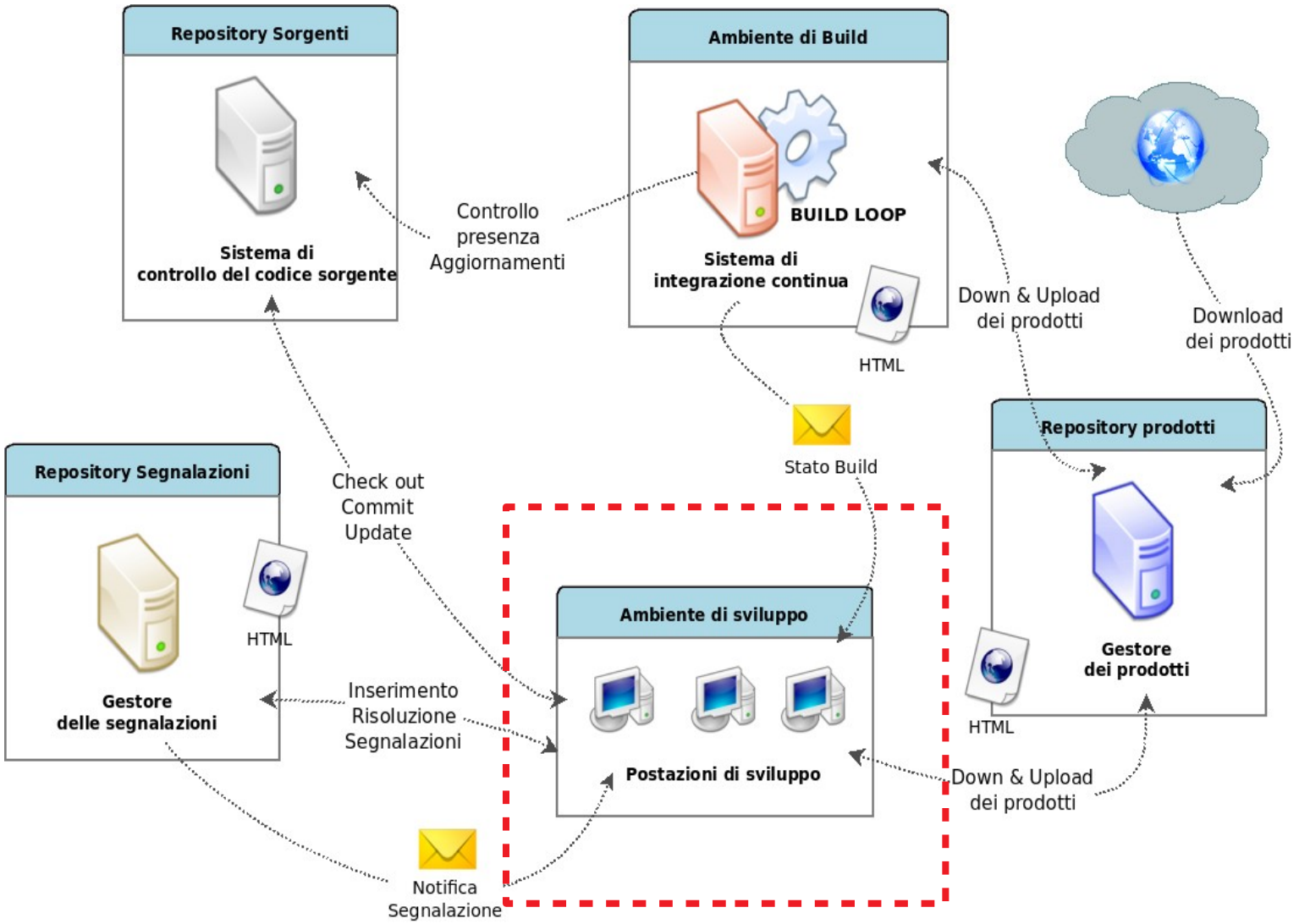
METODI E TECNOLOGIE PER LO SVILUPPO SOFTWARE

Nicola Bertazzo
nicola.bertazzo [at] unipd.it

Università degli Studi di Padova
Dipartimento di Matematica
Corso di Laurea in Informatica, A.A. 2021 – 2022

Visione Generale



Definizione

Software testing is an investigation conducted to provide stakeholders with information about the **quality** of the **software product** or **service under test**. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of **finding software bugs** (errors or other defects), and verifying that the software **product is fit for use**.

