# Estudo de Caso de mudanças disruptivas em argumentos padrão em bibliotecas de aprendizado de máquina

**Arthur de Brito Bonifácio**[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

Projeto Orientado em Computação II

arthurbonifacio@dcc.ufmg.br

***Abstract.*** *This article explores the concept of "Default Argument Breaking Changes"(DABC) and their implications for software projects utilizing open-source libraries. The research delves into the analysis of alterations in default arguments, categorizes distinct types of DABCs, presents recurring patterns, and formulates key questions for a thorough examination. The study focuses on the ScikitLearn, Pandas, and NumPy libraries, each characterized by diverse functionalities and structures. Occurrences of DABCs are extracted and subjected to statistical analysis, yielding preliminary results and insightful comparisons. The initial findings indicate a significant reduction in the number of DABCs in the studied libraries, with unique library-specific characteristics contributing to this disparity. These results provide valuable insights into the prevalence of DABCs and the impact of library philosophies on software projects. Notably, the study leverages newly developed automated tools, demonstrating the creation of tools to facilitate the extraction and analysis of DABCs, contributing to the advancement of research in this domain.*

***Resumo.*** *Este artigo aborda o conceito de "Default Argument Breaking Changes"(DABC) e suas implicações em projetos de software que fazem uso de bibliotecas de código aberto. A pesquisa explora a análise de alterações em argumentos padrões, categoriza diferentes tipos de DABCs, apresenta padrões recorrentes e formula questões-chave para uma avaliação abrangente. O estudo concentra-se nas bibliotecas ScikitLearn, Pandas e NumPy, cada uma caracterizada por funcionalidades e estruturas distintas. Ocorrências de DABCs são extraídas e submetidas a análises estatísticas, proporcionando resultados preliminares e comparações esclarecedoras. As descobertas iniciais indicam uma redução significativa no número de DABCs nas bibliotecas estudadas, com características específicas de cada biblioteca contribuindo para essa disparidade. Esses resultados oferecem insights valiosos sobre a prevalência de DABCs e o impacto das filosofias de bibliotecas em projetos de software. Destaca-se que o estudo faz uso de ferramentas automatizadas recém-desenvolvidas, evidenciando a criação de instrumentos para facilitar a extração e análise de DABCs, contribuindo para o avanço da pesquisa nesse domínio.*

## 1. Introdução

O uso de valores padrão para argumentos de função é uma técnica comum em programação para simplificar o código e fornecer um comportamento padrão para funções

quando nenhum valor é fornecido. Esses valores padrão são conhecidos como ”default argument values”em inglês. No entanto, quando os valores padrão de um argumento de função são alterados em uma atualização de software, isso pode quebrar o código existente que depende desses valores padrão antigos. Esse tipo de mudança é chamado de ”Default Argument Breaking Change” ou DABC. Por exemplo, suponha que uma função chamada ”add_numbers” tenha um argumento opcional com um valor padrão de 0:

```
def add_numbers(x, y=0):
return x + y
```

Se uma atualização de software alterar o valor padrão de ”y”para 1, isso pode quebrar o código existente que depende do comportamento anterior da função. Isso ocorre porque a chamada ”add_numbers(2)” anteriormente retornava 2, mas agora retornará 3. Isso pode causar erros em outras partes do código que dependem do comportamento anterior da função. Portanto, é importante ter cuidado ao alterar os valores padrão de argumentos de função para evitar DABCs.

O desenvolvimento de bibliotecas de aprendizado de máquina tem permitido que muitas aplicações da inteligência artificial sejam implementadas de forma mais rápida e eficiente. No entanto, com o lançamento de novas versões dessas bibliotecas, é possível que mudanças sejam feitas em parâmetros padrão que podem impactar diretamente na execução de códigos já existentes. Essas mudanças, conhecidas como ”default argument breaking changes”(DABC), podem causar um comportamento diferente do funcionamento de modelos de aprendizado de máquina que antes estavam em pleno funcionamento e gerar resultados completamente diferentes do esperado, afetando projetos de pesquisa, negócios e outras áreas em que o uso dessas bibliotecas seja necessário. Por esse motivo, torna-se relevante o estudo dessas mudanças e analisar o impacto em projetos que as utilizam.

