IFFT: Uma ferramenta de alerta de código para alterações coordenadas

Aluno: Thiago Henrique Moreira Santos
Orientador: André Hora

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

thiago.santos@dcc.ufmg.br

Abstract. *O "If-Flow-Then"(IFFT) é uma ferramenta projetada para gerenciar dependências dentro de projetos de desenvolvimento de software. A ideia principal é ajudar os desenvolvedores na coordenação de alterações de código em arquivos de projetos de software de larga escala. Diferente dos "linters"clássicos, o IFFT se propõe na identificação de dependências entre arquivos e na notificação dos desenvolvedores quando modificações em segmentos de código específicos podem necessitar de alterações em outras partes relacionados do projeto. A ferramenta se integra muito bem ao fluxo de trabalho do desenvolvedor, oferecendo recursos como "pre-commit hooks", uma representação em grafos das dependências e uma "interface web"para analisar os resultados.*

O desenvolvimento do IFFT envolveu a implementação de um "backend"capaz de analisar os blocos de dependências, rastrear alterações e gerenciar os metadados dos blocos, em conjunto com um "front end"dinâmico que inclui visualizações mais intuitivas da saída da ferramenta do que a versão de terminal. Dentre os desafios no desenvolvimento dessa segunda etapa do projeto estão: lidar com a escalabilidade do projeto e criar uma experiência de usuário intuitiva, que foram abordados por meio de soluções criativas como a criação de representações simplificadas dos blocos e utilizando bibliotecas de visualização interativa como o "Vis.js".

O IFFT tem o potencial de reduzir significativamente a supervisão no gerenciamento de dependências, melhorando a qualidade do código e a manutenibilidade do projeto. Este ***relatório técnico*** *descreve a motivação, ao mesmo tempo em que discute suas contribuições e possíveis caminhos para a continuidade do projeto.*

1. Introdução

No cenário moderno de desenvolvimento de software, gerenciar dependências, especialmente em projetos de larga escala, é um desafio muito relevante. Com o aumento da complexidade das "codebases"e a natureza colaborativa inerente aos times de desenvolvimento, garantir que as mudanças em uma parte do código se alinhem com os componentes relacionados dentro do código se tornou uma tarefa crítica. Atualizações sem o cuidado na coordenação de dependências normalmente levam a erros em tempo de execução e comportamentos incorretos dentro da aplicação, isso acaba gerando um aumento no custo de manutenção.

Para endereçar esses problemas, ferramentas como ”linters”e ferramentas de análise estática têm sido amplamente adotadas. Essas ferramentas auxiliam desenvolvedores identificando potenciais problemas de código, ajudar a reforçar padrões e alertar inconsistências. Contudo, as soluções existentes atualmente focam primariamente na análise ”intra-arquivo”e faltam mecanismos para assistir com dependências ”inter-arquivo- essa lacuna se torna particularmente evidente em projetos com arquiteturas fortemente modulares ou projetos que necessitam de condições transversais dentro dos módulos.

A ferramenta IFFT foi concebida para preencher essa lacuna. O IFFT introduz uma nova abordagem para gerenciamento de dependências ao focar em relações ”inter-arquivo”. Ele aproveita os blocos IFFT definidos pelo desenvolvedor para identificar regiões de código que, quando modificadas, podem exigir alterações correspondentes em outra seção de código ou fornecer uma análise abrangente por meio de uma ”interface web”. O IFFT visa reduzir a supervisão e agilizar o fluxo de trabalho com respeito ao gerenciamento de dependências em grandes projetos.

O desenvolvimento do IFFT seguiu uma abordagem estruturada, abrangendo funcionalidades essenciais como:

Análise de Blocos de Dependências A capacidade de identificar e analisar blocos IFFT definidos pelo usuário dentro do código.

Gerenciamento de Metadados Armazenar e recuperar informações eficientemente sobre dependências rastreadas.

Integração com Ferramentas de Desenvolvedor Recursos como ”pre-commits hooks”do Git para integrar perfeitamente aos fluxos de trabalho existentes.

Interface Web Fornecer uma UI para analisar saídas, incluindo uma representação gráfica de dependências e resultados detalhados.

O presente relatório técnico é dividido nas seguintes seções: Motivação, Metodologia, Detalhes de Implementação e, por fim, a Conclusão do projeto. As seções a seguir fornecerão uma discussão detalhada dos recursos da ferramenta, desafios encontrados durante o desenvolvimento e seu impacto potencial nas práticas modernas de Engenharia de Software.

2. Motivação e Caracterização do Trabalho

O desenvolvimento de software moderno demanda altos níveis de modularização e colaboração, levando a um aumento de ”codebases”independentes e distribuídas. Enquanto a modularidade provê flexibilidade e manutenibilidade, ela também introduz um grande desafio: garantir que todas as mudanças numa parte da ”codebase”não introduzam inadvertidamente inconsistências ou erros em componentes relacionados.

2.1. O desafio do gerenciamento de dependências

Em projetos de larga escala, dependências entre arquivos normalmente surgem de lógicas compartilhadas, configurações ou funcionalidades. Quando essas dependências não são

gerenciadas de forma adequada, isso pode levar a problemas como:

- **Erros não detectados:** Uma modificação em um arquivo pode requerer mudanças complementares em outras partes do código e falhar em gerenciar essas dependências pode levar a erros futuros.
- **Aumento no custo de manutenção:** Desenvoldedores normalmente passam parte significamente do seu tempo diagnosticando problemas causados por dependências.
- **Sobrecarga na coordenação dos times:** Em projetos colaborativos, os membros do time podem não ter visibilidade na forma com que mudanças afetam outras partes do código, levando a ineficiências no fluxo de trabalho.

2.2. Exemplo Ilustrativo

Imagine um projeto com um módulo que gerencia a autenticação dos usuários (auth.py) e um módulo relacionado que gerencia notificações de e-mail (email.py). Uma mudança pequena na lógica de autenticação pode necessitar de uma mudança complementar no módulo de notificações de e-mail. Ao deixar escapar essa possível coordenação das mudanças, erros em tempo de execução podem aparecer na medida em que o sistema evolui, por exemplo, esses erros poderiam aparecer no sistema de notificação quando novos usuários forem adicionados.

2.3. Como as ferramentas atuais funcionam?

As ferramentas atuais, como ”linters”tradicionais ou analisadores estáticos, focam primariamente na corretude de um arquivo individualmente. Ainda que isso seja uma preocupação importante, isso não resolve os seguintes problemas:

- **Dependências ”Inter-arquivo”:** Essas ferramentas raramente dão informações sobre como os arquivos interagem ou dependem entre si.
- **Alertas Proativos:** Desenvolvedores normalmente tem conhecimento sobre problemas de dependência quando erros em tempo de execução são constatados ou em testes, uma abordagem reativa.

