

# Artificial Intelligence Driven Development (AIDD)

---

FEDERICA CITARRELLA



# DEFINIZIONE

---

L' **Artificial Intelligence Driven Development** (AIDD) è una metodologia di sviluppo software basato sull'utilizzo dell'intelligenza artificiale e in particolare Large Language Model (LLM). Il software engineer è responsabile di guidare l'intelligenza artificiale al fine di massimizzare i risultati.

# AIDD VANTAGGI



## **Automazione dei compiti ripetitivi:**

L'AI può automatizzare compiti noiosi e ripetitivi, consentendo agli sviluppatori di concentrarsi su compiti di valore aggiunto.



## **Aumento della produttività:**

L'IA può accelerare il processo di sviluppo software, riducendo i tempi di progettazione e sviluppo, il che è particolarmente utile in progetti con scadenze strette.

# AIDD SVANTAGGI



## **Mancanza di comprensione del contesto:**

L'IA può creare codice e prendere decisioni basate sui dati disponibili, ma spesso manca della comprensione del contesto aziendale e delle esigenze specifiche del cliente.

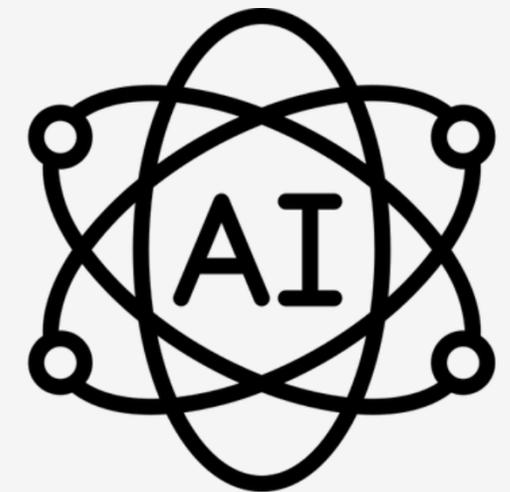
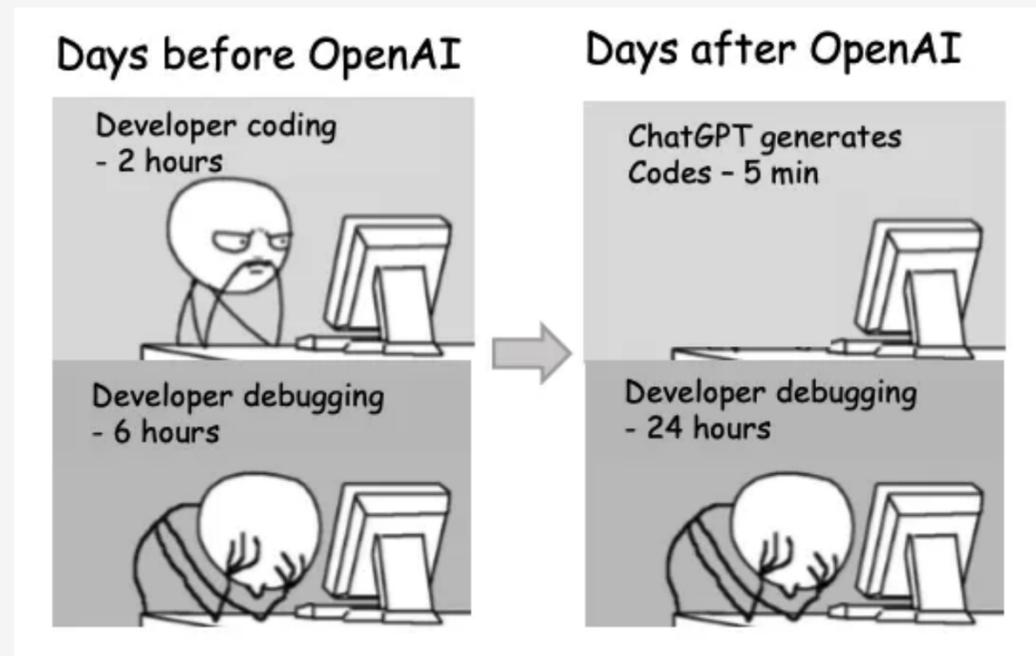


## **Comprensione limitata delle decisioni dell'IA:**

Gli algoritmi di apprendimento automatico possono essere difficili da interpretare. Ciò rende difficile capire il motivo per cui l'IA ha preso determinate decisioni.