### 1.1. Objetivos Gerais

A crescente adoção de bibliotecas e frameworks de código aberto leva muitos desenvolvedores a contar com recursos e funcionalidades fornecidos por essas bibliotecas para acelerar o processo de desenvolvimento de software. No entanto, com a evolução contínua dessas bibliotecas, novas funcionalidades são adicionadas e, às vezes, recursos existentes são removidos ou modificados, o que pode resultar em quebra de compatibilidade e impactar o desenvolvimento e a manutenção do software.

A compreensão das mudanças introduzidas em uma biblioteca é fundamental para que os desenvolvedores possam tomar decisões informadas e mitigar possíveis problemas que podem surgir com a atualização de uma biblioteca. Nesse contexto, este artigo tem como motivação a análise de breaking changes em uma biblioteca de código aberto, com o objetivo de fornecer um estudo detalhado de como essas mudanças podem impactar um projeto de software que utiliza essa biblioteca.

Tem-se como objetivo a análise do impacto das mudanças em argumentos padrões e como isso afeta grande projetos. A análise será feita de forma estatística e visando responder perguntas-chave que guiam a conclusão sobre o estudo de caso. Categorizar os tipos de DABC’s, apresentar valores de recorrência e levantar questionamentos para análise que dão um panorama sobre o assunto além da biblioteca em questão.

### 1.2. Objetivos Específicos

Para a primeira etapa o objetivo é escolher uma biblioteca e estudá-la com o intuito de entender sua organização. Desde a estrutura do repositório e a forma de mudanças até a documentação final. Com isso, entender a fundo a biblioteca e definir o que pode ser utilizado para análse do processo de desenvolvimento.

Na segunda etapa, a análise qualitativa do código se concentrará na extração e descrição minuciosa das ocorrências de DABCs por meio da geração de uma tabela específica. Este processo envolverá a criação manual da tabela de descrição de DABCs, utilizando-a como uma ferramenta fundamental para a identificação e catalogação das diferentes instâncias identificadas. A tabela de resultados resultante será um recurso valioso para formulação de questões-chave e permitirá uma análise estatística mais aprofundada.

A análise qualitativa também se estenderá à investigação dos notebooks de desenvolvimento, visando identificar casos de uso relevantes relacionados às ocorrências de DABCs. A abordagem incluirá a extração de estatísticas específicas de trechos de código, proporcionando uma compreensão abrangente das implicações das dependências antiéticas nas diferentes partes do projeto. Essa análise profunda permitirá uma visão mais holística das práticas de desenvolvimento e fornecerá insights valiosos para a melhoria contínua do código.

Além disso, a análise qualitativa incluirá a identificação e documentação dos cenários em que as DABCs são mais prevalentes, fornecendo uma compreensão contextual de como essas dependências impactam o sistema na totalidade. Ao final desta etapa, os resultados serão apresentados de maneira estruturada em tabelas, facilitando a interpretação dos dados e possibilitando a formulação de estratégias eficazes para mitigar as dependências antiéticas identificadas.

Na terceira etapa, objetivamos a automação do processo de extração de Default Argument Breaking Changes (DABCs) por meio da análise e parse do código-fonte, bem como da mineração do repositório para uma análise abrangente da biblioteca em questão. Para atingir esse objetivo, implementaremos um conjunto de ferramentas e scripts que possibilitarão a identificação automatizada de DABCs, proporcionando eficiência e escalabilidade ao processo.

### 1.3. Estrutura do Trabalho

O restante deste trabalho é composto pela seção 2 com a revisão da literatura. A seção 3 apresenta a metodologia usada desde o estudo da biblioteca até a forma de obtenção dos dados quantitativos e os resultados propriamente ditos. A seção 4 é destinada à conclusão do trabalho e os trabalhos futuros.

