UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

MARCOS GÊNESIS DA SILVA

CIREF: UMA FERRAMENTA PARA VISUALIZAÇÃO DO CONTEXTO HISTÓRICO DE REFATORAÇÕES EM PROJETOS JAVA

QUIXADÁ

2022

MARCOS GÊNESIS DA SILVA

CIREF: UMA FERRAMENTA PARA VISUALIZAÇÃO DO CONTEXTO HISTÓRICO DE REFATORAÇÕES EM PROJETOS JAVA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientadora: Profa. Dra. Carla Ilane Moreira Bezerra.

QUIXADÁ

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S581c Silva, Marcos Gênesis da.
CIREF: uma ferramenta para visualização do contexto histórico de refatorações em projeto java / Marcos Gênesis da Silva. – 2022.
66 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2022.
Orientação: Profa. Dra. Carla Ilane Moreira Bezerra.

1. Software - Refatoração. 2. Mineração de dados. 3. Visualização de dados. I. Título.

CDD 005.1

MARCOS GÊNESIS DA SILVA

CIREF: UMA FERRAMENTA PARA VISUALIZAÇÃO DO CONTEXTO HISTÓRICO DE REFATORAÇÕES EM PROJETOS JAVA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: 12 de Dezembro de 2022

BANCA EXAMINADORA

Profa. Dra. Carla Ilane Moreira Bezerra (Orientadora)
Universidade Federal do Ceará (UFC)

Profa. Dra. Mairieli Santos Wessel
RADBOUD UNIVERSITY

Prof. Dr. Anderson Gonçalves Uchôa
Universidade Federal do Ceará (UFC)

Prof. Me. Camilo Camilo Almendra
Universidade Federal do Ceará (UFC)

A Deus pela oportunidade que me proporcionou, a minha família que fez de tudo para que estivesse hoje aqui e aos meus amigos que fizeram dessa jornada o mais divertido possível de se viver.

AGRADECIMENTOS

Aos meus pais, Maria Clesilda da Silva e José Diniz da Silva, que fizeram de tudo o possível para eu chegar neste momento final da graduação, me colocando como prioridade e adiando os próprios sonhos em prol do meu.

Aos meus amigos de mesmo semestre, Ítalo Lima, Eric Rodrigues, Gustavo Colombo e Jeferson Gonçalves por todos os trabalhos realizados em conjunto, ideias tidas, projetos concluídos e o mais importante: as risadas durante esses 4 anos!

Agradeço a Profa. Dra. Carla Bezerra, por aceitar o desafio de me orientar durante a produção deste trabalho. Que as contribuições em conjunto para a Engenharia de Software continuem após a graduação.

Agradeço a mim mesmo por não ter desistido em momentos de dificuldade durante a graduação, ter tido calma e paciência para superar todos os obstáculos. Ter sido resiliente quando necessário, para que tudo ocorresse bem.

A todos que se fizeram presente nesta minha jornada, o meu muito obrigado!

“A mudança é a única constante. Mas mude devagar, pois a direção é mais importante do que a velocidade.“
(Edson Marques e Heraclitus)

RESUMO

Refatorações são ações que os desenvolvedores não fazem com frequência ou de forma padronizada. Um dos motivos é a falta de visualizações nas ferramentas atuais capazes de identificar refatorações. As visualizações de código podem auxiliar os desenvolvedores na tomada de decisões de analisar a qualidade do código e de possíveis refatorações no código. Neste contexto, este trabalho apresenta o desenvolvimento da ferramenta CIref, uma ferramenta capaz de analisar o contexto histórico das refatorações em projetos escritos em linguagem JAVA e fornecer visualizações sobre as refatorações encontradas neles. Também permite que cada projeto tenha seu próprio ranking de importância dos tipos de refatorações, duelos entre desenvolvedores para conhecer o perfil de cada um dos colaboradores e a visualização da linha do tempo do número de refatorações executadas no projeto. A ferramenta foi validada a partir do modelo TAM (Technology Acceptance Model) com um grupo de 8 desenvolvedores com mais de 2 anos de experiência. A avaliação mostrou que 75% dos participantes concordaram com o uso futuro da ferramenta e com a facilidade de uso da ferramenta.

