

Week 8 – PC Test



Antonia Sacchitella

Analyst @icubedsrl

antonia.sacchitella@icubed.it



Suddivisione del codice

«L'architettura software è l'organizzazione di base di un sistema, espressa dai suoi componenti, dalle relazioni tra di loro e con l'ambiente, e i principi che ne guidano il progetto e l'evoluzione.»

Informalmente, un'architettura software è la struttura del sistema, costituita dalle varie parti che lo compongono, con le relative relazioni. L'architettura di un sistema software viene definita nella prima fase di Design (definizione dei sottosistemi).

Suddivisione del codice

La definizione dell'architettura viene di solito fatta da due punti di vista diversi, che portano alla soluzione finale:

Identificazione e relazione dei sottosistemi

Definizione politiche di controllo

Entrambe le scelte sono cruciali: è difficile modificarle quando lo sviluppo è partito, poiché molte interfacce dei sottosistemi dovrebbero essere cambiate.

Suddivisione del codice

La definizione dell'architettura viene di solito fatta da due punti di vista diversi, che portano alla soluzione finale:

- Identificazione e relazione dei sottosistemi
- Definizione politiche di controllo

Entrambe le scelte sono cruciali: è difficile modificarle quando lo sviluppo è partito, poiché molte interfacce dei sottosistemi dovrebbero essere cambiate.

Suddivisione del codice

Sono due visioni ortogonali tra loro:

Un «layer» è uno strato che **fornisce servizi**.

- Es. interfaccia utente, accesso al database, messaggistica, ecc.

Una «partizione» è **un'organizzazione di moduli**

- Es: funzionalità «carrello» di un sito di commercio, magazzino, gestione ordini

Mentre per definire una «partizione» è sufficiente identificare il contesto di esecuzione inteso come «argomento» trattato (e quindi richiede della conoscenza del business), per segregare un «layer» si tratta di applicare alcune semplici regole di base:

un layer è un raggruppamento di sottosistemi che forniscono servizi correlati, eventualmente realizzati utilizzando servizi di altri layer

un layer può dipendere solo dai layer di livello più basso

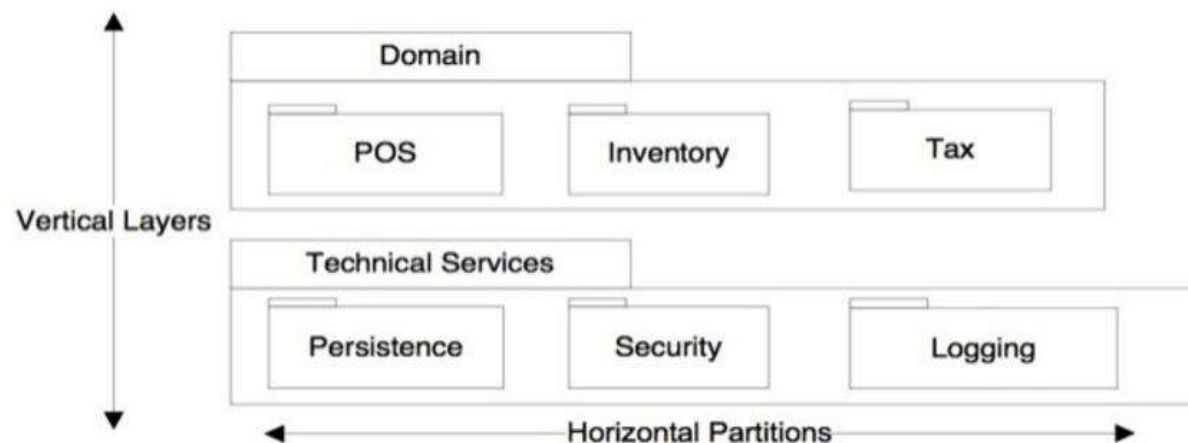
un layer non ha conoscenza dei layer dei livelli più alti.

Suddivisione del codice

In generale una decomposizione in sottosistemi è il risultato di un'attività di partitioning e di layering.

Prima si suddivide il sistema in sottosistemi al top-level che sono responsabili di specifiche funzionalità («partitioning»).

