EECP0003 - ENGENHARIA DE SOFTWARE

Relatório do Desenvolvimento de Software
CRUD

Alunos
Emanuel Rodrigues Valentim da Silva
Engenharia da Computação

João Batista Santos Silva Neto
Engenharia da Computação

Orientador
Davi Viana dos Santos
Coordenação do Curso de Engenharia da Computação

12 de junho de 2023

Sumário

1 Introdução

O Test-Driven Development (TDD), também conhecido como desenvolvimento orientado por testes, é uma abordagem de desenvolvimento de software que prioriza a criação de testes automatizados como ponto central do processo de desenvolvimento. Nessa metodologia, os testes são elaborados antes mesmo da implementação do código de produção, o que contribui para elevar a qualidade do software desde suas fases iniciais.

O TDD transcende a mera técnica de teste e configura-se como uma filosofia de desenvolvimento que proporciona uma série de benefícios e vantagens consideráveis para as equipes de desenvolvimento de software. Ao adotar o TDD, tais equipes podem aprimorar a qualidade do código, aumentar a produtividade, reduzir a ocorrência de bugs e facilitar a manutenção e evolução do sistema.

Essa prática tem sido amplamente empregada em diversas áreas de desenvolvimento de software, resultando em relevantes aplicações e avanços. O TDD desempenhou um papel fundamental no impulsionamento do desenvolvimento de frameworks e bibliotecas de testes automatizados, tais como JUnit (Java), NUnit (.NET), pytest (Python) e Jasmine (JavaScript). Essas ferramentas tornaram-se elementos fundamentais para a execução de testes automatizados e contribuíram para a disseminação do TDD em diferentes linguagens de programação. Além disso, o TDD configura-se como uma prática central nas metodologias ágeis, como Scrum e XP (Extreme Programming), as quais priorizam a entrega incremental, a colaboração e a qualidade do software. No contexto ágil, o TDD desempenha um papel essencial ao fornecer feedback rápido e permitir que as equipes se adaptem prontamente a mudanças nos requisitos do software.

O TDD está intimamente relacionado à prática de Integração Contínua (CI) e ao movimento DevOps. Com a adoção do TDD, os testes automatizados são executados continuamente como parte do processo de Integração Contínua, auxiliando na detecção ágil de problemas e garantindo a estabilidade do software. Além disso, o TDD é compatível com as práticas de entrega contínua e implantação contínua, que são pilares do DevOps. Vale ressaltar que o TDD não se limita apenas ao código de software, mas também pode ser aplicado à infraestrutura. A ideia do Test-Driven Infrastructure (TDI) consiste em escrever testes automatizados para validar a infraestrutura, tais como scripts de configuração, implantação e orquestração. Essa abordagem contribui para assegurar que a infraestrutura seja confiável e consistente em todo o seu ciclo de vida.

A aplicação do TDD (Desenvolvimento Orientado por Testes) ao desenvolvimento de um CRUD (Create, Read, Update, Delete) representa uma abordagem eficaz para assegurar a qualidade e a funcionalidade do sistema. O TDD enfatiza a criação de testes automatizados antes da implementação do código de produção, e essa abordagem se adequa perfeitamente ao desenvolvimento de um CRUD. Essa prática promove a garantia de funcionalidade, uma vez que a elaboração dos testes automatizados antes da implementação das funcionalidades do CRUD oferece a certeza de que as operações básicas (criação, leitura, atualização, exclusão) estão funcionando corretamente. Os testes auxiliam na definição do comportamento esperado do sistema e servem como um guia durante todo o processo de implementação. Além disso, o TDD possibilita uma refatoração segura, permitindo que ajustes sejam realizados com maior confiança. Uma vez que se possui uma suíte de testes automatizados que abrange todas as funcionalidades do CRUD, é possível efetuar alterações no código de produção e executar os testes para garantir que tudo continue funcionando conforme o esperado. Esse processo facilita a melhoria contínua do código, mantendo a funcionalidade intacta. O TDD incentiva uma abrangente cobertura de testes para todas as funcionalidades do CRUD, incluindo diferentes cenários, tais como criação de registros válidos e inválidos, leitura de registros existentes e inexistentes, atualização de registros com dados corretos e incorretos, e exclusão de registros. Possuir uma cobertura abrangente de testes auxilia na identificação e correção de problemas antes que eles afetem o funcionamento do sistema.