Tabela 1. Comparação do IFFT com outros linters conhecidos

Feature/Tool	**IFFT**	**ESLint/Prettier**	**SonarQube**
Propósito	Monitorar dependências no projeto	Garantir corretude de sintaxe e estilo	Identifica problemas de qualidade de código
Escopo	Lógica de dependência entre arquivos	Syntax e estilo intra-arquivo	Qualidade de código, bugs e vulnerabilidades
Linguagens Suportadas	Inicialmente Python	Primariamente Javascript e Typescript	Suporte multi-linguagem (Java, Python, C++, etc)
Configuração	Regras customizadas de como rastrear dependências	Configurações de estilo baseada em regras	Regras para obtenção de qualidade de código
Visualização	Visualização das dependências por grafo interativo	Não-aplicável	Reports e dashboard

2.4. O que o IFFT propõe?

O IFFT oferece uma nova abordagem preventiva para gerenciar dependências.

1. Monitoramento a partir de blocos de dependência. Permite desenvolvedores anotar regiões do código (blocos IFFT) e especificar os relacionamentos deles com outros arquivos ou componentes.
2. Notificações Proativas Durante a fase anterior ao ”commit”ou na fase de análise, o IFFT identifica e alerta desenvolvedores sobre uma potencial descoordenação em pontos dependentes.
3. Visualizações Gráficas Através da sua ”interface web”, o IFFT oferece informações sobre as dependências por meio de recursos visuais como tabelas, grafos interativos, dentre outros.

2.5. Objetivos

Os objetivos para este projeto foram:

- Construir uma ferramenta capz de identificar dependências inter-arquivos de forma proativa e eficiente.
- Criar uma interface ”user-friendly”para analizar e entender estruturas de dependências dentro do projeto.
- Integrar-se facilmente com fluxos de trabalhos existentes, incluindo ”git hooks”e pipelines de CI/CD.

3. Esclarecimento: IFFT e o Acoplamento de Código

A primeira vista, o IFFT pode parecer como uma ferramenta onde sua utilidade é condicionada a aplicações com alto grau de acoplamento, o que, no mundo da engenharia de software é geralmente visto como uma falha de ”design”. Aplicações fortemente acopladas são aplicações em que os componentes são fortemente dependentes entre si. Mudanças em uma parte do código repercutem em várias outras partes do código, aumentando a complexidade, reduzindo a modularidade e fazendo com que a manutenção do código se torne uma tarefa muito complicada. As melhores práticas da engenharia de software advogam para que sistemas tenham baixo acoplamento e alta coesão, onde componentes interagem minimamente e são independentemente funcionais.

Então, porque uma ferramenta como o IFFT seria útil? Não seria o caso em que ao adotar melhores práticas dentro do projeto, como eliminar essas dependências, faria com que o IFFT se tornasse obsoleto? A realidade é mais sutil do que parece. Ainda que reduzir o acoplamento seja quase sempre uma boa ideia, existem diversos cenários onde isso não é desejável ou mesmo possível!

3.1. O real valor do IFFT

O IFFT não foi feito para ser um ”suporte”para um design ruim de projeto, mas sim para auxiliar os desenvolvedores no gerenciamento de dependências de forma eficaz onde as dependências são necessárias ou até mesmo inevitáveis.

Em projetos de larga escala, ou projetos legado, o desacoplamento completo pode ser proibitivamente caro, arriscado ou até mesmo impossível. Estes sistemas geralmente possuem decisões arquiteturais históricas que foram adequadas para o momento em que

foram construídos, mas que agora trazem desafios. Por exemplo, refatorar um código de décadas de idade que lida com um aspecto crítico de um empreendimento pode envolver um tempo de indisponibilidade significativo ou graves riscos à integridade da aplicação, fazendo com que a refatoração completa seja inviável. O IFFT está mais interessado em contextos como esse, provendo aos desenvolvedores um auxílio para ”trackear”e gerenciar essas dependências.

3.2. Quando dependências são intrínsecas, um exemplo ilustrativo

Nem todas as dependências são ruins. Algumas delas refletem aspectos lógicos do relacionamento inerente entre domínios da aplicação, tanto é que existe um padrão arquitetural largamente utilizado chamado de ”injeção de dependência”. Por exemplo, considere um sistema onde:

- **Arquivo A**: Gerencia a lógica que guarda consultas dos usuários.
- **Arquivo B**: Gerencia a lógiica que faz algumas análises a partir dos dados dos usuários

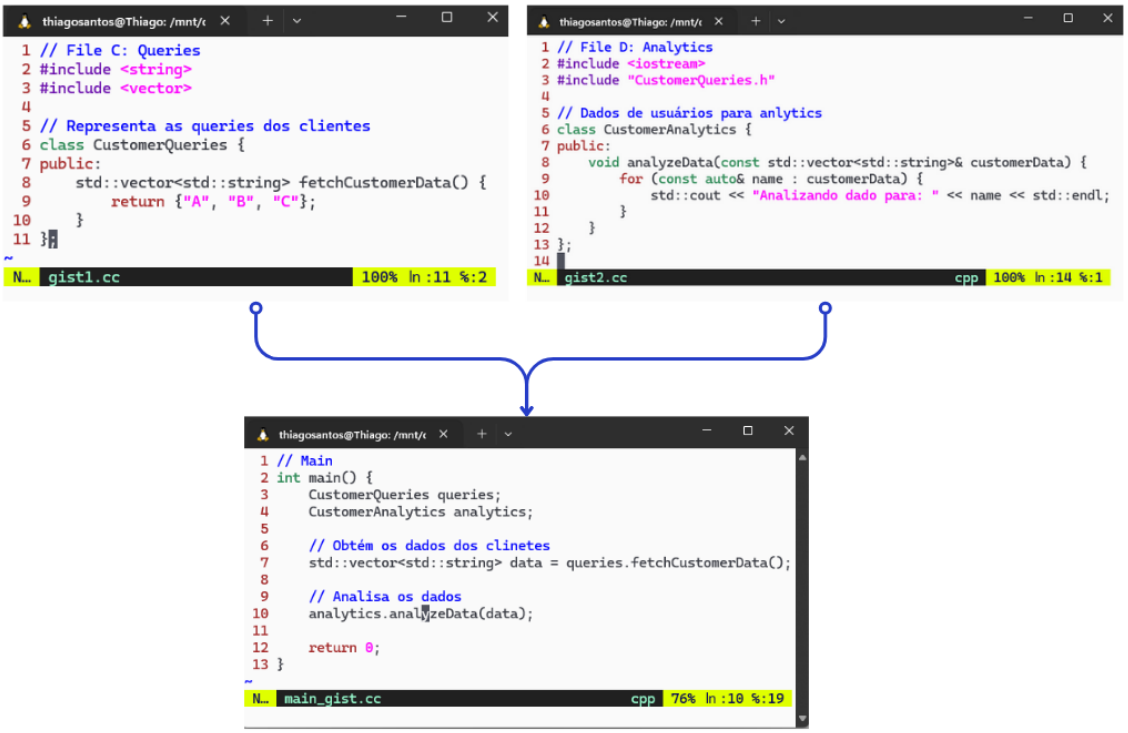

Figura 1. Exemplo de um sistema fictício com dependência intrínseca

Se a estrutura de papéis dentro do arquivo A muda, pode ser que seja necessário atualizar o arquivo B para garantir que as opções de análise dos dados estejam funcionando em conformidade. Estes arquivos são inter-relacionados porque a lógica de negócio demanda isso. Refatorar o código para remover essa dependência pode violar outros princípios extremamente importantes dentro da engenharia de software, em especial, nesse exemplo, estaríamos violando o princípio da separação de responsabilidades,

