Prática de Teste em OSS 1 – Estratégia de Testes

FGA 0238 - Testes de Software

Turma	Turma 01	**Semestre**	2024.1

Equipe	Iceberg

Nome	**Matrícula**
Arthur Ribeiro e Sousa	221007850
Caetano Santos Lucio	180144979
Felipe de Sousa Coelho	211061707
Henrique Azevedo Batalha	211061850
Lucas Henrique de Lima Queiroz	190091703
Luis Eduardo Carneiro Miranda	211063200

1 Projeto

O projeto em qual a equipe irá contribuir se chama Dotenv-Linter, um projeto em que se pode verificar, corrigir ou comparar arquivos **.env** em busca de problemas que possam causar mau funcionamento de um aplicativo.

Link para o repositório: https://github.com/dotenv-linter/dotenv-linter

Link para a documentação do projeto: https://dotenv-linter.github.io

2 Tipos e Níveis de Teste

O projeto usa somente testes unitários, de forma que não é aplicada a pirâmide de testes. Sobre os testes não funcionais, é realizado um benchmark (teste de performance) com outro linter de arquivos .env.

3 Automação de Testes

Os tipos de testes que estão automatizados dentro do projeto são testes funcionais e testes não funcionais, e ambos estão em níveis de testes unitários (que no caso é o único tipo de nível no projeto)

4 Ciclo de Vida das Issues

Em geral as *issues* são abertas, depois se espera que alguém interessado se atribua a responsabilidade e a issue é fechada depois de um pull request aceito ou commit contendo a feature/fix que corresponda a issue.
Em alguns pull requests existe uma automatização que mede a diferença de cobertura dos testes e em pull requests de releases novas é rodado um teste de comparação de benchmark.

5 Papeis e Responsabilidades

O projeto é Open Source. Portanto, todos podem contribuir. No entanto, o projeto faz parte de um projeto maior, cujo foco é o de aprender Rust. Esse superprojeto tem diversos mentores, além de outros projetos, que podem ser encontrados em : https://rustbeginners.github.io/awesome-rust-mentors/.

6 Ferramentas de Teste

O benchmark é realizado com a ferramenta Hyperfine (https://github.com/sharkdp/hyperfine). Os testes unitários são realizados com a ferramenta padrão do Rust (https://doc.rust-lang.org/rust-by-example/testing/unit_testing.html).

7 Métricas de Teste

As métricas de testes atualmente previstas na documentação do projeto, são fornecidas pelas ferramentas de teste acima mencionadas, e que as preveem as métricas da quantidade de testes unitários executados, passados, com aviso e não passados, além da velocidade de comparação com outra ferramenta de análise de variáveis do ambiente escrita em Python.

Para além das métricas encontradas de forma explícita na documentação, outras métricas foram observadas e podem ter uma relevância nas tomadas de decisão da organização como um projeto e como utilitário para outros desenvolvedores:

Testes Unitários	T
Quantidade de testes realizados com sucesso	T01
Quantidade de testes com aviso	T02
Quantidade de testes com falha	T03

Testes de Serviço	T
Resultados do Github Actions	T10
Resultados do Hyperfine	T11
Número de variáveis de ambiente analisados no programa	T12
Velocidade de execução	T13

Testes de Integração	T
Comparação da quantidade dos resultados dos testes unitários com versões anteriores do programa	T20
Comparação da quantidade dos resultados dos testes unitários com outros programas que executam a mesma função	T21

Testes não funcionais	NF
Legibilidade do arquivo .env	NF01
Compreensão, por parte dos utilizadores, do texto que descreve a otimização necessária para o arquivo .env	NF02

As tabelas anteriores fornecem uma percepção das funções provavelmente precisam de atenção para cada iteração ou modificação do código, pois leva em conta tanto métricas para os desenvolvedores, quanto para os usuários finais da ferramenta.

8 Necessidade de Testes

Por se tratar de um linter para .env files, o componente que demanda maior cobertura de testes é o validador de sintaxe, para dar garantia que o .env file seja interpretado corretamente, garantindo a eficiência do software.

9 Análise da Estratégia e Situação dos Testes do Projeto

Como foi mencionado anteriormente, o projeto possui apenas testes unitários e não existe uma documentação para testes, o que torna a criação e manutenção dos testes mais difícil. Isso pode ser especialmente prejudicial quando novas atualizações forem lançadas e os testes precisarem de ser atualizados ou novas funções precisam ser testadas.

10 Recomendações

Uma possível melhoria na cobertura de testes do *linter* é criar testes de desempenho que verificam o desempenho do software lidando com arquivos **.env** de tamanhos diferentes. Os testes podem ser implementados para verificar algumas métricas como tempo de execução e uso de recursos. Essa melhoria pode ajudar a detectar limitações no software.