# 12-Verification &Validation

Va eseguita dopo ogni fase, in modo tale da verificare che non si propaghino errori alle fasi successive.

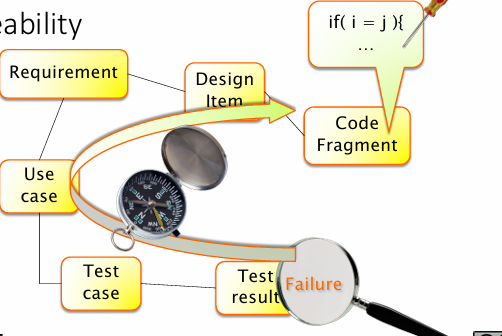
**Validation**:

* Immagine che contiene testo, diagramma

  Descrizione generata automaticamenteis it the right software system?
* Effectiveness
* external (vs user)
* reliability

**Verification**:

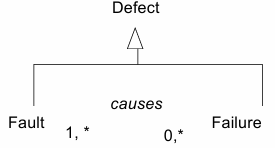
* is the software system right?
* Efficiency
* internal (correctness of vertical transformations)
* correctness



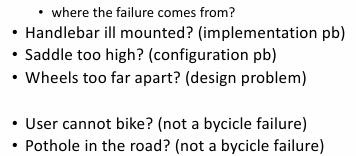
**Tracciabilità**: svolgere varigication & validation dopo ogni fase, aiuta a tracciare e a trovare errori

Il ***costo di fixare difetti nel codice cresce all’aumentare nel tempo del momento in cui me ne accorgo.***

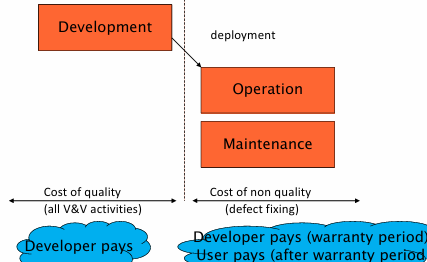
**Terminologia**:

* **Failure**: evento che fa comportare il sw in maniera non attesa
* **Fault**: particolare feature del mio sw che causa la failure
  + Errore nel codice (cnt non incrementato)
  + Faccio in maniera corretta una cosa ma non come richiesto
* Immagine che contiene testo, linea, Carattere, diagramma

  Descrizione generata automaticamente**Defect**: failure/fault

Difetto è caratterizzato da:

* Attività di inserimento
* Attività di rimozione

**Chi paga?**

Se si rompe qualcosa durante la garanzia, paga azienda; altrimenti paga utente.   
Bisogna decidere a priori chi è responsabile della qualità e che deve pagare in caso di problematiche.

**Technical Dept**: faccio una funzione adesso che mi costa meno che mi tornerà prima o poi con i problemi da risolvere.

* Variabili con nomi senza senso
* Copia incolla a caso

**Tipi di V&V:**

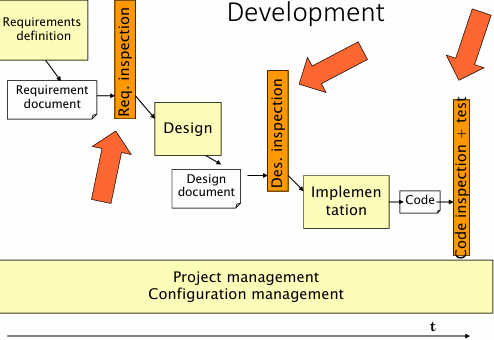
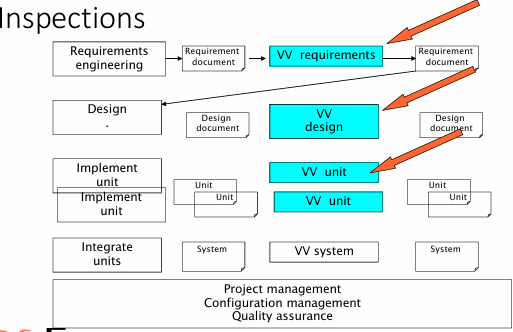
* **Statico**:
  + Inspections
  + Analisi del codice sorgente
* Dinamico
  + **Testing**

Le diverse attività, hanno diverse **tecniche di V&V**:

* **Requirements**:
  + Inspection del dcoumento dei requisiti
  + Prototipo GUI
  + Prototipo prodotto (dyn)
* **Design**
  + Inspection del documento di design
* **Implementation**:
  + Test (dyn)
    - Unit
    - Integration
    - Application
  + Inspection del codice sorgente

### **Inspection**: (test statico)

Persone leggono i derivable e cercano dei defects → trovano i defects senza trovare soluzione.  
**🡪 segnalano i problemi.**



