Sviluppo software guidato dal testing

metodologie e strumenti

Sommario

- Testing, software a oggetti
- Metodologie di sviluppo
  - Test-Driven Development
  - Customer Test-Driven Development
- Strumenti Open-Source: Java e .NET
  - xUnit
  - Mock Objects
  - FIT

- Demo!

Testing

- In piccolo
  - Testing di unità: classe e singoli metodi
- In grande
  - Testing di integrazione (insieme di classi)
  - Testing di accettazione (intero sistema o sottosistema)

... nuovo significato

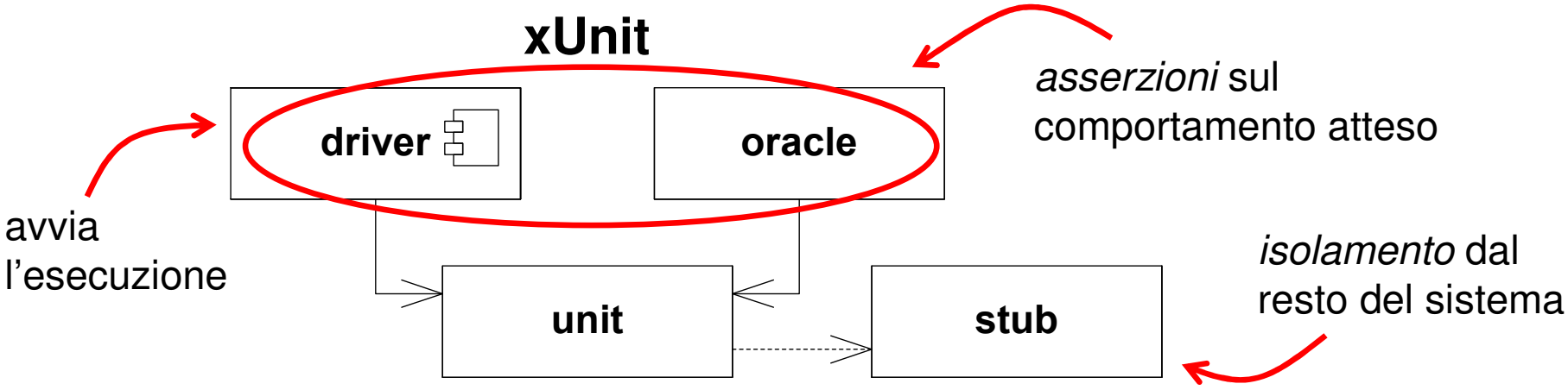
- Guida alla progettazione e specifica requisiti
- Attività svolta presto e spesso
- Coinvolti gli sviluppatori

Test-Driven Development

[K.Beck, 1999 – K.Beck, 2003]

- Attività di progettazione:

  scrivere test di unità prima del codice
  - red: definire comportamento atteso, con *asserzioni*
  - green: rendere eseguibili con successo i test, le asserzioni sono "*verificate*"
  - refactor: effettuare *refactoring* mantenendo verificate le asserzioni, introduzione *design pattern*



xUnit: esempi

jUnit 3.x

```
public class TestMyMath extends TestCase {
  private MyMath fixture;

  protected void setUp() {
    fixture = new MyMath();
  }

  public void testSum() {
    int result = fixture.sum(1, 2);
    assertEquals(3, result);
  }
}
```

- driver: estendere TestCase
- test: nomi metodi iniziano con *test*
- oracolo: metodi *assertXX*

NUnit 2.2.x

```
[TestFixture]
public class TestMyMath {
  private MyMath fixture;

  [SetUp]
  public void setUp() {
    fixture = new MyMath();
  }

  [Test]
  public void Sum() {
    int result = fixture.sum(1, 2);
    Assert.Equals(3, result);
  }
}
```

- driver: attributo TestFixture
- test: attributo Test
- oracolo: classe Assert

Struttura dei test: *build*, *operate*, *check*

(elenco strumenti su http://www.xprogramming.com/software.htm)

Isolamento

- Sostituire dipendenze con istanze “fittizie”
  - accesso a DBMS
  - risorse file-system (locale, di rete, ecc...)
  - ... collaborazioni con altri oggetti
- Tipi di classi “fittizie”
  - classi fake
  - classi stub

  } Testing basato sullo stato
  - valore degli attributi
  - valore di ritorno dei metodi
  - classi mock

  } Testing basato sulle interazioni
  - invocazione metodi di altri oggetti (collaboratori)
  - ordine, parametri, quante volte, ecc...

