UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
FACULDADE DE CIÊNCIAS – CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LEONARDO SCARMATO JORGE DE PAULA

MINERAÇÃO DE REPOSITÓRIOS PARA AVALIAR A INFLUÊNCIA DAS MUDANÇAS DE CÓDIGO AO LONGO DO TEMPO

BAURU
2024

LEONARDO SCARMATO JORGE DE PAULA

MINERAÇÃO DE REPOSITÓRIOS PARA AVALIAR A INFLÛENCIA DAS MUDANÇAS DE CÓDIGO AO LONGO DO TEMPO

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru

Orientador:
Prof. Dr. Higor Amario de Souza

BAURU
2024

De Paula, Leonardo Scarmato Jorge.
Mineração de Repositórios Para Avaliar A Influência das Mudanças de Código ao Longo do Tempo / Leonardo Scarmato Jorge de Paula. – Bauru, 2024
51 f. : il.

Trabalho de conclusão de curso (Bacharelado-Ciência da Computação) – Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru
Orientador: Higor Amario de Souza

1. Mineração de repositórios. 2. Bibliotecas. 3. Frameworks. 4. Análise I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru.

Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada

LEONARDO SCARMATO JORGE DE PAULA

MINERAÇÃO DE REPOSITÓRIOS PARA AVALIAR A INFLUÊNCIA DAS MUDANÇAS DE CÓDIGO AO LONGO DO TEMPO

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru

BANCA EXAMINADORA

Prof. Dr. Higor Amario de Souza – Orientador(a)
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profa. Dra. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Prof. André Luis Debiaso Rossi
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências

Bauru, 13 de Novembro de 2024.

AGRADECIMENTOS

Agradeço, primeiramente à minha família por me prestar condições financeiras e psicológicas para realizar este trabalho e graduação, especialmente à minha mãe, Luciana, e ao meu pai, Roberto, que sempre estiveram do meu lado com apoio, zelo e amor, minhas maiores inspirações. Por fim, agradeço também aos meus amigos Giuliano, Gabriel e Gustavo, que fizeram parte da minha jornada acadêmica e sempre se mostraram presentes.

“A humildade exprime uma das raras certezas de que estou certo: a de que ninguém é superior a ninguém”

Paulo Freire

RESUMO

Sempre em constante evolução, a tecnologia utiliza de métricas expostas em códigos fonte, tomando por base soluções e funções que facilitem o desenvolvimento e a manutenção do software. Tendo essa análise como base, surge a mineração de repositórios, uma técnica valiosa para a coleta de grandes quantidades de dados e informações a partir de um repositório. Possibilitando fundamentar decisões estratégicas, análises micro que aceleram a manutenção e diminuem o retrabalho, ou análises macro, com perspectivas abrangentes que podem direcionar o software e seu desenvolvimento. Este trabalho analisou dez projetos Python amplamente utilizados, abrangendo bibliotecas e *frameworks* de diversas áreas, como aprendizado de máquina, processamento de imagens e desenvolvimento web. A análise centrou-se nas alterações realizadas em métodos e arquivos ao longo de três períodos, com o objetivo de identificar padrões de mudança e relacioná-los à necessidade de manutenção direcionada. Por meio de métricas quantitativas e comparativas, foi possível analisar qual porcentagem do código que muda ao longo do tempo, e quais são os projetos com maior ou menor concentração de alterações, além de fornecer subsídios para estudos futuros.

Palavras-Chave: Mineração de repositórios, bibliotecas, análises, qualidade, software, ciência de dados.

ABSTRACT

Constantly evolving, technology relies on metrics derived from source code, leveraging solutions and functions that facilitate software development and maintenance. Based on this analysis, software repository mining emerges as a valuable technique for collecting and interpreting large amounts of data and information from repositories. This approach enables strategic decision-making, micro-level analyses that accelerate maintenance and reduce rework, or macro-level perspectives that can guide software development and evolution. This study analyzed ten widely used Python projects, including libraries and *frameworks* from diverse fields such as machine learning, image processing, and web development. The analysis focused on changes made to methods and files over three distinct periods, aiming to identify change patterns and relate them to the need for targeted maintenance. Through quantitative and comparative metrics, the study assessed the percentage of code altered over time and identified projects with higher or lower concentrations of changes, providing insights for future research and development practices.

Keywords: Repository mining, libraries, analysis, quality, software, data science

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

É notória a importância da Ciência de Dados como uma parte essencial para a extração e análise de grandes quantidades de informações e dados, desempenhando um papel crucial em diversas áreas da tecnologia, incluindo a engenharia de software. Nesse contexto, a Mineração de Repositórios de Softwares (MRS) permite uma coleta eficaz de dados de um código fonte alinhado com o controle de versão, oferecendo *insights* valiosos e direcionamentos para o desenvolvedor traçar o caminho do software (BIRD et al., 2015).

Tendo em vista que alterações frequentes em componentes específicos podem gerar retrabalho e dificultar a manutenção de longo prazo, uma análise de alterações no código-fonte permite identificar áreas de instabilidade e facilitar o meio de promover melhorias direcionadas. Assim, compreender padrões de mudanças permite que equipes priorizem recursos, otimizem processos e direcionem esforços para as partes mais críticas do projeto, reduzindo falhas e custos operacionais. Nesse sentido, avaliar quais partes do código sofrem maior número de alterações ao longo do tempo contribui diretamente para a qualidade do projeto e para sua evolução contínua.

Dessa forma, este trabalho concentra-se na análise das alterações em métodos e arquivos de 10 projetos Python, selecionados por sua relevância no cenário atual. O objetivo é relacionar as alterações em componentes específicos do código à necessidade de manutenção, estabelecendo métricas para avaliar a estabilidade de cada projeto. A pesquisa foi realizada a partir de extrações diretamente dos *commits*, utilizando ferramentas de análise como o PyDriller e o Pandas para identificar alterações relevantes nos métodos e arquivos, e calcular o percentual de código alterado em períodos específicos.

Por fim, são disponibilizadas duas análises complementares, inicialmente uma análise percentual das mudanças em métodos e arquivos ao longo de 30, 60 e 120 dias, permitindo avaliar a dinâmica de manutenção em curto e médio prazo. Em seguida, uma análise comparativa identifica padrões de alterações entre os projetos, destacando diferenças e semelhanças que indicam áreas críticas ou estáveis do código. Essa abordagem visa fornecer subsídios para futuras análises e decisões de priorização de manutenção e testes direcionados.

Os resultados indicam que projetos como Scrapy, Pillow e Scikit-Learn apresentaram alterações concentradas em áreas de manutenção, enquanto TensorFlow e Django mantiveram o foco em mudanças mais gerais. A pesquisa limita-se a períodos de até 120 dias, mas estabelece uma base para estudos mais aprofundados e abrangentes, fornecendo insumos para otimizar estratégias de priorização de testes e manutenção.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, serão apresentados conceitos base sobre engenharia de software, controle de versão, mineração de repositórios de software, evolução de software e ciência de dados. Estes fundamentos serão relevantes para o entendimento do projeto.

2.1 Engenharia de Software

A Engenharia de Software é a disciplina responsável pela aplicação de princípios e práticas sistemáticas no desenvolvimento e manutenção de software. Seu objetivo é garantir que sistemas sejam construídos de maneira eficiente, confiável e de acordo com os requisitos dos usuários. Ela envolve desde a análise de requisitos e o design do sistema até a codificação, testes e manutenção contínua, buscando sempre melhorar a qualidade e reduzir custos ao longo do ciclo de vida do software.

Segundo Sommerville (2011), a Engenharia de Software se baseia em métodos e processos estruturados, que incluem o uso de métricas, técnicas de gestão de configuração e práticas de integração contínua. Tais práticas são essenciais para garantir que o software seja desenvolvido com qualidade e que problemas sejam identificados e corrigidos de forma eficiente, garantindo o sucesso do projeto.

2.2 Sistemas

Os sistemas de software modernos frequentemente utilizam *frameworks* **e** bibliotecas como elementos essenciais para acelerar o desenvolvimento, promover reutilização de código e padronizar funcionalidades. *Frameworks* fornecem uma estrutura robusta e predefinida para a criação de aplicações, enquanto bibliotecas oferecem funcionalidades específicas que podem ser integradas ao projeto de forma modular.

Segundo Pressman (2015), “um framework é um esqueleto de software que pode ser estendido para criar uma aplicação específica”. Desse modo, compõem parte crucial no desenvolvimento de um projeto, acelerando e integrando a tecnologia por completo, além de auxiliar a economizar o tempo de criação ou manutenção, proporcionando componentes prontos que atendem a funções específicas, como por exemplo, o gerenciamento de dados.

Por exemplo, *frameworks* web, como Django e Flask no ecossistema Python, facilitam a criação de aplicações baseadas em servidor, oferecem funcionalidades prontas que autenticam usuários com maior facilidade, não requisitando um desenvolvimento h*ard-coded* HTTP, assim como o FastAPI, que traz rapidez e simplicidade ao lidar com essas mesmas

requisições HTTP, mas de forma assíncrona. Já bibliotecas como Pandas, servem para a análise de dados e a extração de informações, fornecendo os *insights* ao desenvolvedor para que ele mapeie os próximos passos, assim como o Scikit-Learn, uma biblioteca robusta para machine learning que fornece medidas e algoritmos estatísticos. Seguindo com OpenCV, essencial para o desenvolvimento de uma visão completa computacional, como por exemplo, reconhecimento gráfico, assim como o Pillow, que oferece ferramentas de manipulação e edição de imagens. E por último o SQLAlchemy, principal facilitador de interação com bancos de dados sem necessitar um *hard coding*.

Em constante evolução, sistemas, bibliotecas e *frameworks* bem utilizados garantem longevidade e estabilidade, com análises frequentes nas mudanças, melhoria de padrões e correções se tornam fundamentais para um desenvolvimento coeso, além de possíveis prevenções de obsolescências.

2.3 Controle de versão

Controle de versão é um mecanismo que salva o registro histórico de todas as alterações feitas em um projeto, possibilita reversões e informações sobre os *commits*, além de possibilitar que equipes colaborativas com mais de um desenvolvedor trabalhem de forma simultânea, mantendo o código integrado e coerente.

2.3.1 Git

Git é uma ferramenta de controle de versão que permite que equipes acompanhem as modificações no código e colaborem de maneira organizada, garantindo o histórico de alterações e possibilitando a criação de ramificações para o desenvolvimento paralelo de novas funcionalidades. Facilmente integrado em diversos projetos atualmente.

2.4 Mineração de repositórios de software

A mineração de repositórios de software é uma técnica utilizada para obter informações que sejam úteis ao desenvolvimento a partir das informações armazenadas nos códigos.

Uma técnica que consiste em analisar uma grande quantidade de códigos-fonte contidos em um repositório, utilizando de algoritmos e métodos a fim de extrair informações valiosas sobre a evolução do projeto, identificar padrões, comportamentos e trazer *insights*, que ao fornecer uma visão abrangente do ciclo de vida do software podem aprimorar o desenvolvimento do projeto (HASSAN, 2008) e direcionar mudanças futuras.

2.5 Evolução de Software

A evolução de software é um processo contínuo e necessário para garantir que sistemas sejam adaptados às mudanças e necessidades dos usuários, à evolução tecnológica e principalmente à correção de falhas. A manutenção de software, que representa uma parte relevante dos custos de desenvolvimento, envolve a modificação do código após a finalização do sistema, com o objetivo de corrigir erros, adaptar o software a novos requisitos ou melhorar seu desempenho (Sommerville, 2011). Esse processo é essencial para prolongar a vida útil do software e assegurar que ele continue a atender aos objetivos para os quais foi criado.

Além disso, testes e depuração são atividades críticas no ciclo de vida do software. Os testes têm como objetivo identificar defeitos e garantir que o sistema funcione conforme esperado, enquanto a depuração envolve a análise detalhada do código para encontrar e corrigir erros específicos. Ambos são processos iterativos e essenciais para a qualidade do software, já que garantem que as modificações durante a evolução não introduzam novos problemas, mantendo a confiabilidade e estabilidade do sistema.

