Unicesumar

Curso de Engenharia de Software

Disciplina: PROJETO, IMPLEMENTAÇÃO E TESTE DE SOFTWARE

**Atividade Prática – TDD (Test-Driven Development)**

Aluno: Murilo Zessin Miotta

Data: 16/10/2025

# 1. Introdução

Esta atividade tem como objetivo aplicar a metodologia de Test-Driven Development (TDD) no desenvolvimento de um sistema simples de Lista de Tarefas, implementando os requisitos funcionais por meio da escrita de testes antes da codificação (Red → Green → Refactor). A aplicação do TDD garante que o código seja desenvolvido de forma incremental, com testes guiando cada funcionalidade desde o início.

# 2. Requisitos Funcionais

O sistema de Lista de Tarefas deve atender aos seguintes requisitos:

1. Adicionar tarefas à lista, com nome e descrição.

2. Marcar uma tarefa como concluída.

3. Marcar uma tarefa como em andamento.

4. Editar nome e descrição de uma tarefa existente.

5. Excluir tarefas da lista.

# 3. Etapas do Desenvolvimento TDD

O desenvolvimento foi conduzido seguindo as etapas apresentadas no material da disciplina:

• Passo 1 – Definir os Testes: Foram escritos os testes para cada requisito, um de cada vez, e feito um esboço textual do comportamento esperado.

• Passo 2 – Escrever o Primeiro Teste: O primeiro teste foi escrito e executado sem implementação prévia, resultando em falha (Red).

• Passo 3 – Aplicar o Ciclo TDD (Red → Green → Refactor): Para cada teste: Red → Teste falha inicialmente, Green → Implementação mínima para passar, Refactor → Refatoração sem alterar comportamento.

• Passo 4 – Repetir o Ciclo: O processo foi repetido até todos os testes passarem com sucesso.

# 4. Ciclo Red → Green → Refactor – Testes 01 a 10

## TEST-1 – Adicionar tarefa válida

### Red

O teste foi escrito para verificar se o sistema permite adicionar uma tarefa com nome e descrição válidos. Como o método de adicionar tarefa ainda não existia, o teste falhou ao ser executado, caracterizando a etapa Red.

### Green

Foi implementada uma solução mínima que permitisse criar e armazenar uma tarefa com os campos necessários, fazendo o teste passar com sucesso.

### Refactor

Após o teste passar, a estrutura do código foi aprimorada, tornando-o mais organizado e claro, sem alterar o comportamento.

## TEST-2 – Rejeitar criação de tarefa com nome vazio

### Red

O teste foi escrito para garantir que o sistema rejeite a criação de tarefas com nome vazio ou espaços. Inicialmente, o método de adição aceitava qualquer nome, então o teste falhou.

### Green

Foi adicionada a validação mínima para impedir nomes vazios, fazendo com que o teste passasse corretamente.

### Refactor

Depois, a validação foi integrada de forma mais elegante ao fluxo de criação, mantendo o código limpo.

## TEST-3 – Marcar uma tarefa como concluída

### Red

O teste foi escrito para verificar se, ao marcar uma tarefa como concluída, seu status mudaria corretamente. Como não havia implementação para isso, o teste falhou.

### Green

Foi criada uma implementação mínima para alterar o status da tarefa para concluída, permitindo que o teste passasse.

### Refactor

Em seguida, o código foi ajustado para reutilizar métodos internos e manter a clareza.

## TEST-4 – Impedir remarcar tarefa já concluída

### Red

O teste foi escrito para garantir que uma tarefa concluída não possa ser marcada como concluída novamente. Como a lógica de validação não existia, o teste falhou.

### Green

Foi adicionada uma verificação simples que impede a alteração se a tarefa já estiver concluída, fazendo o teste passar.

### Refactor

Posteriormente, a verificação foi integrada de forma coesa à estrutura da classe.

## TEST-5 – Marcar tarefa como em andamento

### Red

O teste foi criado para verificar se o sistema permite marcar uma tarefa como 'em andamento'. Sem implementação, o teste falhou inicialmente.

### Green

Foi feita uma implementação mínima que muda o status da tarefa para 'em andamento', permitindo que o teste passasse.

### Refactor

Depois, o método foi refatorado para seguir o mesmo padrão de clareza e consistência dos demais métodos.

## TEST-6 – Impedir marcar 'em andamento' se já estiver concluída

### Red

O teste foi escrito para impedir que uma tarefa marcada como concluída possa voltar para 'em andamento'. Sem a validação, o teste falhou.

### Green

Foi adicionada a verificação mínima para lançar erro quando isso ocorresse, fazendo o teste passar.

### Refactor

A validação foi então incorporada de forma clara e consistente ao restante da lógica.

## TEST-7 – Editar nome e descrição de tarefa

### Red

O teste foi escrito para validar se é possível editar o nome e a descrição de uma tarefa existente. Sem o método correspondente, o teste falhou.

### Green

Foi implementado um método básico de edição que atualiza os campos indicados, permitindo que o teste passasse.

### Refactor

Posteriormente, o método foi reorganizado para garantir clareza e tratar casos opcionais de forma elegante.

## TEST-8 – Impedir edição de tarefa inexistente ou com nome inválido

### Red

O teste foi criado para garantir que o sistema não permita editar tarefas inexistentes ou com nome vazio. Como não havia validações, o teste falhou inicialmente.

### Green

Foram implementadas verificações para lançar erros quando a tarefa não existe ou o nome é inválido, permitindo que o teste passasse.

### Refactor

Depois, as validações foram integradas ao fluxo principal de edição de maneira mais estruturada.

## TEST-9 – Excluir tarefa existente

### Red

O teste foi escrito para validar se uma tarefa pode ser excluída corretamente. Sem o método de exclusão, o teste falhou.

### Green

Foi criado um método simples para remover a tarefa pelo ID, o que fez o teste passar.

### Refactor

Em seguida, a lógica foi alinhada com os padrões internos da classe, mantendo consistência.

## TEST-10 – Excluir tarefa inexistente

### Red

O teste foi escrito para garantir que, ao tentar excluir uma tarefa inexistente, um erro apropriado seja gerado. Sem tratamento, o teste falhou inicialmente.

### Green

Foi adicionada uma verificação mínima para lançar erro quando a tarefa não existir, permitindo que o teste passasse.

### Refactor

Depois, a verificação foi mantida de forma padronizada com os demais métodos da classe.

# 5. Resultados Finais

Ao final do desenvolvimento, todos os 10 testes foram implementados e executados com sucesso. A execução final apresentou a mensagem 'Ran 10 tests in ... OK', indicando que todos os requisitos funcionais foram satisfeitos e validados pelos testes automatizados.

# 6. Conclusão

A utilização do TDD possibilitou um desenvolvimento incremental e seguro, com testes guiando cada funcionalidade implementada. O ciclo Red → Green → Refactor assegurou que o código fosse desenvolvido de forma organizada, garantindo clareza e correção desde as primeiras etapas. Todos os requisitos foram implementados e validados com sucesso.