## 2. Revisão da Literatura

[Ferreira et al. 2020] apresenta como soluções a identificação das áreas de pesquisa em Engenharia de Software (SE) que têm sido beneficiadas pela aplicação de técnicas de Deep Learning (DL), bem como os tipos de problemas que têm sido solucionados com essas técnicas. Além disso, o estudo mapeia as técnicas de DL mais utilizadas e as fontes de dados empregadas nos estudos sobre DL em SE. O artigo contribui para a comunidade científica ao fornecer uma visão geral das pesquisas mais recentes nessa área, permitindo

aos pesquisadores identificar gaps e oportunidades para novas pesquisas. A partir dos resultados encontrados, é possível perceber que a aplicação de DL em SE tem se expandido ao longo dos anos, e que a documentação, a predição de defeitos e os testes são os principais problemas de pesquisa abordados pelos estudos analisados.

[Montandon et al. 2020] apresenta uma solução para identificar especialistas em bibliotecas e frameworks de software no GitHub. Os autores propõem uma abordagem baseada em análise de rede, que utiliza informações de colaboração, atividade e conhecimento técnico dos usuários em questão. A abordagem é validada com dados reais de repositórios do GitHub e os resultados indicam que ela é capaz de identificar com sucesso os principais especialistas em determinadas bibliotecas e frameworks. Os autores concluem que sua abordagem pode ser útil para empresas e comunidades de software que buscam identificar especialistas para contribuir com projetos ou fornecer suporte técnico.

[Brito et al. 2020] apresenta uma análise sobre as motivações para Breaking Changes (BCs) em Application Programming Interfaces (APIs), a partir da análise de mais de 4000 pull requests no Github. As soluções apresentadas incluem: identificação dos motivos para BCs, como mudanças de requisitos, correção de bugs e melhorias de performance; uma classificação dos tipos de BCs mais comuns, incluindo DABCs; e uma avaliação do impacto dos BCs em projetos dependentes das APIs, sugerindo a necessidade de cuidado na comunicação das mudanças e na manutenção de documentação clara e atualizada das APIs.

[Brito et al. 2018] apresenta uma ferramenta chamada APIDiff, que é capaz de detectar mudanças em APIs que podem quebrar aplicações clientes. A ferramenta compara duas versões da mesma API e gera um relatório indicando as mudanças encontradas e se elas podem ter impacto em aplicações clientes. O APIDiff usa análise estática de código-fonte e considera diferentes tipos de mudanças, incluindo mudanças em assinaturas de método, mudanças em tipos de retorno e mudanças em valores padrão de argumentos. Além disso, o artigo apresenta um estudo empírico que avalia a eficácia do APIDiff em detectar mudanças que podem quebrar aplicações clientes. Os resultados mostram que o APIDiff é capaz de detectar a maioria das mudanças relevantes, com uma taxa baixa de falsos positivos.

[Mezzetti et al. 2018] apresenta uma solução para detectar mudanças que possam quebrar o código em bibliotecas do Node.js usando testes de regressão de tipo. A ideia é executar testes em diferentes versões da biblioteca e comparar os resultados, verificando se houve alguma alteração no tipo das variáveis ou na assinatura das funções que possa causar problemas em código que depende da biblioteca. Os autores propõem uma ferramenta chamada Turtler, que automatiza o processo de execução de testes e análise de mudanças de tipo, mostrando quais versões da biblioteca são compatíveis com um determinado projeto.

[Grotov et al. 2022] apresenta uma análise comparativa de mais de 1,5 milhão de arquivos Python em dois formatos diferentes: notebooks Jupyter e scripts. Os autores investigam as diferenças e semelhanças entre os dois tipos de arquivos em termos de sintaxe, estilo, bibliotecas, documentação e qualidade do código. Eles também exploram os fatores que influenciam a escolha do formato, como o domínio, a finalidade e o público-alvo do código. O artigo mostra que os notebooks Jupyter são mais usados para análise

de dados, visualização e aprendizado de máquina, enquanto os scripts são mais usados para desenvolvimento de software, automação e testes. O artigo também revela que os notebooks Jupyter tendem a ter mais comentários, células de markdown e importações relativas, enquanto os scripts tendem a ter mais funções, classes e importações absolutas. Além disso, o artigo avalia a qualidade do código usando métricas como complexidade ciclomática, taxa de comentários e taxa de erros de sintaxe. O objetivo do artigo é contribuir para uma melhor compreensão dos benefícios e desafios de cada formato e fornecer recomendações para os desenvolvedores e pesquisadores que usam Python para diferentes tarefas.