***Nota****: Early defect detection improves product quality and reduces avoidable rework (down to 10-20%)*

**Ruoli in un gruppo di lavoro**:

* **Moderatore**: guida il processo di ispezione e in particolare la riunione di ispezione
  + Seleziona i partecipanti, prepara il materiale
    - Di solito esternamente dal progetto che produce documenti da ispezionare
* **Lettori**: leggono i documenti da ispezionare
* **Autori**: Rispondono alle domande
* **Scribe**: scrivono log delle ispezioni

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente  
  
  
**Overview**: presentare rapidamente l'ispezione: obiettivi del gruppo e il documento da ispezionare

**Preparation**: lettura individuale applicando tecniche di inspection

**Meeting**: letture di gruppo, discussioni su problemi, trovare accordi. Scrivere logs. Moderatore deve mantenere il focus, la pace e stoppare le discussioni.  
**Rework**: autore fixa difetti/problemi

**Follow up**: ripetere ispezione o chiudere e passare alla prossima fase

##### *Prerequisiti per svolgere correttamente ispenction*:

* Impegno da parte del management
  + Sforzo investito in anticipo, che "non produce nulla"
* Trovare i difetti, non correggerli
* Il documento in fase di ispezione soddisfa gli standard di qualità
* Risultati non utilizzati per valutare le persone (e in particolare l'autore)
* Approccio costruttivo
  + Il gruppo ha l'obiettivo di produrre il miglior documento possibile
    - * Nessun gioco "uccidi l'autore"
      * Immagine che contiene testo, schermata, Carattere

        Descrizione generata automaticamenteNiente riunioni "rilassati e chiacchiera"

DA SLIDE 38 A SLIDE 65

Nota **su Defect taxonomy**: può essere:

Su un livello (**Brasili**)

Omissioni

Fatti incorretti

Inconsistenza

Ambiguità

Infomrazioni estranee

Su due livelli (**Porter**)

Omissioni

Funzionalità mancanti

Performance mancante

Environment mancante

Interfaccia mancante

Commissione

Informazioni ambigue

Informazioni inconsistenti

Funzionalità extra non corrette

Sezione errata

Nota:

Effort per code inspection

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

* Checklist for requirement: verifico che tutti I requisiti siano stati soddisfatti
* Checklist for code: dipendente da linguaggio usato e dai requisiti
  + Diversi fault
* Defect-based-reating: uno scenario ha delle specifiche classi si defect
* Perspective-based reading: scrivo scenario e poi codice e test

### TESTING (test dinamico)

Può essere di sistema o unitario. Può essere fatto per far girare un sistema in determinate condizioni per cercare di trovare differenze tra il comportamneto attuale e quello atteso.

Scopo: trovare i difetti di sw. (test ha successo se trova dei defect)

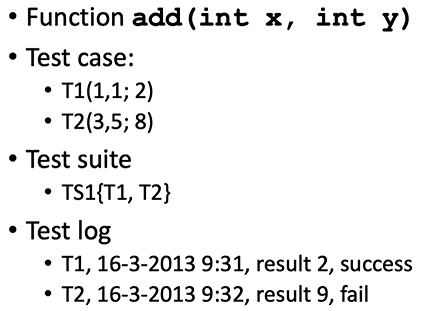
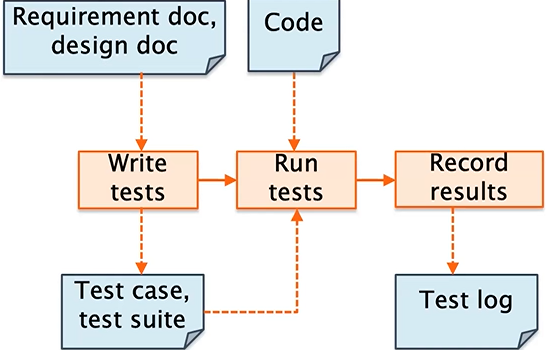
**Testing!=debugging**

* A seguito di test non funzionanti, seguirà una fase di debug per risolvere errori

**Fare una test suite:**

* Raggruppare i test secondo una certa logica
* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamenteBisogna specificare il contesto in cui devono essere validi



## ORACLE 15/05/2024 \_65

Immagine che contiene testo, linea, Carattere, diagramma

Descrizione generata automaticamentel’idea è di avere qualcosa di automatico che effettui i test.

Bisogna conoscere il comportamento atteso del programma, prima di eseguirlo → test vanno effettuati in base allo scenario e non in base alla funzione.