Palavras-chave: Refatoração; Visualização; Mineração de dados;

ABSTRACT

Refactorings are actions developers don't often do or in a standard way. One of the reasons is the lack of visualizations in current tools capable of identifying refactorings. Code views can help developers make decisions about analyzing code quality and possible code refactorings. In this context, this work presents the development of the CIref tool, a tool capable of analyzing the historical context of refactorings in projects written in JAVA language and providing visualizations of the refactorings found in them. It also allows each project to have its ranking of importance of the types of refactorings, duels between developers to know the profile of each of the collaborators, and the visualization of the timeline of the number of refactorings carried out in the project. The tool was validated based on the TAM (Technology Acceptance Model) model with 8 developers with more than two years of experience. The evaluation showed that 75% of the participants agreed with the future use of the tool and the ease of use of the tool.

Keywords: Refactoring; Visualization; Data mining;

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Refatoração é um processo de software que busca modificar o *design* interno de um código sem que esta mudança altere o comportamento externo deste trecho de código (FOWLER *et al.*, 1999). Refatorar é um processo de software fundamental, principalmente por afetar aspectos não funcionais de um código de software como a diminuição da complexidade, aumento da modularidade, reusabilidade e a manutenibilidade do código (MENS; TOURWÉ, 2004). Esses aspectos compensam o alto custo e esforço de desenvolvimento requeridos por este processo (ALSHAYEB *et al.*, 2001) (e.g., em um projeto de tamanho médio — com cerca de 30 mil linhas de código — o tempo de refatoração é de cerca de 6 semanas (LIN *et al.*, 2016)). Um dos tópicos importantes dentro da refatoração, são as refatorações de *code smells*.

Code smells têm como sua definição, pequenos problemas inseridos ao decorrer do percurso natural do desenvolvimento de software onde a estrutura interna de um projeto pode se deteriorar, ocasionando situações como má legibilidade e baixa modificabilidade (FOWLER, 2018). *Code Smells* também podem ser frutos de resquícios de deterioração de funcionalidades do sistema, em que não foram completamente removidas com sucesso, afetando outras partes do sistema com os sintomas acima (FOWLER, 2018).

As refatorações, geralmente, são feitas de forma não frequente ou de maneira que não contém um padrão, muito por conta das ferramentas atuais serem confusas e não estimularem a participação direta dos desenvolvedores das equipes (KIM *et al.*, 2014). Além de não possuírem visualizações variadas sobre as refatorações encontradas, fator esse, crucial para o aumento de refatorações nos projetos (BOGART *et al.*, 2020a). O que se soma com os resultados obtidos por (ALOMAR *et al.*, 2021), no qual é constatado que os desenvolvedores que refatoram o código esporadicamente, tendem a documentar o conhecimento sobre aquela refatoração mais vezes, em comparação com os desenvolvedores mais recorrentes a executarem refatorações. Inferindo-se que as ferramentas existentes atualmente não suprem os desenvolvedores que mais realizam refatorações.

Diante deste contexto, o objetivo do presente trabalho é a criação de uma ferramenta útil para visualização do contexto de refatorações de *code smells*, em projetos com a linguagem JAVA. A partir da ferramenta será possível visualizar as refatorações de um ou mais projetos representadas por números absolutos e por desenvolvedores, além da quantidade de refatorações conforme o tempo e a priorização dos tipos de refatorações individualizadas para cada projeto. Dessa forma, os desenvolvedores e pesquisadores podem utilizar a ferramenta para entender um

pouco mais do contexto das refatorações e tipos de refatorações realizados ao longo do tempo nos projetos.

1.1 Objetivos

Nesta seção, é apresentado o objetivo geral e os objetivos específicos do presente trabalho.

