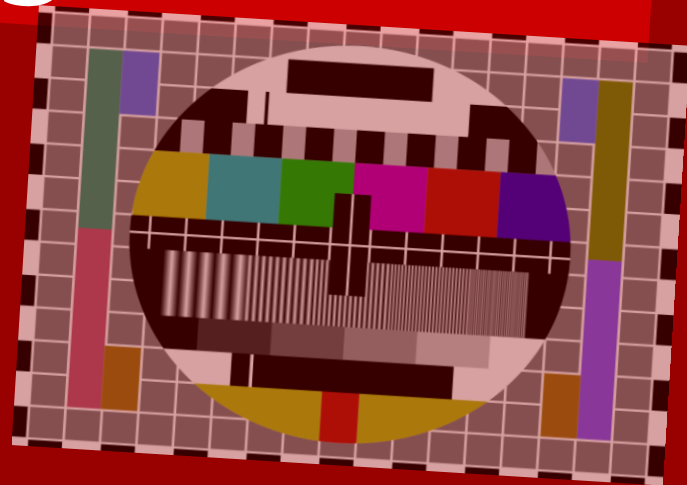


Unit Testing alla portata di tutti con il linguaggio C#

Webinar



Marco Breveglieri

*Sviluppatore software
consulente e trainer*

👉 <https://www.breveglieri.it>

👉 <https://www.compilaquindiva.com>



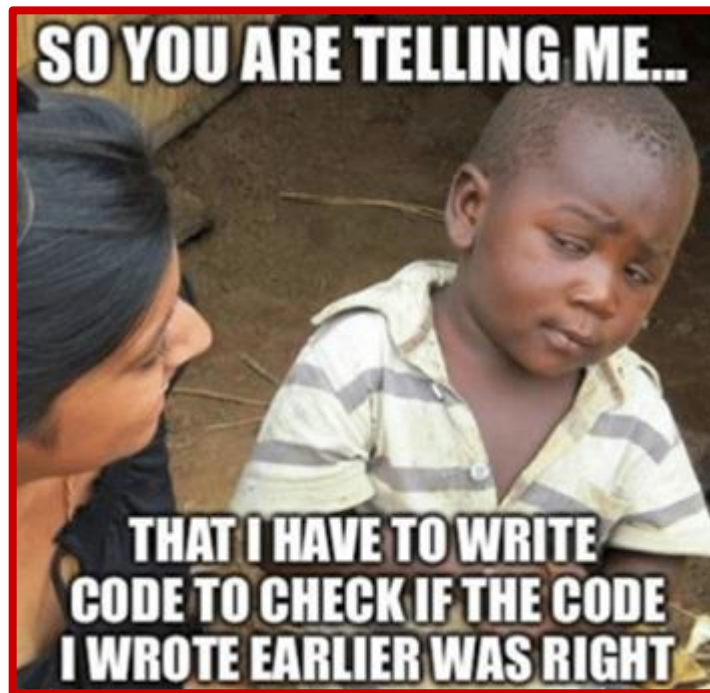
Testing... perché?

I benefici del testing

- Consente l'identificazione e risoluzione (preventiva) dei bug
- Agevola il mantenimento del codice cosiddetto "legacy"
- Incrementa la fiducia dello sviluppatore nella qualità del proprio codice (e in quello di altri)
- Forza il rispetto dei principi SOLID (e altri pattern)
- Permette il controllo dei difetti di regressione nel software
- Aiuta a censire e tracciare i problemi riscontrati dai clienti (ed evitare che si ripetano)
- E' un trampolino di lancio nell'uso di tecnologie e metodologie avanzate di sviluppo (es. TDD)
- Soddisfa i requisiti per l'implementazione della Continuous Integration/Delivery (CI/CD)

Scettici?

- Milioni di righe di codice vengono testate in modo "automatizzato"
- Numerose software house adottano la tecnica TDD (Test Driven Development)
- Non è (o non dovrebbe essere) un argomento nuovo, anzi...



Va fatto bene!