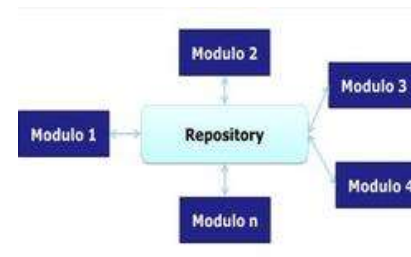
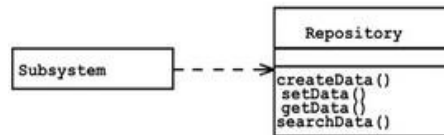
Poi, ogni sottosistema è organizzato in diversi layer, se il livello di complessità lo richiede, finché non sono semplici abbastanza da poter essere implementate da un singolo sviluppatore («layering»)



Suddivisione del codice

Nell'ingegneria del software sono stati definiti vari stili architetturali che possono essere usati come base di architetture software: «Repository Architecture», «Client/Server Architecture» e «Peer-to-peer Architecture».

All'interno di una «**Repository Architecture**» i sottosistemi accedono ad una singola struttura dati, chiamata repository e sono "relativamente indipendenti" (interagiscono solo attraverso il repository).



Suddivisione del codice

- Vantaggi:
 - Modo efficiente di condividere grandi moli di dati: «write once for all to read».
 - Un sottosistema non si deve preoccupare di come i dati sono prodotti/usati da ogni altro sottosistema.
 - Gestione centralizzata di backup, security, access control, recovery da errori.
 - Il modello di condivisione dati è pubblicato come repository schema: è molto facile aggiungere nuovi sottosistemi.
- Svantaggi:
 - I sottosistemi devono concordare su un modello dati «di compromesso»: minori performance.
 - «Data evolution». La adozione di un nuovo modello dati è difficile e costosa: deve venir applicato a tutto il repository tutti i sottosistemi devono essere aggiornati.
 - Diversi sottosistemi possono avere diversi requisiti su backup, security e non possono essere supportati con politiche differenti

Suddivisione del codice

- Vantaggi:
 - Modo efficiente di condividere grandi moli di dati: «write once for all to read».
 - Un sottosistema non si deve preoccupare di come i dati sono prodotti/usati da ogni altro sottosistema.
 - Gestione centralizzata di backup, security, access control, recovery da errori.
 - Il modello di condivisione dati è pubblicato come repository schema: è molto facile aggiungere nuovi sottosistemi.
- Svantaggi:
 - I sottosistemi devono concordare su un modello dati «di compromesso»: minori performance.
 - «Data evolution». La adozione di un nuovo modello dati è difficile e costosa: deve venir applicato a tutto il repository tutti i sottosistemi devono essere aggiornati.
 - Diversi sottosistemi possono avere diversi requisiti su backup, security e non possono essere supportati con politiche differenti

Suddivisione del codice

Per «Client/Server Architecture» si identifica un'architettura distribuita dove dati ed elaborazione sono distribuiti su una rete di nodi di due tipi:

i «server» sono processori che offrono servizi specifici

i «client» sono macchine meno prestazionali sulle quali girano le applicazioni-utente, che utilizzano i servizi dei server.

Il «Server» fornisce servizi ad istanze di altri sottosistemi, detti «Client» che sono responsabili dell'interazione con l'utente.

Suddivisione del codice

Praticamente qualunque applicazione software può essere suddivisa logicamente in tre parti:

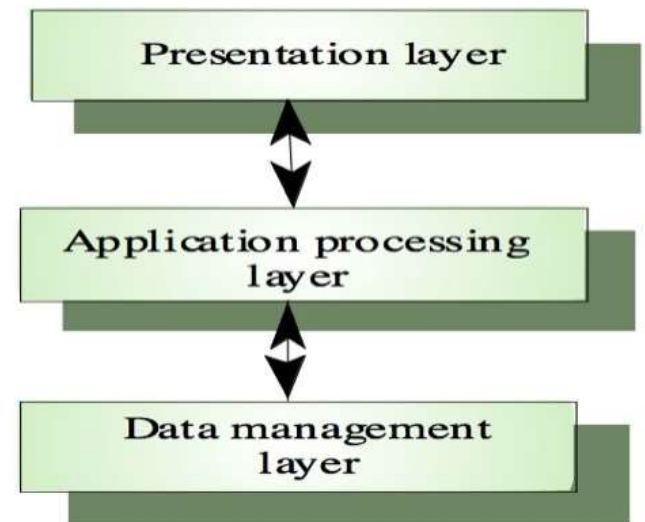
la presentazione che gestisce l'interfaccia utente (gestione degli eventi grafici, controlli formali sui campi in input, ecc)

la logica applicativa vera e propria;

la gestione dei dati persistenti su database.