Outro aspecto crucial é o feedback rápido proporcionado pelos testes automatizados, os quais podem ser executados de forma ágil e repetitiva durante todo o processo de desenvolvimento. Isso

permite a detecção precoce de problemas e erros, facilitando sua depuração e correção imediata. Além disso, a contínua realização de testes nas funcionalidades do CRUD contribui para a redução da probabilidade de erros e problemas no código de produção, resultando em um sistema mais robusto e confiável. Os testes automatizados funcionam como uma documentação viva do sistema, descrevendo os casos de uso e os comportamentos esperados. Assim, eles se tornam uma fonte confiável de referência para desenvolvedores e equipes de controle de qualidade.

Ao adotar o TDD no desenvolvimento de um CRUD, é possível se beneficiar de uma abordagem orientada por testes, que resulta em um código mais testável, modular e de alta qualidade. Isso contribui para um processo de desenvolvimento mais eficiente, com menor incidência de erros e retrabalho, e culmina em um sistema final que atende às expectativas de funcionamento e qualidade.

A aplicação do TDD ao desenvolvimento de um CRUD com fins acadêmicos de aprendizado é extremamente valiosa. Essa abordagem permite aos estudantes familiarizarem-se com a estrutura e a lógica de um CRUD, além de praticarem a escrita de testes automatizados para validar o correto funcionamento das operações. O TDD auxilia na consolidação dos conceitos de desenvolvimento orientado por testes, fomentando a disciplina na elaboração de testes antes da implementação. Isso resulta em uma aprendizagem sólida, onde os estudantes adquirem um entendimento mais profundo das operações do CRUD, assim como das boas práticas de teste e desenvolvimento de software. A aplicação dessas tecnologias no contexto acadêmico proporciona aos estudantes uma base sólida para suas futuras carreiras no desenvolvimento de software, ao mesmo tempo em que ressalta a importância da qualidade e da validação contínua do código.

2 Objetivos

2.1 Objetivo Geral

Implementar um sistema CRUD (Create, Read, Update, Delete) utilizando a metodologia TDD para garantir a qualidade e a funcionalidade do sistema.

2.2 Objetivos Específicos

- Definir as entidades do sistema, identificando os atributos necessários e suas relações;
- Escrever testes automatizados para cada operação CRUD (criação, leitura, atualização, exclusão) das entidades;
- Implementar as funcionalidades básicas do CRUD, seguindo os testes escritos previamente;
- Verificar se os testes automatizados estão passando corretamente após a implementação de cada funcionalidade;
- Realizar refatorações no código, se necessário, mantendo a cobertura dos testes
- Garantir que todas as funcionalidades do CRUD estejam devidamente testadas e funcionando corretamente;
- Integrar o sistema com o repositório de dados, como um banco de dados, e garantir a persistência das informações;
- Realizar testes de integração para verificar o correto funcionamento do sistema como um todo;

- Assegurar que a documentação do sistema esteja atualizada, incluindo a descrição das funcionalidades do CRUD e como executar os testes automatizados;
- Realizar revisões e testes de validação para garantir a qualidade e a adequação do sistema em relação aos requisitos definidos;

3 Materiais e Métodos

3.1 Materiais

Nesta seção, serão listados os materiais necessários para a implementação do projeto CRUD, seguindo a metodologia TDD. A utilização destes materiais proporcionou um ambiente adequado para o desenvolvimento, teste e versionamento do código fonte.

3.1.1 Ambiente de Desenvolvimento

- Linguagem de Programação: Python (versão 3.11);
- Editor de Código: Visual Studio Code (versão 1.79)
- Controle de Versão: Git

3.1.2 Bibliotecas e Dependências

- Biblioteca de Testes: unittest
- Biblioteca para Simulação de API: fastAPI

3.1.3 Gerenciamento de Dependências

- Gerenciador de Pacotes: pip
- Arquivo de Requisitos: requirements.txt

3.1.4 Repositório de Códigos

- Plataforma de Hospedagem: GitHub

3.2 Metodologia

Nesta seção, será descrita a metodologia adotada para a implementação de um CRUD empregando a abordagem de Desenvolvimento Orientado por Testes (TDD) e a utilização do GitHub como plataforma de versionamento do código fonte.

3.2.1 Desenvolvimento Orientado por Testes (TDD)

O Desenvolvimento Orientado por Testes (TDD) é uma metodologia que enfatiza a escrita de testes automatizados antes da implementação do código de produção. O objetivo é garantir a qualidade e a funcionalidade do sistema, bem como promover um processo de desenvolvimento mais eficiente e livre de erros.