Mock Objects (dinamici): es.

EasyMock 2.x (con jUnit)

```
import static org.easymock.EasyMock.*; ...

public void testAlchoolProhibitedUnder18() {
  Seller fixture = new Seller();
  Person teen = createMock(Person.class);

  expect(teen.getAge()).andReturn(14);
  replay(teen);

  boolean canDrink = fixture.giveAlchoolTo(teen);

  assertFalse(canDrink);
  verify(teen);
}
```

- gestione mock e expectations: import *statico* classe EasyMock (Java 5)

Rhino.Mocks 2.x (con NUnit)

```
[Test]
public void AlchoolProhibitedUnder18() {
  MockRepository mocks = new MockRepository();

  Seller fixture = new Seller();
  Person teen = mocks.CreateMock<Person>();

  Expect.Call(teen.Age).Return(14);
  mocks.ReplayAll();

  bool canDrink = fixture.giveAlchoolTo(teen);

  Assert.IsFalse(canDrink);
  mocks.VerifyAll();
}
```

- gestione mock: oggetto MockRepository
- expectations: classe Expect

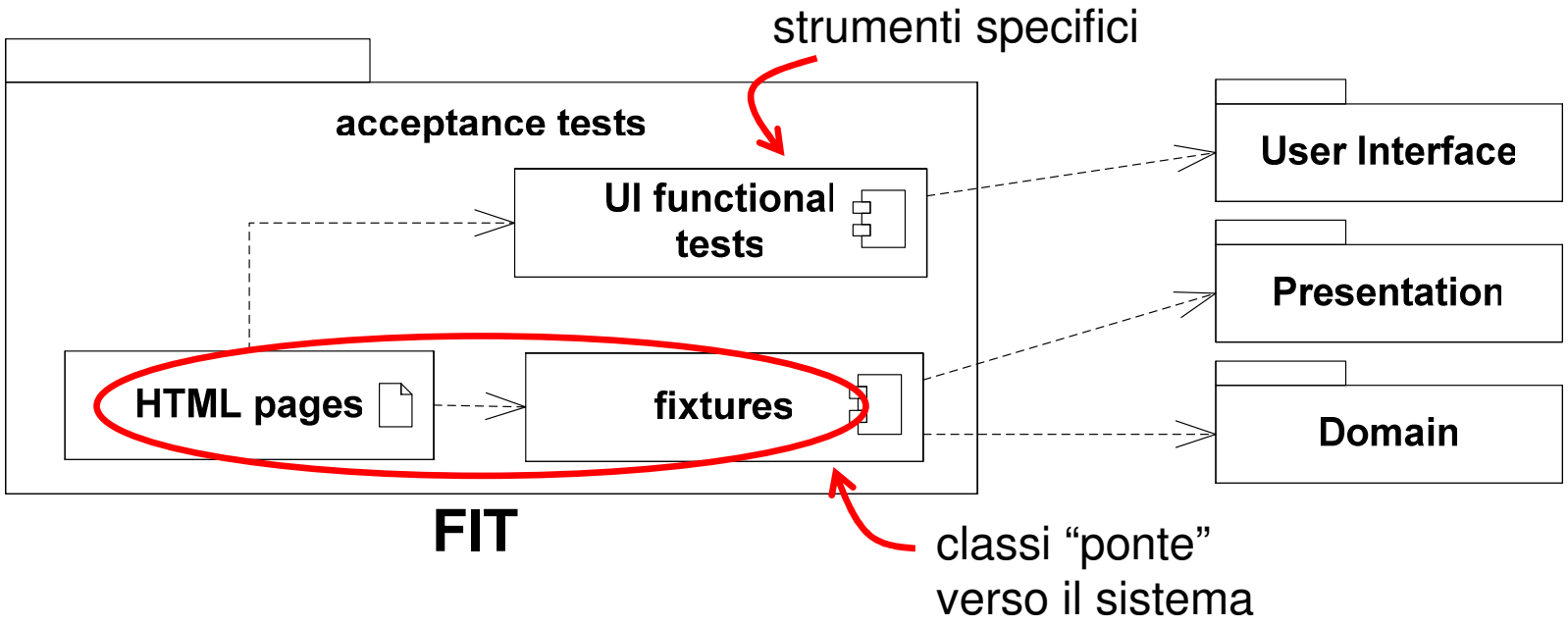
Struttura dei test: *build*, *expect*, *operate*, *check*, *verify*

