


# Introduzione al testing

---

Introduzione al testing 1



## Contents

---

- Fornire una overview sullo stato dell'arte delle metodologie di testing
- Discussione:
  - Problemi
  - Possibili soluzioni

Introduzione al testing 2



## Qualità

---

- L'ingegneria del software degli anni 90 si caratterizza come l'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 ....



## Testing

---

- Il TESTING costituisce una delle più importanti e concrete vie di attacco al problema della qualità del software
  - ... analizzare e valutare (e quindi promuovere il miglioramento della) correttezza di una implementazione con riferimento alle caratteristiche definite nei requisiti nelle specifiche e nel progetto....



## Preistoria

---

- Tracce di testing nella preistoria della IS
  - con la prima linea di codice e' 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 DEBUGGING e TESTING ovvero fra attività di ricerca dei bugs ed attività di eliminazione degli stessi.
  - con il tempo le due attività saranno separate non solo nel tempo ma anche nello SPAZIO.



## 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 e' quello di dimostrare la correttezza dei programmi ... quindi un test perfetto si ha quando il programma non contiene errori



## Le Speranze degli Anni 70

- Esempio 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 ecc.



## 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)



## Anni 80: Punto di Vista

---

- il testing e' il processo di esecuzione di un programma con l'obiettivo di trovare gli errori
  - *Un test che non rivela errori e' un test fallito*
- una nuova conquista e' 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



## 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.



## Fattori implicanti la qualità

- La determinazione delle proprietà che devono essere garantite ad un prodotto software dipende da fattori quali:
  - Il tipo di applicazione;
  - L'ambiente in cui il software viene utilizzato;
  - Il modo di integrazione con l'ambiente;
  - La vita prevista per l'applicazione;
  - L'evoluzione prevista.



## Anni 90 Slogan

- il testing e' 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 Herzl: " il testing e' pianificazione, progettazione, costruzione manutenzione e realizzazione di test ed ambienti di test"



## Sfide Anni 90

- Tra le molte sfide :
  - come abbattere i costi di test (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.



## Convalida e verifica

- Il problema di controllare se un software sia corretto è un problema diverso a seconda se si considerino requisiti (informali) espressi dall'utente o specifiche (rigorose/formali) prodotte dall'analista a partire dai requisiti iniziali.
- **Convalida** è l'attività volta a esaminare se si sia costruito il sistema che risolve il problema applicativo che ha motivato la realizzazione;
- **Verifica** è l'attività volta a esaminare se il software prodotto è corretto rispetto alle specifiche dell'analista.



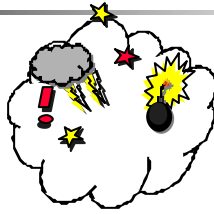
## Chi Deve Fare Testing

### Lo sviluppatore



Ha sviluppato il sistema quindi i suoi test sono biased da aspetti psico-cognitivi: perché deve distruggere il suo sistema?

Obiettivo: la qualità interna



### Un team esterno



Deve comprendere il sistema fa tutti gli sforzi per poterlo distruggere

Obiettivo: la qualità esterna



## Testing (Standard IEEE)

- 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 829-1983) 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



## Related Terms

---

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



## Key Definitions

---

- **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, ecc.) che quando processate producono un fallimento
- In sostanza failure e' un comportamento anomalo, inatteso o errato del sistema mentre il fault e' la sua causa identificata o ipotizzata



## Definitions

- **Defect (difetto)** quando non e' importante distinguere fra fault e failure si puo' usare il termine defect per riferirsi sia alla causa (fault) che all'effetto (failure)
- **Error (errore):** ha 2 significati diversi:
  - una discrepanza fra un valore calcolato, osservato o misurato e il valore corretto, vero
  - l'azione di una persona che causa la presenza di un fault in un software



## Esempio

ERRORE di editing

FAULT

--> "\*" invece di "+"

FAILURE

--> il valore visualizzato

```
program RADDOPPIA ...
```

```
....  
read (x);  
y := x*x;  
write (y)
```

