Test Driven Development

- Definizione
- Passi di partenza
- Refactoring
- Terminologia TDD
- Benefici
- Junit
- Mocktio
- Integrazione continua

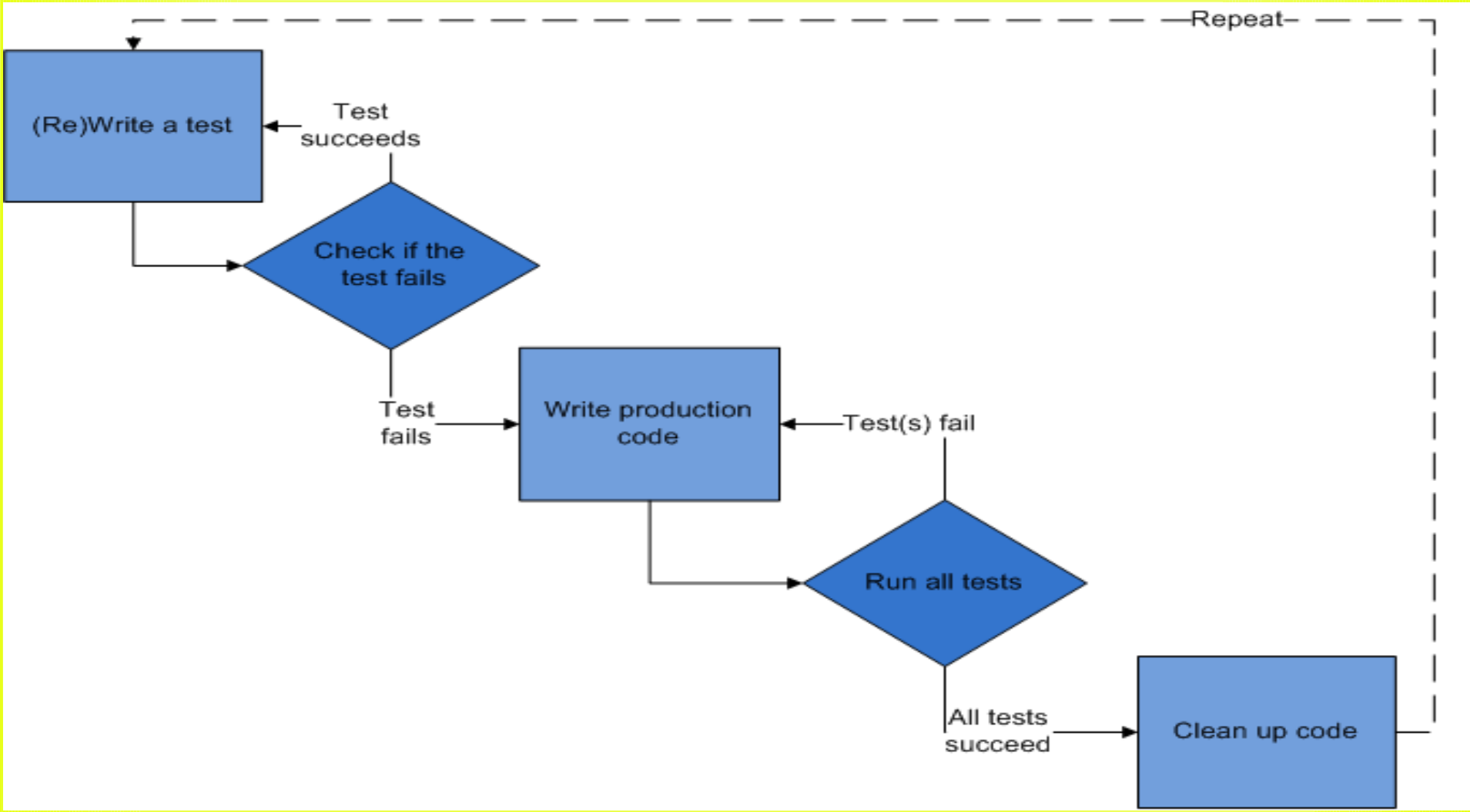
Test Driven Development

- Test driven development o test first development e' una tecnica di sviluppo ed uno dei fattori chiave della metodologia di Extreme Programming

Approcci

- Sviluppo ad hoc
- Scrivere prima pseudo-codice
- Model Driven Development
- Test Driven Development

TDD


Repeat
(Re)Write a test
Test succeeds
Check if the test fails
Test fails
Write production code
Test(s) fail
Run all tests
All tests succeed
Clean up code

TDD aiuta perche'

- Assicura che le funzioni create sono chiamabili e testabili
- Accorcia il feedback per il programmatore
- Fornisce una specifica dettagliata attraverso I test
- Dimostra che il vostro SW funziona
- Aiuta lo sviluppo evolutivo

TDD dice...