2.6 Ciência de Dados

A Ciência de Dados é definida como "a disciplina que combina métodos científicos, processos, algoritmos e sistemas para extrair conhecimento e insights de dados estruturados e não estruturados" (PROVOST; FAWCETT, 2013). Seu objetivo é transformar dados brutos em informações úteis que podem ser utilizadas para apoiar a tomada de decisões estratégicas em diversas áreas, como negócios, saúde, finanças, entre outras. Técnicas como aprendizado de máquina, análise preditiva e visualização de dados são frequentemente aplicadas para identificar padrões e tendências.

3 METODOLOGIA

Este capítulo apresenta os procedimentos e ferramentas utilizados para o levantamento de dados, e sua posterior análise, comparando a seleção dos softwares com as bibliotecas e *frameworks*.

A metodologia utilizada neste trabalho é baseada na Análise Exploratória de Dados (AED), cujo objetivo é resumir as principais características de um grande conjunto de dados, comumente utilizando-se de gráficos e métodos estatísticos para identificar padrões e anomalias. Esta abordagem é utilizada para obter insights a partir de grandes volumes de dados.

A seguir, serão detalhadas as etapas realizadas, incluindo a extração dos *commits*, cálculo de percentuais de alterações e o tratamento dos dados para garantir uma maior confiabilidade e consistência da análise. As ferramentas utilizadas também serão descritas, explicando como cada dado foi coletado com o Pydriller, quais as suas tratativas com a biblioteca Pandas, como foram geradas visualizações gráficas com o Matplotlib e como conectar à uma análise da pesquisa em questão.

3.1 Ferramentas utilizadas

A seguir, são apresentadas as principais ferramentas utilizadas na coleta e análise dos dados para este trabalho.

3.1.1 Python

Python é uma linguagem de programação versátil, amplamente utilizada para análises de dados e construção de *scripts* de automação devido à sua sintaxe moderna e grande disponibilidade de bibliotecas, se torna amplamente utilizada para manipulação de dados e mineração de repositórios

Python fornece a possibilidade de integrar múltiplos pacotes e gerar visualizações textuais e gráficas detalhadas por meio de bibliotecas, sendo assim ideal para extração e análise de métricas relacionadas ao código.

3.1.2 Pydriller

Pydriller é uma biblioteca em Python que facilita a mineração de repositórios Git, trazendo informações sobre alterações em arquivos e sua autoria. Extremamente acessível e amplamente utilizada para gerar históricos de desenvolvimento do ciclo de vida do projeto (MCKINNEY, 2018, p. 50).

3.1.3 Matplotlib

Matplotlib é uma biblioteca para criação de visualizações gráficas em Python, normalmente utilizada para representar dados de forma visual. Nesta análise, a biblioteca foi utilizada para a criação de gráficos das métricas extraídas dos repositórios, permitindo uma interpretação mais clara das tendências de alteração no código.

3.1.4 Pandas

Pandas é uma biblioteca de manipulação e análise de dados em Python, fundamental para o tratamento de grandes volumes de dados de forma eficiente. Ela facilita a limpeza, filtragem e organização dos dados, possibilitando a preparação deles para visualizações ou cálculos mais complexos.

3.2 Seleção de softwares

O critério principal para a seleção dos softwares a serem analisados, foi a relevância em cada uma das áreas da ciência de dados, processamento de imagens (BLUM; HOPCROFT; KANNAN, 2020, p. 89), banco de dados machine learning e desenvolvimento web. Além de um grande número de utilizadores na comunidade, possibilitando assim uma amostra variada e ampla de bibliotecas e *frameworks* em Python, sendo as selecionadas: TensorFlow, Scrapy, Pandas, Flask, Django, FastAPI, OpenCV, Pillow, Scikit-Learn e SQLAlchemy.

Além disso, foram selecionados projetos de código aberto de uso real, com atualizações constantes ao longo do tempo, no caso, com mais de 150 c*ommits* feitos na maior margem de tempo analisada. Essa seleção permitiu investigar as diferenças nos percentuais de alterações entre métodos e arquivos, conforme será apresentado na Seção 4.1.

Segue o Quadro **1**, que exibe uma amostra dos softwares selecionados, e sua área de atuação principal:

Quadro 1 - Seleção de software e Área de atuação

Software	**Área de atuação**	**URL**
TensorFlow	Machine Learning	https://www.tensorflow.org/

Scrapy	Web Scraping	https://scrapy.org/
Pandas	Ciência de Dados	https://pandas.pydata.org
Flask	Desenvolvimento Web	https://flask.palletsprojects.com
Django	Desenvolvimento Web	https://www.djangoproject.com
FastAPI	Desenvolvimento Web	https://fastapi.tiangolo.com
OpenCV	Processamento de Imagens	https://opencv.org
Pillow	Processamento de Imagens	https://python-pillow.org
Scikit-Learn	Machine Learning	https://scikit-learn.org
SQLAlchemy	Banco de Dados	https://www.sqlalchemy.org

Fonte - Elaborada pelo autor.

Os dez projetos selecionados para o estudo foram escolhidos devido à sua relevância em diferentes áreas da tecnologia e à sua ampla utilização pela comunidade. TensorFlow e Scikit-Learn são reconhecidos como ferramentas indispensáveis para o aprendizado de máquina, com suporte a modelos e algoritmos complexos. Pandas é pioneiro na manipulação e tratamento de dados, OpenCV e Pillow se destacam na utilização em processamento de imagens, área de visão computacional e edição de imagens, respectivamente. No contexto de desenvolvimento web, Django, Flask e FastAPI fornecem *frameworks* que auxiliam na criação de aplicações. Já o Scrapy, é uma ferramenta especializada em *web scraping*, possibilitando uma extração automatizada de dados. Por fim, SQLAlchemy facilita a interação com bancos de dados, permitindo abstrações que otimizam consultas e integrações com aplicações.

Ao abrangerem diferentes áreas de atuação, os projetos possibilitam uma análise diversificada.

3.3 Períodos selecionados

Os períodos selecionados foram de 30, 60 e 120 dias, iniciando a análise em julho de 2024. A margem de tempo analisada permite uma boa abrangência entre curto e médio prazo, além de serem períodos amplamente utilizados em estudos de ciclo de vida e gestão de projetos para representar diferentes marcos de produtividade, planejamento e execução (FORSGREN; HUMBLE; KIM, 2018).

Essas datas foram selecionadas a fim de equilibrar a representatividade dos dados e a viabilidade técnica do estudo, considerando a capacidade computacional disponível e o grande volume de informações geradas pelos repositórios analisados. A limitação técnica se

reflete no detalhamento necessário dentro do trabalho, que foi estabelecido em 120 dias para garantir que a análise cumprisse esses requisitos.

Importante também, ressaltar que todos os períodos foram contabilizados a partir do mês de início definido (julho de 2024), permitindo uma consistência e uniformidade na comparação entre os projetos.

3.4 Método de obtenção de dados

Para a obtenção dos dados, o PyDriller foi utilizado para extrair dados históricos dos *commits* de cada projeto, número de arquivos alterados, número de métodos alterados e número de linhas alteradas. Além disso, especificar quais métodos e arquivos foram os mais alterados no período especificado e suas linhas relacionadas, com uma tratativa para não contabilizar duplicatas. Este estudo empírico precede a análise exploratória dos dados afim de comparar os softwares selecionados entre si.

3.4.1 Commits

O PyDriller possibilitou a extração detalhada dos *commits*, permitindo identificar o número de alterações em métodos e arquivos, e quais foram os mais alterados de cada um, respectivamente, permitindo a análise exploratória posteriormente apresentada na Seção 4.1.

A Figura **1** abaixo exemplifica o uso do PyDriller:

Figura 1 - *Commits*

```
# Itera sobre os commits e as modificações
for commit in repo.traverse_commits():
    try:
        print(f"Analisando commit {commit.hash}...")
        for modified_file in commit.modified_files:
            # Ignora arquivos de teste
            if is_test_method_or_file(modified_file.filename):
                continue

            # Adiciona arquivos alterados
            arquivos_alterados.add(modified_file.filename)

            # Adiciona classes alteradas
            if modified_file.source_code is not None:
                classes_alteradas.add(modified_file.filename)  # Considera o arquivo como classe

            # Adiciona métodos alterados e totais
            if modified_file.methods:
                metodos_totais.update(modified_file.methods)
            if modified_file.changed_methods:
                # Conta o número de alterações por método
                for metodo in modified_file.changed_methods:
                    if is_test_method_or_file(metodo.name):
                        continue  # Ignora métodos de teste

                    metodo_contagem[metodo.name] += 1

                    # Adiciona ao conjunto de métodos alterados apenas os métodos que foram alterados uma vez
                    if metodo_contagem[metodo.name] == 1:
                        metodos_alterados.add(metodo)
```

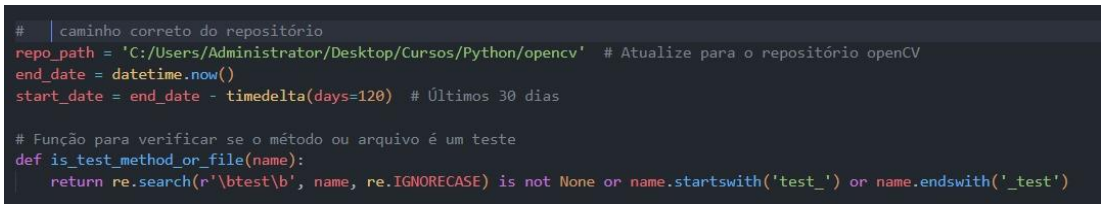
Fonte - Elaborada pelo autor.

Na Figura **1**, a função *trasverse_commits() itera* sobre todos os *commits* do repositório especificado. Para cada *commit*, existe uma grande gama de informações a serem trazidas, no caso, as relevantes e selecionadas foram: Classes alteradas, métodos alterados e suas respectivas quantidades. Essa abordagem facilita uma análise macro, ao longo do tempo, permitindo analisar o código e como ele evolui ao longo dos 30, 60 e 120 dias. Além de identificar padrões de mudança e manutenção, indispensáveis para uma estabilidade do projeto. Posteriormente, na Subseção 3.4.2, serão discutidas as medidas tomadas para tratar os dados, evitando duplicatas e arquivos de teste, para tornar o estudo mais fidedigno.

3.4.2 Tratamento de dados

O retorno da extração realizada na etapa anterior, gerava dados sem tratamento, com padrões distintos e trazendo informações irrelevantes para a análise final, como arquivos ou métodos de teste, e, duplicatas ao contabilizar as mudanças mais de uma vez, o tratamento de exceções foi aplicado diretamente no código.

Figura 2 - Tratamento de *Commits*



```
#   caminho correto do repositório
repo_path = 'C:/Users/Administrator/Desktop/Cursos/Python/opencv'  # Atualize para o repositório openCV
end_date = datetime.now()
start_date = end_date - timedelta(days=120)  # Últimos 30 dias

# Função para verificar se o método ou arquivo é um teste
def is_test_method_or_file(name):
    return re.search(r'\btest\b', name, re.IGNORECASE) is not None or name.startswith('test_') or name.endswith('_test')
```

Fonte - Elaborada pelo autor. 1

Na Figura **2**, é trazido um exemplo do OpenCV, onde é selecionado o repositório e a data de início do estudo, além disso, o tratamento de remoção de arquivos que contém “test”, “test_” ou “_test” no nome garante uma maior confiabilidade de que os arquivos analisados são, de fato, relacionados ao desenvolvimento, alinhando-se à finalidade deste trabalho. Com os dados preparados, a próxima etapa consiste em analisar os resultados obtidos, explorando padrões e comparando os projetos selecionados.

3.4.3 Margem de estabilidade

Com base no estudo "Mining Software Defects: Should We Consider Affected Releases?" de Li et al. (2020), a instabilidade do código pode ser inferida a partir de alterações frequentes e concentradas em componentes específicos do software. Quando mais de 10% dos arquivos ou métodos são alterados em um período de 30 dias, considera-se que o código demonstra instabilidade. Para períodos de 60 e 120 dias, 15% e 20%, respectivamente, são usados para caracterizar instabilidade no código, essa métrica será utilizada posteriormente no trabalho.