- Esistono direttive da rispettare per scrivere test "a regola d'arte"
- Implementare i test in modo errato può portare più danni che benefici
- Scrivere i test deve essere "divertente" (nonostante le scadenze incombenti)
- Possono richiedere più tempo per lo sviluppo, ma è possibile recuperarlo
- Il codice deve essere testabile: usate i principi SOLID! (*)

(*) ne abbiamo parlato pochi giorni fa in un [altro webinar](#): non perdetelo!



Obiettivo?

Diventare un
"mago" del
testing! ✨ 🧙



L'ABC...



Unit Test: una definizione

*A **Unit Test** is a piece of code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward.*

If the assumptions turn out to be wrong, the unit test has failed.

A «unit» is a method or function.

Unit Test: la struttura

La somma delle azioni che hanno luogo invocando un metodo pubblico del sistema sotto test e producono un risultato finale che può essere osservato.

Il risultato finale può essere

- *un valore restituito dal metodo invocato se è una funzione*
- *un cambiamento verificabile nello stato dell'oggetto sottoposto a test*
- *la chiamata di un metodo su una dipendenza dell'oggetto testato*



Caratteristiche di un buon Unit Test

- Automatizzato
- Ripetibile
- Consistente
- Facile
- Accessibile
- Riutilizzabile
- Veloce
- Onnipotente (*)

(*) rispetto al sistema sottoposto a test

Integration Test

Integration Testing is testing a unit of work with one or more of its real dependencies such as time, network, database, or code you cannot control such as threads and random number generators.



Unit vs Integration Tests

Unit Test

- Dipendenze isolate e simulate
- Nessun setup richiesto
- Nessun cleanup necessario
- Devono essere tanti
- Devono essere veloci
- Vengono eseguiti spesso

Integration Test

- Dipendenze concrete (tutte o alcune)
- Possibile inizializzazione necessaria
- Possibile cleanup richiesto al termine
- Sono meno rispetto agli Unit Test
- Possono essere più lenti
- Si possono lanciare più raramente

Qualche esempio?



Qualche esempio?



Qualche esempio?



Qualche esempio?



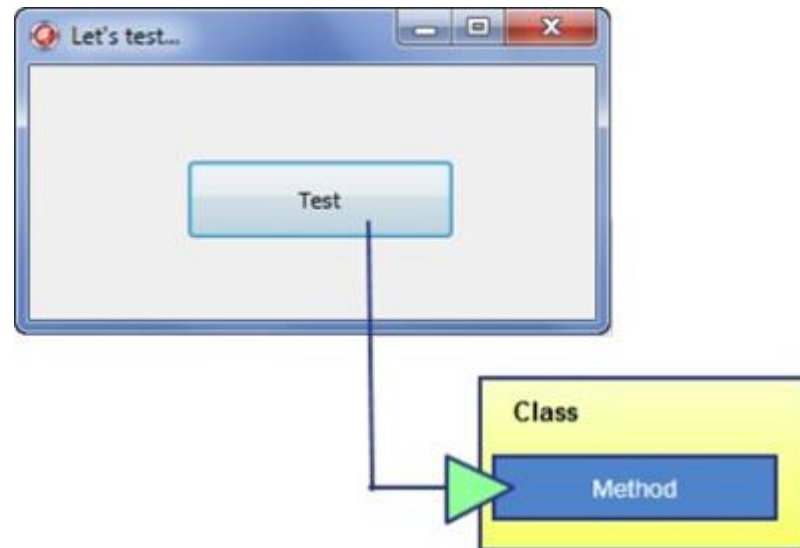
Qualche esempio?



Dalla teoria alla pratica

Il più semplice degli Unit Test

Rapido, immediato e pratico...
...ma forse poco professionale.



Test Framework alla riscossa!

E' indispensabile
usare un
Test Framework!

