Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

João Vítor Bicalho da Silva

MONOGRAFIA EM SISTEMAS DE INFORMAÇÃO - PESQUISA TECNOLÓGICA

Desenvolvimento de uma extensão para o VS Code para identificação de *Breaking Changes* em Parâmetros *Default*

Orientador
João Eduardo Montandon

1. Introdução

Nos últimos anos, a área da Computação e Programação apresentou um crescimento significativo, sendo impulsionado pelo avanço tecnológico e pela demanda por profissionais da área. Dessa forma, a habilidade de programar e gerar códigos para diversos objetivos, como, por exemplo, desenvolvimento de software e análise de dados, se torna essencial para quem almeja entrar para essa área.

Desse modo, para facilitar esse trabalho, o uso de editores de código tornou-se uma prática comum entre desenvolvedores, destacando-se o Visual Studio Code (VS Code) como uma das ferramentas mais populares e versáteis. Com funcionalidades que incluem autocompletar, depuração, controle de versão e integração de extensões, o VS Code é amplamente adotado para facilitar e otimizar o processo de desenvolvimento. Além disso, é prática comum entre os programadores o uso de frameworks e bibliotecas externas, que trazem funcionalidades pré-construídas e contribuem significativamente para a agilidade no desenvolvimento de software.

No entanto, o uso dessas bibliotecas e frameworks externos também apresenta desafios. Muitas dessas ferramentas estão em constante atualização, e a incorporação de uma versão desatualizada de uma biblioteca em um projeto pode resultar em problemas, pois mudanças em suas funções ou estrutura podem afetar o comportamento esperado do programa. Esse tipo de inconsistência pode prejudicar o desempenho, a segurança e a estabilidade de aplicações, criando a necessidade de soluções que ajudem a acompanhar e gerenciar essas atualizações de maneira eficiente.

Dessa forma, o presente trabalho tem como objetivo desenvolver uma extensão para o VS Code que identifique as bibliotecas e funções externas presentes no código que sofreram alterações em atualizações recentes e que possam impactar o comportamento padrão do programa. A ferramenta proposta busca auxiliar desenvolvedores a manterem seus projetos alinhados com as últimas versões das bibliotecas utilizadas, minimizando possíveis conflitos e garantindo a robustez e atualidade do código desenvolvido.

2. Referencial Teórico

2.1. Extensões do Visual Studio Code (VS Code)

Extensões para o Visual Studio Code (VS Code) são pequenos pacotes ou plugins que adicionam funcionalidades ao editor, personalizando e melhorando a experiência de desenvolvimento. Elas são projetadas para estender as capacidades padrão do VS Code, permitindo que os usuários adaptem o editor às suas necessidades específicas.

Entre os recursos disponibilizados, destacam-se:

- **Apoio a linguagens de programação**: adicionam suporte para sintaxe, auto-completar e formatação para linguagens específicas (por exemplo, Python, JavaScript, C#, etc.).
- **Ferramentas de depuração**: auxiliam na depuração de código em várias linguagens e frameworks.
- **Integração com controle de versão**: facilitam o uso de sistemas como Git e GitHub, integrando-os diretamente ao VS Code.
- **Produtividade e automação**: oferecem funcionalidades como snippets, atalhos personalizados e geração de código para acelerar o desenvolvimento.
- **Temas e customização de interface**: alteram a aparência do editor, incluindo esquemas de cores, fontes e layout.

2.2. Breaking Changes

*Breaking change*s se referem a alterações em um sistema, código, API ou biblioteca que, ao serem implementadas, causam incompatibilidades com versões anteriores. Essas mudanças podem gerar problemas para o desenvolvedor, pois exigem que o código ou as dependências que utilizavam a versão anterior sejam atualizados ou adaptados para evitar erros.

São exemplos de *Breaking Changes*:

- **Remoção de Funções ou Métodos**: Funções que existiam em uma versão anterior e foram removidas, o que faz com que chamadas a essas funções resultem em erro.
- **Mudanças de Parâmetros**: Modificar o número, tipo ou ordem de parâmetros de uma função.
- **Alterações em Estruturas de Dados**: Modificar a estrutura de objetos ou classes, como renomear atributos ou remover elementos.
- **Mudança na Interface de uma API**: Modificar o comportamento de uma API ou seus endpoints.
- **Alterações de Comportamento**: Alterações no funcionamento esperado de uma função ou método, o que faz com que o código deva ser ajustado para lidar com o novo comportamento.