A abordagem TDD aplicada ao desenvolvimento de um CRUD consiste nos seguintes passos:

1. Definição dos Requisitos: Inicia-se o processo identificando e definindo os requisitos do sistema CRUD. Isso envolve determinar as operações de criação, leitura, atualização e exclusão que o sistema deve suportar;
2. Escrita dos Testes: Com base nos requisitos definidos, são escritos testes automatizados que verificam o comportamento esperado das funcionalidades do CRUD. Cada teste é projetado para validar uma operação específica e é implementado utilizando uma biblioteca de testes, como a unittest do Python;
3. Execução dos Testes Iniciais: Antes de implementar o código de produção, os testes iniciais são executados. Nesta etapa, espera-se que todos os testes falhem, uma vez que o código ainda não foi implementado. Isso serve para garantir que os testes estão corretamente configurados e refletem os requisitos definidos;
4. Implementação do Código de Produção: Com os testes escritos, inicia-se a implementação das funcionalidades do CRUD. O código de produção é desenvolvido para satisfazer os requisitos e fazer com que os testes sejam aprovados. É importante seguir as práticas de desenvolvimento, como a utilização de boas práticas de codificação e a criação de funções modulares e reutilizáveis;
5. Execução dos Testes de Validação: Após a implementação do código de produção, os testes são executados novamente. Agora, espera-se que todos os testes sejam aprovados, demonstrando que as funcionalidades do CRUD estão corretamente implementadas. Caso algum teste falhe, é necessário revisar e corrigir o código para garantir a conformidade com os requisitos;
6. Refatoração do Código: Após a aprovação dos testes, é realizada a etapa de refatoração do código. O objetivo é melhorar a estrutura, a legibilidade e a eficiência do código, removendo duplicações e otimizando o desempenho. A refatoração deve ser feita com base nos testes existentes para garantir que as alterações não afetem o funcionamento correto do sistema;
7. Execução dos Testes de Regressão: Com as refatorações implementadas, os testes são executados novamente para garantir que as alterações não introduziram problemas nas funcionalidades.

Estes passos, tal qual como foram executados serão explorados na Seção 4.

4 Procedimento Experimental (Implementação)

Nesta seção, serão apresentados os procedimentos experimentais realizados para a implementação do CRUD utilizando a metodologia TDD. Os procedimentos incluem a definição dos requisitos do sistema, a elaboração de um diagrama de casos de uso e a execução dos passos do TDD.

4.1 Definição dos Requisitos

Os requisitos do sistema CRUD foram previamente definidos pelo professor (cliente) e são categorizados em requisitos funcionais. Esses requisitos fornecem as funcionalidades essenciais do sistema e orientam o desenvolvimento do projeto. A seguir estão os requisitos funcionais definidos:

- RF1 - Adicionar produto

  Este requisito tem como objetivo permitir que o usuário adicione um novo produto ao sistema CRUD. O usuário deve fornecer as informações necessárias sobre o produto, como nome, descrição, preço, etc. O sistema deve validar e armazenar os dados do produto adequadamente.

- RF2 - Deletar produto

  Este requisito permite que o usuário delete um produto existente do sistema CRUD. O usuário deve fornecer a identificação única do produto a ser excluído. O sistema deve confirmar a exclusão e remover o produto do banco de dados.

- RF3 - Fazer update do produto

  Este requisito possibilita ao usuário atualizar as informações de um produto existente no sistema CRUD. O usuário deve fornecer a identificação única do produto a ser atualizado e as informações atualizadas. O sistema deve validar as informações e atualizar os dados do produto no banco de dados.

- RF4 - Listar todos os produtos Este requisito permite que o usuário visualize uma lista de todos os produtos cadastrados no sistema CRUD. O sistema deve exibir as informações relevantes de cada produto, como nome, descrição e preço, de forma organizada e legível para o usuário.

- RF5 - Consultar um produto

  Este requisito possibilita ao usuário realizar uma consulta específica sobre um produto do sistema CRUD. O usuário deve fornecer a identificação única do produto a ser consultado. O sistema deve buscar as informações correspondentes ao produto e apresentá-las ao usuário de forma clara e precisa.