1.1.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver o **CIref**, uma ferramenta WEB para a visualização do contexto histórico de refatorações em projetos escritos na linguagem JAVA.

1.1.2 Objetivos Específicos

a) Identificar ferramentas capazes de apoiar a visualizações de refatorações nos projetos.
b) Propor e implementar visualizações de refatorações que possa auxiliar os desenvolvedores a analisar o histórico de refatorações do projeto e tomar decisões.
c) Validar se as visualizações oferecidas pela ferramenta são úteis para os desenvolvedores.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos mais relevantes necessários para o entendimento deste trabalho. Na Seção 2.1 são apresentados conceitos sobre *code smells* e quais suas tipificações. Na Seção 2.2 são descritos conceitos básicos sobre refatorações e ferramentas para identificação de refatorações. Por fim, na Seção 2.3 são apresentados conceitos de visualizações em engenharia de software.

2.1 *Code Smells*

Hoje em dia, falar sobre *code smells*, é falar sobre algo que está consolidado na literatura referente a engenharia de software. O termo foi primariamente utilizado por Fowler *et al.* (1999), porém, outras denominações anteriores e posteriores já davam significados similares ao que foi dado ao termo designado por Fowler e Kent (e.g., descrito também como Desarmonias Lanza *et al.* (2005)). A definição proposta por Fowler e Kent, surgiu quando no mesmo trabalho citado anteriormente, a dupla definira 22 categorias de "maus cheiros" (tradução literal do termo em inglês *code smells*), sendo cada mau cheiro, uma série de características em comum, usadas como potenciais indicadores para falhas de *design* em projetos que podem dificultar a manutenibilidade de softwares, e também que podem ser uma oportunidade para a refatoração do trecho de código em questão.

É importante ressaltar, que um *smell* não é necessariamente ruim por si só, e se deve fazer uma investigação a fundo para verificar se existe de fato um problema (FOWLER, 2006). Como dito em Fowler (2006): "Muitas vezes (*smells*) são um indicador de problema, e não o problema". Estudos como Zhang *et al.* (2011) confirmam a afirmação de Fowler, mostrando que nem todos os *code smells* reduzem a confiabilidade do software.

Ao decorrer dos anos, muitos estudos investigam os efeitos do code smells e tentam aprimorar as formas de tratá-los (NUCCI *et al.*, 2018; FONTANA *et al.*, 2012; PALOMBA *et al.*, 2015a). Encontrar *code smells* não é uma tarefa difícil, visto que, são fáceis de serem detectados (e.g., *Data Classes*: Classes que contêm todos os dados sem comportamento) por desenvolvedores inexperientes, mesmo que estes não saibam como resolvê-los ou avaliá-los para saber de fato, se são um problema (FOWLER, 2006). Abaixo, é apresentado uma lista de alguns *code smells* listados em Fowler *et al.* (1999) e descritos por (SHATNAWI; LI, 2006):

– ***Data Class***: Classe que contêm somente instâncias de variáveis e os métodos *getters* e

setters.

- ***God Class***: Classe que contêm muitas responsabilidades em comparação às classes acopladas a ela.
- ***God Method***: Um método com muitas responsabilidades comparado com os outros métodos da classe.
- ***Refused Bequest***: Classe que contêm métodos instanciados, que nunca foram utilizados.
- ***Shotgun Surgery***: Quando uma modificação em uma classe, afeta diversos outros trechos de código em classes distintas.
- ***Feature Envy***: Classe que contêm um método que chama vários *getters* de outros objetos para calcular valores de dados.