4 RESULTADOS E ANÁLISES

Para analisar qual é o percentual do código que muda ao longo tempo, e fazer a comparação entre os projetos, serão apresentados os resultados dos 10 projetos de software, considerando três cenários, 30, 60 e 120 dias. Desse modo, proporcionar à equipe de desenvolvimento uma visão clara sobre os métodos e arquivos mais frequentemente modificados, o percentual de código alterado ao longo do tempo, e, identificar os elementos mais críticos para uma manutenção direcionada, permitindo que os recursos sejam destinados para áreas que requerem maior atenção.

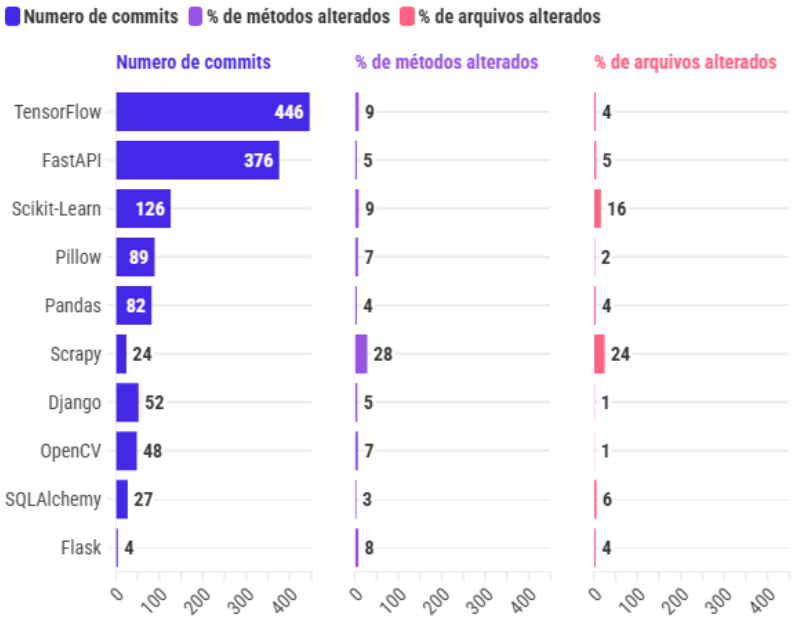
4.1 Análise quantitativa

A análise quantitativa reúne todos os resultados para os softwares estudados, apresentando os números oriundos das métricas selecionadas e suas correlações, de forma geral, sem levar em conta motivos e justificativas sobre cada um dos softwares.

As tendências serão analisadas nos 3 períodos selecionados, 30, 60 e 120 dias, e, sobre a quantidade de *commits*, porcentagem de métodos alterados e, porcentagem de arquivos alterados, sendo uma análise exploratória dos dados.

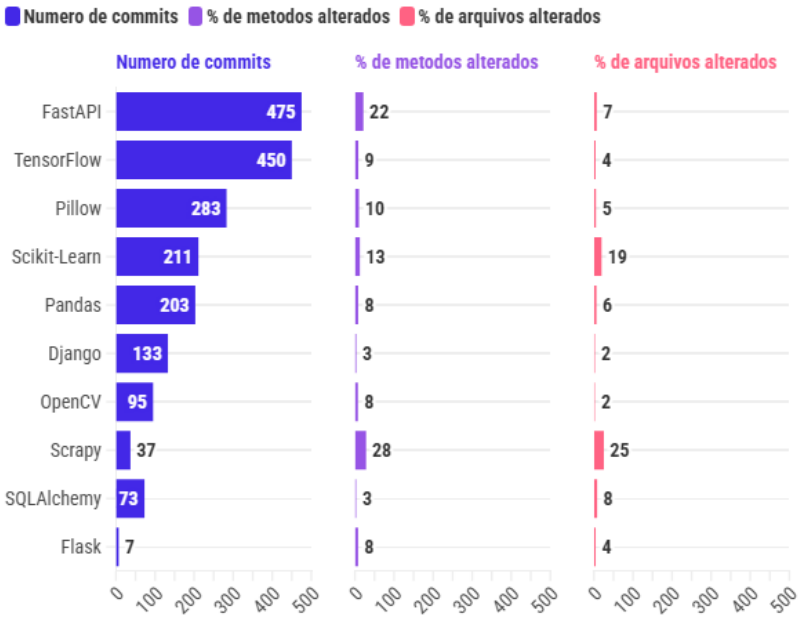
Abaixo, serão representados, a Figura **3**, referente à 30 dias, a Figura **4**, referente à 60 dias e por fim, a Figura **5**, que apresenta as métricas extraídas do período de 120 dias. Os projetos foram avaliados em termos de *commits* realizados, alterações em métodos e arquivos, e os percentuais refletem a proporção de alterações em relação ao total de métodos e arquivos de cada repositório, e serão utilizados para uma comparação posterior na Seção 4.2.

Figura 3 - Métricas de 30 dias



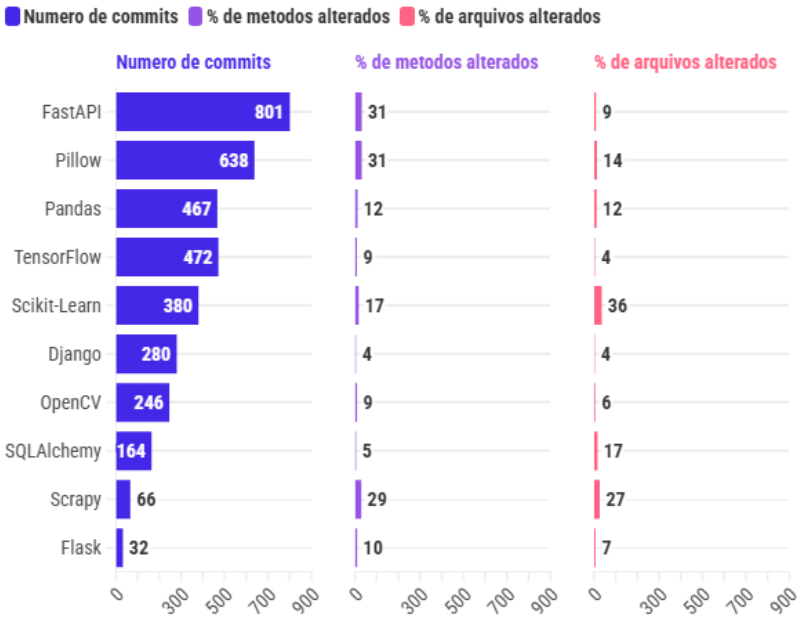
Fonte - Elaborada pelo autor.

Figura 4 - Métricas de dias



Fonte - Elaborada pelo autor.

Figura 5 - Métricas de 120 dias



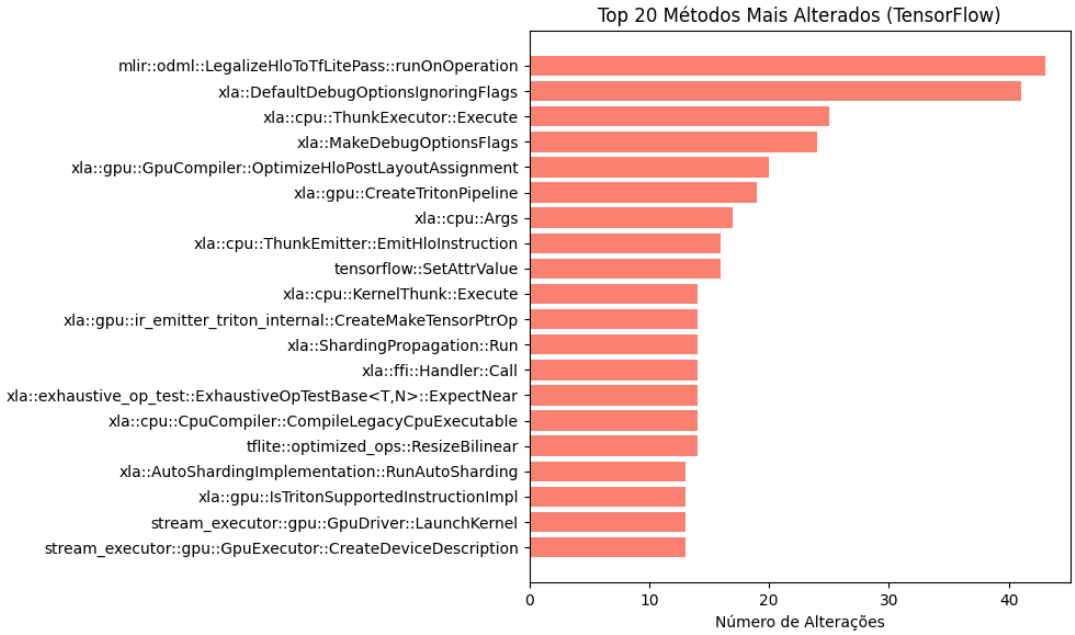
Fonte - Elaborada pelo autor.

4.1.1 TensorFlow

O TensorFlow, amplamente utilizado no domínio de machine learning, apresentou métricas de estabilidade ao longo dos três períodos analisados, com valores percentuais inferiores à margem preestabelecida no Capítulo 2.

4.1.1.1 Métodos mais alterados

Figura 6 - Métodos TensorFlow



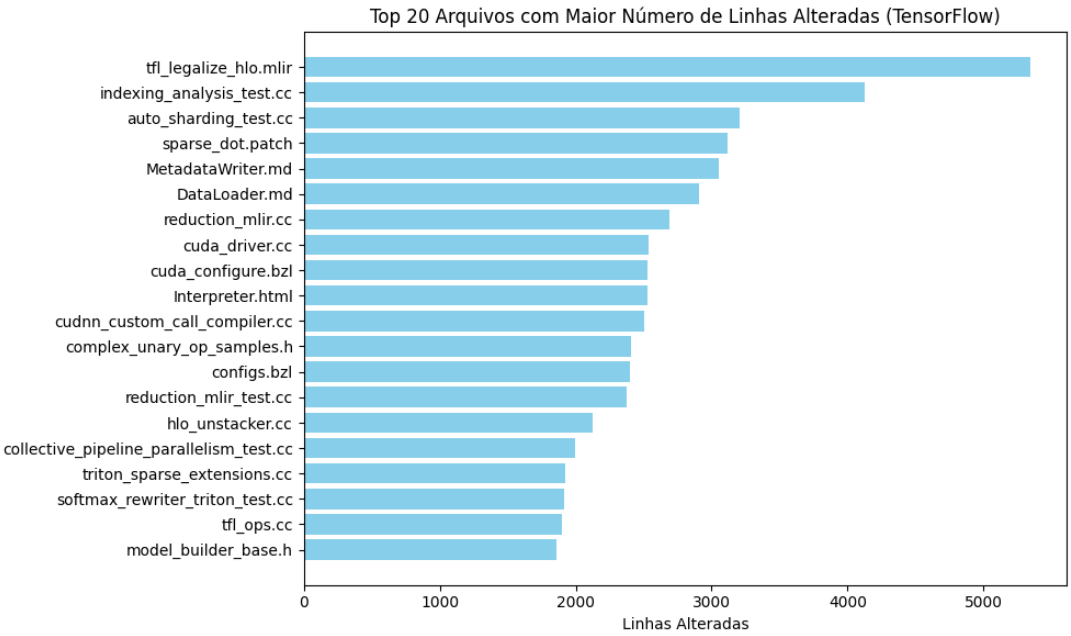
Fonte - Elaborada pelo autor.

A Figura **6** apresenta os métodos mais alterados no TensorFlow ao longo dos 120 dias. Estes métodos refletem esforços contínuos de melhoria em desempenho e compatibilidade.

1. *mlir::odml::LegalizeHloToTfLitePass::runOnOperation*
    - O método mais alterado, evidenciando foco na transformação e otimização de operações específicas para TensorFlow Lite.
2. *xla::DefaultDebugOptionsIgnoringFlags*
    - Frequentemente ajustado para lidar com opções padrão de depuração, sugerindo uma necessidade constante de adaptar ou corrigir funcionalidades.
3. *xla::cpu::ThunkExecutor::Execute*
    - Indica alterações relacionadas ao desempenho em execuções específicas da CPU.
4. *xla::MakeDebugOptionsFlags*
    - Relacionado a melhorias na configuração de opções de depuração.
5. *xla::gpu::GpuCompiler::OptimizeHloPostLayoutAssignment*
    - Alterado para otimizar processos pós-layout em GPUs.