# GAP COGNITIVO

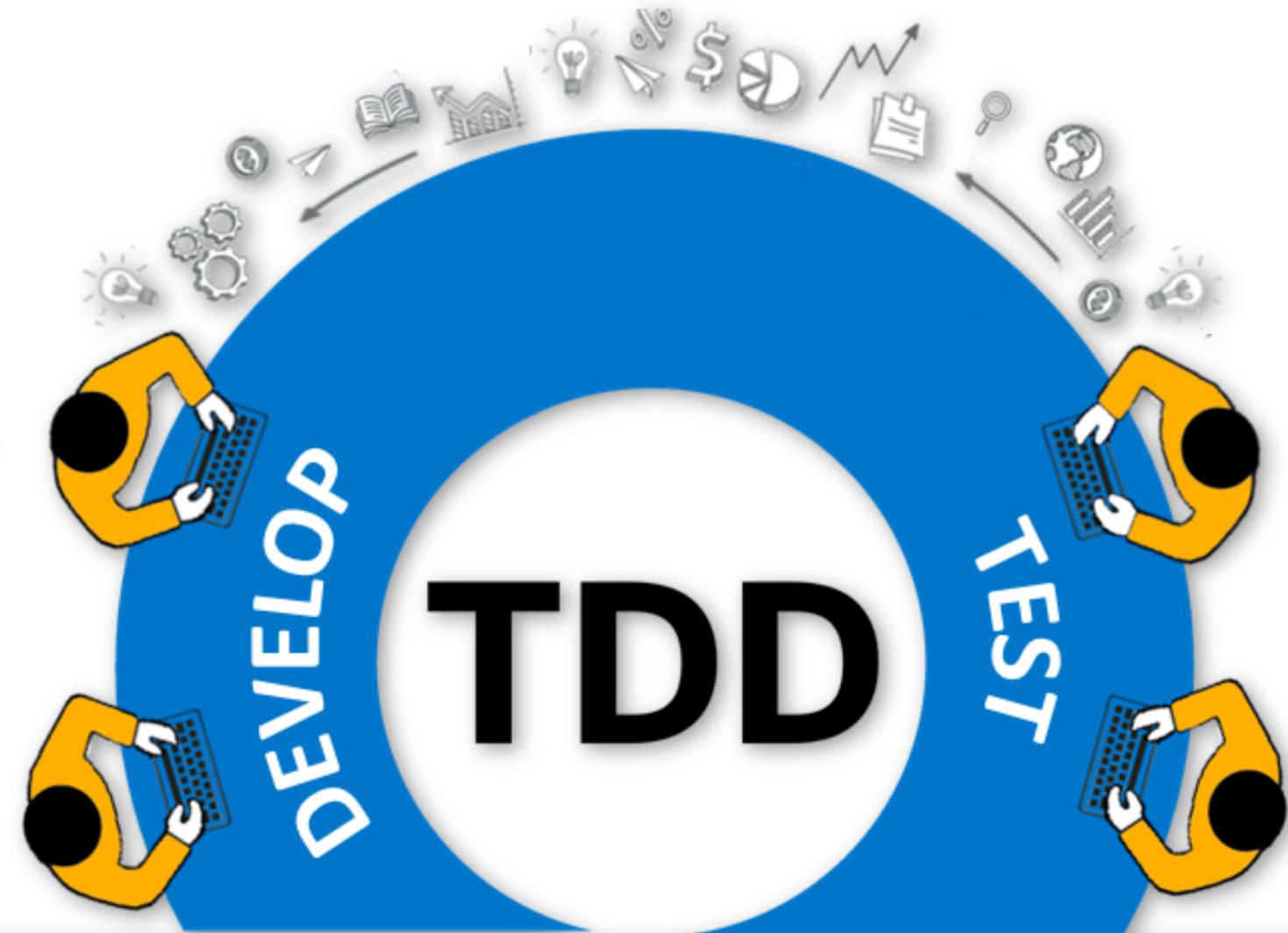
- L'AI manca della completezza del contesto
- Lo sviluppatore ha comprensione limitata delle decisioni dell'IA