Definizione

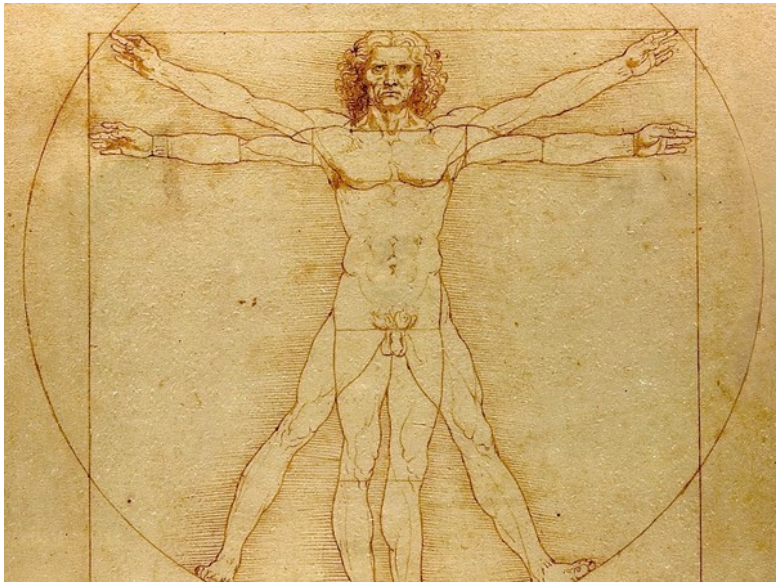
Testing:

The **process** consisting of all lifecycle activities, both **static** and **dynamic**, concerned with planning, preparation and evaluation of software products and related work products to **determine that they satisfy specified requirements**, to demonstrate that they are **fit for purpose** and to **detect defects**.

ISTQB®
International Software Testing Qualifications Board

Perché il Software Testing?

- L'uomo non è perfetto
- Il codice è scritto dagli uomini
- Il codice può contenere errori (bug)
- I bug possono avere delle conseguenze
- Per questo il software deve essere testato



Chi può inserire difetti nel software?

Il Programmatore (45%)

- Il programmatore fa un errore (**mistake**) durante la fase di sviluppo
  - Per esempio non viene considerata la possibilità di ricevere in input una stringa troppo lunga
- Il programmatore quindi inserisce un **difetto** (fault o bug) all'interno del programma
- Quando il programma viene eseguito, e la condizione non considerata dal programmatore si verifica, il difetto provocherà un comportamento inatteso del programma
  - L'utente inserisce nella form di input la stringa troppo lunga
- Il programma avrà quindi una **failure**

Chi può inserire difetti nel software?

L'analista (20% analisi requisiti, 25% progettazione)

- L'analista può introdurre un difetto, interpretando non correttamente un requisito
- Il difetto viene introdotto nella fase di analisi e progettazione
- La progettazione del programma e la codifica può essere influenzata dal difetto

Altre cause

- Condizioni ambientali impreviste (temperatura, umidità, inquinamento, magnetismo, radiazione) che alterano il funzionamento dell'hardware sottostante

Categorie di testing (funzionale)

Funzionale:

Testing conducted to evaluate the compliance of a component or system with **functional requirements.**

Non Funzionale:

Testing conducted to evaluate the compliance of a component or system with non-functional requirements (p.es. Performance, Sicurezza, Usabilità, Accessibilità)

Categorie di testing (Statico/dinamico)

Statico:

Testing a work product without code being executed. (p.es analisi statica del codice, analisi e revisione dei documenti, analisi e revisione dei requisiti)

Dinamico:

Testing that involves the execution of the software of a component or system.

Categorie di testing (Verifica/Validazione)

Cercare difetti per migliorare la qualità del software

Verifica:

Il prodotto è stato realizzato secondo le specifiche (tecniche) e funziona correttamente

Convalidare che il software sia conforme ai suoi obiettivi (fit for purpose)

Validazione:

Il prodotto è stato realizzato rispettando le specifiche dell'utente (requisiti)

Definizione di caso di test

IEEE Standard 610:

Test case: A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

ISTQB Glossary:

Test Case (2018): A set of preconditions, inputs, actions (where applicable), expected results and postconditions, developed based on test conditions.
Test Condition (2018): A testable aspect of a component or system identified as a basis for testing.

Test Condition (2011): An item or event of a component or system that could be verified by **one or more** test cases, e.g. a function, transaction, feature, quality attribute, or structural element.