4.1.1.2 Arquivos mais alterados

Figura 7 - Arquivos TensorFlow



Fonte - Elaborada pelo autor.

A Figura **7** apresenta os cinco arquivos mais modificados no TensorFlow, destacando componentes críticos para o desempenho e documentação do projeto.

1. *tfl_legalize_hlo.mlir*
    - Arquivo mais modificado, alinhado às otimizações observadas no método correspondente.
2. *sparse_dot.patch*
    - Indica foco em melhorias em operações relacionadas a matrizes esparsas.
3. *MetadataWriter.md*
    - Sugere atualizações frequentes em documentação técnica, essencial para a colaboração e manutenção.
4. *reduction_mlir.cc*
    - Evidencia foco em operações de redução, fundamentais para cálculos em machine learning.
5. *cuda_driver.cc*

- Relacionado a suporte para desempenho em GPUs.

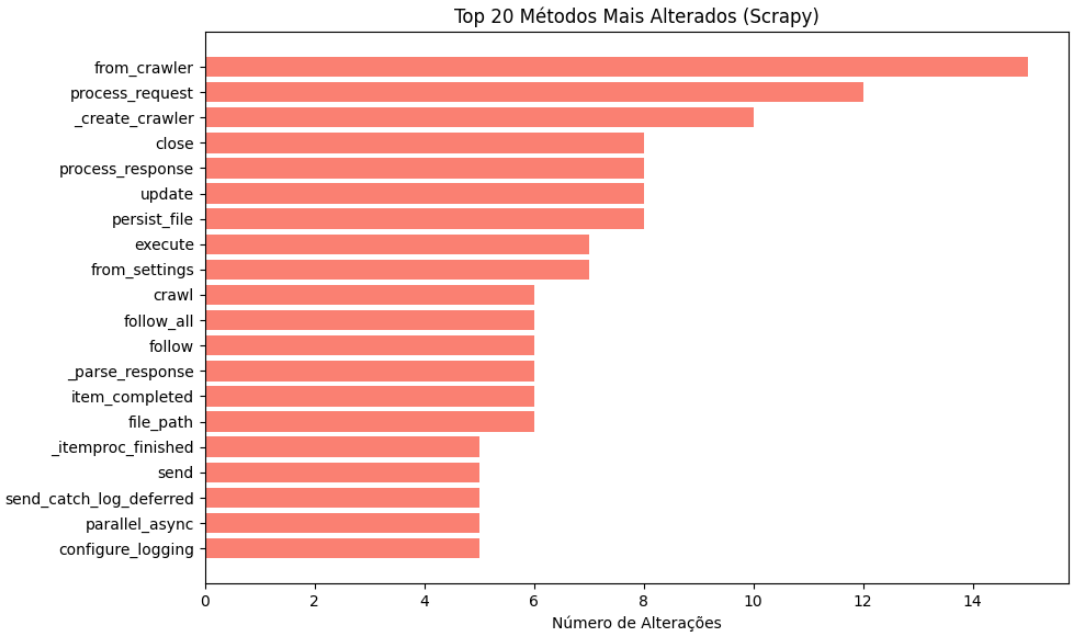
As alterações no TensorFlow estão concentradas em áreas cruciais para desempenho e compatibilidade, especialmente em hardware especializado (GPUs e CPUs). Além disso, a documentação técnica é constantemente atualizada. Mesmo com essas alterações concentradas, os percentuais gerais permaneceram estáveis.

4.1.2 Scrapy

O Scrapy, um framework amplamente utilizado para web scraping, apresentou instabilidade ao longo dos três períodos analisados, com valores percentuais significativamente superior à margem preestabelecida no Capítulo 2.

4.1.2.1 Métodos mais alterados

Figura 8 - Métodos Scrapy



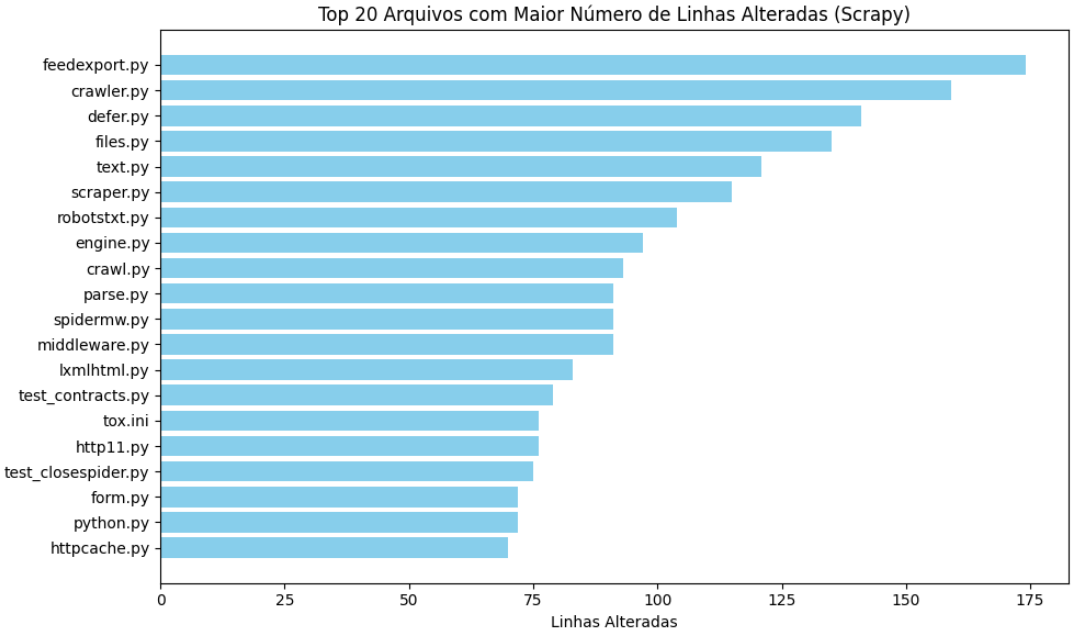
Fonte - Elaborada pelo autor.

A Figura **8**, apresenta os métodos mais alterados no Scrapy ao longo dos 120 dias. Estas alterações refletem melhorias contínuas em funcionalidades iniciais do framework.

1. *From_crawler*
    - O método mais alterado, refletindo uma importância na configuração inicial dos *crawlers.*
2. *Process_request*
    - Indica modificações voltadas para manipulação e otimização de requisições HTTP.
3. *_create_crawler*
    - Ajustes no processo de criação e dinâmica de *crawlers.*
4. *close*
    - Alterado para gerenciamento de *crawlers* de forma eficiente.
5. *Process_response*
    - Tratar e otimizar respostas às requisições HTTP.

4.1.2.2 Arquivos mais alterados

Figura 9 - Arquivos Scrapy



Fonte - Elaborada pelo autor.

A Figura **9** destaca os arquivos com o maior número de linhas modificadas ao longo dos 120 dias, principais pontos focais são funcionalidades e estruturas do framework.

1. *Feedxport.py*

- Arquivo mais modificado, alinhado às otimizações observadas no método correspondente.

2. *Crawler.py*
    - Indica foco em melhorias em operações relacionadas a matrizes esparsas.
3. *Defer.py*
    - Sugere atualizações frequentes em documentação técnica, essencial para a colaboração e manutenção.
4. *Files.py*
    - Evidencia foco em operações de redução, fundamentais para cálculos em machine learning.
5. *Text.py*
    - Relacionado a suporte para desempenho em GPUs.

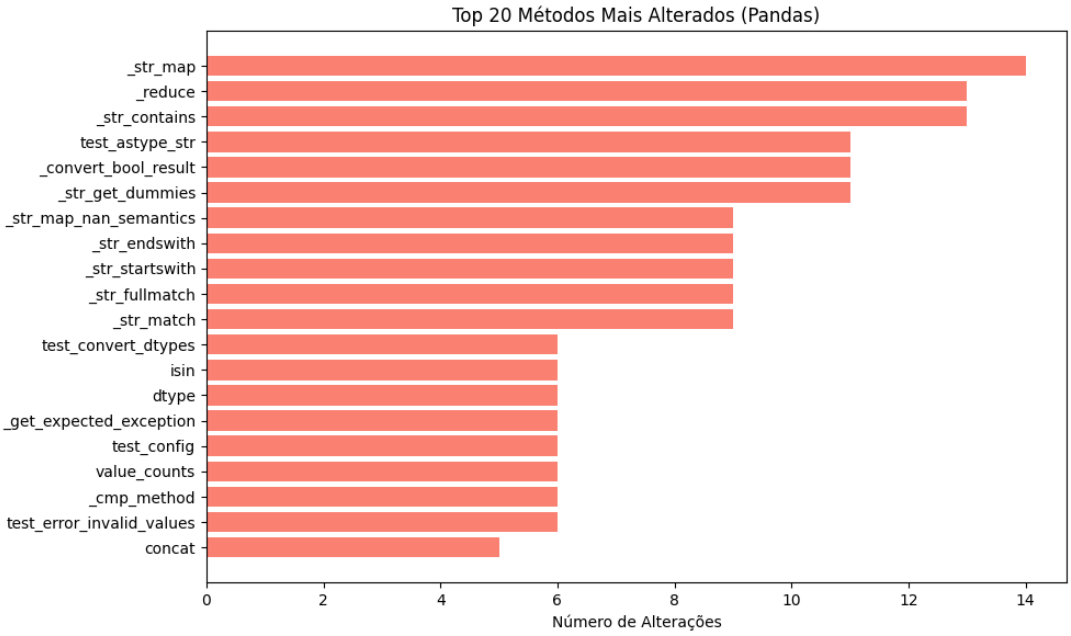
As alterações no Scrapy estão concentradas em áreas funcionais para web scraping, incluindo a manipulação de requisições e respostas, bem como a configuração e execução dos crawlers. As alterações nos arquivos, como feedexport.py e crawler.py, reforçam o foco em melhorias contínuas em desempenho e usabilidade. Diferente de projetos como TensorFlow, o Scrapy apresentou alta instabilidade.

4.1.3 Pandas

O Pandas é uma biblioteca fundamental para manipulação e análise de dados, amplamente utilizada em ciência de dados e machine learning. Sua robustez permite a manipulação eficiente de grandes volumes de dados, com suporte para estruturas como DataFrames e operações avançadas.

4.1.3.1 Métodos mais alterados

Figura 10 - Métodos Pandas



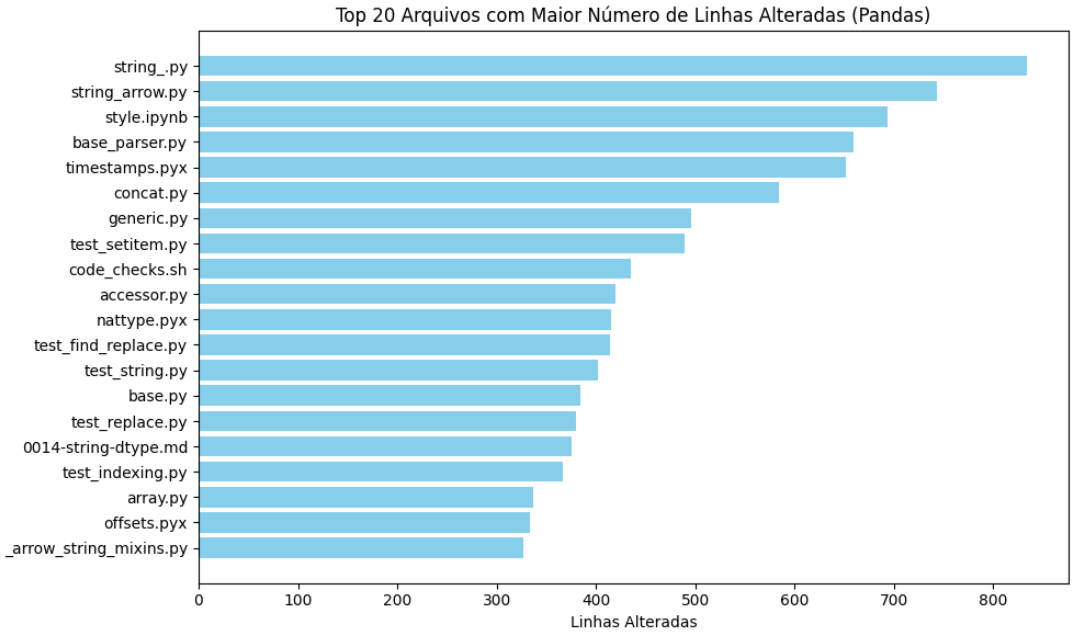
Fonte - Elaborada pelo autor.