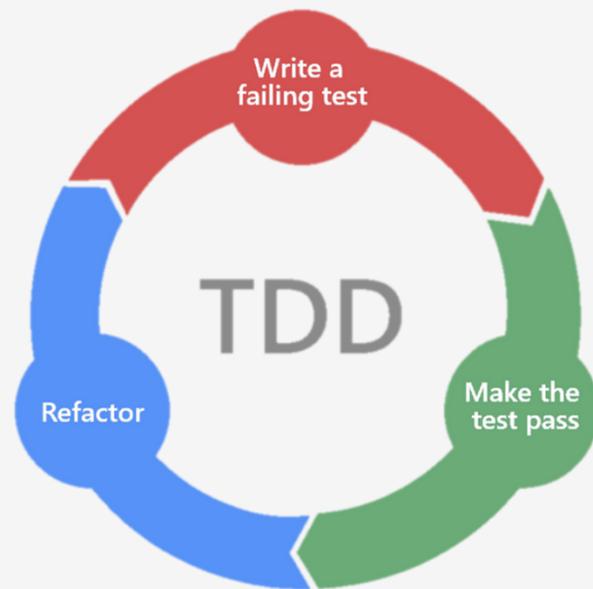
# TEST DRIVEN DEVELOPMENT (TDD)

---

Il **Test-driven development** (TDD) è una metodologia di sviluppo software che si basa sullo scrivere i test prima del codice. Segue un processo ciclico di scrittura di test fallimentari, scrittura del codice minimo per passare il test e infine refactoring del codice.



# TDD + ChatGPT3.5



# TDD base value + ChatGPT3.5



# TIC TAC TOE: TDD + CHATGPT3.5

1. Analisi dei behaviour
2. Definizione del contesto

FE: conosci il gioco tic tac toe?

Si, conosco il gioco del Tic-Tac-Toe, noto anche come Tris o Tre en Raya in alcune regioni. È un gioco da tavolo per due giocatori che si svolge su una griglia 3x3 (anche se possono

FE: puoi fare in modo che la board sia una mappa di position e player?

Certamente, puoi utilizzare una mappa (HashMap) per tenere traccia delle posizioni e dei giocatori sulla board. In questo modo, ogni posizione del gioco sarà mappata a un giocatore. Ecco come puoi farlo:

```
java
import java.util.HashMap;
import java.util.Map;
```

FE: puoi definire la classe del test come "TestThatTicTacToeShould"? e il test come "makeXFirstPlayer"

Certamente, puoi denominare la classe del test "TestThatTicTacToeShould" e il test come "makeXFirstPlayer" in questo modo:

```
java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestThatTicTacToeShould {

    @Test
```

3.

Write a failing test

FE: puoi scrivere un test junit jupiter che testi che il primo giocatore sia X? (codice in inglese)

Of course! Here's a JUnit 5 test for a Tic-Tac-Toe game:

FE: puoi generare una classe TicTacToe che faccia fallire questo test?

Certamente, posso fornirti una classe `TicTacToe` di esempio che farà fallire il test che verifica che il primo giocatore sia "X". Ecco un'implementazione di `TicTacToe` che non rispetta la condizione desiderata:

```
java
Copy code
```

FE: puoi implementare la logica nella classe tic-tac-toe in modo da passare i test?

