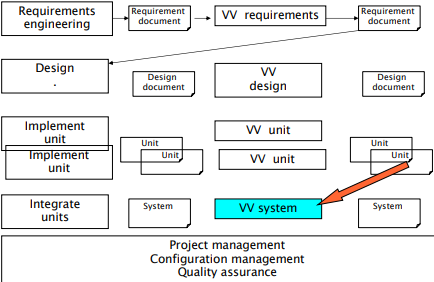
# 12pt3 System test 219

Test di **tutte le unità** in un’applicazione eseguiti sulla piattaforma finale o su una di sviluppo.

**Chi sviluppa** system test:

* Developer, user di piattaforme diverse
  + **Developer**:
  + **End-User**:
    - **Acceptance** testing: user prova l’applicazione in un contesto reale e verifica se c’è corriposondenza con i requisiti
    - **Beta testing**: test informali eseguiti da un gruppo di utenti

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**Come fare** system test:

* **Proprietà funzionali**: partendo dal documento dei requisiti, bisogna effettuare una coverage improntata sul documento dei requisiti o sugli scenari
  + Bisogna considerare come l’utente usa il sistema
    - Bisogna valutare la frequenza di utilizzo dei vari requisiti e i test su quei requisiti saranno più importanti
      * *(es: in questo caso FR3 più usato)*
* **Non funzionali**:
  + Immagine che contiene testo, Carattere, schermata, bianco

    Descrizione generata automaticamenteSi possono testare solo quando il resto è disponibile

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

#### Classificazione dei test:

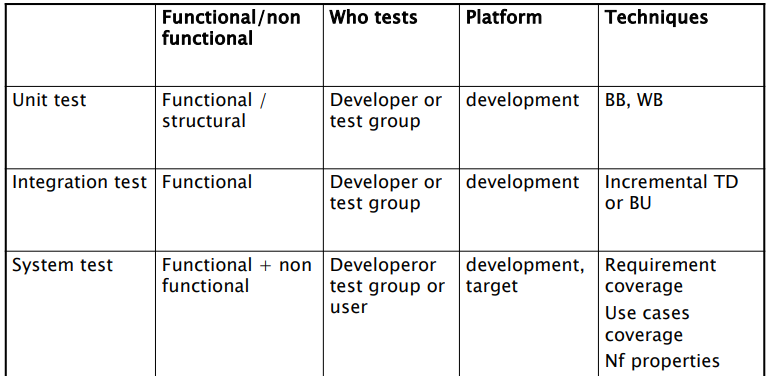


Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

#### Coverage:

dipende dalla fase in cui mi trovo, le entità possono essere diverse:

* Test case
* Requisiti
* Funzioni
* Elementi strutturali
  + Statement, decisioni, moduli

##### Immagine che contiene testo, schermata, Carattere, numero Descrizione generata automaticamenteEsempio su come calcolare coverage:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

→ libreria va testata su diverse piattaforme prima di usarla in un progetto

#### Reliability testing:

Variante del system testing con cui calcolo la probabilità che avvenga una failure in un certo periodo di tempo:

* **Defect rate** = defect/time
* **MTBF**: tempo medio tra due difetti

Vincoli del realiability:

* Largo numero di test cases
* Indipendenti
* Immagine che contiene diagramma, testo, linea, Diagramma

  Descrizione generata automaticamenteFix dei difetti non deve introdurre altri difetti

NOTA su **HW**: quando si sceglie hw bisogna valutare le statistiche fornite dal venditore; defect amentano sul lungo termine.

Immagine che contiene diagramma, linea, Diagramma, testo

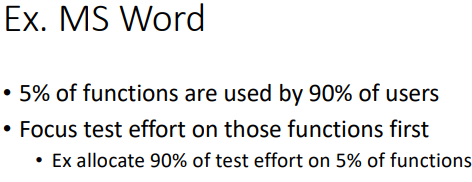
Descrizione generata automaticamente

NOTA SU **SW**: curva scende sul lungo termine

### Immagine che contiene testo, Carattere, schermata, bianco Descrizione generata automaticamenteRISK BASED TESTING

* Identifico rischi
* Caratterizzo rischi
  + Probabilità
  + Effetto
* Rank dei rischi
* Sviluppare test per coprire maggiormente le situazioni rischiose

### USER PROFILES BASED TESTING

Variante del risk-based

* Identifico gli user e usage profiles
* Ranko per frequenza d’uso
* Testo i più usati

### REGRESSION TESTING

Fixo una cosa, se ne rompe un’altra

* Per evitare questo, ogni volta che si pusha, c’è un controllo sulla test suite

## TEST, DOCUMENTAZIONE E AUTOMAZIONE

I casi test devono essere documentati in modo tale che non vadano persi e possano essere riapplicati ma anche **automatizzati**  in modo tale che l’applicazione dei casi di test sia rapida e priva di errori.

Di fatto si distinguono:

* **Informal testing**: casi di test non sono documentati
  + Non ci sono report di test
  + Va bene per chi sviluppa ma stop
* **Test formali:**
  + Si usano applicativi come word o excell per documentare tutto

**Rappresentazione dei test cases:**

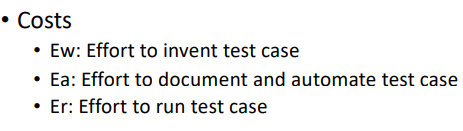
* Non operational
  + Word, excel amano
* Operation:
  + Script o pagine web che interpretano risultato di uno script producendo documentazione html
    - Jest

**NOTA**:

* fare test automatici non è gratis → tutti i test sono scritti e vanno riscritti se si cambia l’architettura del sistema
* Conviene automatizzare un test case se è eseguito per un numero maggiore alle 2/3 volte

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente



**Buon test case:**

* Deve trovare con una buona probabilità gli errori
* Deve testare cose reali, inutile tesatare cose inutili che non saranno mai eseguite
* Non deve essere troppo banale ma neanche troppo complesso
* Deve mettere in risalto le failure
* Test deve essere indipendente dagli altri
* Test suite deve essere esaustiva

Mutuation testing: tecniche per verificare se un test è valido oppure no: 267

* Scrivere vai test
* Inserire variazioni all’interno del nostro programma
* Vedere se test è sensibile a queste variazioni (test dovrebbe trovare l’errore che ho richiesto)

Usata sia per sw che per hw.

Terminologia:

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente• **Mutante**: programma con una modifica   
• **Mutante uccidibile**: non funzionalmente equivalente.

* Un caso di test può ucciderlo

**• Mutante equivalente**: funzionalmente equivalente al programma.

* Nessun caso di test può ucciderlo.

**• Punteggio di mutazione:**

* proprietà di una suite di test (obiettivo: 100%)
* Uccisi mutanti non equivalenti / tutti i mutanti non equivalenti

**Immagine che contiene testo, linea, schermata, diagramma

Descrizione generata automaticamenteMutazioni** che si possono fare:

* Eliminare una riga
* Swap tra righe
* Invertire relazioni booleane
* Cambiare una variabile
* Cambiare un’espressione complicare in true/false
* **Test deve accorgersene**

Come tool per Unit & Integration Testing usiamo Jest

GUI: Cypress mi fa eseguire un test in GUI (utile usando gli id degli elemenyti html)