Tendo em vista os exemplos de *code smells* supracitados acima, a detecção destes se dá hoje por uma lista de formas, incluindo desde o olhar investigatório humano a métodos para a visualização em softwares ou planilhas. Lacerda *et al.* (2020) capturou diversos estudos que relacionam *code smells* com as principais formas de detectá-los e a ferramenta principal que realiza esta detecção, para alguns dos exemplos acima, temos que:

- Para o ***God Class***, suas principais formas de abordagem, a busca baseada em métricas (3 estudos), e também a busca relacionada a regras (3 artigos). Já a ferramenta de detecção mais utilizada, é a DECOR[1], citadas por 5 artigos diferentes. Esta ferramenta utiliza-se de um vocabulário unificado e linguagem dedicada para ser possível identificar esse categoria de *code smells*, entre outros.
- ***Refused Bequest***, já tem como suas principais formas de abordagem a busca baseada em métricas (3 estudos), sendo a ferramenta IPlasma[2], como 3 estudos, sendo a principal ferramenta de detecção existente baseada nos estudos. A ferramenta IPlasma, é uma versão *open-source* da sua versão mais completa *InFusion*.
- ***Shotgun Surgery*** também tem como principal forma de abordagem a busca baseada em métricas (2 estudos), em consonância com a busca baseada em regras (2 estudos). Hist, é a principal ferramenta relacionada em 3 estudos para a detecção dessa categoria de *smell*. Esta ferramenta mira em 5 *code smells* identificados por (FOWLER, 2018) e se baseia no histórico de informações de mudanças para a identificação de *code smell*, como descrito em seu artigo de apresentação (PALOMBA *et al.*, 2015b).
- ***Feature Envy*** conta com a abordagem da busca baseada em métricas e a busca baseada em

[1] http://www.ptidej.net/research/designsmells/
[2] http://loose.upt.ro/iplasma

regras, com 2 estudos cada. Sendo o IPlasma, também como ferramenta principal para detecção desse tipo de *smell* categorizado por 4 estudos diferentes.

Além das ferramentas citadas acima, existem outras ferramentas hoje que também tratam de *code smells* e refatorações (explicada na seção seguinte). Algumas delas, abordando inclusive as sugestões de refatoração. Dentre estas ferramentas, é importante citar:

a) Vidal *et al.* (2015) propôs uma ferramenta para identificação de oportunidades de refatoração de *code smells* na linguagem de programação JAVA. Essa ferramenta é bastante utilizada, pois, além de identificar essas oportunidades de refatoração de *code smells*, também consegue ranquear os maus cheiros conforme a estratégia escolhida pelos desenvolvedores, que conseguem modificar a forma que a ferramenta identifica e ranqueia os *code smells*.

b) **Designite**[3] é outra ferramenta que identifica *code smells*, diferenciados em 2 categorias: *Design Smells* e *Implementation Smells*. Além disso, esta ferramenta também calcula 10 métricas de orientação a objeto, como: (i) Número de linhas; (ii) Número de parâmetros; (iii) Número de métodos, entre outros.

c) **Jdeodorant** é uma ferramenta que sugere as melhores formas de refatorar os 5 tipos de *code smells* suportados pela mesma. (e.g, O *code smell Feature Envy*, tem como sua melhor forma de solução, uma categoria de refatoração, denominado *Move Method* — explicado melhor na sessão seguinte).

d) **SonarQube** [4] é uma ferramenta muito completa e amplamente utilizada no mercado. Dentro de todo o ecossistema abrangido por eles, é possível também encontrar rotas que realizam a identificação de *code smells* e também é possível listar as refatorações sugeridas pelo ecossistema do SonarQube para solucionar estes *code smells*.

2.2 Refatoração

Durante todo o ciclo de desenvolvimento de um software, os envolvidos nos projetos têm que lidar com um dilema bastante recorrente: manter o código funcionando enquanto novas funcionalidades são adicionadas. É sabido que em projeto de desenvolvimento de software, é alocado demasiado tempo para a manutenibilidade do mesmo, onde de acordo com Glass (1998),

[3] https://github.com/tushartushar/DesigniteJava
[4] https://www.sonarqube.org/

nesta mesma fração de tempo, novas funcionalidades estão sendo incrementadas, podendo aumentar novamente a complexidade. Isso acaba se tornando um círculo vicioso.