Esses requisitos funcionais definem as principais operações do sistema CRUD e são essenciais para atender às necessidades da aplicação. Durante o desenvolvimento, os testes automatizados serão elaborados com base nesses requisitos, garantindo a conformidade e a qualidade do sistema.

4.2 Descrição dos Casos de Uso

A descrição dos casos de uso permite compreender as interações entre os usuários e o sistema CRUD. Ele descreve as funcionalidades que cada tipo de usuário pode acessar e como essas funcionalidades se relacionam fornecendo uma visão geral do sistema, ajudando na compreensão dos requisitos e na identificação das operações necessárias. Os casos de uso e suas descrições podem ser observadas abaixo.

4.2.1 Adicionar Produto

Este caso de uso permite ao usuário cadastrar um novo produto no sistema.

- **Atores:** Usuário: Interage com o sistema para adicionar o produto.
- **Fluxo Principal:**
    1. O usuário solicita adicionar um novo produto ao sistema
    2. Sistema abre a tela de cadastro
    3. O Usuário informa id, nome e valor do produto a ser cadastrado
    4. O sistema verifica as informações adicionadas
    5. O sistema retorna a mensagem ”Adicionado com sucesso”
    6. O caso de uso termina
- **Fluxos Alternativos:**
    1. ID já existente: o ID fornecido ja existe no sistema. O sistema emite a mensagem ”Produto com esse ID já existe, tente novamente”
    2. ID não é inserido: O usuário não fornece o ID do produto. O sistema emite uma mensagem ”Produto sem ID, tente novamente”.

4.2.2 Deletar Produto

Este caso de uso permite ao usuário apagar um produto do sistema.

- **Atores:** Usuário: Interage com o sistema para adicionar o produto.
- **Fluxo Principal:**
    1. O usuário solicita apagar novo produto do sistema
    2. Sistema abre a tela para apagar um produto
    3. O Usuário informa ID do produto a ser apagado
    4. O sistema verifica a informação adicionada
    5. O sistema retorna a mensagem ”Produto removido com sucesso”
    6. O caso de uso termina
- **Fluxos Alternativos:**
    1. ID não existente: o ID fornecido não consta no sistema. O sistema emite a mensagem ”Produto com esse ID não encontrado, tente novamente”.
    2. ID não é inserido: O usuário não fornece o ID do produto. O sistema emite uma mensagem ”ID não foi inserido, tente novamente”.

4.2.3 Deletar Produto

Este caso de uso permite ao usuário apagar um produto do sistema.

4.2.4 Atualizar Produto

Este caso de uso permite ao usuário atualizar as informações de um produto existente no sistema.

- **Atores:** Usuário
- **Fluxo Principal:**
  1. O usuário solicita atualizar um produto no sistema
  2. Sistema abre a tela de atualização
  3. O Usuário informa ID do produto a ser apagado
  4. O sistema verifica a informação adicionada e solicita as novas informações
  5. O usuário informa o novo nome e novo preço
  6. O sistema verifica a informação adicionada
  7. O sistema retorna a mensagem ”Produto atualizado com sucesso”
  8. O caso de uso termina
- **Fluxos Alternativos:**
  1. ID não existente: o ID fornecido não consta no sistema. O sistema emite a mensagem ”Produto com esse ID não encontrado, tente novamente”.
  2. Nome não é inserido: O usuário não fornece o nome do produto. O sistema emite uma mensagem ”Nome não foi inserido, tente novamente”.

4.2.5 Listar todos os Produtos

Este caso de uso permite ao usuário receber uma lista com informações de todos os produtos existente no sistema.

- **Atores:** Usuário
- **Fluxo Principal:**
  1. O usuário solicita para que o sistema liste os produtos
  2. Sistema retorna uma tabela com todos os produtos e suas informações
  3. O caso de uso termina
- **Fluxos Alternativos:**
  1. Náo há produto no sistema: Nenhum produro foi cadastrado no sistema. O sistema emite a mensagem ”Nenhum produto adicionado a lista”.

4.2.6 Consultar Produto

Este caso de uso permite ao usuário consultar as informações de um produto existente no sistema.

- **Atores:** Usuário

- **Fluxo Principal:**
    1. O usuário solicita para consultar um produto
    2. Sistema abre a tela de consulta
    3. O Usuário informa ID do produto a ser consultado
    4. O sistema verifica a informação adicionada
    5. O sistema retorna a mensagem uma tabela com as informações do produto
    6. O caso de uso termina
- **Fluxos Alternativos:**
    1. ID não existente: o ID fornecido não consta no sistema. O sistema emite a mensagem ”Produto com esse ID não encontrado, tente novamente”.
    2. ID não é inserido: O usuário não fornece o ID do produto. O sistema emite uma mensagem ”ID não inserido, tente novamente”.