Ricorda:

Non esiste test case, se non esiste il corrispondente requisito da validare

Definizione di caso di test

I casi di test aiutano anche a ridurre l'ambiguità nei requisiti (funzionali e non funzionali), e quindi ne migliorano la descrizione, e in ultima analisi la documentazione: si parla di **testable requirements**. Questo approccio permette inoltre di progettare test che possano essere eseguiti automaticamente in catene CI/CD.

Esempio 1:
- la funzione di inserimento di un ordine deve essere di facile utilizzo (requisito non chiaro)
- l'operatore deve poter inserire un ordine nel sistema in meno di 3 minuti, senza bisogno di consultare il manuale operatore (requisito espresso in maniera più chiara)

Esempio 2:
- il sito deve essere veloce
- il sito deve presentare le sue pagine agli utenti finali con un tempo medio non superiore a 500 ms

Per approfondire (Craig Larman):
- Specification by example: approccio collaborativo alla stesura di requisiti con esempi pratici
- Requirements and business rules as executable tests → pre-requisito per Test Automation

Processo di test

1) **Test planning** → The activity of establishing or updating a test plan
2) **Test control** → azioni correttive e di controllo se il piano non viene rispettato
3) **Test analysis** → Cosa testare
4) **Test design** → Come testare
5) **Test implementation** → attività propedeutica all'esecuzione (p.es definizione dei casi di test)
6) **Test execution** → eseguire il test
7) **Checking result** → verificare i risultati e i dati collezionati dalla test execution per capire se il test è stato superato/fallito
8) **Evalutating exit criteria** → verificare se sono stati raggiunti gli exit criteria definiti nel test plan
9) **Test results reporting** → Riportare il progresso rispetto agli exit criteria definiti nel test plan
10) **Test closure** → Chiusura del processo e definizione azioni di miglioramento

Seven testing principles

1- Testing show presence of defects

Testare un'applicazione può solo rivelare che **uno o più difetti esistono** nell'applicazione

il test non può dimostrare che l'applicazione sia priva di errori.

Pertanto, è importante progettare casi di test per trovare il maggior numero possibile di difetti.

Seven testing principles

2- Exhaustive testing is impossible

A meno che l'applicazione in prova abbia una struttura logica molto semplice e un input limitato, non è possibile testare tutte le possibili combinazioni di dati e scenari.

Per questo motivo, il rischio e le priorità vengono utilizzati per concentrarsi sugli aspetti più importanti da testare.

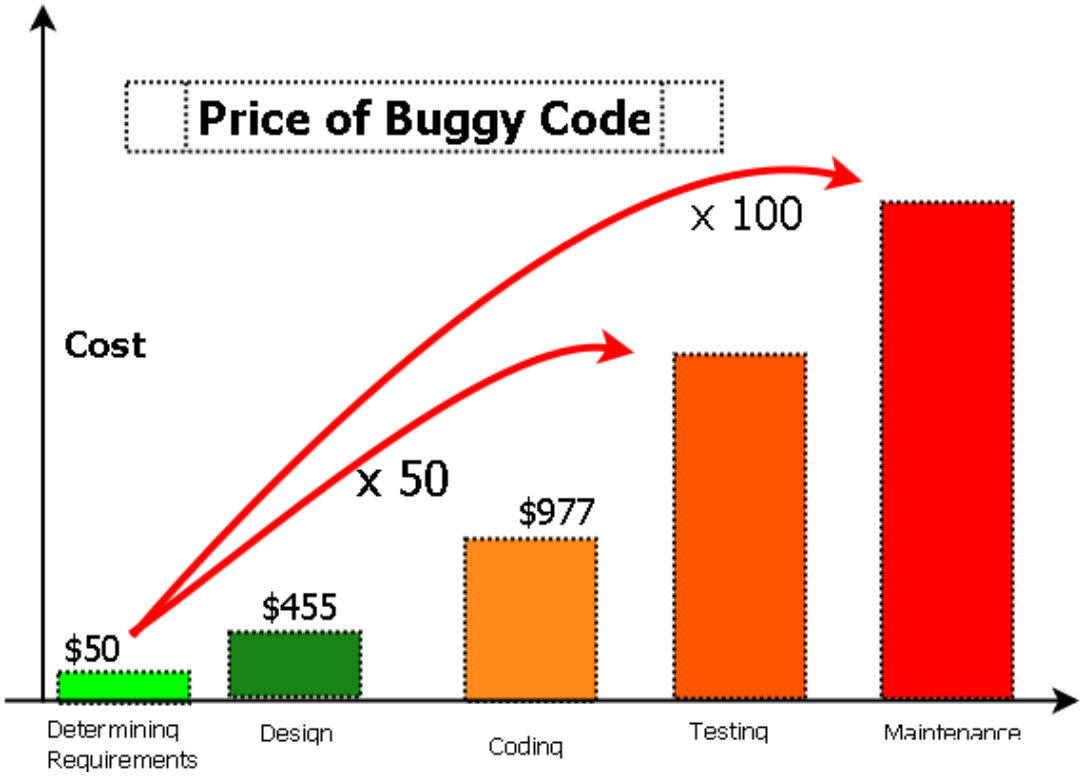
Strategie per selezionare i test case:

- sul rischio (**risk based testing**): funzionalità che hanno impatto sul business
- sui requisiti (**requirement based**)

Seven testing principles

3- Early testing

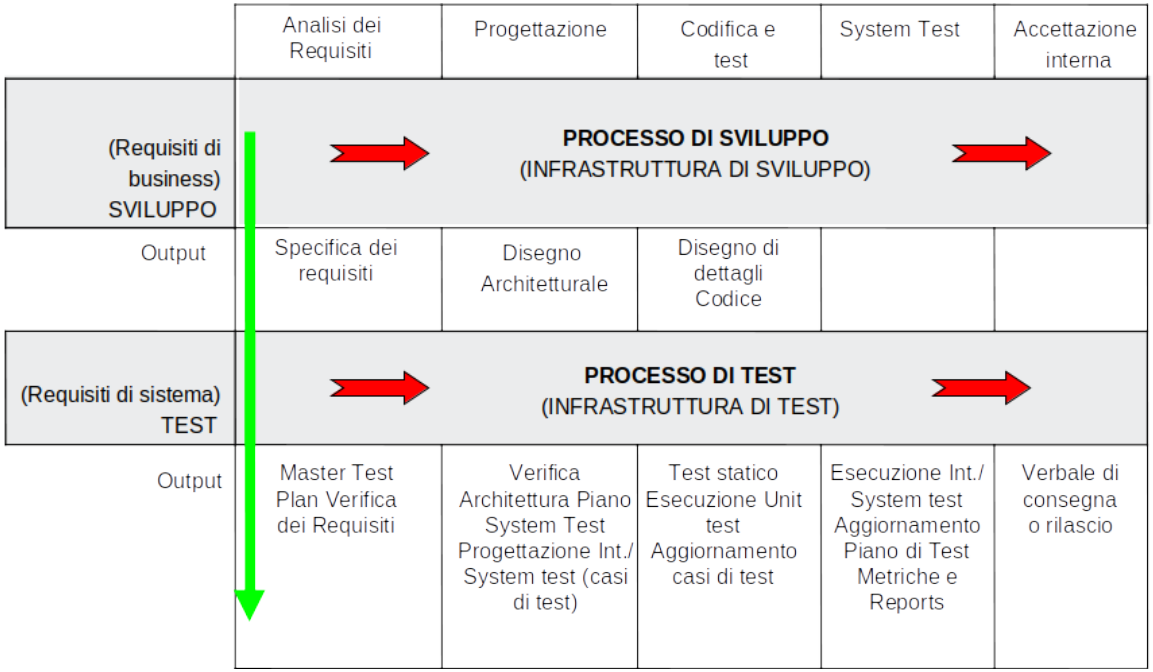
Avviare la fase di test il prima possibile permette di risparmiare sui costi del progetto



Seven testing principles

3- Early testing

Il processo di test non deve essere eseguito quando il progetto è al termine, ma deve andare in parallelo con il processo di sviluppo



Seven testing principles

4 - Defect clustering

Durante i test, si può osservare che la maggior parte dei difetti segnalati sono legati a un numero ridotto di moduli all'interno di un sistema.

Un piccolo numero di moduli contiene la maggior parte dei difetti nel sistema.

Questa è l'applicazione del principio di Pareto ai test del software: circa l'80% dei problemi si trova nel 20% dei moduli.

Seven testing principles

5 - The pesticide paradox

Se continui a eseguire lo stesso set di test più e più volte (ad ogni nuova versione), siamo sicuri che non ci saranno più gli stessi difetti scoperti da quei casi di test.