A Figura **10** apresenta os métodos mais alterados no Pandas, refletindo o foco em funcionalidades relacionadas a operações com strings e tipos de dados.

1. *_str_map*
    - O método mais alterado, utilizado para melhorar manipular e mapear strings.
2. *_reduce*
    - Alterações voltadas para melhorias no desempenho de operações de redução em conjunto de dados.
3. *_str_contains*
    - Verificação de padrões em strings.
4. *_convert_bool_result*
    - Ajustes em conversões de valores booleanos.
5. *_str_get_dummies*
    - Criação de variáveis fictícias, utilizado em abundância no machine learning.

4.1.3.2 Arquivos mais alterados

Figura 11 - Arquivos Pandas



Fonte - Elaborada pelo autor.

1. *String_.py*
    - Foco em manipulação de *strings*, característica central da biblioteca.
2. *String_arrow.py*
    - Suporte a operações envolvendo *strings* e o formato Arrow.
3. *Style.ipynb*
    - Alterações em notebooks de estilo, documentação e demonstração de recursos.
4. *Base_parser.py*
    - Melhorias em estruturas de dados complexas.
5. *Timestampts.pyx*
    - Suporte a operações envolvendo timestamps, utilização em séries temporais.

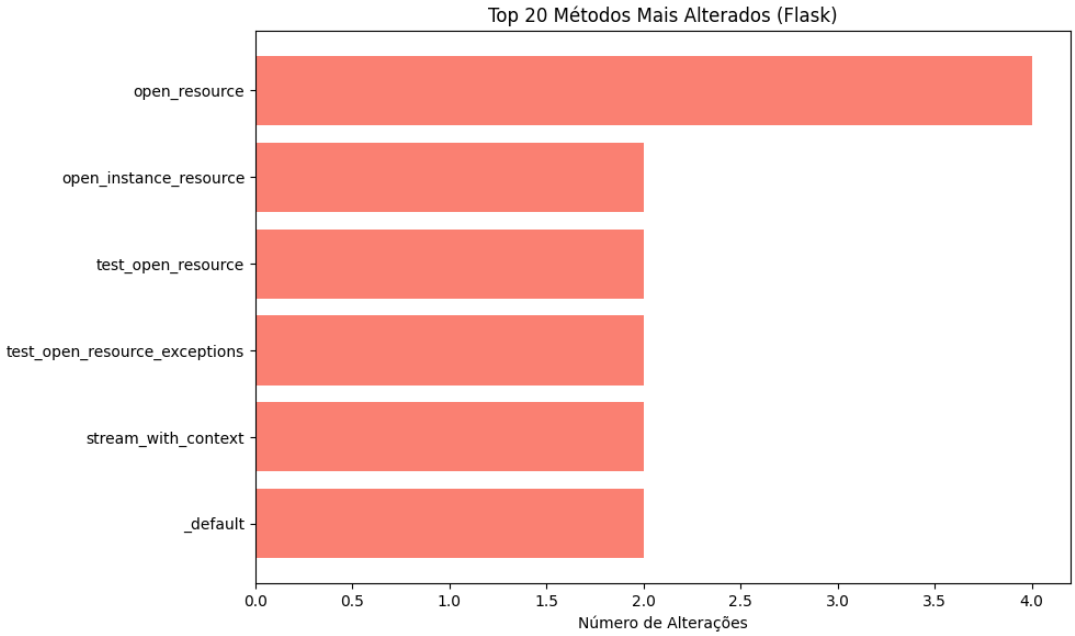
As alterações no Pandas estão fortemente concentradas em métodos e arquivos que lidam com strings, timestamps e operações matemáticas, áreas críticas para sua funcionalidade.

4.1.4 Flask

O Flask é um microframework web em Python amplamente utilizado para desenvolver aplicações web de forma rápida e flexível. Apesar de sua leveza e simplicidade, ele permite extensões para atender a necessidades mais complexas.

4.1.4.1 Métodos mais alterados

Figura 12 - Métodos Flask



Fonte - Elaborada pelo autor.

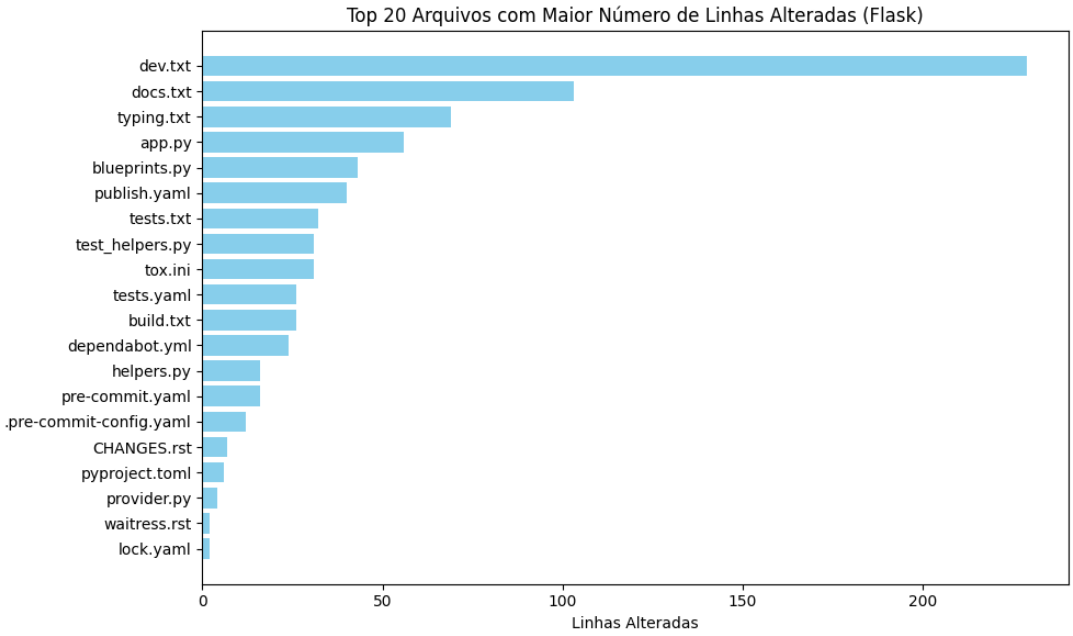
A Figura **12** apresenta os métodos mais alterados no Flask ao longo dos 120 dias analisados, com destaque para aqueles voltados à manipulação de recursos, por conta da capacidade computacional limitada, o Pydriller não permitiu a extração de mais métodos.

1. *Open_resource*
    - Método mais alterado, relacionado à manipulação de recursos.
2. *Open_instance_resource*
    - Foco em abrir recursos de instância.
3. *Stream_with_context*
    - Modificado para lidar com fluxos de dados.
4. *_default*

- Alterado para lidar com configurações padrão em diferentes cenários.

4.1.4.2 Arquivos mais alterados

Figura 13 - Arquivos Flask



Fonte - Elaborada pelo autor.

A Figura **13** apresenta os cinco arquivos mais modificados no Flask, indicando uma predominância de alterações em arquivos de configuração e documentação, com menor foco em componentes de execução direta.

1. *Dev.txt*
    - Relacionado à instruções e configurações para desenvolvedores.
2. *Docs.txt*
    - Alterações em documentações.
3. *Typing.txt*
    - Documentações, tipos e anotações.
4. *App.py*
    - Mudanças no arquivo principal do projeto, funcionalidades centrais.
5. *Blueprints.py*
    - Ajustes no suporte à modularidade de aplicações no Flask.

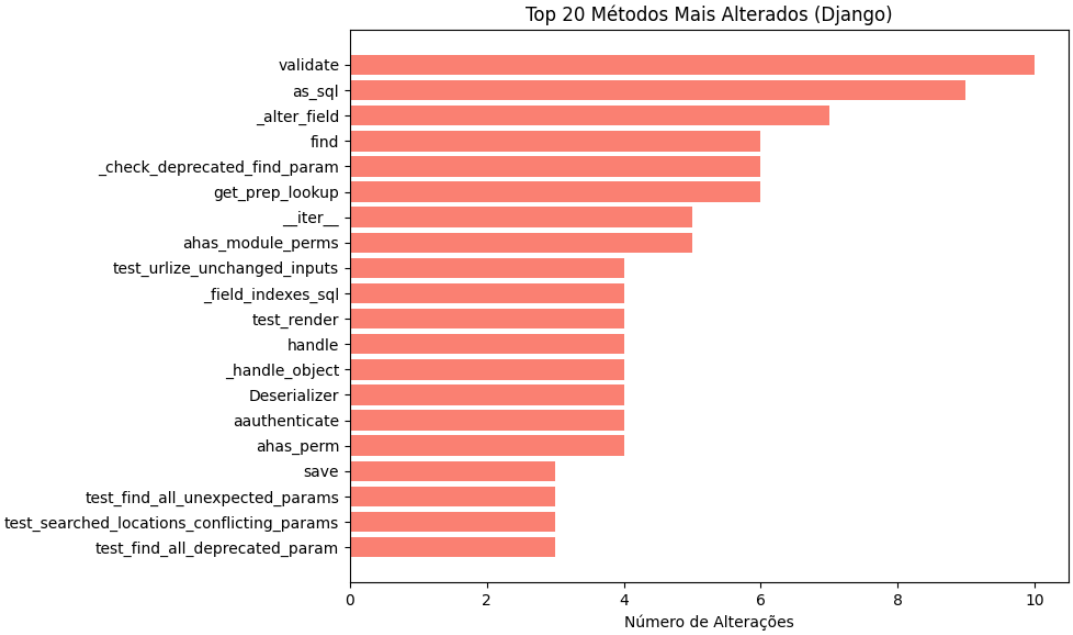
As alterações no Flask apresentam um padrão bastante concentrado, tanto em métodos quanto em arquivos, com foco evidente em documentação e configuração.

4.1.5 Django

O Django é um dos *frameworks* mais utilizados para o desenvolvimento web em Python, oferecendo uma ampla gama de ferramentas para criar aplicações robustas e escaláveis.

4.1.5.1 Métodos mais alterados

Figura 14 - Métodos Django



Fonte - Elaborada pelo autor.

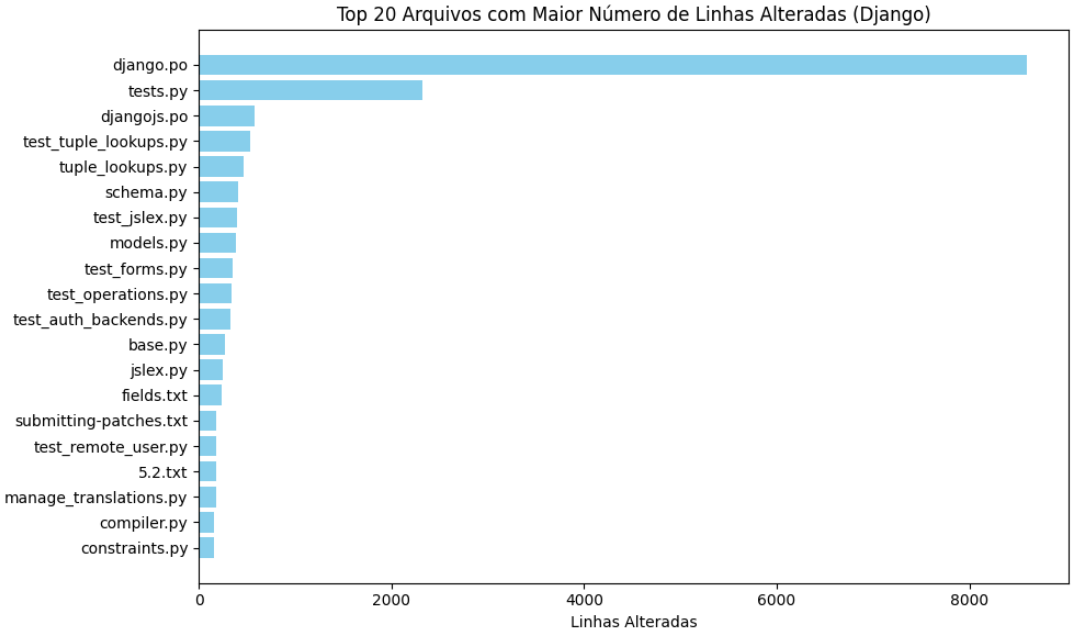
1. *Validate*
    - Verificação de integridade de dados em modelos e entradas de usuários.
2. *As_sql*
    - Geração de consultas SQL a partir de objetos.
3. *_Alter_field*
    - Ajustes em operações que modificam atributos de campos nos modelos.
4. *find*

- Busca de informações dentro da aplicação.