FAULT causata da un errore

FAILURE causato da un'anomalia



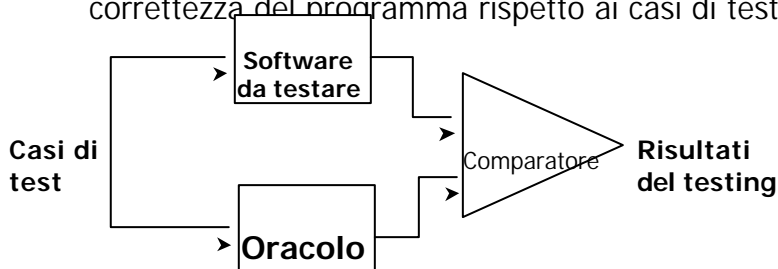
## Tesi di Dijkstra

- Osservando i malfunzionamenti possiamo dedurre la presenza di difetti
- Ma: il testing non può dimostrare l'assenza di difetti, ma può solo dimostrare la presenza di difetti (*Tesi di Dijkstra*)
- 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



## Oracolo

- Per testare un programma è necessario avere una descrizione del comportamento atteso, per determinare se il comportamento osservato coincide con esso. Per tale motivo serve un **oracolo**.
- L'oracolo è un meccanismo per controllare la correttezza del programma rispetto ai casi di test





## Oracolo

---

- Sarebbe opportuno avere a disposizione un oracolo automatico; tuttavia, spesso ciò non è possibile: l'oracolo è umano, e il confronto tra i risultati attesi e quelli ottenuti viene eseguito manualmente.
- Per tale motivo, dato che un oracolo umano è soggetto ad errori, prima di poter affermare la presenza di un fault nel programma, occorre verificare la correttezza della risposta dell'oracolo.
- L'oracolo utilizza le specifiche del programma per valutare l'esito del test; ma anche le specifiche possono contenere informazioni non corrette, imprecisioni, ambiguità, incompletezze, ecc.



## Testing level

---

- Software factory:
  - unit testing
  - integration testing
  - system testing
- Cooperation developer/Customer
  - alpha testing
  - beta testing
- Customer
  - acceptance testing



## Unit Testing

- il testing e' applicato isolatamente ad una unita' (modulo) di un sistema software
- obbiettivo fondamentale e' quello di rilevare errori (logica e dati) nel modulo
- prassi diffusa e' che venga realizzato direttamente dal programmatore che ha sviluppato il modulo l'unita' sottoposta a test

unita': elemento definito nel progetto di un sistema software e testabile separatamente

unita' e modulo sono spesso usati come sinonimi



## Integration Testing

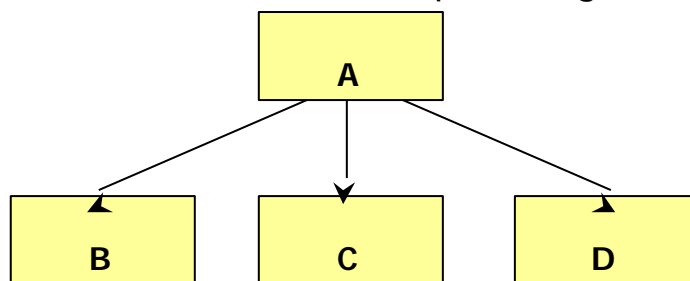
- E' il testing applicato ad un aggregato di due o piu' unita' di un sistema software
- obiettivo fondamentale e' quello di rilevare degli errori nella integrazione fra le unita' e nelle funzioni che l'aggregato deve assolvere
- non e' compito dei programmatori che hanno prodotto le unita' componenti

## Integration Testing

- Le unita' da integrare sono ricavate in base a criteri funzionali ricavabili dall'architettura del sistema
- sono possibili due approcci top-down e bottom up
- talvolta il termine test di integrazione viene riferito anche alla integrazione fra componenti hardware e software

## Bottom-up vs top-down

- Consideriamo l'esempio in figura:





## Bottom-up vs top-down

- Se volessimo testare il modulo A prima ancora di aver realizzato B, C, D (approccio **top-down**), dovremmo utilizzare dei moduli in grado di simulare il comportamento di B, C, D. Tali moduli sono detti moduli fittizi (*stub*)
- Uno stub ha la stessa interfaccia del modulo simulato, ma è più semplice: può ad esempio restituire solo valori costanti, anche se in tal caso occorre fare attenzione (es. un modulo di calcolo radice quadrata non rivelerebbe così dei malfunzionamenti in caso di passaggio di parametri negativi).
- Può talvolta essere necessario realizzare stub interattivi, che richiedono il risultato al testatore.