3. Resultados MSI I

Ao final da MSI I, foi possível alcançar resultados significativos com a criação de dois protótipos funcionais, que, de certa forma, se complementam.

O primeiro protótipo é responsável por identificar chamadas de funções no código, destacando-as visualmente e exibindo uma mensagem contextual ao passar o mouse sobre elas. Esse recurso contribui para uma navegação mais intuitiva e dinâmica no código, facilitando a identificação de funções.

Já o segundo protótipo, mais avançado, está integrado com uma ferramenta de análise de código. Ele não apenas identifica chamadas de funções, mas também exibe uma mensagem contendo a assinatura completa da função. Além disso, essa assinatura é capturada para potencial uso em outras funcionalidades no futuro. No entanto, esse protótipo ainda apresenta uma limitação importante: ele é capaz de capturar apenas as assinaturas de funções declaradas no próprio arquivo. Chamadas de funções provenientes de bibliotecas externas não são processadas corretamente, o que restringe sua aplicação em cenários mais complexos.

Embora essa limitação precise ser abordada em etapas futuras do projeto, o progresso obtido com esses protótipos já representa um avanço significativo em direção aos objetivos finais, demonstrando o potencial das ferramentas e técnicas aplicadas.

Apesar dos desafios, como foi detalhado na seção de Desenvolvimento, foi possível obter 2 protótipos funcionais que de certa forma se complementam. O primeiro identifica chamadas de funções e coloca um destaque sobre elas, além de exibir uma mensagem ao passar o mouse por cima.

Já o segundo, já integrado com a ferramenta de análise de código, identifica a chamada de uma função e exibe uma mensagem com a sua assinatura, além de capturá-la para utilizar de alguma outra forma. Entretanto, como foi dito, esse protótipo possui a limitação de capturar as assinaturas apenas das funções declaradas no arquivo, não funcionando corretamente para chamadas de funções de bibliotecas externas.

4. Desenvolvimento

Com base nos resultados obtidos na primeira etapa da monografia, o objetivo desta fase foi complementar o protótipo inicial, incorporando as funcionalidades necessárias para detectar *Breaking Changes* em chamadas de funções, com foco especial nos parâmetros críticos definidos previamente

Como resultado, a versão atual da extensão para o Visual Studio Code possui as seguintes funcionalidades principais:

- Detecção de chamadas de funções;
- Identificação de parâmetros passados por nome ou posição;
- Verificação da presença de parâmetros críticos (conforme definidos pela DABC);

- Exibição de mensagens informativas por meio de hover (passar o cursor);
- Destaque visual (sublinhado) em chamadas que omitem o parâmetro crítico.

Essas funcionalidades foram implementadas por meio de um algoritmo de análise estática simples, porém eficaz, conforme descrito a seguir.

Algoritmo de detecção de *Breaking Changes*

O algoritmo utilizado pela extensão segue uma série de etapas sequenciais para identificar chamadas de funções e verificar a ocorrência de possíveis breaking changes com base em parâmetros críticos. A seguir, são descritas as etapas principais do processo:

1. Captura do Código

Sempre que o documento é modificado ou o usuário alterna entre abas no editor, o conteúdo completo do código-fonte é lido e armazenado em tempo real. Esse conteúdo serve de base para a análise posterior.

2. Identificação de Chamadas de Função

Através de expressões regulares, a extensão localiza todas as chamadas de função no texto. Em seguida, o nome de cada função é extraído individualmente. Para cada uma delas, é verificado se está presente em uma lista de funções conhecidas com risco potencial de breaking change.

3. Verificação de Parâmetros

Para cada função conhecida, a extensão armazena localmente sua definição, incluindo os nomes e a ordem dos parâmetros esperados.

4. Classificação da Chamada

Com base na análise anterior, a chamada de função é classificada da seguinte forma:

- Função desconhecida: Caso a função não esteja na lista de funções monitoradas, nenhuma marcação é aplicada.
- Função conhecida, parâmetro ausente: Se a função está na lista e o parâmetro crítico não foi passado, a chamada é sublinhada para chamar atenção do desenvolvedor.
- Parâmetro presente (nomeado ou posicional): Se o parâmetro crítico foi identificado corretamente, nenhuma marcação é aplicada.