## 3. Metodologia

A metodologia adotada neste estudo de caso utilizou técnicas de mineração de repositórios de software para identificar e analisar mudanças que quebram argumentos padrões em bibliotecas de software. Foram empregadas diversas técnicas para identificar as DABCs, incluindo análise de commits, análise de pull requests e análise de mudanças de API. A documentação também foi examinada em busca de referências a mudanças que quebram argumentos padrões, como anotações em arquivos de changelog ou notas de versão.

Os projetos selecionados passaram por um processo de triagem para garantir sua relevância para o estudo. Os critérios de seleção incluíram o tamanho do projeto, o objetivo do software, o uso da biblioteca em questão e como a documentação dos métodos é gerada. Além disso, os projetos precisavam ter um histórico de atualizações que permitisse a identificação de mudanças que quebram argumentos padrões e seu impacto nos projetos.

A análise do impacto das mudanças foi realizada por meio de perguntas-chave formuladas para orientar a análise dos dados coletados. Essas perguntas foram respondidas por meio da análise dos dados coletados nos repositórios dos projetos, incluindo registros de mudanças, análise de código e análise da documentação.

As análises consistiram na categorização dos casos encontrados e na quantificação em tabelas com valores recorrentes. Essa abordagem permitiu uma análise técnica e precisa dos efeitos das mudanças na biblioteca, assim como nos projetos que a utilizam. Além disso, a metodologia possibilitou a identificação de padrões de mudança e de problemas recorrentes em bibliotecas de software.

A metodologia adotada para a análise das Default Argument Breaking Changes (DABCs) envolveu a criação de uma Tabela de Descrição de DABCs, sendo este um processo realizado manualmente para identificação e catalogação das diversas instâncias identificadas. Essa tabela foi concebida como uma ferramenta essencial para o mapeamento detalhado das ocorrências de DABCs, proporcionando uma base estruturada para a condução da análise qualitativa.

Na fase de geração da Tabela de Resultados, foram realizadas análises aprofundadas nos notebooks de desenvolvimento. Durante esse processo, buscou-se identificar casos de uso relevantes associados às ocorrências de DABCs. A análise se estendeu à extração de estatísticas específicas de trechos de código, visando proporcionar uma compreensão abrangente das implicações das dependências antiéticas nas diferentes partes do projeto.

A criação manual da Tabela de Descrição de DABCs possibilitou uma abordagem detalhada e personalizada na identificação dessas ocorrências, garantindo a precisão e a contextualização necessárias para uma análise qualitativa robusta. A tabela serviu como um guia fundamental para a formulação de questões-chave e a condução de uma análise estatística mais aprofundada sobre os impactos das DABCs no código-fonte.

Ao analisar os notebooks, foram identificados casos de uso que forneceram insights valiosos sobre a natureza das ocorrências de DABCs. A busca por esses casos de uso teve como objetivo não apenas a identificação de padrões de utilização, mas também a compreensão de como as DABCs impactam o desenvolvimento e a funcionalidade do sistema.

As estatísticas do trecho de código, extraídas durante a análise dos notebooks, complementaram a compreensão global, fornecendo métricas quantitativas sobre a distribuição e frequência das DABCs ao longo do código-fonte. Esses dados estatísticos enriqueceram a análise qualitativa, permitindo uma visão mais holística das práticas de desenvolvimento e fornecendo uma base sólida para a tomada de decisões estratégicas na melhoria contínua do código.

Dessa forma, a metodologia adotada combinou a abordagem manual na criação da Tabela de Descrição de DABCs com análises automatizadas nos notebooks, resultando em uma compreensão abrangente das Default Argument Breaking Changes presentes na biblioteca em estudo. Essa abordagem híbrida proporcionou um equilíbrio entre a precisão na identificação das ocorrências e a eficiência na análise estatística, contribuindo para uma avaliação completa e contextualizada do impacto das DABCs no processo de desenvolvimento.