4.3 Passos do Test Driven Development (TDD)

O TDD foi utilizado como abordagem para o desenvolvimento do CRUD, garantindo a qualidade e a funcionalidade do sistema. Os passos do TDD realizados foram os seguintes:

1. **Definição dos Requisitos:** Os requisitos funcionais do sistema foram previamente definidos e consistem nos seguintes itens:
    - RF1 – Adicionar produto
    - RF2 – Deletar produto
    - RF3 – Fazer update do produto
    - RF4 – Listar todos os produtos
    - RF5 – Consultar um produto
2. **Criação dos Testes:** Com base nos requisitos definidos, foram escritos testes automatizados utilizando a biblioteca unittest da linguagem Python. Para cada funcionalidade do CRUD, foram criados testes específicos que validam o comportamento esperado. Os testes foram estruturados em classes ou funções separadas, seguindo o princípio de teste por unidade. Os nomes dos arquivos podem ser visualizados no repositório com os seguintes nomes:
    - RF1 – teste_AdicionarProduto.py
    - RF2 – teste_DeletarProduto.py
    - RF3 – teste_AtualizarProduto.py
    - RF4 – teste_ListarProdutos.py
    - RF5 – teste_ConsultarProduto.py
3. **Execução dos Testes Iniciais:** Os testes foram executados pela primeira vez, e como ainda não havia implementação para as funcionalidades do CRUD, todos os testes falharam corretamente, indicando a ausência de código funcional.
4. **Implementação Mínima:** Foi realizada uma implementação mínima das funcionalidades do CRUD, abordando apenas os requisitos mais básicos. Além disso, a biblioteca FastAPI foi utilizada para simular um mock de banco de dados, permitindo o armazenamento temporário dos dados em memória. A implementação inicial contemplou a criação das rotas e dos métodos necessários para realizar as operações de adicionar, deletar, atualizar, listar e consultar produtos.

- Produto.py
- RF1 – AdicionarProduto.py
- RF2 – DeletarProduto.py
- RF3 – AtualizarProduto.py
- RF4 – ListarProdutos.py
- RF5 – ConsultarProduto.py

5. **Execução dos Testes:** Após a implementação mínima, os testes foram executados novamente. Esperava-se que a implementação passasse nos testes mais básicos, validando as funcionalidades mais simples do CRUD, enquanto ainda falhava nos testes mais complexos.

6. **Refatoração:** Após a execução dos testes, foi realizada a refatoração do código. Nesse processo, o código foi otimizado e reestruturado, levando em consideração as boas práticas de programação. Foram aplicadas melhorias para aumentar a legibilidade, a manutenibilidade e a qualidade do código. Assim todas as funcionaldades são unidas em uma classe sistema cujo arquivo pode ser visualizado em Sistema.py no repositório.

7. **Execução dos Testes Finais:** Após a refatoração, todos os testes foram executados novamente para garantir que a implementação e as mudanças realizadas durante a refatoração não tenham introduzido novos erros ou quebrado funcionalidades existentes. Esperava-se que todos os testes fossem aprovados, validando o correto funcionamento do CRUD.

8. **Histórico de Commit no GitHub:** Durante todo o processo de desenvolvimento, foram realizados commits no repositório do GitHub para registrar todas as alterações feitas no código-fonte. O histórico de commit permitiu acompanhar todas as etapas do desenvolvimento, fornecendo um registro detalhado das modificações realizadas ao longo do tempo. Essa abordagem facilitou a colaboração e a rastreabilidade do desenvolvimento do CRUD.

5 Resultados

Nesta seção, apresentaremos os resultados obtidos ao aplicar a metodologia do Test-Driven Development (TDD) para a implementação do CRUD, juntamente com a utilização da biblioteca FastAPI para simular um mock de banco de dados.

O CRUD foi implementado com sucesso, incluindo as funcionalidades de adicionar, deletar, atualizar, listar e consultar produtos. Utilizando o TDD, garantimos que todas as funcionalidades atendessem aos requisitos previamente definidos pelo cliente. Foram escritos testes automatizados para cada funcionalidade do CRUD, abrangendo os requisitos funcionais estabelecidos. Esses testes foram executados durante todo o processo de desenvolvimento, permitindo uma validação contínua do código implementado. Todos os testes foram aprovados, indicando que as funcionalidades do CRUD estavam funcionando corretamente.