Resolver este problema, se torna um desafio importante na busca por técnicas que reestruturem o código, reduzindo a complexidade do software por meio do incremento da qualidade interna de código (MENS; TOURWÉ, 2004). Reestruturar, quando falamos neste contexto de orientação a objetos, tem um nome bem conhecido: Refatoração. Termo este, inserido no meio acadêmico por Refactoring () e Fowler *et al.* (1999) como a melhoria da qualidade interna do código, sem a alteração do comportamento externo.

Refatorar, muitas vezes está interligado com *code smells*, isso se deve ao fato de um (*code smell*) ajudar a identificar inconsistências no código, e a (refatoração) tentar melhorar a qualidade interna afetada diretamente pelos *code smells*. Por existir essa correlação direta, existem diversas tentativas de criar melhores ferramentas ou técnicas de abordagem que ajudem os desenvolvedores a refatorar *code smells* com mais facilidade (MEALY *et al.*, 2007; O'KEEFFE; CINNEIDE, 2006). Essas abordagens parecem não refletir um cenário encontrado por Bavota *et al.* (2015) onde cerca de apenas 7% das operações de refatorações de *code smells*, de fato removeu o devido *code smell* — em relação ao total de 42% das operações de refatorações diretamente ligadas a *code smells*.

Algo que pode ajudar a explicar esse baixo índice, é a questão do comportamento dos desenvolvedores em relação a refatoração de *code smells*. Como os achados de (VASSALLO *et al.*, 2018) sugere, a experiência no momento da refatoração importa, onde a principal motivação encontrada dos desenvolvedores para continuar refatorando é a melhora do código. Acrescentando os resultados de Kim *et al.* (2012), mostra que os desenvolvedores necessitam de ferramentas fáceis de integrar e de revisar alterações.

2.2.1 Técnicas de refatoração

Como citado anteriormente neste presente trabalho, Fowler *et al.* (1999) além de identificar as 22 categorias de *code smells*, também trouxe formas de solucionar estes problemas na codificação dos projetos. Abaixo consta 4 exemplos de técnicas de refatoração, técnicas essas, que estão diretamente conectadas com a ferramenta Jdeodorant que será utilizada neste trabalho.

2.2.1.1 Move Method

Como dito por Fowler *et al.* (1999), todos os métodos vivem em um contexto específico, que no cenário de uma aplicação orientada a objetos é a classe. Se esta classe contém muitas referências a outras classes, ou seja, se comunicam com outros contextos constantemente, é nestas ocasiões onde a técnica de *move method* deve ser aplicada.

O procedimento para a aplicação desta técnica de refatoração é simples:

a) Identifique todos os contextos que a função escolhida para ser movida é utilizada, e verifique se estas funções também devem ser movidas para um novo diretório.
b) Cheque se a função escolhida é polimórfica, para serem ajustados, as classes pai e as sub classes por igual.
c) Ajuste a função selecionada ao seu novo contexto, execute análise estática e teste a função agora em seu novo ambiente.

   Esta técnica de refatoração é adotada pelo Jdeodorant para solucionar *code smells* do tipo *Feature Envy*.

2.2.1.2 Replace Conditional

Esta estratégia de refatoração é utilizada principalmente dentro do Jdeodorant para solucionar problemas de tipagem errôneas. Dentro dessa estratégia, é recomendável utilizar-se do polimorfismo — um dos pilares da orientação a objeto — para solucionar os problemas de tipagem, onde uma classe pai é criada com assinaturas de métodos, que as classes filhas vão poder implementar essas assinaturas da forma que o método precisar. Assim, a lógica do método em questão é separada corretamente.

2.2.1.3 Extract Method

Métodos longos podem ser de fato um problema paras as aplicações, isso é o que diz a própria documentação do Jdeodorant. Para solucionar este problema, Fowler também apresenta uma estratégia, que é trabalhar com pequenas funções dentro de toda aplicação, extraindo para funções separadas, contextos separados, ou que envolvem passos semelhantes (e.g, realizar um *print* de alguma variável).