5. *_check_deprecated_find_param*
    - Compatibilidade com versões anteriores.

4.1.5.2 Arquivos mais alterados

Figura 15 - Arquivos Django



Fonte - Elaborada pelo autor.

1. *Django.po*
    - Arquivo de tradução, atualização relacionada à internacionalização.
2. *Djangojjs.po*
    - Continuação sobre tradução.
3. *Tuple_lookups*
    - Funcionalidades de busca em coleções de dados.

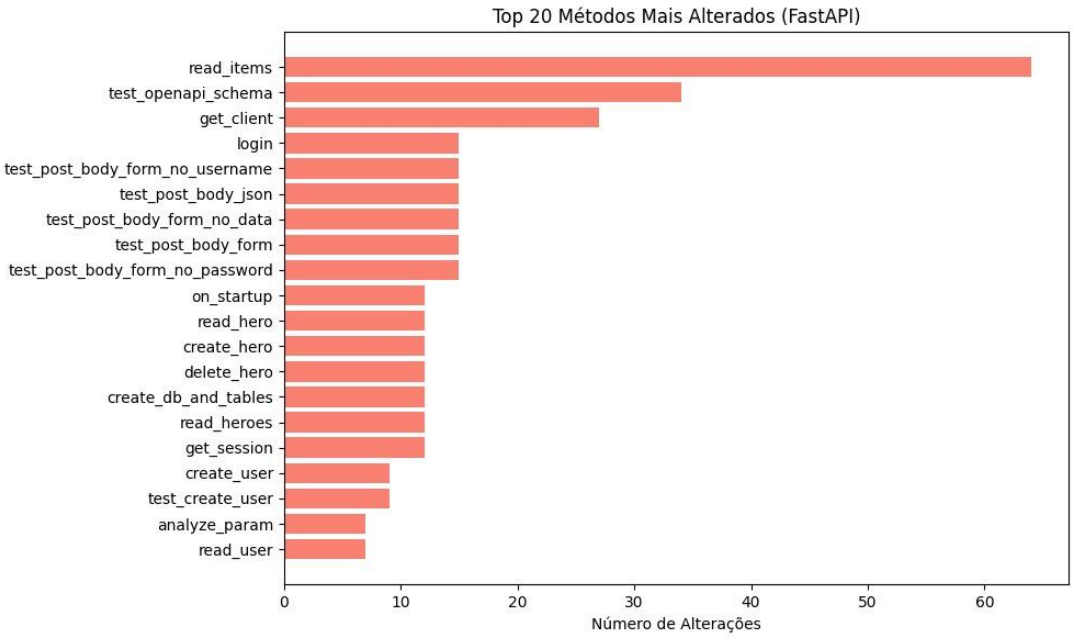
As alterações no Django estão concentradas em arquivos de tradução e testes, além de ajustes em métodos relacionados à compatibilidade e eficiência de operações básicas.

4.1.6 FastAPI

O FastAPI, um framework leve para desenvolvimento de APIs com suporte a Python moderno, possui características que promovem alto desempenho e simplicidade. As métricas analisadas ao longo do tempo destacaram alterações focadas em documentação e funcionalidades principais.

4.1.6.1 Métodos mais alterados

Figura 16 - Métodos FastAPI



Fonte - Elaborada pelo autor.

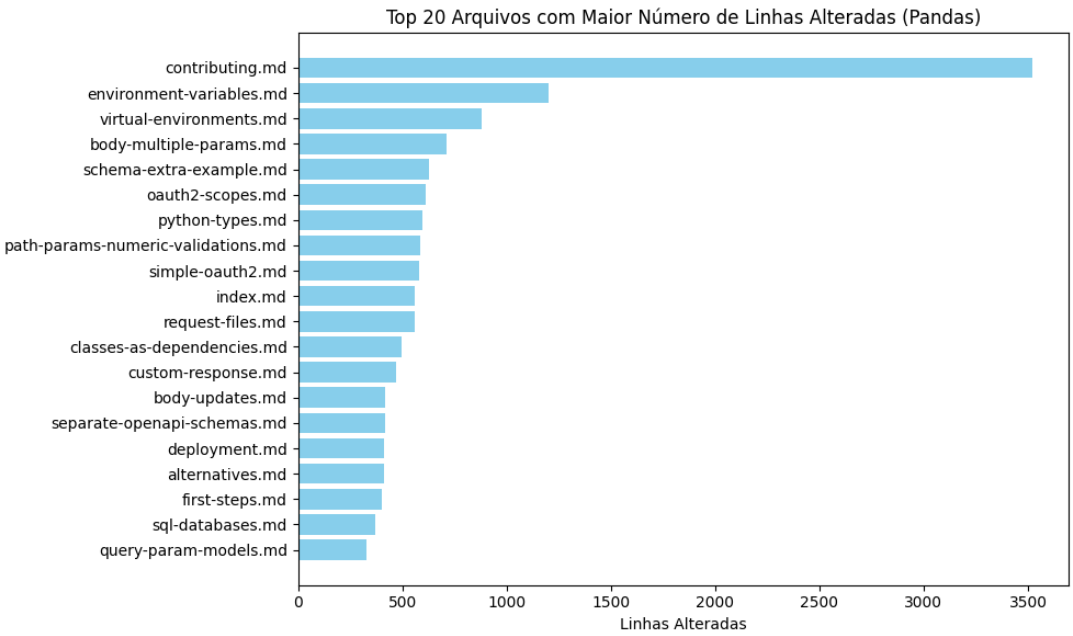
A Figura **16** retorna os métodos mais alterados no FastAPI ao longo do período analisado.

1. *Read_itens*
    - Método central relacionado à leitura de itens, comumente usados em exemplos e rotinas de API.
2. *Openapi_schema*
    - Validação de esquemas OpenAPI, conformidade de padrões.
3. *Get_client*
    - Gerenciamento de clientes em requisições.
4. *login*
    - Autenticação de usuários, aprimorar segurança e usabilidade.

5. *Post_body_form_no_username*
    - Verifica a ausência de parâmetros no corpo de requisições.

4.1.6.2 Arquivos mais alterados

Figura 17 - Arquivos FastAPI



Fonte - Elaborada pelo autor.

1. *Contributing.md*
    - Refino de diretrizes de contribuição entre colaboradores.
2. *Environment-variables.md*
    - Configuração de variáveis de ambiente, deploy.
3. *Virtual-environments.md*
    - Uso de ambientes virtuais no desenvovimento.
4. *Body-multiple-params.md*
    - Manipulação de parâmetros em requisições.
5. *Schema-extra-example.md*
    - Ilustração de padrões em esquemas extras.

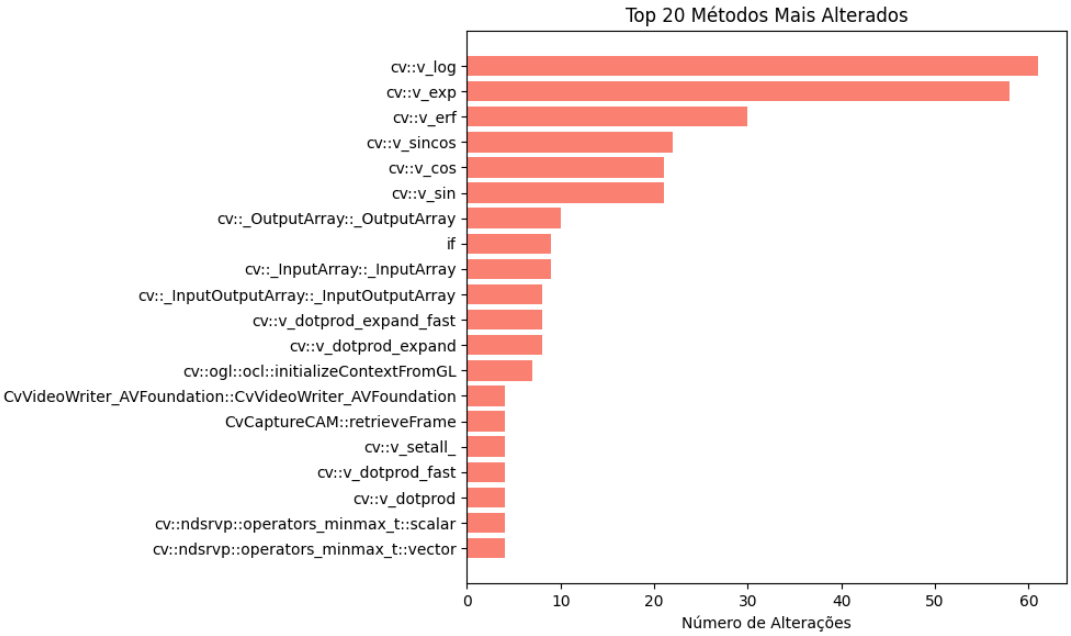
O FastAPI demonstra um padrão de alterações distribuído entre métodos centrais e documentação extensiva, com destaque para refinamentos em práticas de uso e desenvolvimento.

4.1.7 OpenCV

O OpenCV, amplamente utilizado para processamento de imagens e visão computacional, teve suas alterações mais concentradas em arquivos e métodos diretamente relacionados à matemática e ao suporte a dispositivos.

4.1.7.1 Métodos mais alterados

Figura 18 - Métodos OpenCV



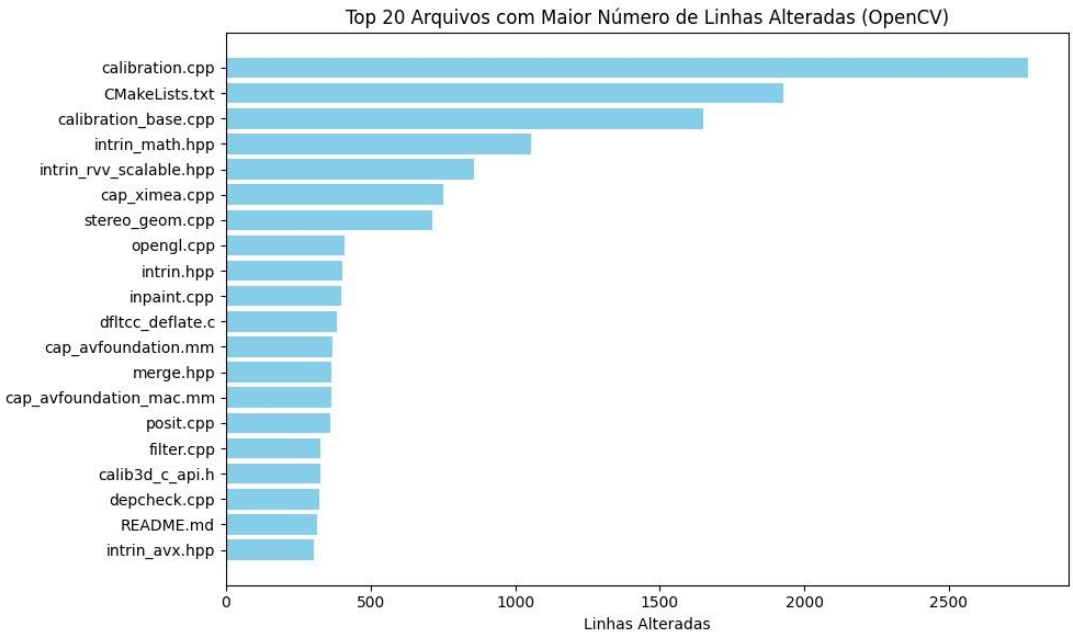
Fonte - Elaborada pelo autor.

