Agile Testing

Corso di Ingegneria del Software

Anno Accademico 2012/13

Introduzione

- **Le pratiche che descriveremo in questa sezione sono tipiche dell'eXtreme Programming (XP)**
- **L'XP si basa su alcune pratiche fondamentali**
  - **Automatizzare tutti i test**
  - **Test Driven Development (TDD)**
  - **Acceptance Test associati ai requisiti (espressi come User Stories)**

Test Automatici

- **Nell'XP c'è una forte enfasi, che si trova sia nel TDD che nei test di accettazione, sul fatto che i test devono essere automatici**
  - **I test stessi sono codice**
- **I test devono poter essere fatti girare in qualunque momento, da chiunque con uno sforzo minimo**
  - **Aumenta la confidenza che il nuovo codice aggiunto non destabilizzi quello che c'era prima**

Test Automatici

- **I test automatici possono formare una suite di regression test**
  - **Il regression test è la ripetizione dell'esecuzione dei test dopo che sono state cambiate parti di codice**
  - **L'obiettivo dei regression test è mostrare che il comportamento del sistema non è cambiato**
- **I test automatici fanno risparmiare tempo e denaro, perché il *debugging* alla cieca può essere molto lungo e costoso**

Benefici dei Test Automatici

- **Sviluppo di sistemi più affidabili**
- **Riduzione dello sforzo per gestire e far girare i test**
- **Possibilità di effettuare refactoring e di modificare i requisiti anche in corso d'opera**
- **Però:**
  - **I test non sono gratuiti**
  - **E' necessario trovare un compromesso tra i costi per automatizzare i test e i loro benefici**

Test Automation Manifesto (Meszaros et al.)

- **Il manifesto contiene le linee guide per creare i test automatici**
- **I test automatici dovrebbero essere:**
  - **Concisi – Il più semplici possibile, ma non banali**
  - **Self Checking – Devono riportare i risultati senza necessità di un'interpretazione umana**
  - **Ripetibili – Devono poter essere eseguiti ripetutamente senza intervento umano**
  - **Robusti – I test producono sempre gli stessi risultati. Non sono influenzati da cambiamenti dell'ambiente esterno, dalla sequenza in cui sono eseguiti, o dal fatto che un test precedente è fallito**

Test Automation Manifesto (Meszaros et al.)

- **Sufficienti – Devono verificare tutti i requisiti del software sotto test**
- **Necessari – In ogni test, tutto contribuisce alla specifica del comportamento desiderato**
- **Chiari – Ogni istruzione è facile da capire**
- **Efficienti – I test sono eseguiti in un tempo ragionevole**
- **Specifici – Ogni punto in cui il test può fallire corrisponde ad una funzionalità specifica**

Stakeholders

- **I tre Stakeholders coinvolti nell'attività di testing automatico sono:**
  - **Lo sviluppatore che deve tener conto del tempo speso per automatizzare i test sia a livello di unit test che di test di accettazione**
  - **Il manager deve concedere agli sviluppatori il tempo per implementare i test. Questo tempo diventa un investimento a lungo termine perché i test diventano uno strumento fondamentale per i test di regressione**
  - **Il cliente deve partecipare alla creazione dei test di accettazione**

Test-Driven Development

- **Test First Programming**
- **Prima di modificare o aggiungere del codice, scrivere i test per verificare il codice stesso**
- **La programmazione guidata dai test è una tecnica che riguarda non solo i test, ma è una pratica di progetto**
  - **Gli sviluppatori iniziano a scrivere i test per una user story e poi scrivono il codice che la implementa**
- **Se è difficile scrivere un test, è un problema di progetto e non di codifica**

Test Driven Development

- **Scrivere i test prima del codice**
- **Tutto il codice deve avere Test Unitari**
- **Tutto il codice deve superare i test unitari al 100%**
- **Quando si trova un bug, si scrivono test**
- **I Test di Accettazione sono eseguiti spesso e se ne pubblicano i risultati**
- **Tutti i test devono essere automatici**

TDD in 6 passi

- **E' una tecnica utilizzata dagli sviluppatori che eseguono rapide iterazioni dei seguenti passi:**
  1. **Si esegue una breve sessione di progetto, identificando gli oggetti e le loro proprietà rilevanti per sviluppare la funzionalità in esame**
  2. **Si scrivono alcuni test unitari automatici**
  3. **Si eseguono tali test, verificando che falliscano (non c'è ancora il codice!)**
  4. **Si implementa giusto il codice che permette di superare i test**

TDD in 6 passi

5. **Si rieseguono tali test, verificando che ora siano superati**
6. **Si esegue un refactoring sul codice e sui test, per migliorarne il funzionamento e l'architettura**