- Quando ho una nuova funzionalita' da realizzare, noi spesso pensiamo: come faccio ad implementarlo?
- TDD dice: non farlo!
- L'enfasi e' posta sull'uso anziche' la implementazione

Come partire

- Analizza I requisiti e scrivi una lista di funzionalita' o task
- Prendine una
- Inventa una lista di test relativi
- Prendi un test e scrivilo
- Scrivi solo quanto basta perche' si riesca a compilarlo
- Esegui il test e scrivi solo codice sufficiente perche' passi
- Rifattorizza il codice eliminando duplicati
- Torna al punto iniziale

Quando trovo un errore

- Create subito un test che lo riveli
- Modifica il codice cosicche' passi correttamente
- Fai girare tutti I test in modo che passino

Refactoring

- Refactoring e' una delle parti piu' importanti del TDD implicando che il codice deve essere continuamente raffinato man mano che si aggiungono codice e test

Terminologia TDD 1

- Testcase: metodo che testa una funzionalita'
- Testsuite: gruppo di Testcase raggruppati da una Fixture
- Fixture: Macro o asserzione che confronta risultato ottenuto col desiderato
- SetUp() metodo in JUnit che crea l'ambiente di esecuzione dei test

Terminologia TDD 2

- Refactoring: manipolazione del codice sorgente per migliorarlo senza alterarne le funzionalita'
- Unit: quello che testiamo, da un singolo metodo ad un insieme di classi
- Mock: oggetto che imita un oggetto del sistema esibendo comportamento deterministico
- Red e Green bar: dall'ambiente grafico di Junit, mostra il progresso e diventa rossa se un test fallisce

Benefici TDD 1

- Copertura dei test: Si e' quasi sicuri che tutte le linee di codice sono testate
- Ripetibilita' dei test: permette di tentare dei cambiamenti troppo rischiosi senza di essi
- Documentazione: I test descrivono la comprensione del comportamento del codice e l'uso delle sue API
- Progetto delle API: scrivendo prima i test, ci si mette nei panni del loro utilizzatore

Benefici TDD 2

- Organizzazione del sistema: un modulo testabile indipendentemente e' staccato dal resto del sistema
- Riduzione del debugging: muovendosi in piccoli passi come raccomandato da TDD, il debugger e' meno necessario
- Fiducia: se si osserva un team, tutti hanno il codice che funziona quasi in continuazione

problemi TDD

- Difficile uso se test completo richiesto (es UI)
- Chi scrive test e poi sviluppa, puo' sbagliare due volte in modo identico
- Manutenzione test piu' complessa perche' molto integrati nel codice
- necessario deciso supporto del management

Librerie

- Java – Junit
- Ruby – Test::Unit
- Javascript – Test.More o JSUnit
- C++ - CPPUnit
- PHP – PHPUnit
- Python - PyUnit

Frameworks

- Java - Mockito crea degli oggetti mock, permettendo di specificare comportamento e valori di ritorno per classi e metodi e verificare le interazioni
- Ruby- ??

Integrazione continua

•Continuous Integration (CI) e' una pratica di ingegneria SW in cui qualunque cambiamento locale isolato e' testato imediatamente e propagato quando viene aggiunto ad un grosso sistema. Obiettivo e' il rapido feedback, per scoprire il prima possibile se esso introduce dei difetti

CI Best Practices

- Automatizzare la costruzione (build)
- Creare Test cosicche' la build si testi automaticamente
- Tutti fanno commit alla baseline ogni giorno
- Per ogni commit una build
- Tutti vedono I risultati dell'ultima build
- Installazione automatica
- Test automatici sono assolutamente essenziali

Tests

- Indispensabile un tester nel team dal giorno 1
- Quando lo sviluppatore ha una feature il tester e' pronto col test per trovare un errore
- I tester devono valutare l requisiti e collaborare all'analisi

Strumenti SW

- Hudson -Java SW per automatizzare la build, supporta CVS SVN Git ant maven
- Bamboo
- Buildmaster by Inedo (link)