uma vez que a única solução viável seria centralizar as duas lógicas dentro de uma única entidade da aplicação. Nesse cenário, o IFFT tem um valor inestimável, servindo como um sistema de lembrete que garante que nenhuma alteração seja feita sem que o desenvolvedor tenha pleno conhecimento da existência dessa dependência.

Nesse cenário, o IFFT garante que nenhuma atualização no que diz respeito às consultas do usuário seja feita sem que o desenvolvedor seja alertado de possíveis mudanças no arquivo de análises.

Nota: *Esse exemplo é apenas ilustrativo. Na verdade, em termos práticos, esse cenário* ***não deveria fazer o uso do IFFT*** *conforme explicado na seção "O real valor do IFFT".*

3.3. O IFFT não encoraja o forte acoplamento nas aplicações

Também é importante enfatizar que o IFFT não é. A ferramenta não foi pensada para gerenciar dependências triviais que poderiam ser eliminadas através de boas práticas de desenvolvimento de software. Por exemplo:

- Sincronizar contadores entre dois arquivos. Um contador de número de logins, por exemplo.
- Manter configurações duplicadas entre arquivos

Estes casos se beneficiarão mais através de módulos centralizados ou mudanças arquiteturais. O IFFT é focado em mudanças não triviais, dependências de domínio específico que persistem mesmo em sistemas bem planejados. Em termos mais específicos, nesse exemplo, nestes cenários o mais adequado seria refatorar o código de modo que ambos os códigos referenciem um terceiro ente compartilhado, eliminando a redundância.

Usar o IFFT em casos como esse mascararia o grave problema de "design", encorajando práticas de desenvolvimento não recomendadas.

3.4. Colocando o IFFT em perspectiva

Ultimamente, o IFFT é uma ferramenta pragmática, não é um substituto para um bom "design"de software. Seu valor reside em complementar o desenvolvimento de sistemas de larga escala a partir da redução do erro humano no gerenciamento de dependências. Mais uma vez, não é sobre encorajar o acoplamento, mas sobre conhecer e mitigar os desafios onde essas dependências não são evitáveis ou muito custosas de serem removidas.

De modo a clarificar a forma com que o IFFT opera, segue um diagrama que mostra como o IFFT processa os arquivos e notifica os desenvolvedores de potenciais atualizações nas dependências:

4. Metodologia

4.1. Design da Ferramenta e Arquitetura

O IFFT foi projetado como uma ferramenta modular e extensível que busca a integração perfeita com fluxos de trabalho de desenvolvimento modernos. Sua arquitetura é formada por quatro componentes principais:

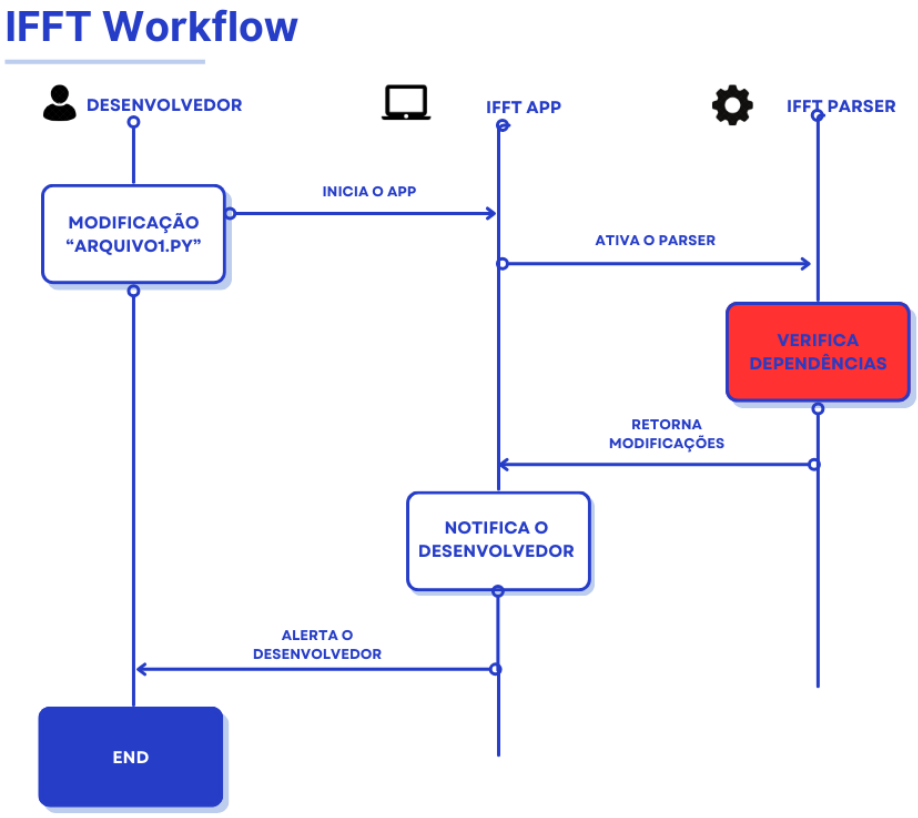

Figura 2. Workflow do IFFT

4.1.1. Parser:

O parser faz o ”scan”do código fonte para identificar as marcações *IFFT.If* e *IFFT.Then*, que definem os blocos de código que estão sendo monitorados. Esse componente usa o módulo **re** do Python para fazer o casamento de padrão usando expressões regulares e também usa o **GitPython** para identificar os arquivos modificados.

4.1.2. Gerenciamento de Blocos:

Este componente gerencia as principais operações da ferramenta, como:

- Remover anotações do IFFT para reduzir a ”poluição”nos arquivos.
- Restaurar anotações usando metadados dos blocos.
- Exportar metadados sobre as dependências para análise.

4.1.3. Integração Git:

”Pre-commit hooks”para automatizar a validação das dependências antes da submissão de ”commits”, garantindo que todas as possíveis modificações necessárias sejam realizadas.