Di conseguenza, ogni applicazione può essere vista come composta da (almeno) tre layers.

Data questa osservazione, le architetture a 2 livelli risultano essere una forzatura, chiedendo di suddividere su 2 strati i 3 layer concettuali che compongono un'applicazione.



Suddivisione del codice

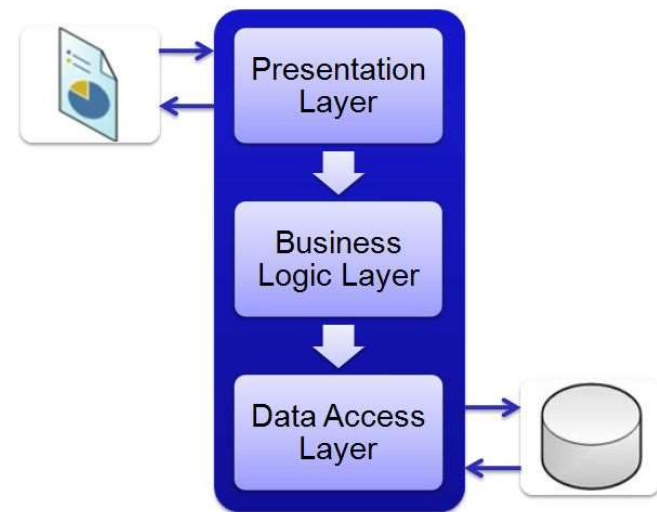
Sono state introdotte all'inizio degli anni '90 per la necessità di definire esplicitamente una business logic che esulasse dalla presentation e dallo storage di un sistema informativo.

Livello 1: gestione dei dati (DBMS, file XML, ecc)

Livello 2: business logic (processamento dati, autenticazione, ecc);

Livello 3: interfaccia utente (presentazione dati, servizi, ecc)

Ogni livello ha **obiettivi e vincoli** di design **propri**: nessun livello fa assunzioni sulla struttura o implementazione degli altri ma si limita ad utilizzare le funzioni pubbliche di servizio esposte dagli altri layer per garantire un colloquio a «black box» con essi.



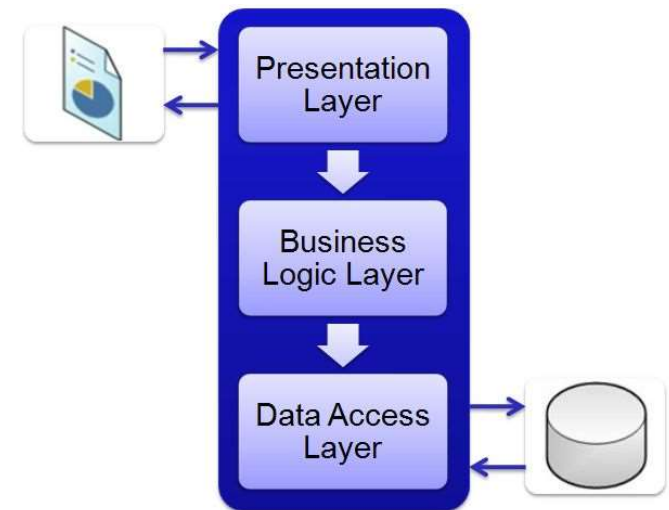
Suddivisione del codice

Non c'è comunicazione diretta tra «livello 1» e «livello 3»:

- L'interfaccia utente non riceve, né inserisce direttamente dati nel livello di data management;
- tutti i passaggi di informazione nei due sensi vengono filtrati dalla business logic

I livelli operano senza assumere di essere parte di una specifica applicazione:

- applicazioni viste come collezioni di componenti cooperanti;
- ogni componente può essere contemporaneamente parte di applicazioni diverse (e.g., database, o componente logica di configurazione di oggetti complessi).



