso che causa l'esecuzione di tutte le istruzioni?
  - ...

Test esuastivo

- Il testing esuastivo è quello in cui T = D.
- Non è praticabile
- ....

Generazione in pratica

- Problema non computabile: trovare un insieme di input che esegua tutte le istruzioni
  - **non esiste** algoritmo per risolvere questo problema per ogni programma
  - esistono però algoritmi e tecniche che sono in grado di risolverlo per molti programmi
  - esistono tool per la generazione di casi di test: non è detto che funzionino sempre, però negli usi pratici
    - es. **CUTE, e altri commerciali**

In sintesi

- Abbiamo visto:
  - non esiste un algoritmo per generare per un qualsiasi programma casi di test ideali
  - il testing può provare la presenza di difetti ma non garantire l'assenza.
- Ricordate che:
  - molti problemi riguardo il testing non sono risolvibili per un qualsiasi programma mediante algoritmi
    - esempio come coprire una certa istruzione;
    - però posso scrivere algoritmi che risolvono il problema per alcuni (molti?) programmi
  - per non potendo trovare criteri di test ideali, alcuni criteri di test sono utili in pratica