4.1.4. Sistema de Saída:

Uma ”interface web”construída com Flask e bibliotecas de ”front-end”como **Bootstrap** e **Vis.js** que permite aos desenvolvedores interagir com as saídas dos IFFT visualmente. O diagrama a seguir mostra, em alto nível, o design da infraestrutura da ferramenta:

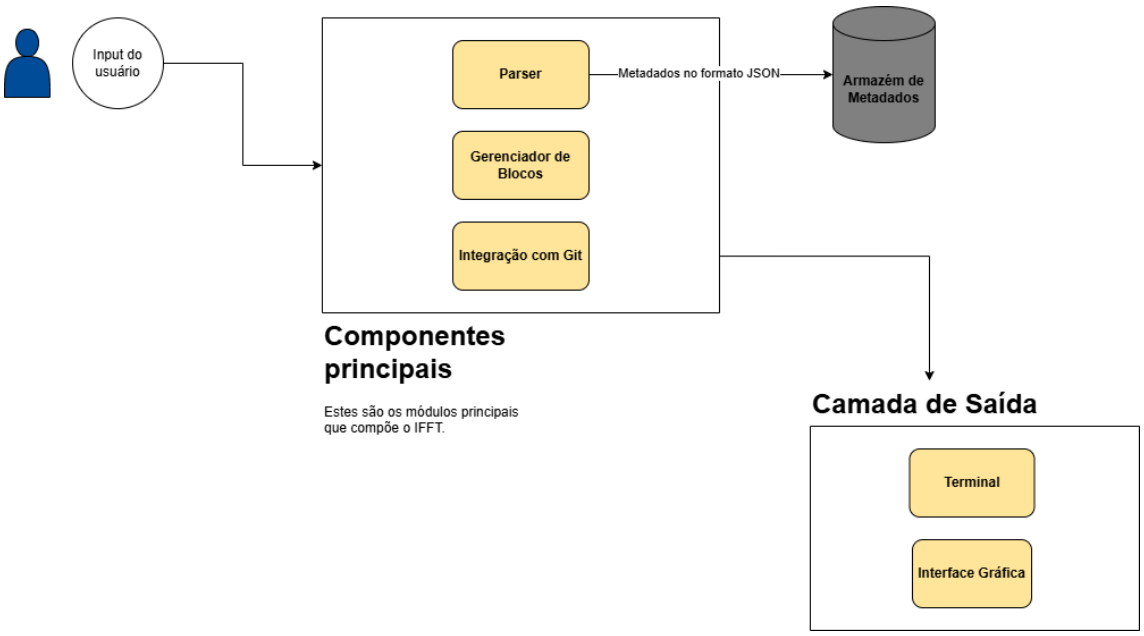

Figura 3. Infraestrutura IFFT

5. Panorama Geral dos Módulos e do Fluxo do IFFT

O ”workflow”do IFFT é sequencial, habilitando um monitoramento e ”feed-back”precisos durante o desenvolvimento. Segue o detalhamento dos módulos que compõem o fluxo geral de funcionamento da ferramenta:

5.1. Scan dos Arquivos

- **Objetivo:** Identifica os arquivos com blocos IFFT e validar se as anotações são válidas.
- **Funcionamento:** Percorre o diretório do projeto ao qual a ferramenta está acoplada e localiza os arquivos python. Uma pequena melhoria realizada durante o desenvolvimento do projeto, foi a criação de uma opção que permite o usuário definir diretórios do projeto onde o ”scan”não precisa ser feito, aumentando a eficiência do processo de ”scan”.

5.2. Análise de Dependências

- **Objetivo:** Validar e estabelecer uma relação de dependência entre os blocos de código.
- **Funcionamento:** A forma com que blocos de código são marcados para ser monitorados funciona a partir da inserção dos blocos de abertura ***IFFT.If*** e posteriormente os blocos de fechamento que realiza os ”links”de dependência. Após

esta etapa, ocorre um processo em que os relacionamentos identificados são guardados como um grafo direcionado, com os nós representando os arquivos e as arestas representando as dependências. Este dado gerado é utilizado dentro da ”interface web”para fornecer uma visualização interativa das dependências para o usuário.

5.3. Ações Disponíveis

- **Remover Anotações:** Esta opção realiza a remoção de todos os blocos IFFT dentro do projeto.
- **Restaurar Anotações:** Re-insere os marcadores dentro dos arquivos. Esse processo é feito a partir dos metadados dos blocos que ficam registrados.
- **Exportar Resultados:** Salva os resultados das análises de dependência como JSON ou CSV.

5.4. Integração com o Git

- **Objetivo:** Automatizar o processo da ativação da análise de dependências de modo que usuário não precise ficar se lembrando de executá-la.
- **Funcionamento:** O funcionamento se dá a partir da configuração de um ”pré-commit hook”que inicia o processo de análise de dependência automáticamente.

6. Novas Features e Detalhes de Implementação

Ao término da segunda parte do projeto, algumas ”features”novas foram adicionadas. Nesta seção vou me aprofundar na ideia por trás de cada uma e em aspectos específicos de implementação.

6.1. Arquivo de Configuração

Foram introduzidas seis novas opções de configurações para que a ferramenta possa se adequar melhor às necessidades do usuário e do projeto.

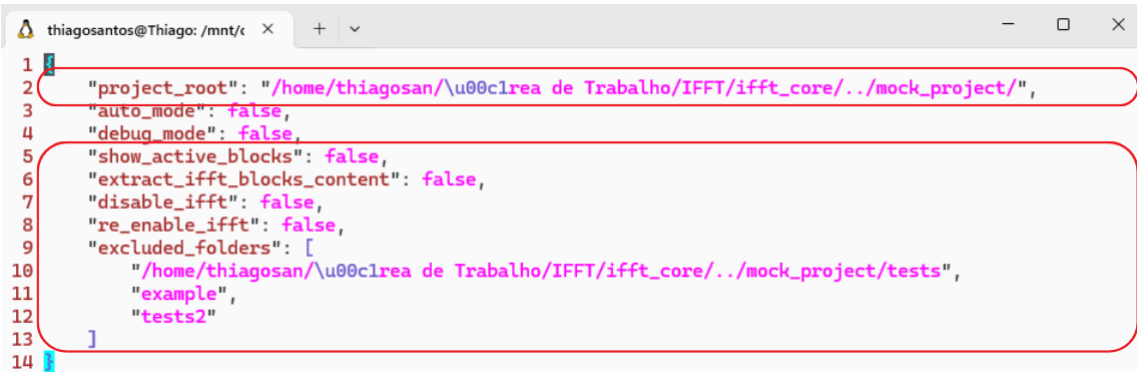

Figura 4. Novas opções de configuração

6.1.1. “project_root“

Esta opção permite ao usuário determinar o caminho (absoluto ou relativo) para o projeto que será monitorado pela ferramenta. Essa opção é essencial para que o usuário tenha

uma forma simples e flexível de determinar onde será localizado o projeto. Por exemplo, o sistema foi todo desenvolvido em um ambiente Linux, porém, para conseguir as imagens, optei por fazer em um sistema Windowns, tudo que tive que fazer foi trocar o caminho do projeto e tudo funcionou sem problemas.