Suddivisione del codice

Rappresentano di fatto una evoluzione delle 3-tier, distribuite su N livelli che permettono configurazioni diverse ed una distribuzione ancora più focalizzata delle competenze su altri layer applicativi.

Gli «n» layer sono variabili a seconda dell'esigenza applicativa e funzionale, ma sono generalmente composte da questi elementi fondamentali:

Suddivisione del codice

Interfaccia utente (UI): gestisce interazione con utente; può essere un web browser, interfaccia grafica desktop, mobile...o IoT (Internet of Things)

Presentation logic: definisce «cosa» la UI presenta e come gestire le richieste utente; è spesso associata con il layer di servizio nel caso di web service ed API

Business logic: gestisce regole di business dell'applicazione, l'autorizzazione all'esecuzione del flusso funzionale e tutte le regole di workflow di sistema

Data access layer: contiene le funzionalità di accesso alle informazioni del sistema, non necessariamente concentrate in una base dati unica, ma potenzialmente in una serie di «storage» generici la cui natura tecnica non è discriminante per l'implementazione degli stati applicativi superiori

Infrastructure services: forniscono funzionalità supplementari alle componenti dell'applicazione (messaging, supporto alle transazioni, notifica, ecc). Spesso sono un layer «trasversale» che viene utilizzato da alcuni o da tutti gli altri layer a seconda della reale necessità

Demo

Suddivisione dell'esercitazione

Si vuole realizzare un sistema per la gestione amministrativa dei corsi di Master.

Il corso del master è caratterizzato da un codice, un nome e una descrizione.

Ciascun corso si articola in un certo numero di lezioni e ogni lezione può essere tenuta da un solo docente per volta.

I docenti sono caratterizzati da nome, cognome, email e numero di telefono.

Possono esistere docenti assegnati al corso che, per vari motivi, non tengono alcuna lezione. I

docenti si distinguono fra docenti interni ed esterni. Per i docenti interni si deve tenere traccia degli anni di anzianità, mentre per i docenti esterni si richiede il nome dell'azienda di appartenenza

Ogni lezione ha una data e un orario di inizio, una durata (in termini di giorni) e un'aula assegnata.

Inoltre ogni lezione può richiedere l'utilizzo di risorse aggiuntive (es. tablet, PC, etc.).



Demo

Ogni corso ha un certo numero di partecipanti per i quali si vuole tenere traccia del nome, cognome, data di nascita, indirizzo email e ultimo titolo di studio.

Le operazioni da implementare sono:

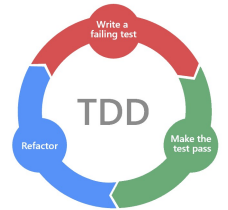
- *inserimento/cancellazioni delle lezioni*
- *inserimento di un docente*
- *assegnazione di un docente ad un corso*
- *visualizzazione delle lezioni di un corso (comprendendo anche i dati relativi ai docenti)*
- *dato un corso visualizzare i dati dei partecipanti iscritti*

NB. Se non specificato, la chiave primaria delle entità è un id intero.

Realizzare il modello E-R della struttura, utilizzare entity-framework per la realizzazione dello strato di persistenza.



Test Driven Development (TDD)



Il test-driven development è un modello di sviluppo del software.