O passo a passo para a execução dessa estratégia é simples como o nome sugere: extraia o trecho de código em uma nova função, analise-a e verifique quais variáveis ela se utiliza,

ajustando as mesmas para parâmetros desta nova função. Pronto! Basta chamar esta função na função anterior a que o conteúdo dela pertencia, passando as variáveis por parâmetro como definido. Vale ressaltar que nas linguagens modernas, os compiladores trabalham melhor quando as funções são pequenas, pois facilita por estas funções em *cache* (FOWLER *et al.*, 1999).

2.2.1.4 Extract Class

Outro *code smells* bastante encontrado nas aplicações é o *God Class* (explicado na sessão 2.1). É esperado que ao decorrer do desenvolvimento de uma aplicação, as classes comecem a ganhar responsabilidades mais complexas, que vão tornando-as mais longas e mais distantes da separação de responsabilidades bem feita que uma classe deve possuir.

Outro tipo de refatoração, bem simples, como o próprio nome já diz, **extrair** responsabilidades de uma **classe** longa, para sub classes filhas. Essa extração é decidida pelo desenvolvedor, que vai encontrar no código onde a classe deve ser dividia, criando uma classe e adicionando este código na mesma. Logo após os ajustes necessários, com a mudança de métodos e atributos, basta instanciar a sub classe na antiga classe longa.

2.2.2 Ferramentas para refatoração

RefactoringMiner (TSANTALIS *et al.*, 2022a) é outra ferramenta, que cumpre propostas similares a ferramenta JDeodorant. Esta ferramenta, como o próprio nome já diz, é um minerador de refatorações. Isso significa que sua principal funcionalidade é percorrer todo o projeto, identificando refatorações que foram executadas ao decorrer do tempo. Apesar de ser construída pelos mesmos autores da ferramenta anterior, o RefactoringMiner contém uma lista recheada com tipos diferentes de refatorações que o mesmo consegue encontrar ao decorrer de um projeto JAVA. Algumas das refatorações identificadas pela ferramenta: (1) *Extract Method*, (2) *Inline Method*, (3) *Rename Method*, (4) *Move Method*, (5) Move Attribute, (6)*Merge Variable*, (7) *Move and Rename Method*, (8) R*eorder Parameter*, (9) *Extract Method* e (10) *Inline Method*.

A lista acima representa 10, dos 90 tipos diferentes de refatorações que essa ferramenta suporta a identificação, tornando a ferramenta mais adequada ao nosso trabalho. Outra grande vantagem dessa ferramenta, é que ela não se limita a ser apenas uma biblioteca que pode ser adicionadas nas IDEs, em geral. Mas na sua concepção, ela foi pensada também para ser uma API de forma nativa. Isso quer dizer que, é possível, via código, utilizar as suas funcionalidades perfeitamente em qualquer projeto JAVA. Essa ferramenta, em suma, sempre

retorna seus resultados em um mesmo formato, o que também facilita a integração desta com outras ferramentas. As principais informações que esta ferramenta traz em todas as extrações de refatorações encontradas, são:

- O caminho absoluto do arquivo modificado relacionado ao projeto. Isso, em si, já é de suma importância, pois conseguimos medir o quanto aquele arquivo em específico foi refatorado ao decorrer do tempo e quais tipos de refatoração predominam dentre essas refatorações.
- O tipo de refatoração, que conseguimos além da medição do item anterior, possuir também números absolutos sobre o número de refatorações em um projeto e quais os tipos mais frequentes dentro do mesmo.
- A ferramenta sempre disponibiliza os estados de antes e depois que determinada refatoração ocorreu, ou seja, ela sempre irá mostrar como o arquivo era antes da refatoração e como ficou após a mesma. Indicando, inclusive, as colunas e linhas corretas para que uma visualização possa ser implementada, comparando esses dois estados.
- A ferramenta também traz informações sobre os tipos de refatorações, de forma geral, com o tipo e a descrição do mesmo.

