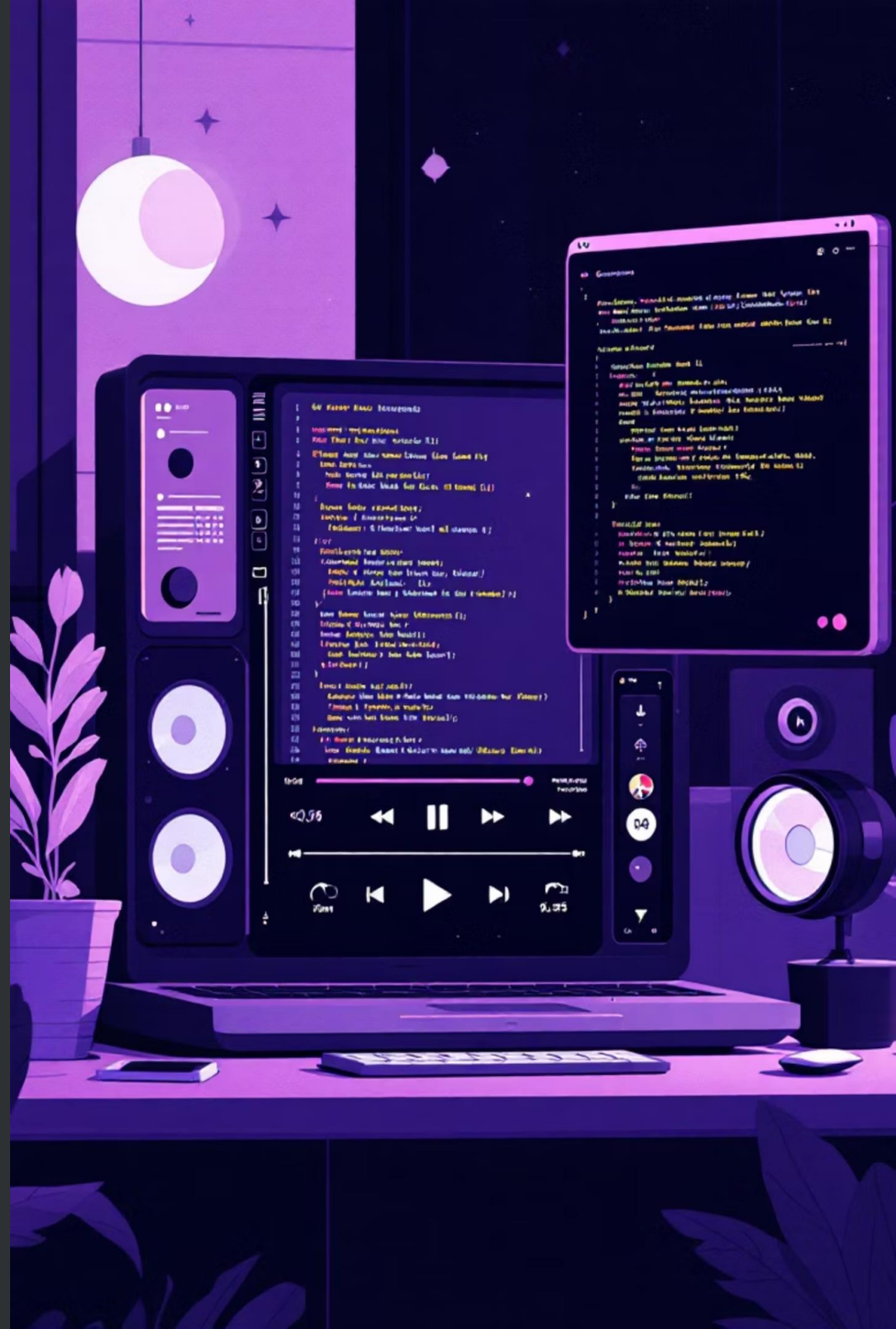
Gestão e Qualidade de Software: Refatorando um Music Player

Este trabalho documenta o processo de análise crítica, refatoração arquitetural e implementação de testes unitários em um projeto de Music Player em Python. Nosso foco foi elevar a qualidade do código e a manutenibilidade do sistema, aplicando princípios de Clean Code e isolamento de dependência.