(elenco strumenti e articoli su http://www.mockobjects.com)

Customer TDD

[W.Cunningham, 1999 - L.Crispin, 2001 - J.Kerievsky, 2004]

- Attività di specifica dei requisiti
  - Esempi di utilizzo del sistema (*Specification by Examples*)
  - Regole del dominio (Domain)
  - Sistema di interazione (UI e Presentation)
- Svolta con il cliente
  - Serve un linguaggio semplice (non *di programmazione*)

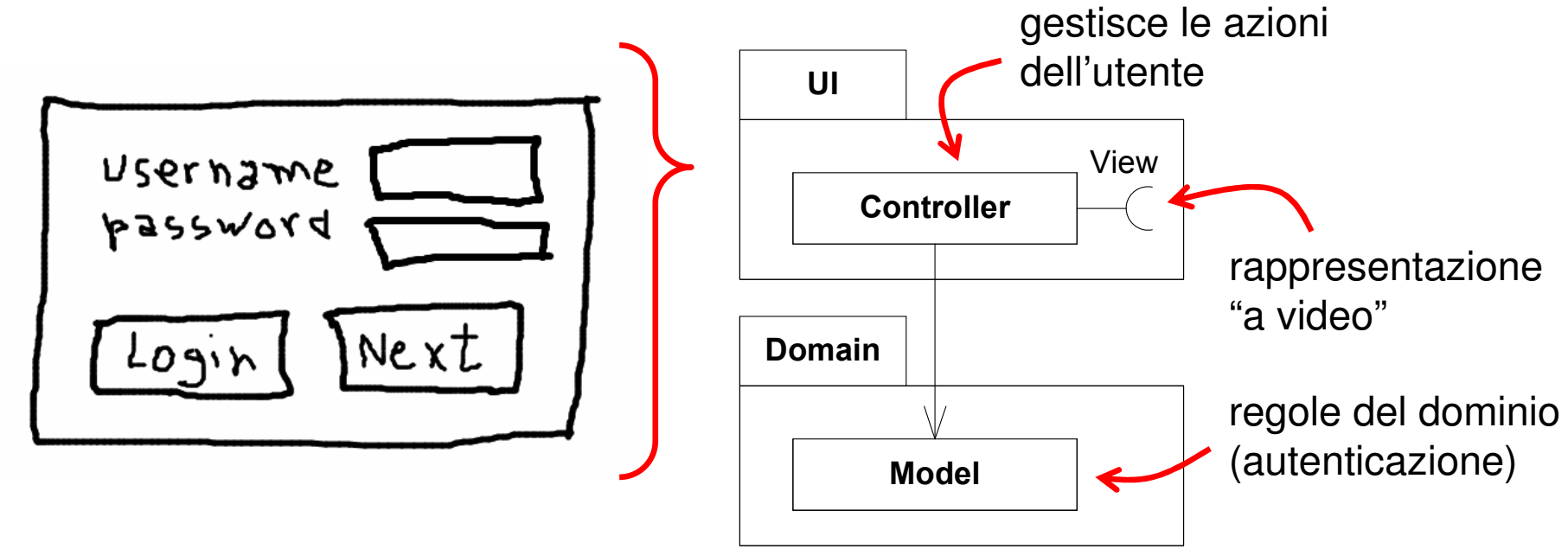


FIT+FitLibrary: principali fixture

- ColumnFixture, “a colonne”
  - Ogni riga calcola valori su dati forniti
  - Es: *username*, *password*, *message*?
- RowFixture, “a righe”
  - Elenca insieme di dati
  - Es: elenco utenti, *username*
- ActionFixture, “con azioni”
  - Simula interazione con l’utente
  - Es: inserisci *username* e *password*, clicca “*login*”
- DoFixture, la più generale
  - Sintassi flessibile, vicina al linguaggio naturale
  - Es: crea un nuovo utente, con *username* e *password*

DEMO: SimpleLogin

- Sistema di autenticazione utenti registrati ad un servizio (es: web, intranet, ecc...)
- Fornisce una pagina di “Login”
- Credenziali: username e password (!!)



SimpleLogin: sviluppo

- CTDD: test di accettazione
  - presentazione (“controller” o “presenter”)
  - regole di dominio (servizio autenticazione, tipi di errori)