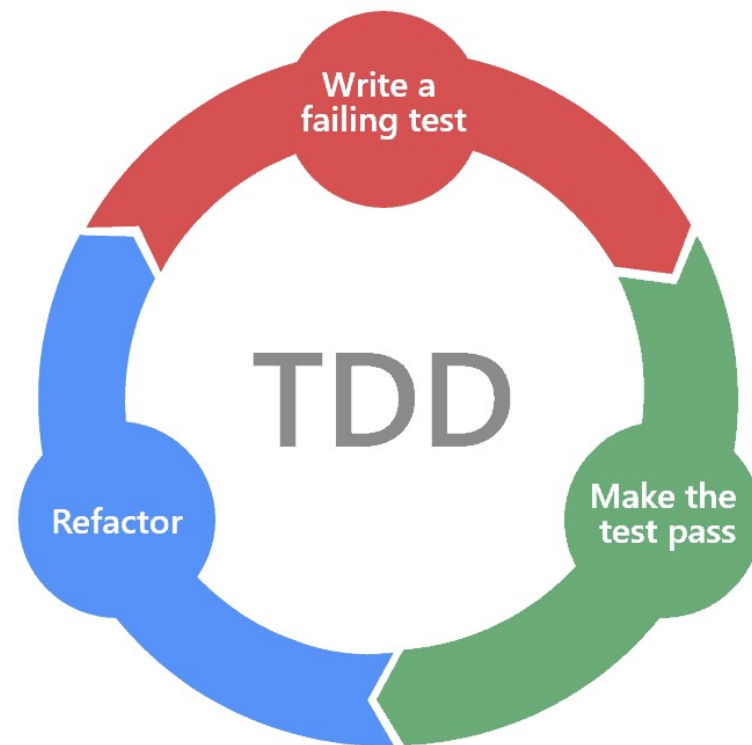
Prevede che la stesura dei test avvenga prima di quella del software che deve essere sottoposto a test.

Test Driven Development (TDD)

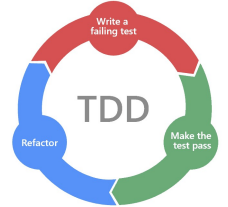


Il processo di TDD prevede di:

- Scrivere un test (*destinato a fallire*)
- Scrivere il codice necessario a passare il test
- Rilavorare il codice scritto per ottimizzarlo (**Refactoring**)



Test Driven Development (TDD)

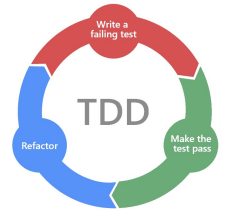


Il principio fondamentale del TDD è che lo sviluppo vero e proprio deve avvenire *solo* allo scopo di passare un test che (inizialmente) fallisce.

Questo vincolo è inteso a impedire che

- il programmatore sviluppi funzionalità non esplicitamente richieste
- il programmatore introduca complessità eccessiva in un progetto, per esempio perché prevede la necessità di generalizzare l'implementazione in un futuro più o meno prossimo (Overdesign)

Test Driven Development (TDD)



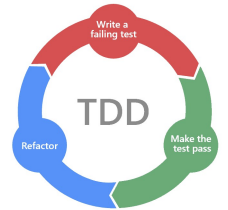
Il principio fondamentale del TDD è che lo sviluppo vero e proprio deve avvenire *solo* allo scopo di passare un test che (inizialmente) fallisce.

In questo senso il TDD è in stretta relazione con numerosi principi della programmazione agile e dell'extreme programming

KISS (Keep It Simple, Stupid)

YAGNI (You Aren't Gonna Need It)

Test Driven Development (TDD)



Dal punto di vista pratico, si realizza una **Batteria di Test**, che possono essere eseguiti anche in modalità automatica.

I test sono frammenti di codice, scritti utilizzando le funzionalità messe a disposizione da alcune librerie (MS Tests, Xunit, Nunit ...). Le batterie di test sono solitamente tutte raggruppate all'interno di progetti dedicati.

```
0 references
public class WeatherForecastTests
{
    [Fact]
    0 references
    public void ShouldReturnFiveForecasts()
    {
        // ARRANGE
        var sut = new WeatherForecastController();

        // ACT
        var result = sut.Get();

        OkObjectResult okResult = (OkObjectResult)result;

        IList<WeatherForecast> data = (IList<WeatherForecast>)okResult.Value;

        // ASSERT
        Assert.True(data.Count ≥ 5);
    }
}
```

Test Driven Development (TDD)



Dal punto di vista pratico, si realizza una **Batteria di Test**, che possono essere eseguiti anche in modalità automatica.