* Generati dai requisiti
* Fatti da altri, non le stesse persone che lo sviluppano
* Può identificare delle problematiche che prima non c’erano → **regressione**

Per verificare la correttezza di un sw devo controllare l’output per tutti gli input possibili → Test **Esaustivi**:

* L’impatto di un testo non fatto potrebbe causare un costo molto elevato
* Immagine che contiene diagramma, linea, Piano, testo

  Descrizione generata automaticamenteEs:

servirebbe troppo tempo,  
  
devo trovare un altro modo

* Bisogna fare in modo che i test abbiano una buona coverage

***NOTA****: i test, rilevano la presenza di errori e non la loro assenza.*

Bisogna trovare il giusto insieme di test cases che devono essere eseguiti

* Test cases per affidabilità e validità

**Concetti base:**

* **D**: input (program domain)
* **d**: singolo input → P(d): output
* Ok(d) <-> P(d) corrispondente su oracle
* **Test**: **T** **⊆** D
* *SUCC(T) <->* ***∀*** *t ∈ T, OK(t)*

**Criterio**: Test possono essere selezionati secondo croterio

**Theoria fondamentale:**

* Il successo di un test T selzionato da un criterio valido e affidabile, implica la correttezza di T
* **Uniformità**: quando assegno un criterio, sto dicendo che sto andando a selezionare delle caratteristiche sono abbastanza uguali in base a quello che gli scrivo
* **Howden**: per un programma generico è impossibile trovare un algoritmmo che ci generi un numero finito di test ideale. → bisogna fare i conti con il fatto che è impossibile trovare il test ideale che trovi tutti i bachi.
* **Winberg**: è impossibile che un developer testi il proprio codice senza alcun pregiudizio in quanto io ho in testa come è questa funziona e non testerei molti casi che portano a bachi.
* **80%** dei difetti avviene nel 20% dei moduli

**Dunque**:

* Bisogna testare tutto
* Bisogna creare ed effettuare test in base al rischio
* **Immagine che contiene testo, schermata, diagramma, Carattere

  Descrizione generata automaticamenteI test si classificano in:**
  + Per **oggetto** **testato** →
    - unità (singola funzione)
    - integrazione (più funzioni iniseme)
    - sistema (tutto)
    - regressioni → se si è rotto qualcosa che prima funzionava
  + Per **approccio**:
    - Black box
    - White box → guardo il codice sorgente e faccio chiamate in base a quello che viene chiamato
    - Reliability → Does it satisfy the customer need?
    - Risk based → identificando i rischi
  + Per **formalità**:
    - Informale → controllo se funiona
    - Formale → è accompagnato da una serie di documenti formali → report
      * Si ha uno storico dei test

#### **Immagine che contiene testo, schermata, Carattere, numero Descrizione generata automaticamenteTest per approccio:**

**Dato un item da testare, l’approccio può essere:**

* Requirement driven
* Structure-driven
* Statistics-driven
* Risk-driven → vedere definizione da slide

*Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteDal punto di vista della* ***copertura****, possiamo valutare quanto il nostro test copra:*

**Coverage**: quanto ho coperto

**Regression testing**: verifica che tutto quello che ho testato precedentemente, continui a funzionare.

### **UNIT-TEST-Black BOX**

**Random**: testo randomicamente le funzioni

* Indipendente dai requisiti
* Richiedere molti test cases

#### **Equivalence classes partition:**

prende tutti i possibili input e li divide in blocchi dove ogni blocco ha un comportamento simile in termini di requisiti.

* **Boundary condition**: valori a cavallo tra due blocchi (es: 0, -1 sono a cavallo tra numeri positivi e numeri negativi, quindi se sto testando per positivi/negativi, molto probabile abbia problemi lì)
  + Se sto testando lunghezza pw>8, controllo 7, 8 ,9
  + Immagine che contiene testo, schermata, Carattere, numero

    Descrizione generata automaticamenteControllo anche maxint+1

**Criterio, predicato, partizioni:**

* Criterio: età
* Predicato: età>30
  + Definisce partizioni

Quando si crea una classe di equivalenza, devo creare **almeno un test che valuti il caso corretto**:

* Effettuare un test valido per quella classe di input
* Le classi vanno testate nei loro bordi in modo tale da trovare in maniera più probabile gli errori
* Testare anche con valori fuori senso (maxVal + 0.1)

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteEsempio:**

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

***Nota****: se la funzione ha degli stati, va testata in diversi stati:*

* *Bisogna capire gli stati leggendo la documentazione*
* *Bisogna chiamare la funzione per riuscire a chiamare questo stato*

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

**esempio**: devo creare una funzione che crea la media degli ultimi 3 nr passati, esclundendo i negativi:

* **Criteri**:
  + **Stato**: n elementi ricevuti
  + **Int i**: positivo, negativo

*Nota: i test devono partire tutti dallo stesso stato e devono arrivare tutti allo stesso stato.*