1. *V_log*
    - Implementação de funções logarítmicas.
2. *Environment-variables.md*
    - Funções exponenciais, para meios gráficos.
3. *V_erf*
    - Calcular a função erro, relevante para operações de ajuste e filtros.
4. *V_sincos*
    - Cálculo de Seno e Cosseno, também relevantes para operações geométricas.
5. *V_cos*

- Precisão de cálculos de cosseno, também para operações geométricas.

4.1.7.2 Arquivos mais alterados

Figura 19 - Arquivos OpenCV



Fonte - Elaborada pelo autor.

1. *Calibration*
    - Rotinas de calibração de câmeras, visão computacional.
2. *CMakeLists*
    - Configuração e inclusão de bibliotecas externas, alterações na *build.*
3. *Calibration_base*
    - Complementação das funções do *calibration*, com foco em algoritmos de calibração.
4. *Intrin_math*
    - Definições matemáticas intrínsecas utilizadas em vários módulos do OpenCV.
5. *Intrin_rvv_scalable.hpp*
    - Operações matemáticas vetorizadas, otimizadas para hardware específico.

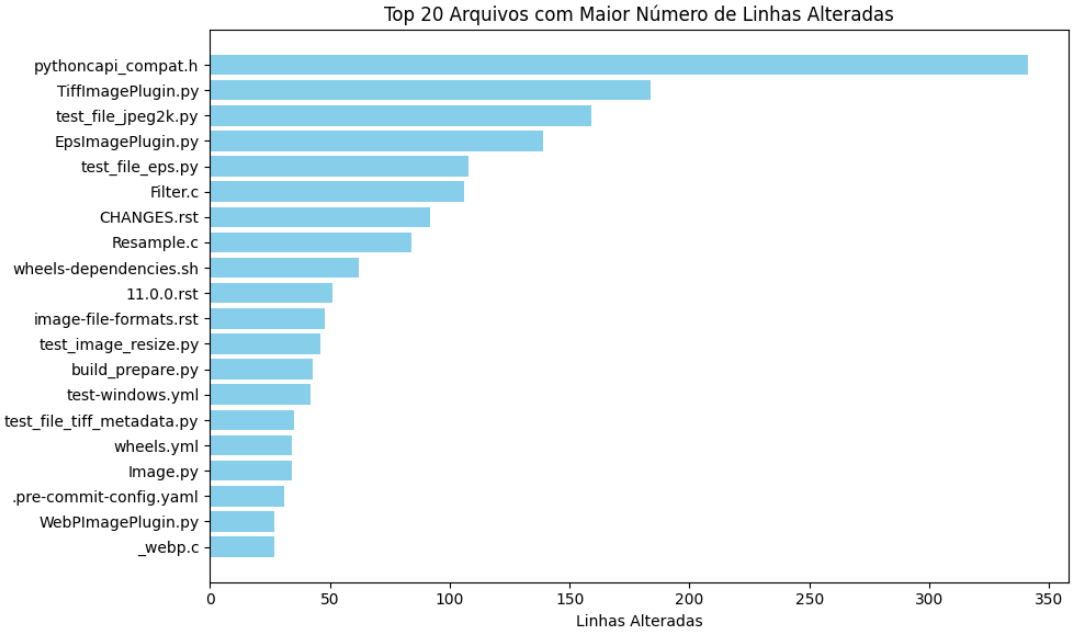
As modificações no OpenCV indicam esforços para manter a biblioteca atualizada com os requisitos de hardware moderno e suportar funcionalidades matemáticas avançada.

4.1.8 Pillow

O Pillow, uma biblioteca popular para manipulação de imagens em Python, apresentou alterações significativas em arquivos específicos ao longo do período analisado. Devido à limitações computacionais, não foi possível obter os dados relacionados aos métodos mais alterados.

4.1.8.1 Arquivos mais alterados

Figura 20 - Arquivos Pillow



Fonte - Elaborada pelo autor.

1. *Pythoncapi_compat.h*
    - Compatibilidade com Python, mantendo suporte às versões recentes da linguagem.
2. *TiffImagePlugin*
    - Suporte ao formato de imagem TIFF, para aplicações profissionais.
3. *File_jpeg2k.py*
    - Relacionado ao formato JPEG2000, ajustes e validações.
4. *EpsImagePlugin*
    - Lidar com o formato EPS e relacionados.

5. *File_eps*
    - Arquivo relacionado ao formato EPS, foco em estabilidade de arquivos.

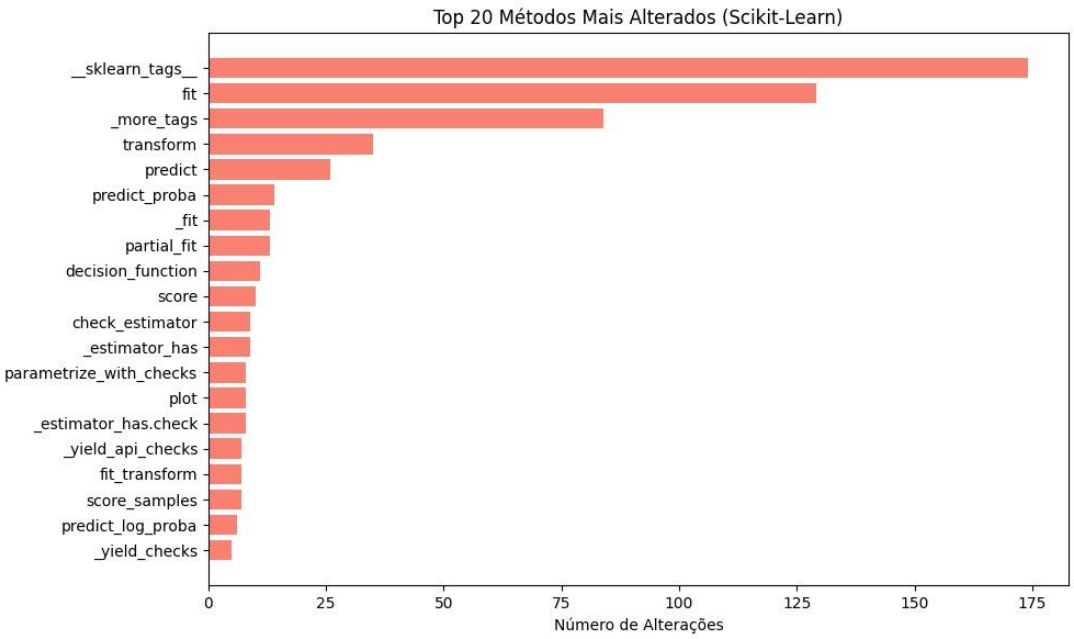
As alterações no Pillow destacam esforços para manter compatibilidade com formatos de imagens amplamente utilizados e melhorias contínuas em testes. A ausência de dados relacionados aos métodos mais alterados limita análises mais detalhadas.

4.1.9 Scikit-Learn

O Scikit-Learn, amplamente utilizado para aprendizado de máquina e processamento de dados, apresentou alterações em componentes relacionados à funcionalidade principal do framework.

4.1.9.1 Métodos mais alterados

Figura 21 - Métodos Scikit-Learn



Fonte - Elaborada pelo autor.

A Figura **21** apresenta os métodos mais alterados no Scikit-Learn, destacando esforços voltados à manutenção e ajustes de funcionalidades.

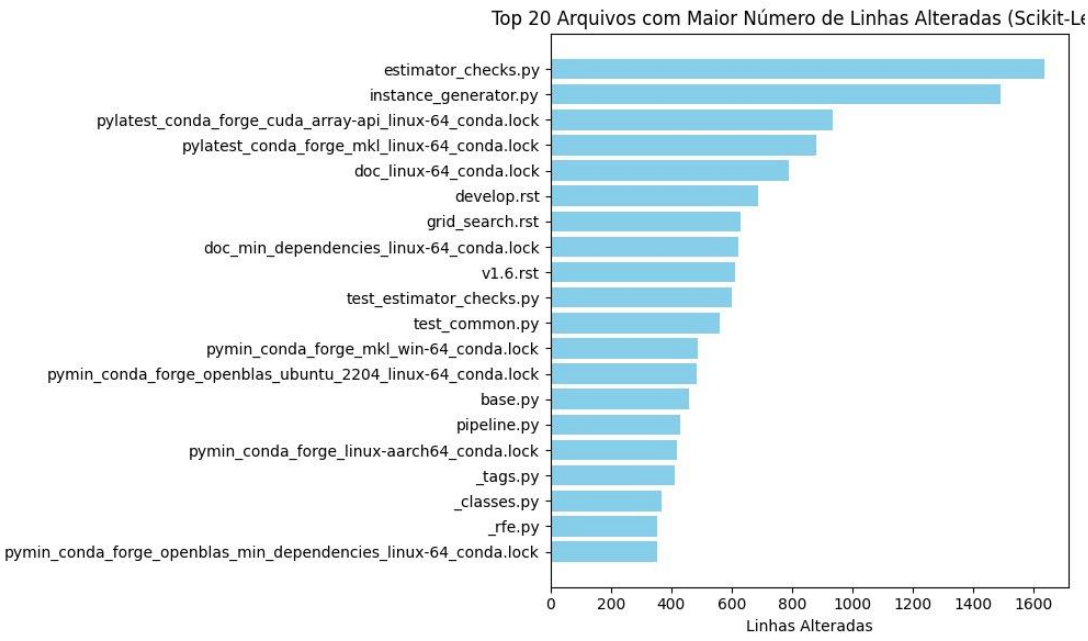
1. *__sklearn_tags__*

- Método relacionado à estrutura interna de identificação de atributos de estimadores.

2. *fit*
    - Método fundamental, garante robustez no ajuste de modelos.
3. *_more_tag*
    - Relacionado à tag interna de classificadores.
4. *transform*
    - Assegurar a funcionalidade de transformação nos pipelines do framework.
5. *Predict*
    - Importante para previsões e frequentemente ajustado para manter a consistência em estimadores.

4.1.9.2 Arquivos mais alterados

Figura 22 - Arquivos Scikit-Learn



Fonte - Elaborada pelo autor.

1. *Estimator_checks*
    - Método relacionado à estrutura interna de identificação de atributos de estimadores.

2. *Instance_generator*
    - Geração de instâncias para testes.
3. *Pylatest_conda_forge_cuda_Array-api-linux-64_conda.lock*
    - Alterações relacionadas à compatibilidade do framework com dependências no ambiente de desenvolvimento.
4. *Doc_linux-64_conda.lock*
    - Documentação e configuração do ambiente Linux.
5. *Develop.rst*
    - Documentação, melhorias e explicação de funcionalidades de desenvolvimento.