Equipe

Este projeto foi desenvolvido por uma equipe dedicada de estudantes, combinando esforços e conhecimentos para alcançar um resultado de alta qualidade:

- Angelo Rodrigues 824139676
- Barbara Tracanella 824124152
- Erick Domingues Soares 82414486
- Eduardo Baptistella Gonçalves 824147595
- Gabriel Prieto Lima 824142064
- Wellington de Oliveira Sousa 825240209

Nosso objetivo comum foi transformar um código funcional, mas com falhas, em um sistema robusto e de fácil manutenção.

Desafios do Código Original: Débito Técnico Elevado

A versão inicial do Music Player, embora funcional, apresentava sérias deficiências que comprometiam sua sustentabilidade a longo prazo. Analisamos o código e identificamos problemas críticos que geraram um **alto débito técnico**.

Fragilidade Arquitetural

A classe principal violava o Princípio da Responsabilidade Única (SRP), resultando em alto acoplamento entre a Interface de Usuário (UI) e dependências de hardware. Isso impossibilitava testes unitários isolados, evidenciando um design incorreto.

Baixa Transparência

O código era desprovido de comentários estratégicos e docstrings, o que aumentava o custo cognitivo para qualquer desenvolvedor que precisasse entender sua lógica. A falta de um README.md completo dificultava a colaboração e a adoção do projeto.

Essas falhas tornavam o software inviável para manutenção e evolução, exigindo uma reforma emergencial.

Nossa Estratégia de Refatoração: Rumo à Qualidade

Para reverter o cenário de débito técnico, aplicamos princípios de Clean Code e Engenharia de Software. Nossa estratégia focou em:

01

Isolamento de Dependência (DIP)

Utilizamos mocking e patching para substituir dependências gráficas e de hardware. Isso permitiu a criação de instâncias da UI em ambientes de teste controlados e viabilizou a testabilidade.

02

Modularidade e SRP

Dividimos o sistema em módulos autônomos, como src/song_controller.py (lógica de negócios e áudio) e src/ui.py (apresentação e orquestração). Essa separação garante baixo acoplamento e facilita a manutenção.

03

Adição de Documentação

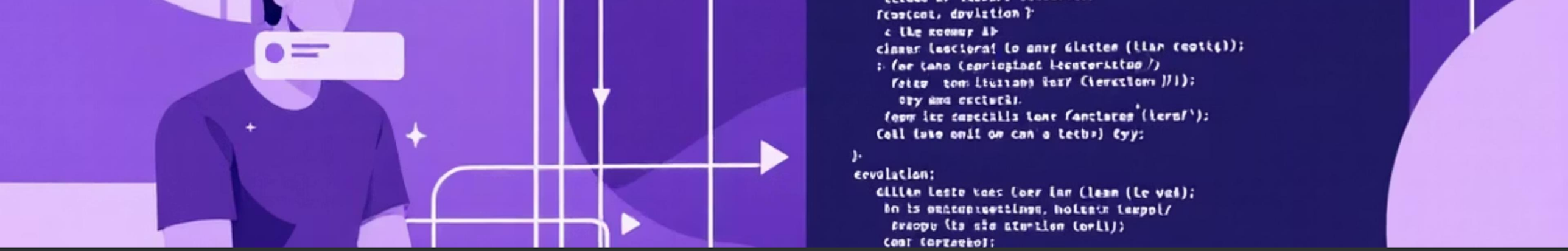
Incluímos docstrings completas para classes e métodos, além de comentários estratégicos. Isso tornou o código autoexplicativo, elevando a legibilidade e reduzindo o custo cognitivo para novos desenvolvedores.

04

Implementação de Testes Unitários

Criamos uma suíte robusta de testes unitários para a lógica de negócios e a interface de usuário, comprovando a testabilidade do código refatorado e garantindo a validação contínua.

Essa abordagem estratégica permitiu resgatar o software e garantir sua longevidade.



A Importância do Clean Code na Manutenção de Software

A refatoração do Music Player demonstrou o valor inestimável do **Clean Code** para a manutenibilidade de software. Um código limpo não é apenas esteticamente agradável; é uma fundação para a sustentabilidade e evolução de qualquer projeto.

Redução de Débito Técnico: Evita custos futuros e retrabalho.

Facilidade de Manutenção: Permite correções e atualizações rápidas.

Colaboração Aprimorada: Facilita o entendimento entre equipes.

Testabilidade Inerente: Torna o software mais robusto e confiável.

Investir em Clean Code desde o início, ou aplicá-lo através de refatorações, é crucial para a qualidade e o sucesso a longo prazo de qualquer sistema de software.