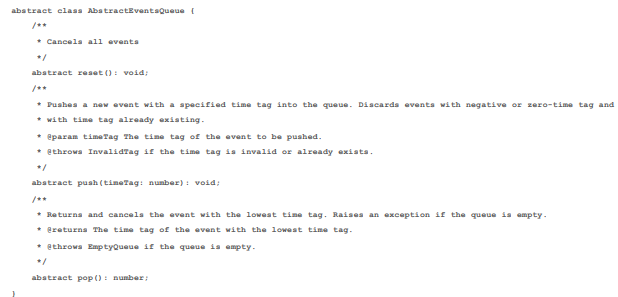
Es: Test of OO classes.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

**Test Case:**

* Insieme di test dello stesso tipo → che testa quella particolare equivalence class

## Unit test -white box 22 mag. 24ok

* **Black box** → funzionale:
  + Random
  + Equivalenti di partitioning di classi
  + Contizioni boundary

*Nota: se non c’è un’ottima documentazione, diventa difficile eseguire un test black box efficace*

* **White box** → strutturale:
  + Copertura degli elementi strutturali
    - Ho a disposizione **codice** e **struttura**, posso scrivere test più specifici
      * Statement
      * Decision, condition
      * Path → percorsi
      * Loop

*Solitamente il white-box testing viene fatto nell’ambiente di sviluppo*

#### **WHITE BOX- statement**

*Immagine che contiene testo, ricevuta, Carattere, linea

Descrizione generata automaticamente→ se questa funzione la testo bene, posso usarla in altri codici portandomi anche i test eseguiti.*

*Immagine che contiene testo, Carattere, bianco, tipografia

Descrizione generata automaticamente*

*(coverage si misura in percentuale)*

Dopo aver fatto i test che fanno eseguire tutti gli statement, bisogna creare il **control flow graph** della nostra funzione:

* Immagine che contiene cerchio, diagramma, clipart, cartone animato

  Descrizione generata automaticamente**Nodo**: istruzione atomica (assegnazione/istruzione) o decisione (if o loop che fa un controllo)
* **Arco/Edge**: trasferimento del controllo da una parte all’altra (da una funzione parto ad una successiva)
* Immagine che contiene testo, schermata, diagramma, Carattere

  Descrizione generata automaticamente**Basic block**: una serie di nodi possono essere accorpati se sono uno di seguito all’altro   
  senza via di uscita

BB: Basic Block

Immagine che contiene testo, diagramma, schermata, cerchio

Descrizione generata automaticamente

**Usando BasciBlock→**

#### **Node coverage**

Per calcolare la node coverage, devo fare un test e vedere **quanti nodi esegue** all’interno del nostro grafo.

*Se eseguo una* ***test-suite****:*

* *Posso avere un node coverage cumulativo sulla test\_suite e non sul singolo test*

In generale, la node coverage tende alla statement coverage, alla stessa maniera la basic block coverage

#### **Decision coverage:**

Quante **decisioni** **copre**, devo provare a coprire tutte le decisioni di un programma con vero/falso

* Decision: espressione booleana
* Decision coverage= #decisions covered/ #decisions

*Nota: Nella decision coverage, bisogna testare ogni decisione in modo indipendente dalle altre*

*In generale, la edge coverage tende alla decision coverage*

#### **Edge coverage:**

Quanti **archi copre**.

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente

In questo caso, con -1 → si avrebbe 100% di node coverage ma 50% di decision; va testato anche numero >0 → testare le decisioni sia quando true che quando false.

*Nota: Edge coverage implies node coverage, not viceversa*

#### **Condition Coverage**

Una decisione è composta da una combinazione di termini/Condizioni

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**Simple Condition Coverage:** voglio testare le condizioni di vero-falso generale

→ con 2 test faccio 100%

**Immagine che contiene testo, numero, Carattere, parole crociate

Descrizione generata automaticamenteMultiple decision coverage**: voglio testare tutte le possibili combinazioni

* Considera indipendenti tutte le possibili decisioni
* Per singola condizione testo sia Vero che Falso

Mi servono 8 test per fare 100% 🡨

*Più efficace ma anche più costoso*

*Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente*

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

Descrizione generata automaticamentePATH coverage**: sequenza di nodi nel nostro grafo:

* Percorsi possibili
* Problema se grafo ciclico → troppi percorsi possibili
  + Immagine che contiene testo, schermata, diagramma

    Descrizione generata automaticamenteBisogna selezionare quelli critici

**Loop coverage**: in caso di loop, si possono testare:

* **Immagine che contiene testo, Carattere, bianco

  Descrizione generata automaticamenteBoundary** → 0 iterazioni (non è detto si possa fare, vedi caso media)
* **Interior**
  + 1 iterazione
  + > 1 iterazione

**→ 3 test case (non estra, entra e cicla una volta, entra e cicla più di una volta)**

**Unit testing:**

* Programmi java → Junit
* Node e typescript → JestImmagine che contiene testo, schermata, software, Software multimediale

  Descrizione generata automaticamente (se usiamo anche instanbul → tool grafico per rendere carino e chiaro)