Disciplina	Prof. Dacio Machado	
PROJETO IMPLEMENTAÇÃO E TESTE DE SOFTWARE	Valor	+01 ATV
ATIVIDADE : Test-Driven Development	Aluno: Vinicius Reginaldo Ferrarini	
ESOFT - 6 - N		23159293-2

Atividade prática de Test-Driven Development:

Descrição da Aplicação

Um sistema simples de **lista de tarefas** (*To-Do List System*), no qual os usuários possam gerenciar suas tarefas diárias, com recursos básicos como um CRUD de tarefas e uma opção de concluído ou em andamento.

Requisitos Funcionais

1. O usuário deve poder **adicionar tarefas** à lista, informando um **nome** e uma **descrição**.
2. O usuário deve poder **marcar uma tarefa como concluída**.
3. O usuário deve poder **marcar uma tarefa como em andamento**.
4. O usuário deve poder **editar** o **nome** e a **descrição** de uma tarefa existente.
5. O usuário deve poder **excluir tarefas** da lista.

ID do Requisito	Descrição do Requisito	ID do Teste	Descrição do Teste	Status Esperado
REQ-01	O sistema deve permitir **adicionar uma tarefa** à lista, informando um **nome** e uma **descrição**.	**TEST-01**	Verificar se uma nova tarefa é criada corretamente quando fornecidos nome e descrição válidos.	Teste deve **passar** (tarefa adicionada com sucesso).
		TEST-02	Verificar se o sistema rejeita a criação de tarefa sem nome (ou com nome vazio).	Teste deve **passar** (erro ou exceção esperada).
REQ-02	O sistema deve permitir **marcar uma tarefa como concluída**.	**TEST-03**	Verificar se, ao marcar uma tarefa como concluída, seu status é atualizado corretamente.	Teste deve **passar** (status alterado para "concluída").
		TEST-04	Verificar se tarefas já concluídas não podem ser marcadas novamente.	Teste deve **passar** (mensagem de aviso ou nenhuma alteração).
REQ-03	O sistema deve permitir **marcar uma tarefa como em andamento**.	**TEST-05**	Verificar se uma tarefa pode ser marcada como "em andamento".	Teste deve **passar** (status alterado para "em andamento").
		TEST-06	Verificar se não é possível marcar como "em andamento" uma tarefa já concluída.	Teste deve **passar** (erro ou aviso emitido).
REQ-04	O sistema deve permitir **editar o nome e a descrição** de uma tarefa existente.	**TEST-07**	Verificar se é possível atualizar o nome e a descrição de uma tarefa existente.	Teste deve **passar** (dados atualizados corretamente).

		TEST-08	Verificar se o sistema impede edição de uma tarefa inexistente.	Teste deve **passar** (erro ou exceção esperada).
REQ-05	O sistema deve permitir **excluir tarefas** da lista.	**TEST-09**	Verificar se uma tarefa pode ser removida da lista com sucesso.	Teste deve **passar** (tarefa removida).
		TEST-10	Verificar se o sistema trata corretamente a tentativa de excluir uma tarefa inexistente.	Teste deve **passar** (erro ou exceção esperada).

ETAPAS DA ATIVIDADE

Passo 1 – Definir os Testes

- Comece **escrevendo os testes** para cada requisito, **um de cada vez**.
- Faça um **esboço textual** descrevendo o comportamento esperado de cada funcionalidade antes de implementá-la.

Passo 2 – Escrever o Primeiro Teste

- Implemente o **primeiro caso de teste** utilizando `unittest` do `Python` (ou outra ferramenta de teste da linguagem escolhida).
- O teste deve inicialmente **falhar** (etapa *Red*).

Passo 3 – Aplicar o Ciclo TDD (Red → Green → Refactor)

- **Red**: execute o teste e confirme que ele falha.
- **Green**: escreva o código mínimo necessário para fazer o teste **passar**.
  Refactor: **refatore** o código, melhorando sua clareza e estrutura sem alterar o comportamento.

Passo 4 – Repetir o Ciclo para Todos os Casos de Teste

- Continue aplicando o ciclo TDD até que **todos os requisitos** do sistema tenham sido implementados e testados com sucesso.

Passo 1 – Definir os Testes

Primeiro, pense nos testes antes de implementar qualquer código. Baseando-se nos requisitos que você passou, podemos definir o seguinte **esboço de testes**:

Requisito	Teste	Comportamento esperado
REQ-01	TEST-01	Criar tarefa com nome e descrição válidos
	TEST-02	Rejeitar tarefa sem nome
REQ-02	TEST-03	Marcar tarefa como concluída
	TEST-04	Não marcar novamente tarefa concluída
REQ-03	TEST-05	Marcar tarefa como em andamento
	TEST-06	Não marcar como andamento tarefa concluída
REQ-04	TEST-07	Editar nome e descrição de uma tarefa existente
	TEST-08	Impedir edição de tarefa inexistente
REQ-05	TEST-09	Excluir tarefa existente
	TEST-10	Tratar exclusão de tarefa inexistente

Passo 2 – Escrever o Primeiro Teste

Vamos criar um arquivo chamado teste.py e escrever o primeiro teste (TEST-01):

codigo:

```
import unittest

from todo import TodoList, Task


class TestTodoList(unittest.TestCase):


    def setUp(self):

        self.todo = TodoList()


    def test_add_task_valid(self):

        task = self.todo.add_task("Comprar leite", "Ir ao mercado comprar leite")

        self.assertEqual(task.name, "Comprar leite")

        self.assertEqual(task.description, "Ir ao mercado comprar leite")

        self.assertEqual(task.status, "em andamento")  # status inicial


if __name__ == '__main__':

    unittest.main()
```

Passo 3 – Implementar o Código Mínimo (Green)

Crie o arquivo tudo.py com a implementação mínima para passar o teste:

```
class Task:
    def __init__(self, name, description):
        if not name:
            raise ValueError("O nome da tarefa não pode ser vazio")
        self.name = name
        self.description = description
        self.status = "em andamento"


class TodoList:
    def __init__(self):
        self.tasks = []


    def add_task(self, name, description):
        task = Task(name, description)
        self.tasks.append(task)
        return task
```

Passo 4 − Refatorar

Aqui você pode melhorar nomes, estruturas ou adicionar validações sem mudar o comportamento. Por enquanto está simples, então pode seguir para o próximo teste.

Passo 5 – Implementar os Próximos Testes

Passo 6 – Repetir o Ciclo para Todos os Testes