Poiché il sistema si evolve, molti dei difetti precedentemente segnalati vengono corretti. Ogni volta che viene risolto un errore o è stata aggiunta una nuova funzionalità, è necessario eseguire tutti i test (di non regressione) per assicurarsi che il nuovo software modificato non abbia interrotto vecchi errori.

Tuttavia, anche questi casi di test di non regressione devono essere modificati per riflettere le modifiche apportate nel software per essere applicabili.

Seven testing principles

6 - Testing is context dependent

Diverse metodologie, tecniche e tipi di test sono legati al tipo e alla natura dell'applicazione.
Ad esempio, un'applicazione software in un dispositivo medico richiede più test di un software di giochi.

Un **software per dispositivi medici** richiede **test basati sul rischio**, essere conformi con i regolatori dell'industria medica e possibilmente con specifiche tecniche di progettazione dei test.

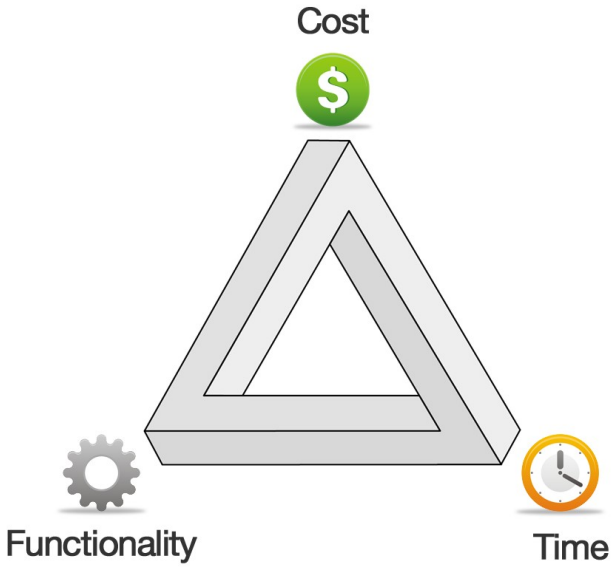
Un **sito web molto popolare** deve superare rigorosi **test delle prestazioni** e test di funzionalità per assicurarsi che le prestazioni non siano influenzate dal carico sui server.

Seven testing principles

7 - Absence of errors fallacy

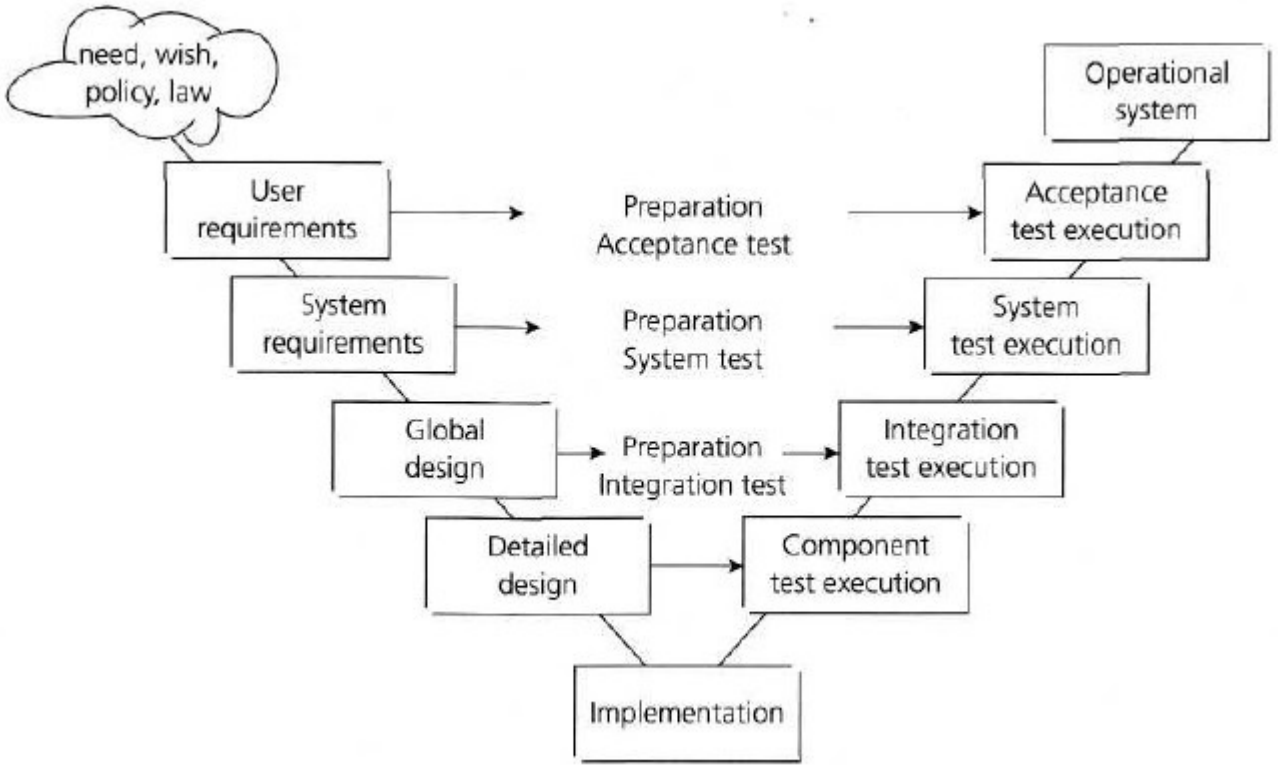
Solo perché il test non ha riscontrato alcun difetto nel software, non significa che il software sia perfetto e pronto per essere rilasciato.
I test eseguiti sono stati davvero progettati per catturare il maggior numero di difetti?