Na etapa subsequente do desenvolvimento, introduzimos duas ferramentas de automatização como prova de conceito, buscando otimizar a análise de código e a mineração de repositórios. Abaixo, detalhamos as características específicas de cada ferramenta:

Inicialmente, implementamos a construção de uma Árvore Sintática Abstrata (AST) como parte da nossa abordagem de análise de código automatizada. Essa estrutura proporciona uma representação organizada e hierárquica do código-fonte, facilitando a identificação de padrões e estruturas complexas.

Além disso, desenvolvemos mecanismos para a extração automatizada de dados do código, permitindo uma análise detalhada das características presentes. A ferramenta foi projetada especificamente para a identificação de Default Arguments (DAs), utilizando algoritmos especializados para reconhecimento eficiente dessas dependências.

A extração de informações relevantes foi incorporada à ferramenta, possibilitando a obtenção de dados específicos relacionados ao DAs. Esses dados são então organizados em uma estrutura de dados pertinente, facilitando a interpretação e análise das informações coletadas.

A segunda ferramenta concentrou-se na mineração de repositórios, oferecendo uma abordagem automatizada para analisar o histórico de desenvolvimento de uma biblioteca específica. A partir da especificação da URL do repositório, a ferramenta é capaz de extrair informações relevantes.

A definição de um range de commits possibilita uma análise focalizada em períodos específicos do histórico de desenvolvimento, proporcionando uma compreensão mais detalhada das alterações realizadas ao longo do tempo.

A ferramenta identifica modificações, adições e remoções de código, fornecendo uma visão abrangente das evoluções no repositório. Adicionalmente, realiza a extração automatizada do código-fonte, tanto antes quanto depois de cada modificação, permitindo uma análise comparativa das alterações realizadas ao longo do tempo.

Essas ferramentas, concebidas como prova de conceito, representam uma abordagem inovadora para automatizar aspectos-chave da análise de código e da mineração de repositórios. Esta implementação inicial serve como base sólida para futuras iterações e refinamentos, visando alcançar uma automação mais robusta e completa. A integração dessas ferramentas na metodologia fortalece a capacidade de análise, fornecendo insights valiosos para o entendimento e aprimoramento contínuo do processo de desenvolvimento da biblioteca em estudo.

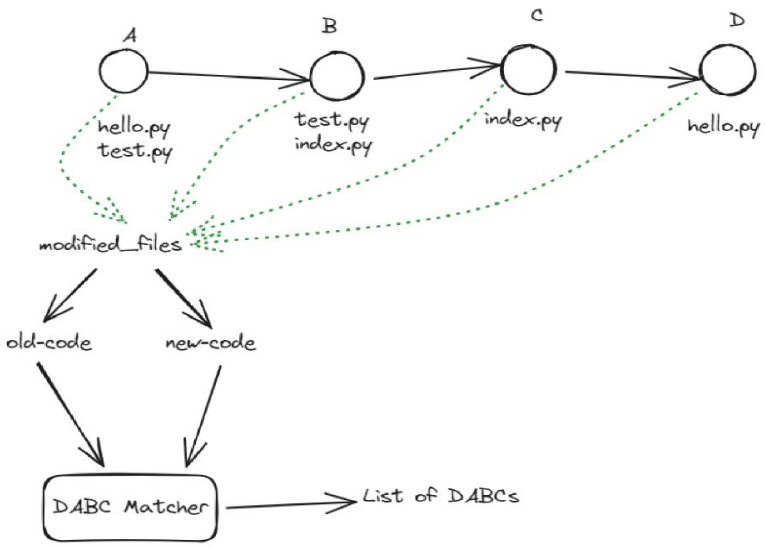


**Figura 1. Modelagem da estrutura e funcionamento das ferramentas.**

## 4. Resultados

O desenvolvimento deste estudo envolveu várias etapas. Inicialmente, realizamos uma busca por mudanças na biblioteca utilizando o termo ".. versionchanged::". Em seguida, identificamos as mudanças no código, os registros de mudança e a justificativa para as mudanças que quebraram os argumentos padrões (DABC).