6.1.2. “show_active_blocks“

6.1.3. “extract_ifft_blocks_content“

Esta opção permite que o usuário **manualmente** decida extrair os metadados dos blocos ativos dentro do projeto. Importante mencionar que essa ação já ocorre de forma automática no momento em que a feature de remoção é ativada. Na prática, com essa opção, o desenvolvedor tem agora uma flexibilidade que o permite ter os dados prontamente disponíveis para verificação ou para posterior análise.

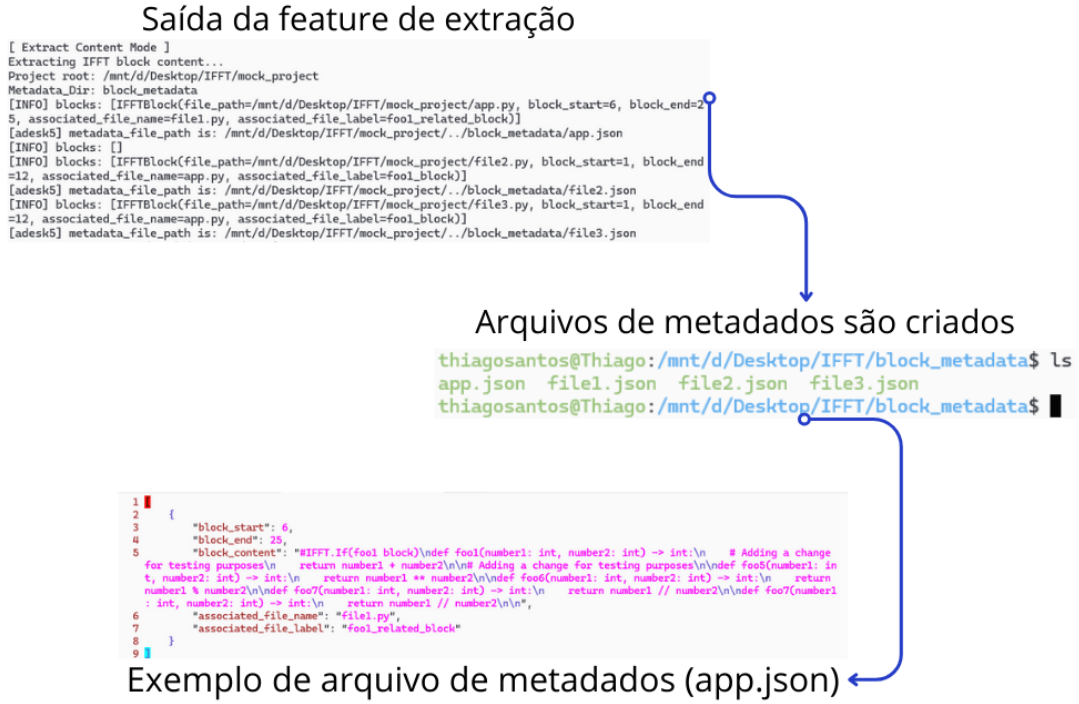

Figura 5. Feature de extração dos metadados de blocos ativos

6.1.4. “disable_ifft“

Esta opção permite ao usuário desabilitar completamente a ferramenta dentro do projeto. Esta ação desabilita toda a infraestrutura da ferramenta dentro do projeto, podendo ser revertida. O que permite que a ação seja desfeita é o salvamento automático dos metadados dentro do diretório de metadados da ferramenta. Em uma melhoria futura, também daria a

opção para o usuário remover os metadados, configurando assim uma ação terminativa da ferramenta dentro do projeto. As imagens a seguir mostram um exemplo de saída dessa feature no terminal e a comparação do ”antes e depois”da utilização da mesma:

Figura 6. Feature que desabilita o IFFT dentro do projeto

6.1.5. “re_enable_ifft“

Funciona de forma similar à feature que remove, porém é a ação contrapositiva. Ela faz o uso dos metadados salvos para re-inserir os marcadores dentro do código. Para evitar redundância, não será exibida imagem para essa feature dado que o funcionamento é muito semelhante à feature anterior.

6.1.6. “excluded_folders“

Permite ao usuário determinar diretórios nos quais não há a necessidade de monitoramento da ferramenta. Essa opção é bem importante no quesito performance, dado que a ferramenta deixará de fazer o ”parsing”de arquivos que não são de interesse do usuário de serem monitorados. Exemplos de diretórios que poderiam ser ignorados: diretórios de testes, diretório de configurações, logs e etc.

Antes

```python
import os

file_dir = os.path.dirname(__file__)

#IFFT.If(foo1_related_block)
def foo1(number1: int, number2: int) -> int:
    # Adding a change for testing purposes
    return number1 + number2

# Adding a change for testing purposes

def foo5(number1: int, number2: int) -> int:
    return number1 ** number2

def foo6(number1: int, number2: int) -> int:
    return number1 % number2

def foo7(number1: int, number2: int) -> int:
    return number1 // number2

def foo7(number1: int, number2: int) -> int:
    return number1 // number2


#IFFT.Then("file1.py", "foo1_related_block")
def foo2(number1: int, number2: int) -> int:
    return number1 - number2

def foo3(number1: int, number2: int) -> int:
    return number1 * number2

def foo4(number1: int, number2: int) -> int:
    return number1 / number2

def foo8(number1: int, number2: int) -> int:
    return 2*number1 + number2

def outsideFunction() -> None:
    print("This is a new function added outside of any IFFT block!")

# main

print(foo1(1, 2))
print(foo2(1, 2))
print(foo3(1, 2))
```

Depois

```python
import os

file_dir = os.path.dirname(__file__)


def foo1(number1: int, number2: int) -> int:
    # Adding a change for testing purposes
    return number1 + number2

# Adding a change for testing purposes

def foo5(number1: int, number2: int) -> int:
    return number1 ** number2

def foo6(number1: int, number2: int) -> int:
    return number1 % number2

def foo7(number1: int, number2: int) -> int:
    return number1 // number2

def foo7(number1: int, number2: int) -> int:
    return number1 // number2


def foo2(number1: int, number2: int) -> int:
    return number1 - number2

def foo3(number1: int, number2: int) -> int:
    return number1 * number2

def foo4(number1: int, number2: int) -> int:
    return number1 / number2

def foo8(number1: int, number2: int) -> int:
    return 2*number1 + number2

def outsideFunction() -> None:
    print("This is a new function added outside of any IFFT block!")

# main

print(foo1(1, 2))
print(foo2(1, 2))
print(foo3(1, 2))
```

Figura 7. Antes e depois da utilização da feature de remoção

7. Interface Gráfica

A criação da interface gráfica do IFFT foi um dos feitos mais interessantes durante o período de desenvolvimento. Este feito transformou o IFFT de uma ferramenta baseada puramente no terminal para uma aplicação que é muito agradável visualmente e com diversos elementos interativos. Esta seção do relatório traz detalhes por trás da motivação para o desenvolvimento da versão da ferramenta com interface gráfica, os detalhes técnicos de sua integração e os aspectos principais introduzidos.