Il software corrispondeva ai requisiti dell'utente?



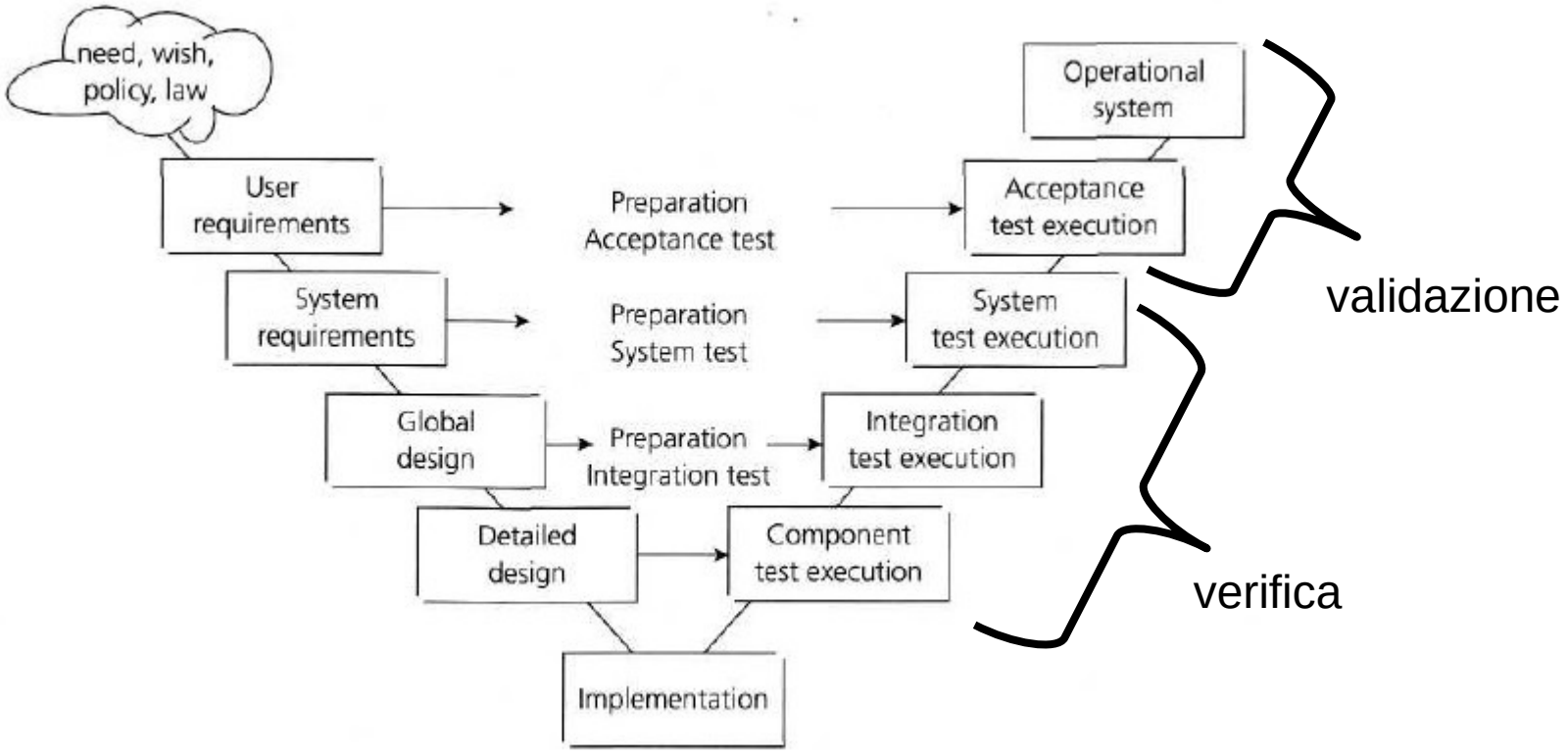
V-model

A sequential development lifecycle model describing a one-for-one relationship between major phases of software development from business requirements specification to delivery, and corresponding test levels from acceptance testing to component testing.



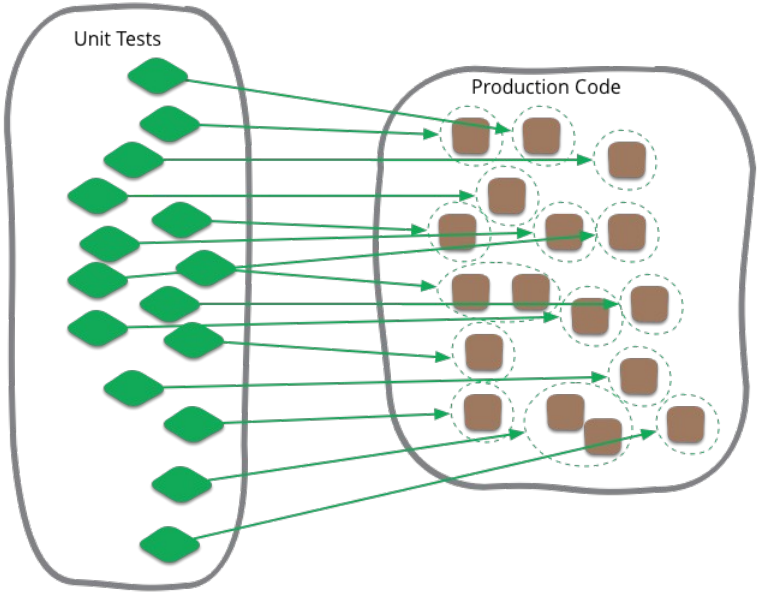
V-model

A sequential development lifecycle model describing a one-for-one relationship between major phases of software development from business requirements specification to delivery, and corresponding test levels from acceptance testing to component testing.



Component testing (Unit testing)

- Verificano l'unità: il più piccolo sottosistema possibile (in Java: classe o metodo) che può essere testato separatamente
- Veloci da eseguire
- Ogni modifica del codice sorgente dovrebbe scatenare l'esecuzione degli unit test
- Sono indipendenti tra di loro
- Non dipendono dall'ordine di esecuzione
- Il sistema sotto test (SUT – System Under Test) è considerato come white box



Integration testing

- Verificano se sono rispettati i contratti di interfaccia tra più moduli o sub-system
- Verificano l'integrazione tra più sub-system
- I Sub-System possono essere:
  - Sub-system **Interni**: Già verificati dagli unit testing
  - Sub-system **Esterni**: p.es database, file system…
- Gli "Integration testing" sono più lenti da configurare e da eseguire
- SUT è considerato come white box