Para extrair os commits do repositório, desenvolvemos um script que gerou uma tabela com informações relevantes. Essas informações incluíam o caminho do arquivo, a linha em que ocorreu a mudança, a descrição da mudança, a versão correspondente, o link para o arquivo, o commit relacionado, a mensagem do commit, o nome e o email do autor.

Além disso, exploramos o conjunto de dados de Notebooks Jupyter do artigo [Grotov et al. 2022]. Esse conjunto de dados inclui todos os Notebooks Jupyter do

GitHub em novembro de 2020, totalizando 9.719.569 arquivos. Após aplicar filtros para selecionar apenas os notebooks com licenças permissivas e escritos em Python, restaram 847.881 arquivos. Focamos nossa análise nos notebooks que importavam a biblioteca NumPy.

Com base nos dados coletados e explorados, realizamos análises e obtivemos insights sobre as mudanças que quebraram os argumentos padrões, além de fazer comparações entre o código nos Notebooks Jupyter e nos scripts.

Os resultados parciais da análise qualitativa revelam uma redução significativa de aproximadamente 100 vezes no número de casos identificados em comparação com a biblioteca NumPy. Essa disparidade sugere particularidades distintas entre as bibliotecas analisadas, possivelmente influenciadas por métodos altamente específicos presentes na biblioteca em foco e pela filosofia única adotada. A observação de um número relativamente limitado de DABCs sugere uma abordagem mais conservadora ou uma menor propensão a alterações de argumentos padrão na filosofia de desenvolvimento dessa biblioteca específica.

Após o desenvolvimento das ferramentas de automatização, obtivemos resultados promissores na extração e organização das Default Argument Breaking Changes (DABCs). A primeira ferramenta permitiu a estruturação dessas mudanças em dicionários de dicionários, onde as chaves são associadas a classes, métodos, argumentos e seus respectivos valores padrão. Essa estrutura detalhada proporcionou uma visão sistemática e organizada das DABCs, facilitando análises mais aprofundadas. Já a segunda ferramenta demonstrou eficácia ao extrair o código antes e depois de mudanças específicas de commits, dentro de um range de hashes determinado. Essa capacidade de rastrear a evolução do código ao longo do tempo permitiu disponibilizar informações cruciais para a ferramenta de análise de código, possibilitando a comparação e identificação eficaz de potenciais DABCs ou modificações significativas.

## 5. Conclusão

**Tabela 1. Resultado da extração de dados das bibliotecas**

| Métrica | NumPy | Pandas | Scikit-Learn |
|---|---|---|---|
| DABCs | 5 | 13 | 77 |
| Imports no dataset | 584.995 | 348.889 | 251.813 |
| Linhas de código | 604.019 | 669.836 | 377.720 |
| Commits na main | 32.714 | 32.678 | 30.101 |

Primeiramente, é importante ressaltar que tanto Pandas quanto Scikit-Learn apresentaram um número significativamente maior de DABCs em comparação com NumPy. Enquanto NumPy registrou apenas 5 DABCs, Pandas teve 13 e Scikit-Learn liderou com 77 ocorrências desse tipo de mudança. Isso indica que as atualizações nessas bibliotecas podem ter um impacto maior em projetos existentes que as utilizam, exigindo uma atenção especial ao realizar atualizações.

Em relação aos imports no dataset, observamos que NumPy foi a biblioteca mais utilizada, com 584.995 imports registrados. Pandas veio em seguida, com 348.889 im-

ports, e Scikit-Learn teve 251.813 imports no dataset analisado. Isso evidencia a popularidade e ampla adoção dessas bibliotecas em projetos que envolvem análise de dados e aprendizado de máquina.

A análise dos dados revelou uma relação inversamente proporcional entre o número de DABCs (Default Argument Breaking Changes) e a quantidade de imports no dataset. Embora NumPy tenha sido a biblioteca mais utilizada, apresentou um menor número de DABCs em comparação com Pandas e Scikit-Learn, que, mesmo com menos imports, demonstraram uma frequência relativamente maior de mudanças em seus argumentos padrões. Essa observação ressalta a importância de considerar não apenas a quantidade de imports, mas também a qualidade das atualizações nas bibliotecas para garantir a compatibilidade e estabilidade dos projetos.