7.1. Objetivos de Design

O objetivo principal da UI foi o de melhorar a acessibilidade e a experiência de uso da ferramenta, oferecendo uma plataforma centralizada para gerenciar, visualizar e analisar dependências em projetos de desenvolvimento de software.

7.2. Implementação

A UI foi construída utilizando **Flask** para o ”backend”, **Boostrap** para os componentes visuais e **Vis.js** para a visualização com o grafo interativo. O sistema também faz uso do **DataTables.js** para apresentar dados tabulares na aba de saída da ferramenta.

Seguindo o padrão adotado para as outras partes do projeto, a UI foi pensada como uma extensão modular do ”backend”do IFFT, com uma clara separação de responsabilidades:

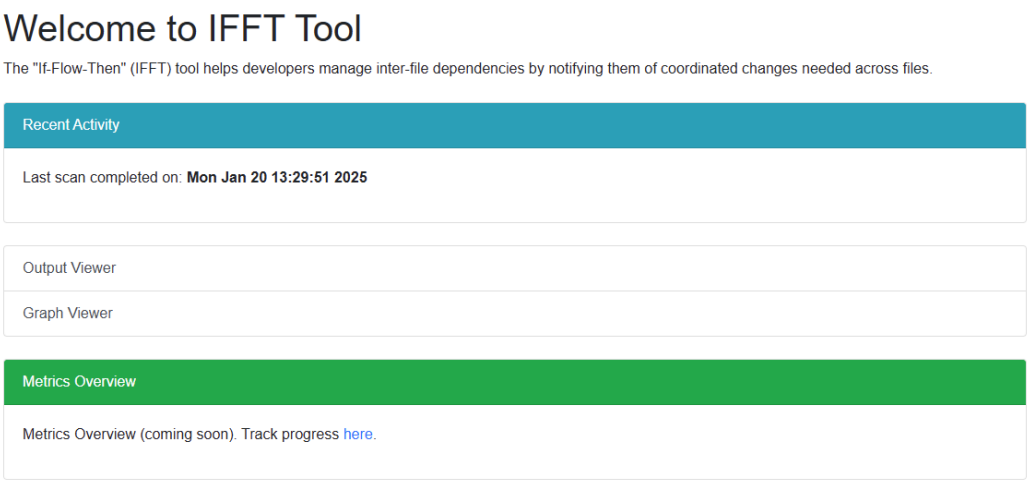

Figura 8. Página inicial da versão gráfica do IFFT

- **API do Backend:** Rotas são definidas para servir dados no formato JSON para o ”frontend”(ex: ***/output-data*** e ***/graph-data***). Essas APIs obtém os dados diretamente dos arquivos do ”ifft core”ou dos arquivos de metadados.
- **Integração com o Frontend:** Templates HTML criados em conjunto com o suporte do **jinja2**, carregam dados dinamicamente dos ”endpoints”mencionados e renderizam os componentes como tabelas e grafos.
- **Gerenciamento de Configurações:** A página de configurações sincroniza as preferências do usuário definidas no arquivo ***ifft_config.json*** e também garante a persistência dos dados entre sessões.

7.3. Feature Principais da Interface Gráfica

7.3.1. Visualizador de Resultados

Este componente exibe os resultados gerados pela ferramenta em formato de tabelas onde cada linha corresponde a um arquivo e traz consigo uma opção em modal que permite visualizar as modificações dentro dos blocos.

Note que essa visualização dos resultados é diferente da visualização apresentada na proposta do projeto, a razão para isso é **escalabilidade**. A versão anterior que mostrava os ”diffs”com o conteúdo que foi adicionado (e ou removido) tinha um problema muito grande de escalabilidade, imagine que o desenvolvedor tivesse feito mudanças dentro de dez blocos diferentes, isso já seria o suficiente para que a página de resultados fosse muito extensa verticalmente, o que tornaria a experiência de navegação nos resultados muito ruim.

Por isso foi utilizada uma versão tabular. O conteúdo alterado ainda pode ser visto clicando no botão ”view modified lines”que vai abrir um modal com o conteúdo. Além disso, outras duas opções novas de exportação foram adicionadas, uma que permite que o usuário exporte os metadados em formato json (o que já era feito na versão de terminal) e também uma versão nova, que permite o usuário exportar os metadados em formato csv. A versão tabular seria muito mais agradável de se navegar nesse exemplo das ”dez

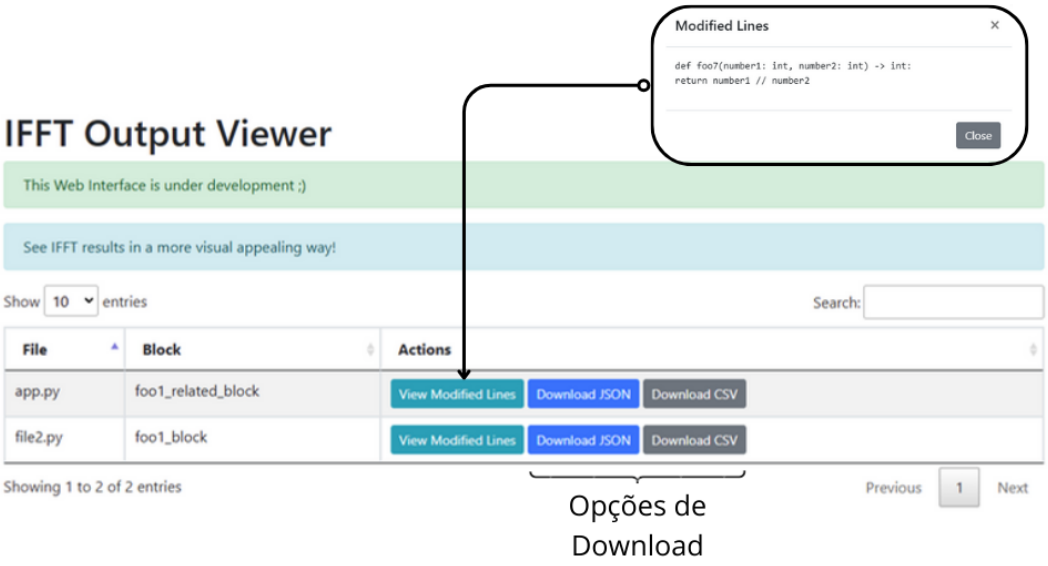

Figura 9. Página de resultados do IFFT

modificações"dado que ela apresenta filtros e páginas que ajudam a organizar e buscar por uma mudança específica de forma mais facilitada.