5. Exibição de Feedback via Hover

Quando o usuário posiciona o cursor do mouse sobre uma chamada de função, a extensão exibe um tooltip com um resumo informativo, contendo:

- Nome da função;
- Lista dos parâmetros identificados na chamada;
- Indicação clara se o parâmetro crítico foi passado ou não.

5. Desafios Encontrados

Ao longo do desenvolvimento, enfrentei diversos desafios, sendo o principal a falta de informações específicas na documentação oficial sobre algumas ferramentas necessárias para o projeto. Apesar de a documentação ser abrangente e bem estruturada em termos gerais, determinados métodos e funções essenciais para atender aos requisitos do projeto não eram detalhados e exemplificados de maneira clara. Isso exigiu uma abordagem de tentativa e erro durante boa parte do processo de desenvolvimento. Cada avanço foi fruto de experimentação: à medida que uma solução ou funcionalidade se mostrava promissora, eu a explorava e refinava para alcançar os resultados esperados.

6. Resultados

Apesar dos desafios enfrentados ao longo do processo de desenvolvimento — especialmente no que diz respeito à análise de chamadas de funções externas e ao tratamento de parâmetros posicionais — foi possível alcançar resultados significativos com a implementação do plugin proposto.

A extensão desenvolvida demonstrou-se eficaz na detecção de potenciais ocorrências de *breaking changes*, especialmente em contextos onde o uso de bibliotecas externas está sujeito a modificações críticas em suas funções. O plugin é capaz de identificar chamadas de funções no código-fonte, verificar se essas funções possuem parâmetros considerados críticos (de acordo com a base de dados utilizada) e alertar o desenvolvedor de forma clara e não intrusiva, por meio de anotações visuais e mensagens contextuais.

Entre os principais resultados obtidos, destacam-se:

- Precisão na detecção de funções e parâmetros: A extensão foi capaz de localizar com alta confiabilidade as chamadas de funções e identificar os parâmetros passados, seja por nome ou por posição.

- Destaque visual inteligente: Chamadas que omitem parâmetros críticos são automaticamente sublinhadas no código, permitindo rápida identificação de possíveis inconsistências.

- Feedback contextual via hover: Ao posicionar o cursor sobre uma chamada de função, o desenvolvedor recebe um resumo contendo a assinatura da função e uma indicação clara sobre a presença (ou ausência) do parâmetro crítico.

- Atualização em tempo real: A análise é executada dinamicamente, reagindo a alterações no conteúdo do arquivo ou na navegação entre abas, sem

necessidade de ações manuais por parte do usuário.

- Arquitetura extensível: O design modular do plugin permite a ampliação futura para suportar outras linguagens, outras bibliotecas, e até mesmo diferentes critérios de verificação (como tipos de retorno, valores padrão, ou novos tipos de breaking changes).

Esses resultados mostram que o plugin cumpre bem sua proposta de auxiliar na manutenção de projetos, oferecendo alertas úteis sobre possíveis mudanças nas dependências externas. A extensão ajuda a identificar pontos que merecem atenção, contribuindo para manter o código mais estável e atualizado no dia a dia do desenvolvimento.

```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

# Load dataset
ds = datasets.fetch_20newsgroups_vectorized()
X = ds.data[:, 2:]
y = ds.target

# Create training/test data split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42, stratify=y)

# Create an instance of SVC Classifier
# DABC ESTÁ AQUI, PARÂMETRO GAMMA NÃO FOI INICIALIZADO
clf = SVC(random_state=42)

# Fit, predict, and measure model's performance
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('Acc: %.3f' % accuracy_score(y_test, y_pred))
```