Quanto ao número de linhas de código, Pandas apresentou o maior volume, totalizando 669.836 linhas. Em seguida, temos NumPy com 604.019 linhas e Scikit-Learn com 377.720 linhas. Essa diferença pode indicar a complexidade e extensão das funcionalidades oferecidas por cada biblioteca, refletindo nas necessidades de manutenção e gerenciamento do código.

Analisando os commits na branch principal (main), observamos uma quantidade semelhante para Pandas e NumPy, com 32.678 e 32.714 commits, respectivamente. Scikit-Learn registrou um pouco menos, com 30.101 commits. Esses números sugerem um nível de atividade e contribuição significativo por parte da comunidade de desenvolvedores em todos os projetos.

Em suma, os dados da tabela fornecem insights valiosos sobre as bibliotecas NumPy, Pandas e Scikit-Learn. Essas informações podem ser usadas para compreender melhor o impacto das mudanças, a popularidade e o nível de atividade em cada biblioteca, auxiliando no desenvolvimento e na manutenção de projetos que as utilizam.

Os resultados obtidos nas análises qualitativas apontam para uma notável diferença no número de Default Argument Breaking Changes (DABCs) identificadas, aproximadamente 100 vezes menor em comparação com a biblioteca NumPy. Essa disparidade reflete não apenas as nuances intrínsecas da biblioteca em questão, evidenciadas por métodos altamente específicos, mas também sugere uma influência direta da filosofia de design adotada. A observação de um número reduzido de DABCs indica uma abordagem mais conservadora ou uma menor predisposição a mudanças significativas nos argumentos padrão nessa biblioteca específica. Essas conclusões destacam a importância de considerar as particularidades de cada biblioteca ao avaliar a presença e o impacto de DABCs, fornecendo insights valiosos para o entendimento e aprimoramento contínuo do processo de desenvolvimento de software.

As ferramentas de automatização desenvolvidas representam contribuições substanciais para a análise de Default Argument Breaking Changes (DABCs). A primeira ferramenta proporcionou uma abordagem estruturada ao organizar as DABCs em dicionários de dicionários, oferecendo uma visão detalhada e sistemática das mudanças nos argumentos padrão. Essa organização refinada facilitou a interpretação e análise das DABCs, promovendo uma compreensão mais profunda de seu impacto. Simultaneamente, a segunda ferramenta, ao extrair o código antes e depois de mudanças específicas, viabilizou uma análise comparativa fundamental para identificar e contextualizar DABCs ao longo

do histórico de desenvolvimento. A integração bem-sucedida dessas ferramentas fortaleceu a capacidade de análise, proporcionando insights valiosos para o entendimento e aprimoramento contínuo do processo de desenvolvimento de software, particularmente no contexto de dependências antiéticas.

## Referências


Brito, A., Valente, M. T., Xavier, L., and Hora, A. (2020). You broke my code: Understanding the motivations for breaking changes in APIs. *Empirical Software Engineering*, 25:1458–1492.

Brito, A., Xavier, L., Hora, A., and Valente, M. T. (2018). APIDiff: Detecting API breaking changes. In *25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Tool Track*, pages 507–511.

Ferreira, F., Silva, L. L., and Valente, M. T. (2020). Software engineering meets deep learning: A mapping study. *Journal of Systems and Software*, 163:110566.

Grotov, K., Titov, S., Sotnikov, V., Golubev, Y., and Bryksin, T. (2022). A large-scale comparison of python code in jupyter notebooks and scripts.

Mezzetti, G., M, A., and Torp, M. T. (2018). Type Regression Testing to Detect Breaking Changes in Node.js Libraries. In Millstein, T., editor, *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*, volume 109 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:24, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Montandon, J. E., Silva, L. L., and Valente, M. T. (2020). Identifying experts in software libraries and frameworks among github users. *Empirical Software Engineering*, 25(1):1–38.