## Bottom-up vs top-down

- Nel caso di approccio **bottom-up**, occorre invece simulare l'ambiente chiamante. A tale scopo occorre realizzare moduli guida (*drivers*)
- I problemi sono analoghi a quelli incontrati nella realizzazione degli stub, in quanto inizializzare l'ambiente chiamante sempre allo stesso modo provocherebbe una non efficace rilevazione dei malfunzionamenti.



## Big bang vs. Incremental

- Si parla di test di integrazione **big bang** quando tutti i moduli (precedentemente sottoposti ad unit testing) sono integrati in un sol colpo
- Si parla di test di integrazione di tipo **incrementale** quando i moduli sono integrati via via che vengono prodotti ed esercitati singolarmente. I vantaggi sono notevoli:
  - Eventuali anomalie nelle interfacce possono essere rilevate ed eliminate durante lo sviluppo, e non si propagano sul prodotto finale;
  - E' più facile localizzare (e quindi rimuovere) eventuali anomalie;
  - Il test incrementale esercita più a lungo ciascun modulo.



## System Testing

- e' il testing applicato sul sistema software completo ed integrato
- l'obiettivo e' quello di valutare l'adesione del sistema ai requisiti specificati
- va eseguito dal team addetto al testing (esterno al gruppo di sviluppo)



## System Testing: Requisiti

- i requisiti di sistema non sono solo le funzionalita' esterne
- fondamentali possono essere i requisiti di qualita', di prestazioni, stabiliti ad esempio sulla base di un modello (o profilo) di qualita' del prodotto opportunamente istanziato
- Prestazioni, Manutenibilita', Usabilita' ...



## System Testing

- **Funzionale:** funzionalita' come viste dall'operatore
- **Configurazione:** tutti i comandi, gli automatismi per scambiare/cambiare le relazioni fisiche logiche dei componenti HW
- **Recovery:** capacita' di reazione del sistema a cadute
- **Stress:** affidabilita' in condizione di carico limite
- **Sicurezza:** invulnerabilita' del sistema rispetto ad accessi non autorizzati



## Acceptance Testing

---

- Testing effettuato sull'intero sistema sulla base di un piano e di procedure approvate dal cliente
- L'obiettivo e' quello di mettere il cliente, l'utente o altri a cio' preposti (collaudatori o enti ad hoc) in condizioni di decidere se accettare il prodotto
- E' a carico del committente
- Segna il passaggio del sistema dal produttore all'ambiente operativo
- Puo' talvolta essere piu' una demo che un test



## Alpha Testing

---

- uso del sistema da parte di utenti reali ma nell'ambiente di produzione e prima della immissione sul mercato
- talvolta riferito per il testing da parte di un cliente (o gruppo di clienti) privilegiato(i)



## Beta Testing

- Installazione ed uso del sistema in ambiente reale e prima della immissione sul mercato
- Strategia adottata da produttori di packages per mercato di massa
- Talvolta il beta testing e' preceduto da un alpha testing, o un beta testing da parte di un gruppo piu' ristretto di utenti
- Problemi di confidenzialita'



## Testing di regressione

- Si parla di test di regressione quando il programma da testare costituisce una nuova versione del prodotto
- Il test di regressione ha l'obbiettivo di verificare compatibilità e differenze della nuova versione rispetto alla precedente
- E' il test più semplice da automatizzare, purché vengano registrati, fin dalla prima versione, casi di test e relativi risultati
- IL test di regressione consiste nell'eseguire la nuova versione con gli stessi dati di test della vecchia, e confrontare i risultati con quelli precedentemente registrati nella base dati.



## Fondamenti teorici

- 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 rispetto ad un ingresso è indicata con  $ok(P,d)$
- Un programma è corretto  $\Leftrightarrow \forall d \in D, ok(P,d)$



## Adeguatezza di un test

- Scopo dell'attività di testing è la rilevazione di malfunzionamenti. Un test  $T$  rileva un malfunzionamento se il programma  $P$  non è corretto per  $T$ .  $T$  *ha successo* per un programma  $P$  se rileva uno o più malfunzionamenti presenti in  $P$ , viceversa il test  $T$  è inadeguato.
- Un test  $T$  è detto *ideale* se l'inadeguatezza di  $T$  rileva la correttezza del programma  $P$ , cioè se  $ok(P,T) \Rightarrow ok(P)$



## Selezione dei casi di test

- Uno degli scopi della teoria del test è la definizione di metodi per la selezione di test che approssimino i test ideali.
- Un *criterio di selezione* di test  $C$  per un programma  $P$  è un insieme di predicati sul dominio di ingresso  $D$  del programma.
- Un test  $T$  è selezionato dal criterio  $C$  se  $\forall t \exists c \in C$  tale che  $c(t)$  è vero e  $\forall c \in C \exists t \in T$  tale che  $c(t)$  è vero.