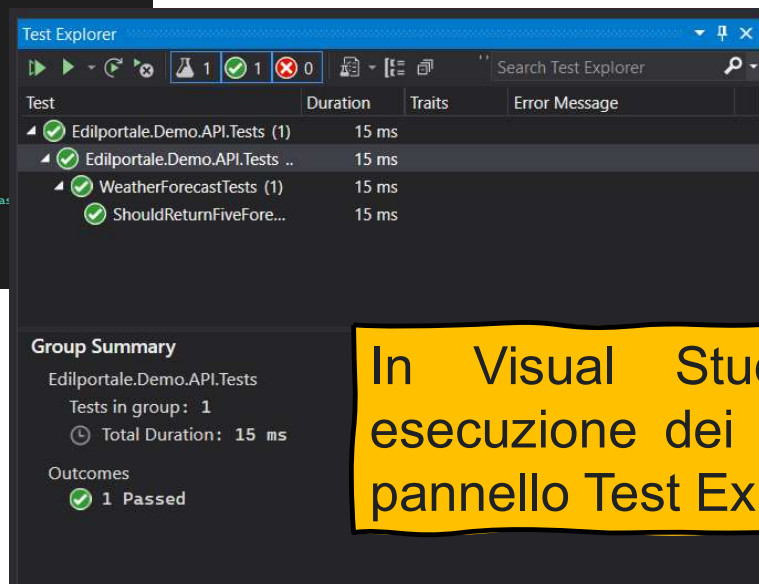
```
0 references
public class WeatherForecastTests
{
    [Fact]
    0 references
    public void ShouldReturnFiveForecasts()
    {
        // ARRANGE
        var sut = new WeatherForecastController();

        // ACT
        var result = sut.Get();

        OkObjectResult okResult = (OkObjectResult)result;

        IList<WeatherForecast> data = (IList<WeatherForecast>)okResult.Value;

        // ASSERT
        Assert.True(data.Count >= 5);
    }
}
```



The screenshot shows the Visual Studio Test Explorer window. At the top, there are icons for running tests: a green play button, a green checkmark, a red X, and a red stop button. Below these icons, a table lists the tests and their results:

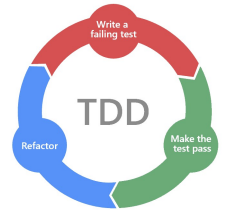
Test	Duration	Traits	Error Message
Edilportale.Demo.API.Tests (1)	15 ms		
Edilportale.Demo.API.Tests ..	15 ms		
WeatherForecastTests (1)	15 ms		
ShouldReturnFiveFore...	15 ms		

Below the table, the 'Group Summary' section shows:

- Edilportale.Demo.API.Tests
- Tests in group: 1
- Total Duration: 15 ms
- Outcomes: 1 Passed

In Visual Studio, per la gestione / esecuzione dei test è possibile sfruttare il pannello Test Explorer.

Test Driven Development (TDD)



```
0 references
public class WeatherForecastTests
{
    [Fact]
    0 references
    public void ShouldReturnFiveForecasts()
    {
        // ARRANGE
        var sut = new WeatherForecastController();

        // ACT
        var result = sut.Get();

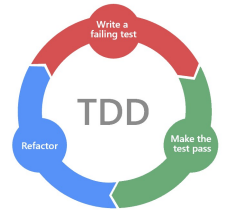
        OkObjectResult okResult = (OkObjectResult)result;

        IList<WeatherForecast> data = (IList<WeatherForecast>)okResult.Value;

        // ASSERT
        Assert.True(data.Count ≥ 5);
    }
}
```

XUnit

Test Driven Development (TDD)



I Test sono organizzati in classi.

```
public class WeatherForecastTests
{
    [Fact]
    0 references
    public void ShouldReturnFiveForecasts()
    {
        // ARRANGE
        var sut = new WeatherForecastController();

        // ACT
        var result = sut.Get();

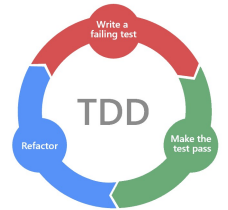
        OkObjectResult okResult = (OkObjectResult)result;

        IList<WeatherForecast> data = (IList<WeatherForecast>)okResult.Value;

        // ASSERT
        Assert.True(data.Count ≥ 5);
    }
}
```

XUnit

Test Driven Development (TDD)



I Test sono organizzati in classi.

```
public class WeatherForecastTests
{
    [Fact]
    0 references
    public void ShouldReturnFiveForecasts()
    {
        // ARRANGE
        var sut = new WeatherForecastController();

        // ACT
        var result = sut.Get();

        OkObjectResult okResult = (OkObjectResult)result;

        IList<WeatherForecast> data = (IList<WeatherForecast>)okResult.Value;

        // ASSERT
        Assert.True(data.Count ≥ 5);
    }
}
```

Ogni metodo pubblico, marcato con l'attributo [Fact] è un test.