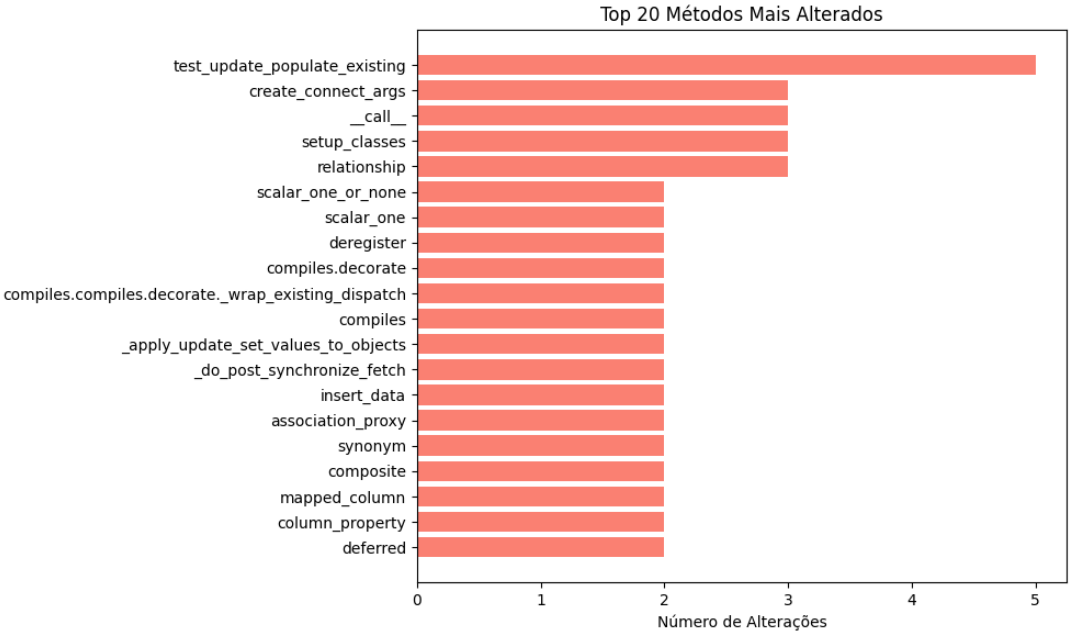
As alterações no Scikit-Learn se concentram principalmente em métodos centrais como fit, transform e predict, que são essenciais para o funcionamento do framework. Já os arquivos alterados incluem tanto aspectos de código quanto de documentação.

4.1.10 SQLAlchemy

O SQLAlchemy é uma biblioteca ORM (Object-Relational Mapping) amplamente usada para manipulação e abstração de bancos de dados em Python. Este projeto apresentou focos de ajustes e aprimoramentos no gerenciamento de dados e na configuração da biblioteca.

4.1.10.1 Métodos mais alterados

Figura 23 - Métodos SQLAlchemy

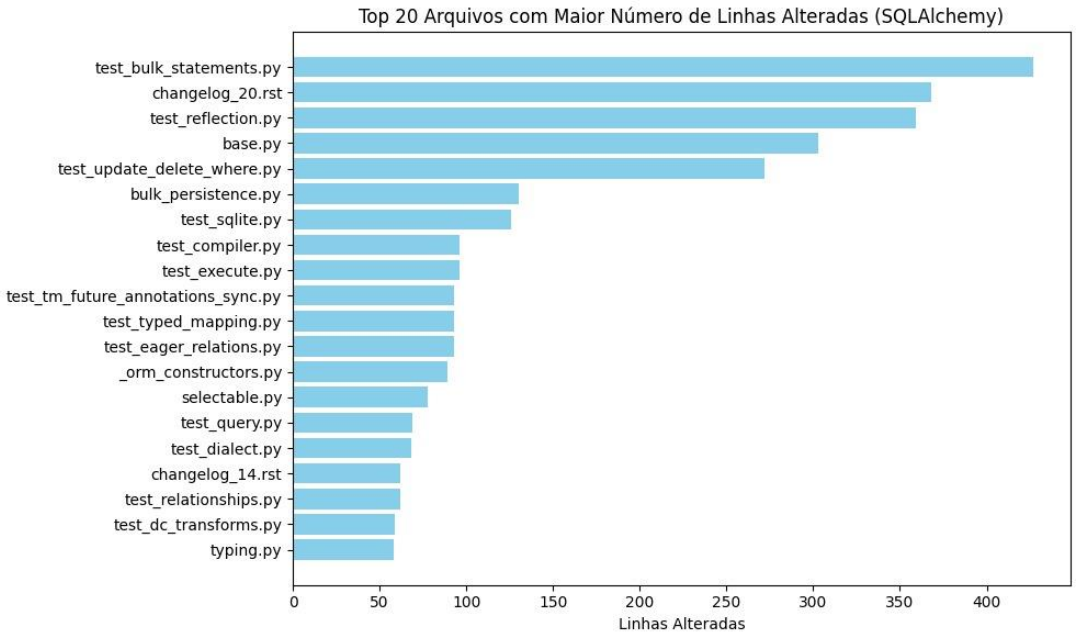


Fonte - Elaborada pelo autor.

1. *Update_populate_Existing*
    - Modificações relacionadas à atualização e preenchimento de objetos existentes na base de dados.
2. *Create_connect_args*
    - Alterado para melhorias na definição de argumentos de conexão com banco de dados.
3. *Call*
    - Ajustes no comportamento de execução de funções ou objetos chamáveis.
4. *Setup_classes*
    - Mudanças na configuração inicial de classes usadas como modelos na ORM.
5. *relationship*
    - Gerenciamento de relações entre tabelas.

4.1.10.2 Arquivos mais alterados

Figura 24 - Arquivos SQLAlchemy



Fonte - Elaborada pelo autor.

1. *Bulk_statements*
    - Arquivo relacionado à execuções em lote, manipulação massiva de dados.
2. *Changelog_20*
    - Documento de registro de mudanças e versões do projeto.
3. *reflection*
    - Funcionalidades de reflexão para inspecionar e mapear estruturas de banco de dados.
4. *base*
    - Definições fundamentais estruturais da biblioteca.
5. *Update_delete_where*
    - Operações de atualização e exclusão em bases de dados.

As alterações nos métodos e arquivos do SQLAlchemy estão concentradas em áreas técnicas que envolvem manipulação de dados, criação e gerenciamento de conexões com bancos de dados, e funcionalidades relacionadas à estruturação de objetos do ORM.

4.2 Análise comparativa

A análise comparativa a seguir examina semelhanças e diferenças acerca dos softwares estudados. Com foco em padrões técnicos que foram observados nos métodos e arquivos mais alterados, analisados na Seção 4.1. Está seção também considera aspectos específicos relacionados às áreas de atuação de cada projeto, como desempenho, modularidade e complexidade das alterações.

4.2.1 Semelhanças

1. Atualizações em documentação
    - Projetos como TensorFlow, FastAPI e Django apresentaram alterações consistentes em arquivos de documentação (*.md*, *.rst*).

2. Presença de testes entre os arquivos alterados
    - Arquivos de teste aparecem com frequência nos projetos SQLAlchemy, Pillow, FastAPI e Scikit-Learn, refletindo uma aderência às boas práticas de qualidade e desenvolvimento orientado à testes.

3. Métodos diretamente ligados à funcionalidades centrais
    - Alterações em métodos que definem a funcionalidade principal
        - Scikit-Learn: *fit* e *predict* são essenciais para o aprendizado de máquina
    - OpenCV: Métodos matemáticos como *v_log* e *v_exp* são voltados para cálculos de alta *performance*
    - TensorFlow: Métodos relacionados à otimização de operações em GPUs e CPUs, por exemplo *runOnOperation.*

4. Aderência a práticas de modularidade

- Evidenciada por alterações concentradas em arquivos ou métodos isolados, padrão observado nos projetos Flask e Scrapy.

4.2.1.1 Diferenças

1. Distribuição das alterações
    - TensorFlow e OpenCV concentraram alterações em um número selecionado de arquivos e métodos como *tfl_legalize_hlo.mlir* (TensorFlow) e *calibration.cpp* (OpenCV). Por outro lado, projetos como Scrapy e SQLAlchemy apresentaram mudanças distribuídas em um número maior de arquivos e métodos.

2. Natureza dos métodos alterados
    - Scikit-Learn e Pandas apresentaram alterações em métodos relacionados a aprendizado de máquina e manipulação de dados, respectivamente. Em contraste, Django e Flask concentraram alterações relacionadas à estruturação e configuração do framework.

3. Enfoque em hardware e desempenho
    - TensorFlow e OpenCV tiveram alterações em métodos e arquivos relacionados à otimização para hardware especializado, como GPUs. Já o FastAPI e Scrapy demonstraram alterações mais voltadas para lógicas de aplicação e APIs.

4.2.2 Mudanças no código ao longo do tempo

A análise dos percentuais de métodos e arquivos alterados ao longo dos períodos revela padrões distintos entre os projetos, indicando diferentes dinâmicas de manutenção, desenvolvimento e foco. Este foco contribui para uma redução de custos, minimização de falhas e uma melhora na qualidade do código, conforme destacado por Li et al. (2020), áreas com alta frequência de mudanças são fortes candidatas à uma manutenção preditiva.

Projetos como Scrapy, Pillow e FastAPI demonstraram percentuais altos de alterações em todos os períodos de estudo (30, 60 e 120 dias), por outro lado, projetos como TensorFlow, Django e SQLAlchemy mantiveram percentuais semelhantes em todos as margens estudadas.

Já no Scikit-Learn, observou-se uma tendência crescente de alterações ao longo dos períodos, especialmente em arquivos. O Flask, mesmo com um número reduzido de mudanças em relação aos outros projetos, apresentou alterações significativas em arquivos relacionados à configuração e documentação.

Os resultados obtidos na Seção 4.1 demonstram que a análise do percentual do código alterado pode ser útil para guiar processos de priorização de testes e revisões futuras. Projetos com maior concentração de alterações em métodos e arquivos ao longo do tempo, como o Scrapy, podem ser utilizados como referência para identificar áreas críticas e frequentes de mudança. Ao mesmo tempo, projetos como TensorFlow e Django servem como exemplos de estabilidade, permitindo um direcionamento dos esforços à outros trechos do código.

Essa abordagem ressalta a importância de associar os percentuais observados à uma priorização futura de recursos. Por exemplo, projetos que demonstraram maior variação ao longo do tempo, como Pandas e OpenCV, podem ser explorados em estudos posteriores para investigar padrões de alteração, e assim, avaliar como esses padrões podem ajudar a diminuir esforços desnecessários em áreas estáveis do código.

5 CONCLUSÃO

Este trabalho buscou investigar qual o percentual de código que muda ao longo do tempo em diferentes projetos de software, por meio de uma análise das alterações em métodos e arquivos ao longo de três períodos distintos definidos (30, 60 e 120 dias). A pesquisa foi fundamentada na Mineração de Repositórios de Software (MSR) e utilizou o Pydriller, Pandas e o Matplotlib como ferramentas para extração e tratamento de dados de 10 projetos Python, amplamente utilizados na comunidade. Os resultados permitiram identificar padrões de manutenção e desenvolvimento, fornecendo insumos para futuras melhoras no processo de engenharia de software.

As limitações do trabalho, como a restrição do período de análise a 120 dias e a não-possibilidade de extração de métodos de todos os projetos devido à limitações computacionais, não comprometem os resultados, e abrem caminho para estudos futuros, tendo em vista que investigações de períodos mais amplos podem incluir analises qualitativas para complementar os insights obtidos.

Por fim, este estudo contribui para o campo da engenharia de software ao apresentar uma metodologia que auxilia a avaliar a estabilidade de projetos, com base em métricas de alterações percentuais. Os resultados demonstram a importância de aplicar a abordagem para priorizar testes, alocar recursos e direcionar esforços de desenvolvimento, promovendo a evolução contínua dos projetos analisados, e potencialmente, outros projetos de código aberto.

REFERÊNCIAS

BLUM, Avrim; HOPCROFT, John; KANNAN, Ravindran. *Foundations of Data Science: With Applications in R and Python*. Cambridge University Press, 2020.

FISCHER, Michael; PINZGER, Martin; GALL, Harald C. *Populating a Release History Database from Version Control and Bug Tracking Systems*. Proceedings of the International Conference on Software Maintenance (ICSM), IEEE, 2003.

FORSGREN, Nicole; HUMBLE, Jez; KIM, Gene. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 2018.

HASSAN, Ahmed E. *Mining Software Repositories*. In: *Proceedings of the 2008 Frontiers of Software Maintenance*. IEEE, 2008. p. 48-57.

IEEE**.** Mining software defects: Should we consider affected releases? *In: IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020. Disponível em: https://ieeexplore.ieee.org/document/8811982. Acesso em: 27 nov. 2024

LI, W.; HUANG, Q.; ZHANG, Y. Mining Software Defects: Should We Consider Affected Releases? *IEEE Transactions on Software Engineering*, v. 46, n. 11, p. 1241-1255, 2020.

MCKINNEY, Wes. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 2018.

PRESSMAN Roger S. *Software Engineering: A practitioner's Approach*. 9ª ed. New York: McGraw-Hill, 2015

PROVOST, F.; FAWCETT, T. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. O'Reilly Media, 2013.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011.