- Consente di dare una struttura ai test
- Rende i test automatizzati e ripetibili
- Riduce gli errori nel codice dei test
- Produce un rendiconto dell'esito dei test
- Fornisce all'occorrenza una comoda GUI, integrata o fruibile direttamente nell'IDE
- Può fornire (o meno) supporto alla gestione delle dipendenze del sistema sotto test
- Può dialogare con l'eventuale sistema di Continuous Integration & Delivery (CI/CD)

Demo

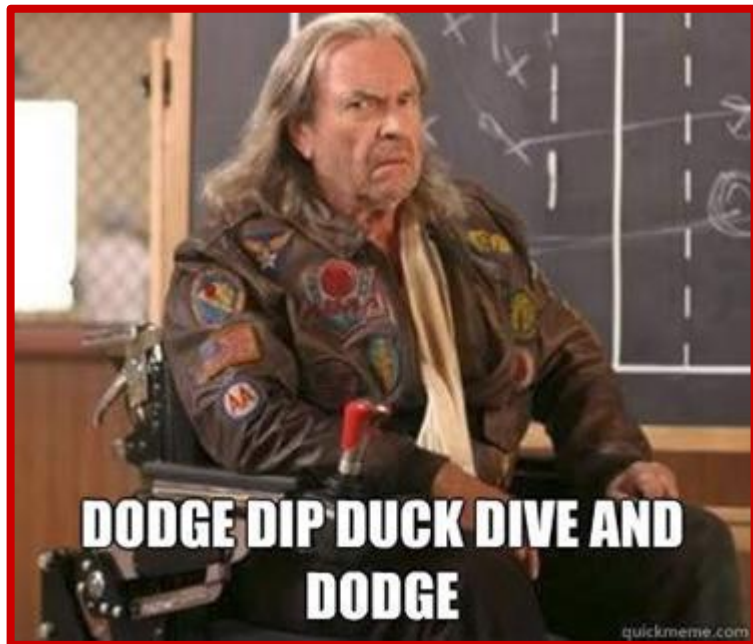


Piccolo glossario



- **Test Fixture / Test Suite / Test Class:**
classe che raggruppa una serie di test
- **Setup:** metodo che inizializza il sistema per l'esecuzione di un test
- **TearDown:** metodo che finalizza il sistema dopo l'esecuzione di un test
- **Test Case / Test Method:** metodo che implementa un test specifico (è il nostro *Unit Test*!)

Azioni di un test



Assign: creiamo gli oggetti e assegnamo i valori iniziali alle variabili.

Act: agiamo sull'oggetto sottoposto a test (es. chiamiamo un suo metodo).

Assert: verifichiamo che l'esito corrisponda a quello che ci attendiamo.

Regole d'oro



- Attribuire al test un nome significativo
- Seguire una convenzione nel naming
- Non eseguire più di una asserzione
- Non usare valori apparentemente insoliti
- Non introdurre logica complessa nel test
- Non dipendere mai dall'esecuzione di un test precedente

State-Based Testing

Il controllo dell'esito del test si basa sullo stato finale dell'oggetto coinvolto (*result driven*).

- *L'erba del prato è verde?*
- *La terra è sufficientemente umida?*



Un momento...

*Come faccio se
l'oggetto da testare
ha delle
dipendenze?*



Cos'è una dipendenza?

“Una dipendenza esterna è un oggetto del sistema software con cui il codice sottoposto a test interagisce e sul quale non ha controllo”.

() es. file system, database, network, I/O in genere, ecc.*

*(**) in breve, qualsiasi implementazione diversa da quella sotto test*

"Stub" alla riscossa!

Lo **Stub** è un oggetto "controllabile" che sostituisce una dipendenza reale.

- *Restituisce valori alla classe sotto test*
- *Contiene una implementazione parziale*
- *I valori restituiti sono quelli che forniscono la risposta attesa per l'asserzione che si intende verificare*



Demo



Interaction-Based Testing

Il controllo dell'esito del test si basa sulla interazione con oggetti che ricevono input o forniscono output (*action driven*).

