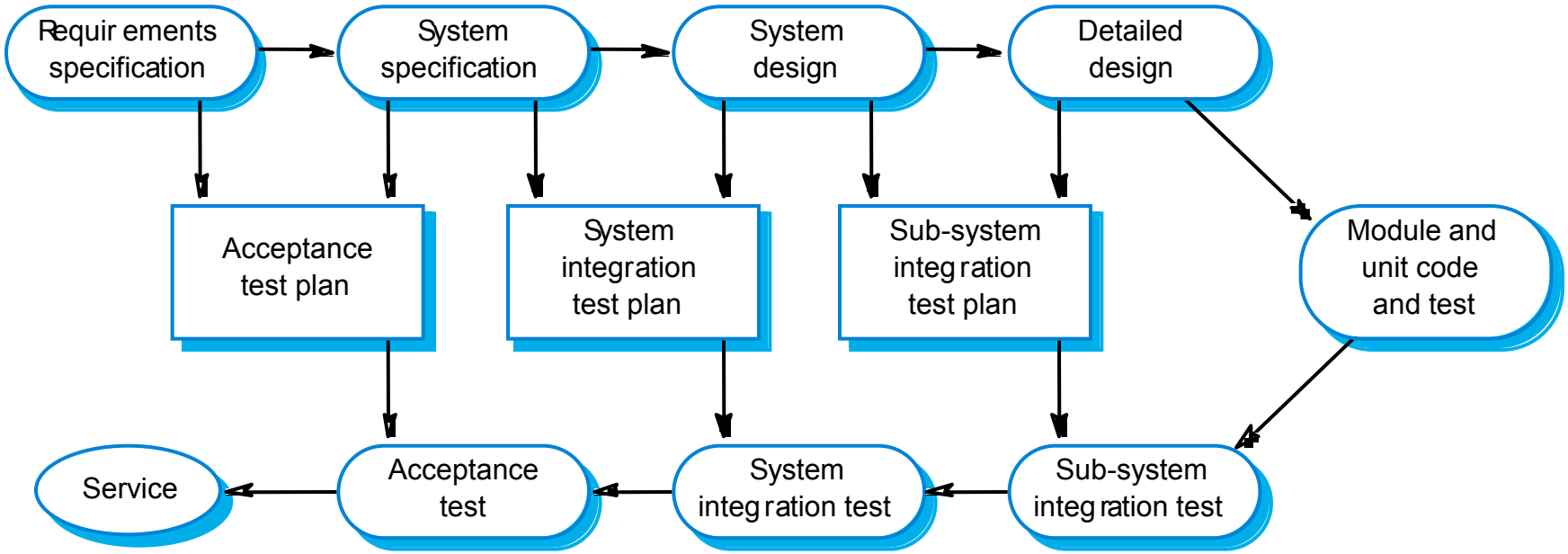
Verifica e Validazione del Software

Testing processes

Riferimenti

- Ian Sommerville, Ingegneria del Software, capitoli 22-23-24 (più dettagliato sui processi)
- Pressman, Principi di Ingegneria del Software, 5° edizione, Capitoli 15-16
- Ghezzi, Jazazeri, Mandrioli, Ingegneria del Software, 2° edizione, Capitolo 6 (più dettagliato sulle tecniche)

Modello V


Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Livelli di Testing

- System Testing in tre stage
  - *Simulation stage*
    - Il software viene testato in isolamento in un ambiente completamente simulato
  - Prototyping stage
    - L'ambiente reale inizia a rimpiazzare quello simulato, ma il software è eseguito sempre su di un emulatore
      - Un emulatore è un componente software che emula il comportamento di un componente reale da cui il software da testare dipende
  - Pre-production stage
    - Il software è eseguito su di un device reale in un ambiente reale

Livello produttore-utente privilegiato

- Alpha testing:
  - uso del sistema da parte di utenti reali ma nell'ambiente di produzione e prima della immissione sul mercato.
- Beta Testing:
  - installazione ed uso del sistema in ambiente reale prima della immissione sul mercato.
- ***tipicamente adottati dai produttori di packages per mercato di massa!***

Testing di Accettazione

- Testing effettuato sull'intero sistema sulla base di un piano e di procedure approvate dal cliente (o utente);
- l'obiettivo é quello di mettere il cliente, l'utente o altri a ciò preposti (collaudatori o enti ad hoc) in condizione di decidere se accettare il prodotto;
- Occorre provare che il software fornisce tutte le funzionalità, le prestazioni, l'affidabilità, etc. richieste, e che non fallisca.
- É in genere un testing black-box (a scatola nera):
  - Basato solo sulle specifiche del software;
  - I Tester non accedono al suo codice.
- è a carico del committente;
- .. è più 'una dimostrazione che un test'

- Il Testing dei Requisiti Non Funzionali

Performance testing

- Dopo che il sistema è stato completamente integrato, è possibile testarne le proprietà emergenti, come le prestazioni ed affidabilità.
- I test di prestazione in genere riguardano il carico e si pianificano test in cui il carico viene incrementato progressivamente finché le prestazioni diventano inaccettabili.
- Il carico deve essere progettato in modo da rispecchiare le normali condizioni di utilizzo.

Stress testing

- Nello stress testing si sollecita il sistema con un carico superiore a quello massimo previsto: in queste condizioni in genere i difetti vengono alla luce.
- Stressando il sistema si può testare il comportamento in caso di fallimento.
- L'eventuale fallimento dovrebbe essere 'leggero' e non produrre effetti catastrofici. Si deve controllare che non ci siano perdite inaccettabili di servizio e di dati.
- Lo Stress testing è particolarmente rilevante per sistemi distribuiti che possono mostrare severe degradazioni delle prestazioni quando la rete è sommersa di richieste.