7.3.2. Visualização em Grafo

A introdução da visualização em grafos para o IFFT representou uma abordagem transformadora de entender as dependências dentro do projeto. Enquanto a representação tabular dos resultados é funcional e concisa, a necessidade de uma forma mais interativa e intuitiva de explorar as dependências inter-arquivos se tornou evidente.

Em qualquer projeto de larga escala, dependências entre arquivos e blocos de código formam redes de relacionamentos que são difíceis de desvendar a partir de representações estáticas. Uma abordagem baseada em uma visualização de grafos desses relacionamentos oferece aos desenvolvedores uma forma imediata, de alto nível, de compreender a arquitetura do projeto. Nós representam arquivos ou blocos, enquanto as arestas direcionadas representam as dependências, permitindo aos usuários rapidamente identificar caminhos críticos e componentes que dependem entre si.

Construir essa visualização em grafos começou com a escolha de quais tecnologias seriam mais adequadas para o objetivo final. A biblioteca **Vis.js** foi escolhida por apresentar diversas funcionalidades interativas e suporte para "layouts"dinâmicos e com o aspecto da física dos objetos implementados. A escolha permitiu que os usuários pudes-

sem arrastar os nós, dando a eles a flexibilidade de fazer um re-arranjo das dependências para uma melhor visualização.

A integração com o ”backend”também apresentou seus desafios. Dado que o grafo depende dos metados dos processos de ”scan”da ferramenta, um esforço significativo foi feito para garantir que o ”endpoint”***/graph-data*** providenciasse os dados no formato correto.

Um dos aspectos mais interessantes do grafo são as arestas direcionadas. Essas arestas indicam claramente ao usuário o fluxo de dependência dentro do projeto, deixando muito claro quais são os componentes ”upstream”e ”downstream”dentro do projeto. Essa simples adição melhora significativamente a usabilidade do grafo, especialmente em projetos complexos, por exemplo, uma aresta apontando de *app.py* para *file1.py* sinaliza que mudanças foram identificadas no primeiro arquivo e que talvez elas devam ser refletidas no segundo. Para finalizar, essa feature oferece um balanço entre funcionalidade e simplicidade.

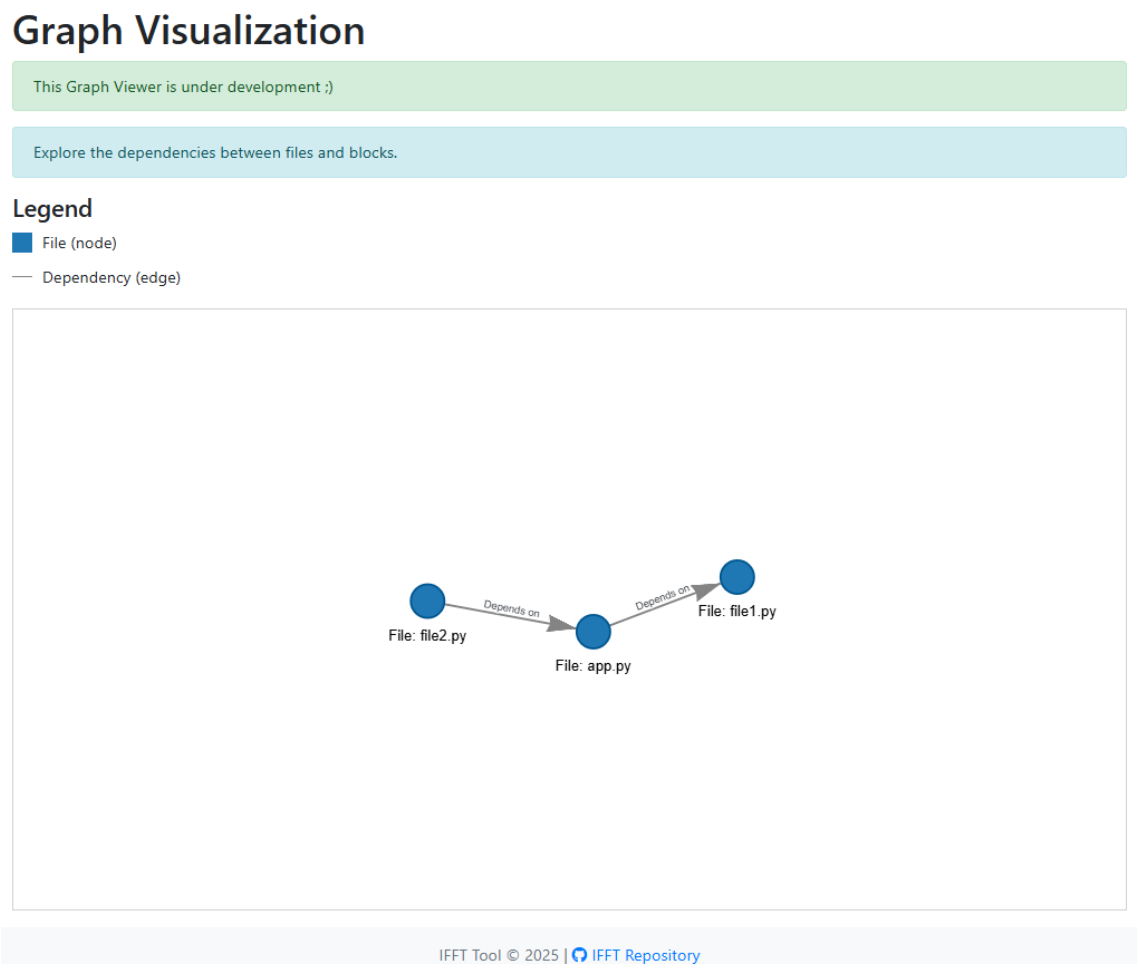

Figura 10. Visualização em grafo das dependências do projeto

7.4. Gerenciador de configurações

Continuando com as features novas implementadas nessa segunda fase do projeto, temos a interface gráfica para controle de configurações do IFFT. A **figura 4** mostra como os

aspectos de configuração da ferramenta cresceram em tamanho e complexidade, tornando muito complexo para o usuário ter que entender todas aquelas diferentes opções e ter que editar o arquivo sempre que deseja alguma alteração. Para endereçar esse problema, a interface de configuração do IFFT foi criada. Essa página permite configurar a ferramenta de forma muito mais simplificada.

Além do aspecto da usabilidade, uma preocupação considerada foi o do aumento da complexidade das operações da ferramenta. Features como o ”Auto Mode”por exemplo, são muito fáceis de serem utilizadas de forma incorreta uma vez que deixamos a atribuição de configuração bruta para os desenvolvedores.

O funcionamento da página torna tudo mais simples, cada opção de configuração é representada com um elemento gráfico intuitivo. Caixas de seleção fazem o chaveamento de features como ”Auto Mode”e ”Debug Mode”, entradas de texto lidam com caminhos relativos e absolutos dentro do projeto e, mais importante, ***todas essas mudanças são validadas em tempo real para previnir entradas inválidas***, esse não era o caso quando as edições eram feitas diretamente no arquivo de configuração.