Si, posso mostrarti come implementare la logica nella classe `TicTacToe` in modo da far superare i test "makeXFirstPlayer" e "makeOSecondPlayer". Ecco un'implementazione possibile:

```
java
Copy code

public class TicTacToe {
    private String currentPlayer;

    public TicTacToe() {
        // Inizializziamo il simbolo del primo giocatore come "X"
```

Refactor

5.

Make the test pass

4.

TDD

# TIC TAC TOE: TDD BASE VALUE + CHATGPT3.5

1. Analisi dei behaviour
2. Definizione del contesto

FE avrei bisogno dei test junit jupiter per testare i seguenti test cases:

1. che il primo giocatore sia X
2. che il secondo giocatore sia O
3. che il gioco alterni correttamente X e O
4. ritorni pareggio se la board è piena
5. che negli 8 casi di vittoria (righe, colonne e diagonali) il winner sia X o Y

Ecco i test JUnit Jupiter per i casi che hai menzionato:

FE su tutti i test, al posto di utilizzare le stringhe per X e O, crea un enum

Puoi utilizzare un enum per rappresentare i giocatori X e O nei

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class TicTacToeTest {

    private enum Player { X, O }

    @Test
    public void testCurrentPlayerIsX() {
        TicTacToe game = new TicTacToe();
        Player currentPlayer = game.getCurrentPlayerEnum();
        assertEquals(Player.X, currentPlayer, "The current player should be X");
    }

    @Test
    public void testCurrentPlayerIsOAfterOneMove() {
        TicTacToe game = new TicTacToe();
        game.move(0, 0, 'X');
```

3.

4.

TESTS

CODING

FE potresti ora generarmi la classe TicTacToe che passi tutti i test che ti ho descritto?

Ecco una possibile implementazione della classe `TicTacToe` che dovrebbe superare tutti i test che abbiamo definito finora:

```
java Copy code

public class TicTacToe {
    private enum Player { X, O }
    private Player currentPlayer;
    private Map<Position, Player> board;
    private Player winner;
    private int moves;

    public TicTacToe() {
        currentPlayer = Player.X;
        board = new EnumMap<>(Position.class);
        winner = null;
        moves = 0;
    }

    public Player getCurrentPlayerEnum() {
        return currentPlayer;
    }
}
```

FE potresti fare in modo che la board sia una mappa position - player?

Certamente, è possibile utilizzare una mappa che associ le posizioni ai giocatori invece di una matrice bidimensionale. Ecco una versione aggiornata della classe `TicTacToe`:

```
java Copy code

import java.util.EnumMap;
import java.util.Map;

public class TicTacToe {
    private enum Player { X, O }
    private enum Position { TOP_LEFT, TOP_CENTER, TOP_RIGHT, MIDDLE_LEFT, MIDDLE_CENTER, MIDDLE_RIGHT, BOTTOM_LEFT, BOTTOM_CENTER, BOTTOM_RIGHT }
    private Map<Position, Player> board;
    private Player currentPlayer;
    private Player winner;
    private int moves;

    public TicTacToe() {
        currentPlayer = Player.X;
        board = new EnumMap<>(Position.class);
        winner = null;
        moves = 0;
    }

    public Player getCurrentPlayerEnum() {
        return currentPlayer;
    }
}
```

Send a message

## Conclusioni TDD + AI

---



- Processo troppo guidato
- Si perdono i vantaggi dell'AI
- No ottimizzazione tempo

## Conclusioni TDD VALUE + AI

---



- Ottimizzazione tempo
- Codice di alta qualità



**GRAZIE!**

[federica.citarrella@eoc.ch](mailto:federica.citarrella@eoc.ch)



## References

[https://medium.com/neudesic-innovation/ai-driven-development-the-future-of-software-engineering\\_a4b72199365d](https://medium.com/neudesic-innovation/ai-driven-development-the-future-of-software-engineering_a4b72199365d)

<https://betterprogramming.pub/now-tdd-is-more-important-than-ever-dfaf65024d9>

<https://chat.openai.com/>

<https://www.encora.com/insights/ai-driven-development>