Altri tipi di Testing (di requisiti Non-Funzionali)

- Testing di Compatibilità
  - Consiste nel provare la stessa applicazione al variare di diversi sistemi e piattaforme e al variare di diverse configurazioni
    - E' difficilissimo poter testare su tutte le piattaforme possibili
    - In ogni caso, è impossibile testare sulle piattaforme di *futura* generazione!
    - Ogni applicazione dovrebbe dichiarare l'insieme di piattaforme per le quali è stata testata la compatibilità

- Testing di Accessibilità
  - Consiste nel verificare che sia possibile l'accesso ai contenuti dell'applicazione in presenza di capacità hardware/software ridotte (sul lato client) e di disabilità dell'utente
    - Particolarmente importante per applicazioni Web, in particolare per quelle dirette ad un pubblico globale (es.: applicazioni istituzionali)
      - **Esistono standard nazionali (es.: Legge Stanca) e internazionali (es.: W3C) contenenti requisiti di accessibilità a regole da verificare**

Altri tipi di Testing

- Testing di Sicurezza
  - Consiste nel verificare l'efficacia del sistema nel difendersi e prevenire accessi da utenti non autorizzati, utilizzo improprio delle risorse, etc.
    - *Studiato a fondo nell'esame di Secutity and Dependability*
- Testing di Usabilità
  - consiste nel verificare e misurare l'usabilità dell'applicazione per gli utenti, valutando se soddisfa degli standard minimi
    - *La misura dell'usabilità non è completamente una misura oggettiva, per cui deve essere valutata con metodologie statistiche*

Test Driven Development (TDD)

Riferimenti

- S. Amber. Introduction to Test Driven Development (TDD). www.agiledata.org
- http://www.testdriven.com (online community/forum for TDD)
- D. Janzen, Test-Driven Development:Concepts, Taxonomy,and Future Direction, IEEE Computer-2005

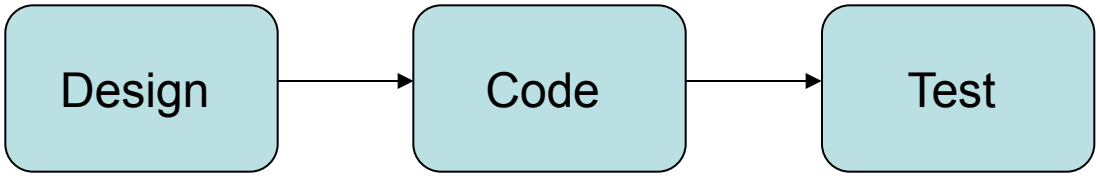
Cosa significa TDD?

- È' un approccio evolutivo allo sviluppo software in cui *si scrive prima un test che fallisce*, e poi si scrive il relativo codice.

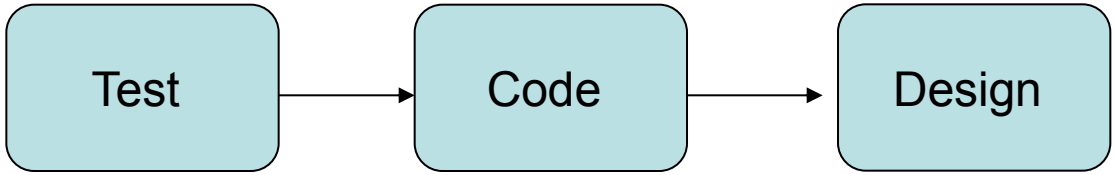
- Differisce dagli approcci tradizionali, in cui prima si scrive il codice e poi (forse) lo si testa.

La soluzione offerta dal TDD

- ***Ciclo di sviluppo Tradizionale***



- ***Sviluppo con TDD***



Cosa è il TDD

- TDD è uno stile di sviluppo/ progettazione, non è una tecnica di testing
- In TDD la programmazione e lo unit test non sono attività separate

- In TDD, si può scrivere nuovo codice solo se un test automatico è fallito
- TDD Mantra: "**Only ever write code to fix a failing test**"

Vantaggi del TDD

- Permette di scrivere software migliore e più rapidamente (costruendo software in piccoli incrementi- 1 test per volta)

- Evita la paura nello sviluppatore che :
  - induce a procedere per tentativi
  - fa comunicare meno
  - fa temere il feedback
  - rende *acidi*!

Vantaggi

- Produce *codice pulito e che funziona* (in modo opposto allo sviluppo architecture driven, in cui si fanno prima tutte le decisioni)

- Consente agli sviluppatori di produrre un insieme di *test di regressione automatizzabili*, man mano che sviluppano

Qualità interna col TDD

- La presenza di difetti nel codice può dipendere dalla mancanza di test, o dalla mancata esecuzione di test che abbiano evidenziato tali difetti.
- Invece:
  - Col TDD non ci sarà mai, praticamente, una parte di codice non testata
  - Scrivendo i test prima del codice, si è portati a cercare test che esercitano sia i casi normali, che i casi limite e di errore
  - Si riduce il tempo per la correzione dei difetti (si sa esattamente quali nuove linee possano aver fatto fallire il test: non serve neanche il debugger!)

Il Ciclo di Sviluppo TDD

RED **GREEN** **REFACTOR**

TEST-DRIVEN APPROACH

- Scrivi un test che non funzionerà (**red**)
- Fai in modo che il test funzioni, scrivendo solo il codice necessario a superarlo (**green**)
- Migliora (**ristruttura** – il codice ed i test)

- Esegui tutti i test per assicurarsi che non ci siano problemi
- Ripeti finchè non trovi altri test