- Fixture di FIT (per eseguire i test)
- TDD: test di unità
  - di ogni classe
  - stub e mock per le dipendenze

- Strumenti, Open-Source in Java
  - FIT, FitLibrary, FitNesse
  - jUnit e EasyMock
  - Eclipse

DEMO !!

Cosa manca?

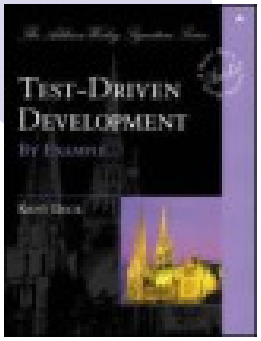
- TDD delle View UI
  - tool specifici (AWT/Swing, .NET, web)
- TDD del DAL
  - persistenza (DMBS/SQL, XML/XPath/XQuery)

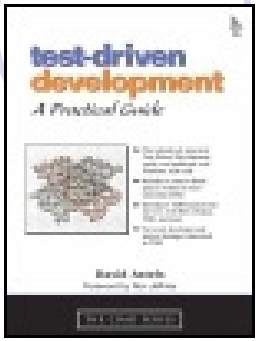
... testing di integrazione

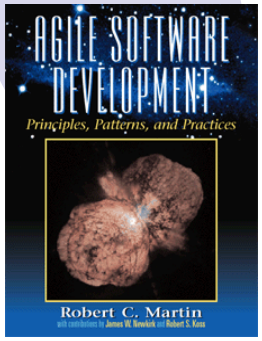
- Efficacia dei test
  - misura copertura istruzioni (EMMA, NCover)
  - analisi mutazionale (Jester, Nester)
- Processo di sviluppo
  - eXtreme Programming (XP), Scrum, Agile UP

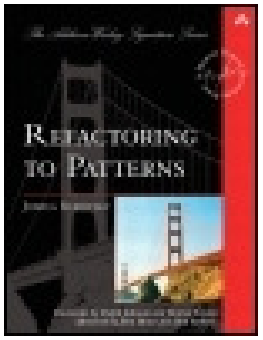
Riferimenti

- TDD by example
- TDD a pratical guide
- Agile software development
- Refactoring to patterns
- FIT for developing software
- XP explained, 2ed.









http://www.testdriven.com (notizie, articoli e recensioni strumenti)
http://www.objectmentor.com/resources/ (articoli, esempi e capitoli di ASD)
... e quelli citati per xUnit e Mock Objects

alcuni blog:

http://codebetter.com/blogs/jeremy.miller/ (US)
http://www.ayende.com/blog/ (Israele)
http://www.jpboodhoo.com/blog/ (UK-Canada)

Chiarimenti? Mi trovate qui:
jacopo.franzoi@gmail.com
http://blogs.ugidotnet.org/papo