XUnit

Test Driven Development (TDD)



I Test sono organizzati in classi.

```
public class WeatherForecastTests
```

```
{
```

```
[Fact]
```

```
0 references
```

```
public void ShouldReturnFiveForecasts()
```

```
{
```

```
// ARRANGE
```

```
var sut = new WeatherForecastController();
```

```
// ACT
```

```
var result = sut.Get();
```

```
OkObjectResult okResult = (OkObjectResult)result;
```

```
ICollection<WeatherForecast> data = (ICollection<WeatherForecast>)okResult.Value;
```

```
// ASSERT
```

```
Assert.True(data.Count ≥ 5);
```

```
}
```

```
}
```

Ogni metodo pubblico, marcato con l'attributo [Fact] è un test.

Ogni test segue le stesse tre fasi:

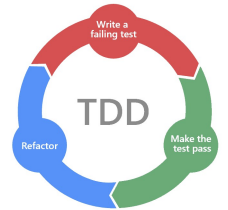
ARRANGE

ACT

ASSERT

XUnit

Test Driven Development (TDD)



I Test sono organizzati in classi.

```
public class WeatherForecastTests
```

```
{
```

```
[Fact]
```

```
0 references
```

```
public void ShouldReturnFiveForecasts()
```

```
{
```

```
// ARRANGE
```

```
var sut = new WeatherForecastController();
```

```
// ACT
```

```
var result = sut.Get();
```

```
OkObjectResult okResult = (OkObjectResult)result;
```

```
ICollection<WeatherForecast> data = (ICollection<WeatherForecast>)okResult.Value;
```

```
// ASSERT
```

```
Assert.True(data.Count ≥ 5);
```

```
}
```

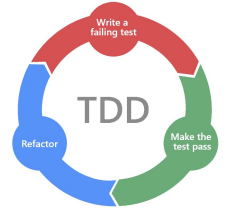
```
}
```

Ogni metodo pubblico, marcato con l'attributo [Fact] è un test.

Ogni test segue le stesse tre fasi:
ARRANGE
ACT
ASSERT

XUnit

Test Driven Development (TDD)



I Test sono organizzati in classi.

```
public class WeatherForecastTests
```

```
{
```

```
[Fact]
```

```
0 references
```

```
public void ShouldReturnFiveForecasts()
```

```
{
```

```
// ARRANGE
```

```
var sut = new WeatherForecastService();
```

```
// ACT
```

```
var result = sut.GetForecasts();
```

```
(OkObjectResult)result;
```

```
WeatherForecast> data = (IList<WeatherForecast>)okResult.Value;
```

```
// ASSERT
```

```
Assert.True(data.Count ≥ 5);
```

```
}
```

```
}
```

Ogni metodo pubblico, marcato con l'attributo [Fact] è un test.

Il codice del test esegue un Happy Path

segue le stesse tre fasi:

ARRANGE

ACT

ASSERT

XUnit

Demo

Test Driven Development



Demo

Realizzare una Class Library che contenga una classe Equation.

L'unico metodo di questa classe è

```
double[] ResolveSecondDegreeEquation (double a, double b, double c)
```

che risolve l'equazione di secondo grado $ax^2 + bx + c = 0$

Predisporre un Progetto con una batteria di test (Xunit) che verifichi il corretto funzionamento del metodo di risoluzione nei seguenti casi:

a	b	c	Risultato
1	-3	2	$X_1 = 1 ; X_2 = 2$
1	-5	6	$X_1 = 2 ; X_2 = 3$
1	2	1	$X_1 = X_2 = -1$
1	-3	10	Nessuna Soluzione



Esercitazione

Realizzare un'applicazione (utilizzando opportunamente la suddivisione in progetti) che implementi una calcolatrice.

Per la classe calcolatrice devono essere disponibili i metodi di somma, divisione, sottrazione e moltiplicazione.

Utilizzare l'approccio del Test Driven Development specificando una batteria di test per tutte le operazioni.

Il presentation layer verrà realizzato con un'applicazione console ed uno specifico menù.