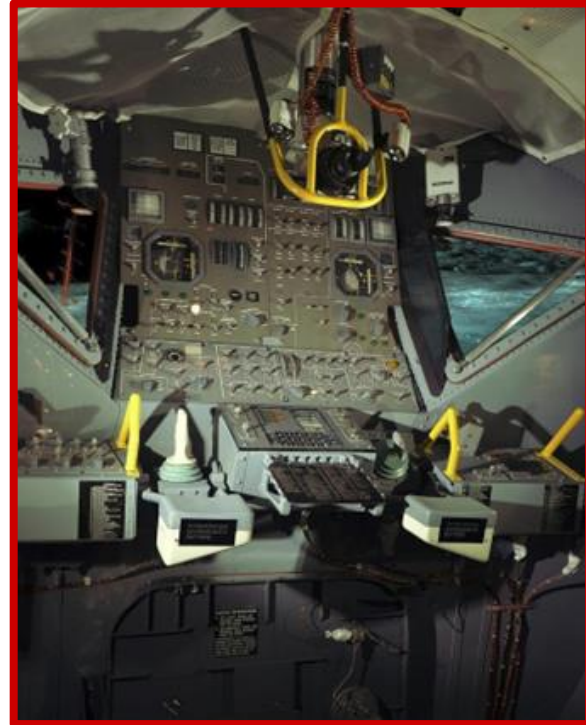
- *Quante volte viene irrigato il campo in un determinato lasso di tempo?*
- *Quanta acqua viene erogata ogni volta?*



"Mock" alla riscossa!

Il **Mock** è un oggetto che interagisce con il sistema sotto test e indica se il test è stato superato oppure no.

- *Viene coinvolto dall'oggetto sottoposto a test*
- *Subisce chiamate a uno o più dei suoi metodi*
- *Verifica di essere stato usato nel modo corretto*



ATTENZIONE: ogni test dovrà usare **uno e un solo mock!**

Demo



Scrivere "fake" è faticoso 😞



- Scrivere le classi può richiedere parecchio tempo
- Le classi possono diventare complesse all'aumentare del numero di condizioni da verificare
- Le classi di stub/mock possono diventare numerose, non riutilizzabili oppure ingestibili
- Possono compromettere l'efficacia del test

Usiamo gli "Isolation Framework"!



Sono **librerie, package e tool** che **offrono "oggetti programmabili"** in grado di

- *aiutare nella creazione di "fake"*
- *sollevare lo sviluppatore dall'onere di scrivere codice*
- *acquisire informazioni sulle interazioni (mock) e fare asserzioni*

Demo



Conclusioni

Testare bene... conviene!

- Forza la scrittura di codice semplice, manutenibile e verificato
- Aumenta la confidenza nel tuo codice e in quello degli altri
- Consente di “giocare d’anticipo” rilevando i bug prima che li scopra il cliente
- E’ una tecnica abilitante per l’adozione di metodologie e architetture moderne
 - Continuous Integration / Delivery
 - Scrum, Agile e Kanban Board
 - TDD (Test Driven Development)
- Nella creazione di librerie e framework di alto livello, l’adozione dei test è (quasi) imprescindibile
 - Gli utenti orientano le loro scelte in base alla presenza o meno di test



Tool e librerie consigliate

- **xUnit.NET** (tool open-source per unit testing in .NET)
<https://xunit.net/>
- **Moq** (Isolation Framework per testing in .NET)
<https://moq.github.io/moq4/>
- **FluentAssertions** (Extension Methods per scrivere meglio le asserzioni)
<https://github.com/fluentassertions/fluentassertions>
- **AutoFixture** (Tool per automatizzare la creazione dei test)
<https://github.com/AutoFixture/AutoFixture>

Risorse e approfondimenti

- Libro **“The Art of Unit Testing (with examples in C#), Second Edition”**
<https://www.manning.com/books/the-art-of-unit-testing-second-edition>
- Libro **“Unit Testing Succinctly”**
<https://www.syncfusion.com/succinctly-free-ebooks/unittesting>
- Post **“The Practical Test Pyramid” (Martin Fowler)**
<https://martinfowler.com/articles/practical-test-pyramid.html>
- Articolo con **tool aggiuntivi per Unit Testing in .NET**
<https://methodpoet.com/unit-testing-tools/>

Q & A

Domande e risposte



Grazie!