Figura 1: exemplo de codigo python

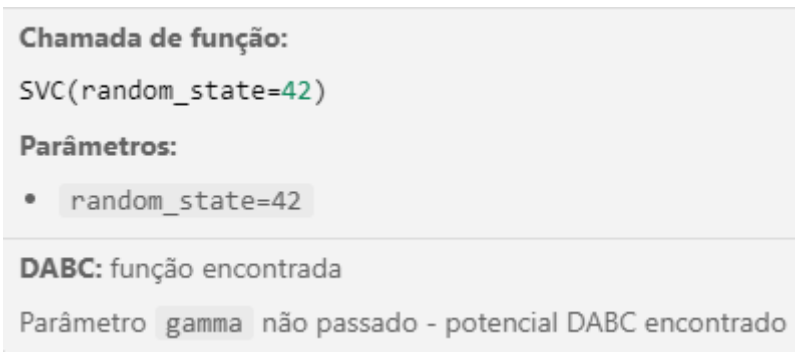

Figura 2: Hover exibido no caso em que o parâmetro não é passado

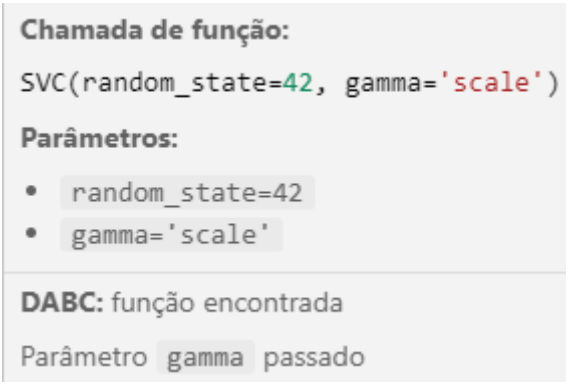

Figura 3: Hover exibido no caso em que o parâmetro é passado

Por fim, destaca-se que este trabalho serviu de base para o desenvolvimento do artigo intitulado *"DABCheck: Um Plugin para Detecção de Mudanças Disruptivas em Argumentos-Padrão de Bibliotecas Python"*, submetido à trilha de ferramentas do Simpósio Brasileiro de Engenharia de Software 2025 (SBES 2025). Essa submissão reforça a relevância prática e acadêmica da proposta, bem como seu potencial de contribuição para a comunidade de engenharia de software.

7. Próximos Passos

Com a finalização da primeira versão funcional da extensão, abre-se espaço para o aprofundamento e a expansão de suas capacidades. Um dos caminhos mais promissores para a evolução da ferramenta está na ampliação do seu suporte para outras bibliotecas Python. Atualmente, a análise de *breaking changes* se baseia em um conjunto restrito de funções previamente mapeadas. A inclusão de novas bibliotecas amplamente utilizadas, como pandas, numpy, matplotlib e scikit-learn, permitiria à extensão abranger um conjunto mais representativo de aplicações reais, tornando sua utilização mais útil e escalável para diferentes contextos de desenvolvimento em Python.

Outro possível avanço está relacionado à adaptação da extensão para outras linguagens de programação que, assim como o Python, fazem uso extensivo de bibliotecas externas e frequentemente enfrentam problemas decorrentes de mudanças disruptivas em suas APIs. Linguagens como JavaScript ou TypeScript, por exemplo, poderiam se beneficiar de uma abordagem semelhante. Essa expansão exigiria a reimplementação dos mecanismos de detecção de chamadas e extração de parâmetros conforme as particularidades sintáticas de cada linguagem, mas manteria a lógica central da ferramenta. Esse tipo de generalização representaria um passo importante em direção à consolidação da extensão como uma solução mais ampla para o monitoramento de breaking changes em ambientes de desenvolvimento modernos.

Essas possíveis evoluções reforçam o potencial da ferramenta como uma aliada na manutenção da estabilidade e da atualidade dos projetos de software, especialmente em cenários onde as bibliotecas externas evoluem com rapidez e introduzem alterações que impactam diretamente o comportamento do código.

8. Referências Bibliográficas

MICROSOFT. *Visual Studio Code Extension API Documentation*. Disponível em: https://code.visualstudio.com/api. Acesso em: 25 out. 2024.

Montandon, J. E., Silva, L. L., Politowski, C., El Boussaidi, G., & Valente, M. T. (2023, October). Unboxing default argument breaking changes in Scikit Learn. In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 209-219). IEEE.

Montandon, J. E., Silva, L. L., Politowski, C., Prates, D., Bonifácio, A. D. B., & Boussaidi, G. E. (2024). Unboxing Default Argument Breaking Changes in 1+ 2 Data Science Libraries.

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