Todo o progresso do desenvolvimento foi registrado no repositório do GitHub através de commits. Isso permitiu acompanhar e revisar todas as alterações feitas no código-fonte ao longo do tempo. O histórico de commit serviu como uma forma de documentação do processo de desenvolvimento e facilitou a colaboração entre os membros da equipe. o repositório pode ser visualizado em `https://github.com/batista-neto/TDD_ES/tree/main`

6 Dificuldades Encontradas

Durante o desenvolvimento do projeto, foram encontradas algumas dificuldades relacionadas ao uso do repositório e das APIs. Uma das principais dificuldades encontradas foi lidar com a configuração inicial do repositório no GitHub. Embora a plataforma seja amplamente utilizada e forneça uma série de recursos, a configuração inicial, como a criação do repositório, a definição de permissões e a configuração de branches, pode ser um processo complexo para aqueles que estão iniciando no uso da plataforma.

Além disso, ao trabalhar com APIs, a equipe enfrentou o desafio de entender a estrutura e a documentação adequada. Em alguns casos, a documentação pode ser escassa ou pouco clara, dificultando a compreensão dos endpoints, parâmetros necessários e formatos de dados esperados. Isso exigiu um esforço adicional de pesquisa e experimentação para garantir que estavam utilizando corretamente as APIs e obtendo os resultados esperados.

Outra dificuldade encontrada foi a integração entre as diferentes tecnologias e ferramentas utilizadas no projeto. Embora existam muitos recursos disponíveis para facilitar essa integração, como bibliotecas e frameworks, a configuração inicial e a comunicação adequada entre as partes podem ser desafiadoras. Isso exigiu um bom entendimento de cada tecnologia envolvida e a capacidade de solucionar problemas e depurar possíveis erros.

Além disso, o trabalho com repositório e APIs também pode ser impactado por limitações de acesso ou de taxa de solicitação. Em algumas situações, pode ser necessário lidar com autenticação, chaves de acesso e cotas de uso, o que requer um planejamento cuidadoso e consideração das restrições impostas pelos serviços utilizados.

Apesar dessas dificuldades, a equipe foi capaz de superá-las com pesquisa, trabalho em equipe e perseverança. Com o tempo, foi possível compreender melhor o funcionamento do repositório e das APIs, otimizando suas abordagens e encontrando soluções para os desafios encontrados. Essas dificuldades também proporcionaram uma valiosa experiência de aprendizado, aprimorando suas habilidades de resolução de problemas e adaptação a diferentes tecnologias e ferramentas.

[1][2][4][5][3]

7 Conclusão

Durante o procedimento experimental, a equipe adquiriu valiosas aprendizagens. A aplicação do Test-Driven Development (TDD) permitiu-lhes compreender de forma significativa a importância de escrever testes automatizados antes da implementação do código, resultando em um código mais robusto e confiável. A utilização da biblioteca FastAPI para simular um mock de banco de dados proporcionou-lhes uma experiência enriquecedora, permitindo a criação eficiente de rotas e métodos para o CRUD. Além disso, a utilização do GitHub como plataforma de colaboração ensinou-lhes a trabalhar de forma colaborativa, realizando commits, revisões de código e resolução de conflitos de maneira adequada. A execução contínua dos testes automatizados também mostrou-lhes a importância vital da validação contínua do código, identificando e corrigindo problemas de forma ágil. Essas aprendizagens forneceram-lhes uma base sólida para o desenvolvimento de software, enfatizando a qualidade, a colaboração e a melhoria contínua do código, aspectos cruciais para um desenvolvimento de software bem-sucedido.

Referências Bibliográficas

[1] Doyle, W. Tdd full course (learn test driven development with python). https://www.youtube.com/watch?v=eAPmXQ0dC7Q, Dec. 2020.

[2] em Foco, W. Crud com python. https://www.youtube.com/watch?v=-vrXnewHrwA&list=PLbnAsJ6zlidvszSXnxplfYgtB6KQ-fZ-N, Dec. 2020.

[3] GitHub. Github documentation, 2021.

[4] Python Software Foundation. *Python Documentation*. Python Software Foundation, 2021.

[5] Python Software Foundation. unittest – unit testing framework, 2021.