TDD: risolve quattro problemi

- **cowboy coding: imporsi di scrivere test, e di farli superare dal codice scritto, evita il c.c.**
- **accoppiamento e coesione: il codice a basso accoppiamento e alta coesione è facile da testare**
- **fiducia: chi scrive codice e supera i test otterrà la fiducia dei propri compagni**
- **ritmo: è facile divagare per ore quando si codifica; se ci si abitua al ritmo: scrivi test, scrivi codice, esegui refactoring, scrivi test... ciò non avviene**

Test Unitari

- **Sono test scritti per ogni modulo di codice**
- **Ogni classe del sistema sviluppato avrà una corrispondente classe di test**
- **Sono scritti dagli stessi programmatori del codice**
- **L'insieme dei test unitari è raggruppato in una suite ed i test sono eseguiti in modo automatico**
- **Devono sempre essere superati al 100%**

Il framework XUnit

- **Per molti linguaggi esiste un framework di test gratuito chiamato Xunit**
- **Sviluppato originariamente da Kent Beck per il linguaggio Smalltalk (SUnit)**
- **Facilita la scrittura e l'esecuzione dei test unitari, permette la ripetibilità e l'automazione dei test**
- **Esistono versioni per Java (JUnit), C++, VB**
- **Ada, C, Delphi, .Net, Eiffel, Forte, Objective-C, Oracle, Perl, Power Builder, Python, Ruby, VOC**

X-Unit

- **X-Unit si basa su quattro pattern:**
  - **Fixture: set di variabili o oggetti, inizializzati a valori opportuni, usati ripetutamente come dati di input per i test. Ogni volta che un test è "lanciato", la sua Fixture è rinizializzata.**
  - **Test-Case: è un singolo test, o una serie di test atomici, definito su una Fixture e implementata in codice, in modo tale da essere ripetibile e trasmissibile ad altri sviluppatori.**
    - ***setUp()*** **crea o rinizializza la Fixture**
    - ***tearDown()*** **rilascia ogni risorsa usata per il test**

X-Unit

- **Check: un test case stimola una Fixture e controlla i risultati attesi. Si usano funzioni di controllo (Check) capaci di testare condizioni Booleane e riportare i risultati e lo stato del sistema in caso di fallimenti.**
  - **assert(*boolean expression*)**
  - **equals(*expression1, expression2*)**
  - **equalsDouble(*num expression1, num expression2*)**

X-Unit

- **Test Suite: i test case sono raggruppati e organizzati in Test Suites, sets di test cases che possono essere lanciati automaticamente e ripetutamente. Quando una Test Suite è "lanciata" tutti i Test Case sono eseguiti e tutti gli error e i failure sono mostrati.**
  - **Error: errore di esecuzione che porta a un'eccezione o a un crash di sistema**
  - **Failure: errore dei dati: un Check non è superato**

JUnit

- **E' un framework di testing open source per Java scritto da Eric Gamma e Kent Beck**
- **Si usa per white box testing**
- **Disponibile su www.junit.org**
- **Permette di scrivere codice più velocemente, aumentando la qualità**
  - **Diminuisce i tempi di debug**
  - **Aumenta la confidenza tra gli sviluppatori**
  - **Permette il refactoring**

Test di Accettazione

- **Sono test di tipo black box, sono scritti dal cliente in collaborazione con gli sviluppatori**
- **Sono creati a partire dalla storie (User Stories)**
- **Sono di proprietà del cliente**
- **Sono un potente strumento di documentazione del sistema**
- **Quando un test di accettazione passa, il cliente considera la storia implementata**
- **La percentuale di TA superati denota lo stato di sviluppo del sistema**
- **Dovrebbero essere automatici**

Test di Accettazionec

- **Hanno lo stesso template dei test case di tipo black box**
  - **Test ID**
  - **Description, che descrive le precondizioni del test e i suoi passi**
  - **Expected Results**
  - **Actual Results**

Framework for Integrated Test (FIT)

- **Ideato e scritto da Ward Cunningham**
- **E' un framework per scrivere test di accettazione automatici**
- **Open Source, basato su Java**
- **Utilizza HTML ed un semplice linguaggio:**
  - **per scrivere i test**
  - **per specificare i risultati attesi**
  - **per verificare i risultati**

Riassumendo... test automatici

- **Test unitari**
  - **I programmatori comunicano tramite il codice e i test unitari**
  - **Abilitano la proprietà collettiva del codice**
  - **Abilitano il refactoring e le modifiche**
- **Test di accettazione**
  - **I clienti comunicano tramite le storie e i test di accettazione**
  - **Mostrano le aspettative del cliente (semplicità)**
  - **Mostrano i progressi effettuati (fiducia)**