System testing

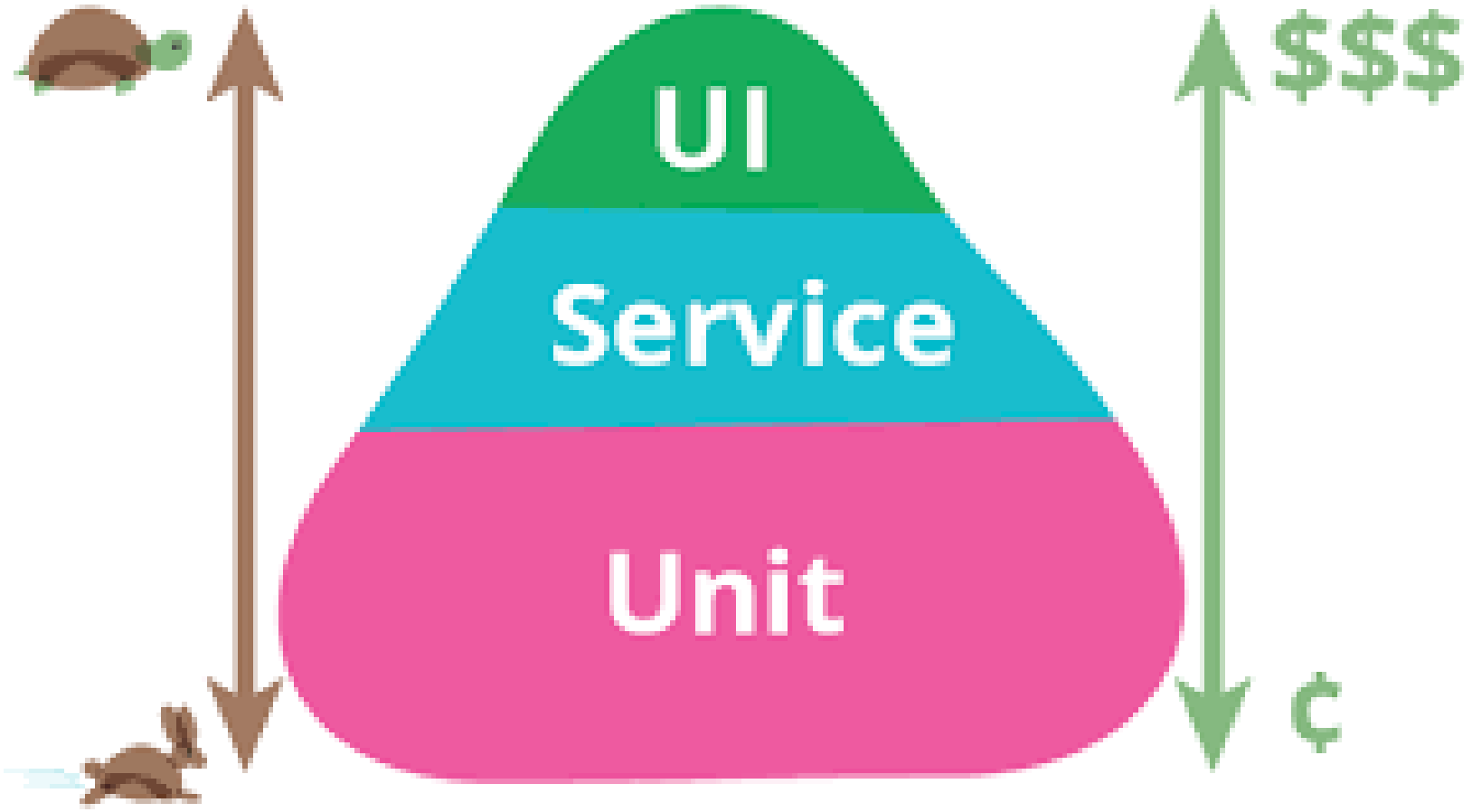
- Verificano il comportamento dell'intero sistema
- Lo scopo principale dei system test è la verifica rispetto alle specifiche tecniche
- Il sistema può essere considerato come white box o black box

Un tipo di system testing sono gli smoke testing:

- L'obiettivo degli smoke test è quello di trovare degli errori il **prima possibile** (prima della produzione) o eseguire altri test sul SUT.
- Verificano le funzionalità del SUT
- conosciuti anche come "Sanity Checking"

Acceptance testing

- Anche conosciuti come **"UAT": User Acceptance Testing** o "End User testing"
- Test suite su tutto il SUT, relativi agli use case e ai requisiti concordati con l'utente finale/cliente
- Svolti con l'utente finale/cliente
- SUT è considerato come black box


UI
Service
Unit
$$$
¢

Riferimenti

https://en.wikipedia.org/wiki/Software_testing
http://glossary.istqb.org/search/testing
http://glossary.istqb.org/search/functional%20testing
http://glossary.istqb.org/search/non-functional%20testing
http://glossary.istqb.org/search/verification
http://glossary.istqb.org/search/validation
https://www.testingexcellence.com/seven-principles-of-software-testing/
http://learnthebasicsofsoftwaretesting.blogspot.com/2013/04/v-model.html
https://martinfowler.com/bliki/TestPyramid.html
http://www.testingeducation.org/course_notes/bach_james/
cm_2002_rapidsoftwaretesting/
rapidsoftwaretesting_13_heuristic_risk_based_testing.pdf