FAULT : non mi accorgo perché si sia attivato

FAILURE : per causa scatenante (errore umano)

↳ nel tuo software c'è un fault (nell'implementazione: moltiplicazione al posto della divisione)

↓ la [illegible] **intensità** si chiama errore

causa: software (fault)

↓ si fanno test sul piccolo per risalire la catena causale

↓ il comportamento di un pezzo di software: **variano gli input e i cammini**

↓ pochi input → un cammino di esecuzione

↓ al massimo una verità sul prodotto (POCO!)

pochi parametri

↓ + facile fare test

le condizioni di un test:

- cosa sto guardando
- input esplicito (arg) o input impliciti
- stato dell'ambiente di esecuzione (variabile presa da un'altra parte)

intensità di test ⇒ determina alta qualità (PdQ)

PdQ in ordine con il PdP (deve versare risorse perché ci sia qualità)
↑ tempo

LEGGE DEL RENDIMENTO DECRESCENTE

↳ ascisse = 'quanto ci costa'

↳ ordinata = 'beneficio' (numero cumulativo di failure che traccio)

inizialmente smaschero fault: failure ⟹ poi non ce ne accorgiamo

⟹ l'atto di correggere fault introduce fail
(tolgo uno e ne introduco 2), non posso fare test all'infinito

in questo caso il test è utile ⇑

test si progetta e si implementa — bisogna trovare failure ⇑

⇓

atteggiamento senza pietà verso i fault
(merciless)

trovo a cost → stipendio dipendente dal prodotto
(+ consegno + soldi)

poi bisogna toglierli

PRINCIPI TESTING:

⟹ se faccio un test lo faccio dove è più facile sbagliare

⟹ cosa serve per lavorare bene (dicevo)

⟹ test va per scoprire nel dettaglio regressioni se scoperto un failure

⟹ faccio un test solo dopo che so cosa aspettarmi

⟹ un test è tale se riproducibile (se possiamo ripeterlo)

Cosa deve fare il test? (attività di prova)

- ...
- ...
- compilazione → dal la parte eseguibile da salvare
-
- analisi → confronto con attese (si trova in specifica)

i test si possono archiviare (archivio → contiene anche l'esito del test)

↓

modello A V

unità : non c'è in un linguaggio di programmazione
è non smontabile

↳ **tipo funzionale** (black box) : riguarda solo la funzione
scruta non l'implementazione

↳ **tipo strutturale** (white box) : guardo come lavori

tipo funzionale : decido che input (esplicito o impliciti) dare e quali output (espliciti o impliciti) attendersi

misuro l'output ←

↓ non guardo dentro la cosa che sto guardando

(se esito negativo ⇒ scoperchio la black box !

quando do gli input cerco di ridurre il numero di test da fare :

• input ⇒ fisso : valori attesi
(intervalli, confini, valori illegali)

↓ quante classi di equivalenza :

- illegali fuori
- legali dentro
-
-
-

test per ogni classe di equivalenza ←

↓ almeno 5 test per ogni parametro

tipo strutturale : guardo la sintassi, mi serve sapere il cammino

↳ misura di copertura

↳ è massimo quando : miei test sono eseguito tutti : cammini e hanno prodotto esito atteso (disincentivo software corretto)

Copertura strutturale ⇒ ossia:

- ho eseguito tutti gli statement dell'unità (statement coverage)
- (branch coverage) ho un vero e un falso nel branch, è coperto tutto se ho navigato tutte le opzioni, si parla di: cammini

↓

decisione = ciò che governa un branch

↓

if complicati fanno MALE!!

if (A && B or (b > 3))

:o

Ritorno in aula ...

TESTING: • arrivare al collaudo sapendo che andrà bene

• il software è grande e complesso (non basta una singola persona, troppo complicato)

COLLAUDO → esplora scenari concordati

TEST DI SISTEMA non può essere sufficiente per il collaudo ↓ richiede cliente

↙ utente ⇐⇒ siamo andati in profondità

↓ esplora tutto ciò che dentro dagli scenari

↪ mostro a me stesso che il sistema faccia i requisiti sotto il cartoncino (sopra ci sono assistenza e vincoli)

↓

requisiti di sistema visti come prodotto (sopra ci sono gli scenari d'uso)

↓ così si assegna usando il prodotto

requisiti = albero da esplorare, foresta

sono le cose che gli scenari implicano

riguardano l'input, output e footprint

SCENARIO D'USO (succede questo)
↓
REQUISITI (serve questo, questo)

↪ poi ci saranno aspettative precise sull'implementazione

↓

specifici per evitare WORKAROUND!!! ← non lasciare al libero arbitrio dell'implementatore

E SE VA MALE IL TEST DI SISTEMA? COME CAPIRE DOVE È IL PROBLEMA?

E SE GLI SCENARI NON FOSSERO SUFFICIENTI?

↓

C. DEVE ESSERE QUALCOSA CHE CI PRECEDA! TEST SULLE PARTI PICCOLE

<u>ASSEMBLO PARTI</u> CHE PRODUCANO SCENARI ALTI