O arquivo de configuração do projeto ainda é soberano, portanto é necessário que essa página ”converse”com o arquivo, para fins de sincronização, de alguma forma. Isso acontece por meio de uma rota dedicada, */seetings*, que orquestra a comunicação entre a UI e o backend. Uma vez que uma mudança é submetida na página, ela é refletida instantaneamente no arquivo ***ifft_configs.json***, garantindo a persistência entre diferentes sessões. No sentido inverso, a página popula os campos dinamicamente a partir da leitura do estado mais recente do arquivo de configuração naquele momento. Isso ajuda a garantir a sincronização entre o arquivo e a página.

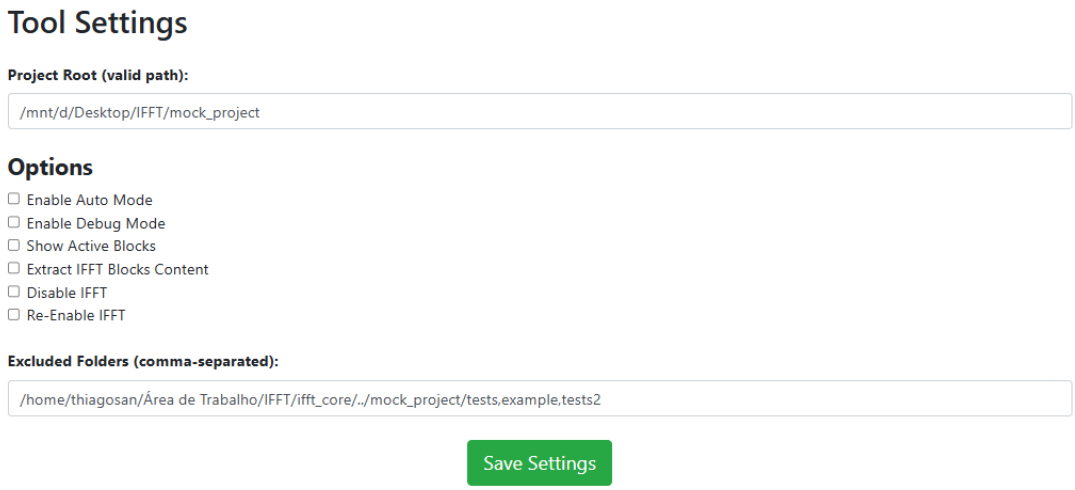

Figura 11. Página de configuração da ferramenta

8. Conclusão e Trabalho Futuro

O desenvolvimento do IFFT foi um projeto muito interessante de desenvolver e se mostrou bem promissor na atividade de auxiliar os desenvolvedores com o gerenciamento de dependências em projetos de larga escala.

A ideia do projeto começou a partir de uma experiência pessoal trabalhando em uma empresa com projetos de larga escala e, consequentemente, da constatação de que, apesar dos avanços em práticas de desenvolvimento de software, desenvolvedores frequentemente encontram cenários onde modificações em uma seção de código necessitam de uma mudança complementar em outra parte. Através de features como **monitoramento de blocos de código**, **integração com os pre-commit hooks** e uma interface gráfica , o IFFT auxilia os times a reduzirem erros causados por problemas de dependências.

Um ponto que gostaria de destacar nessa segunda fase de desenvolvimento da ferramenta foi a introdução da interface gráfica. O visualizador em grafo introduziu uma abordagem interativa de entender as dependências, enquanto que a página com os resultados da ferramenta providencia uma saída detalhada das mudanças que foram realizadas.

É claro que, apesar da ferramenta apresentar recursos muito promissores para endereçar os problemas apresentados no relatório, um esforço grande foi feito para que os limites da ferramenta fiquem claros. O IFFT não foi pensado para substituir a refatoração, garantir padrões arquiteturais de projetos ou gerenciar dependências triviais que poderiam ser resolvidas através de uma melhor modularização do projeto. Ao invés disso, ele serve como um auxílio prático para times navegando por desafios de codebases grandes e estabilizadas onde a refatoração clássica é muito custosa ou mesmo inviável.

9. Considerações Finais e Trabalho Futuro

Do que foi proposto desde a primeira etapa do projeto até o presente, cerca de 85% foi implementado com sucesso, o que considero um grande resultado. Todavia, ainda consigo vislumbrar diversas oportunidades de desenvolvimento para a ferramenta, dentre elas:

- **Suporte para outras linguagens:** Uma premissa do projeto foi a de que focaria apenas na linguagem Python por questões simplicidade de conseguir valida a ideia do projeto como um todo. Expandir a ferramenta para outras linguagens como C++ e Java, linguagens que são utilizadas amplamente em grandes bases de código, faria com que a ferramenta fosse útil para mais pessoas.
- **Melhoria na análise de dependências:** Versões futuras poderiam incorporar formas de análises mais sofisticadas, usando predições de IA por exemplo. Uma outra coisa que pensei era em tornar mais robusta a carga computacional de fazer ”scan”de todos os arquivos do projeto, é muito claro para mim que a ferramenta precisaria de um processamento paralelo se ela tem pretenções reais de atender grandes projetos.
- **Integração com pipelines de CI/CD:** Essa era uma das coisas que estavam na proposta do projeto mas que foi caindo de prioridade na medida em que ele foi sendo desenvolvido, até o ponto em que isso ficou de fora. Mas certamente é algo que seria muito bem vindo no projeto.
- **Utilização de ”protocol buffers”:** Num ponto muito longíquo do projeto, eu deveria começar a pensar em como as estruturas chave do projeto como os objetos de metadados, de blocos serão transmitidos em rede, utilizar protocol buffers poderia auxiliar a melhorar a performance e escalabilidade, especialmente em projetos grandes com extensas árvores de dependências.

- **Melhoria nos testes:** Confesso que utilizei de uma abordagem incorreta para testar a ferramenta inicialmente, foquei muito em testar os módulos separadamente em testes de unidade, mas quando eles interagiam entre sí tive muita dificuldade de conseguir testá-los, isso estava tomando muito tempo de desenvolvimento para as features da especificação do projeto. Uma melhoria que faria era pensar num arcabouço de testes robusto do zero.

Essas ideias que tive para um eventual esforço futuro na ferramenta concluem o presente relatório técnico.

Referências

[1] Git Hooks by Atlasian. https://www.atlassian.com/br/git/tutorials/git-hooks

[2] Making a python l. https://python.plainenglish.io/asts-and-making-a-python-linter-from-scratch-79e66e8d99b8

[3] Git Hooks. https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

[4] Pydriller. https://github.com/ishepard/pydriller

[5] GitPython. https://gitpython.readthedocs.io/en/stable/