## Affidabilità di un criterio di test

- Un criterio di selezione di test  $C$  è affidabile per un programma  $P$  se per ogni coppia di test  $T_1, T_2$  selezionati dal criterio  $C$ , se il test  $T_1$  ha successo, allora anche  $T_2$  ha successo e viceversa
- Un criterio di selezione  $C$  è *valido* per un programma  $P$  se, qualora il programma non sia corretto, esiste almeno un test  $T$  selezionato da  $C$  che ha successo per il programma  $P$ .



## Teorema di Goodenough e Gerhart

---

- Il fallimento di un test  $T$  per un programma  $P$  selezionato da un criterio  $C$  affidabile permette di dedurre la correttezza del programma  $P$ , cioè:

*affidabile*  $(C, P) \wedge$  *valido*  $(C, P) \wedge$

*selezionato*  $o(C, T) \wedge \neg$  *successo*  $(T, P) \Rightarrow ok(P)$

- L'affidabilità del criterio  $C$  garantisce lo stesso risultato (in termini di rilevamento dei malfunzionamenti) per tutti i test selezionati.



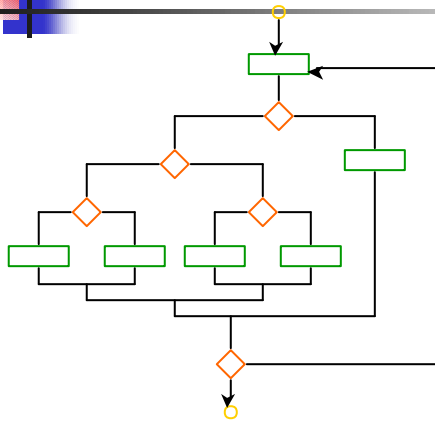
## Validità dei criteri di test

---

- Un criterio di selezione  $C$  che non esclude alcun elemento del dominio di ingresso  $D$  del programma  $P$  è valido, ma non affidabile. Infatti, se esiste almeno un malfunzionamento, questo si manifesterà in corrispondenza di un dato ingresso  $d$ , ma il criterio  $C$  seleziona almeno un test  $T$  che contiene  $d$  e quindi rileva il malfunzionamento; non tutti i test selezionati dal criterio  $C$  conterranno però un dato di test che rileva il malfunzionamento.
- Un criterio di selezione  $C$  non soddisfatto da alcun elemento di  $D$  è affidabile, ma non valido.
- Un criterio  $C$  che richiede che il test selezionato contenga tutto  $D$  è valido e affidabile (*test esaustivo*)



## Test Esaustivo???



$10^{14}$  cammini possibili

1 test / millisecondo

Ciclo da  
1 a 20



3170 anni per  
testare questo programma

*E non e' tutto ....*



## Teorema di HOWDEN

- 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.
- Il teorema di HOWDEN corrisponde alla tesi di Dijkstra: Il test di un programma può rivelare la presenza di malfunzionamenti, ma mai dimostrarne l'assenza.



## Amarezze del Testing

- 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 ... (macchina di turing, halting problem, teorema della esistenza di funzioni non calcolabili, etc...)
- i problemi indecidibili esaltano l'ingegnere
- es. stabilire se l'esecuzione di un programma termina a fronte di un input arbitrario e' un problema indecidibile
- ma se stiamo costruendo una societa' basata sul software...



## Equivalenza di Funzioni

- 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
- altre indecidibilita' derivano direttamente da quelle enunciate
  - 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?



## Teorema di Weyuker

- 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?
- Dato un problema indecidibile, è però possibile individuare sottoproblemi significativi decidibili. Inoltre l'ingegno umano è pur sempre in grado di risolvere problemi indecidibili in maniera creativa e non meccanica