2.3 Visualização em Engenharia de Software

Caserta e Zendra (2011) indicam a busca por soluções que facilitem para os desenvolvedores, a leitura, as mudanças e a manutenção dos softwares construído por eles. Uma dessas soluções são ferramentas para visualização de software, que segundo apresentado em (PRICE *et al.*, 1993) representa a utilização de recursos gráficos e textuais para a navegação, análise e a apresentação de informações de software para aumentar o entendimento sobre o mesmo. Com esta visualização os desenvolvedores conseguem entender melhor o sistema e reduzir o tempo necessário para alcançar boas soluções programáveis (STOREY *et al.*, 2000).

Tendo como objetivo a representação da estrutura, comportamento e evolução dos *softwares* (CARPENDALE; GHANAM, 2008), uma visualização em engenharia de *software* é proposta através de 3 passos: (i) aquisição dos dados, através da extração de informações de alguma base de dados; (ii) Análise dos dados coletados, visando identificar os mais importantes para diminuir a quantidade de dados. (iii) visualização dos recursos mapeados no passo ii, por componentes visuais. A maneira com que estes componentes vão ser representados não apresentam uma forma pré-definida, possibilitando que a criatividade do autor da visualização se

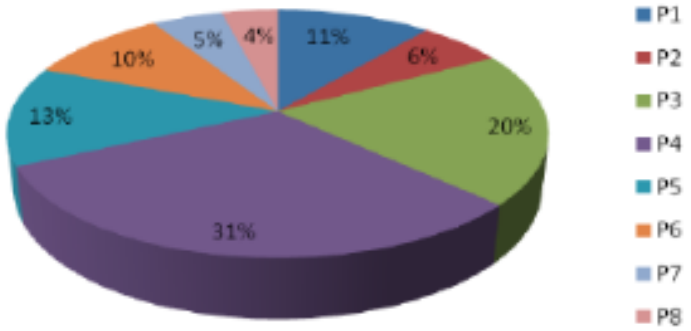
sobressaia (DIEHL, 2007).

Apesar de não existir maneiras pré-definidas de como se representar visualmente os componentes visuais, existem técnicas de representação visual 2D que já são encontrados na literatura e também podem representar esses componentes visuais.

2.3.1 PieChart

Uma das técnicas que utilizamos nestre trabalho para a visualização dos dados é o chamado gráfico circular, que também pode ser chamado de gráfico de torta - tradução direta do termo em inglês *pie chart* - ou gráfico de pizza. Este tipo de gráfico é dividido em algumas seções, em que cada seção desta representa uma porcentagem do valor total apresentado no gráfico. O tamanho de cada parte do círculo que forma este gráfico é avaliada em relação as outras partes do gráfico (SPENCE, 2005). Num gráfico desse tipo, cada fatia representa a proporção dos dados que possuem uma ou mais características em comum. Diferentes rótulos podem ser utilizados para distinguir cada parte do gráfico, porém, é usualmente utilizado porcentagens (ARCHAMBAULT *et al.*, 2015). Gráficos deste tipo são bastante semelhantes a outros tipos, como o gráfico de *donut*, que como o próprio nome indica, tem um formato de um donut, diferindo do gráfico de pizza por possuir um buraco no centro, dando o formato característico.

Figura 1 – Exemplo genérico de gráfico do tipo pizza



Fonte: (ARCHAMBAULT *et al.*, 2015)

3 TRABALHOS RELACIONADOS

Foram identificados na literatura alguns estudos que se relacionam com este projeto de pesquisa tanto com relação à proposta de resolução do problema apresentado - Construção da ferramenta para visualização do contexto histórico das refatorações em projetos JAVA - quanto com relação à melhoria da qualidade deste artefato. Nesta seção estes estudos são apresentados e suas contribuições e o modo como se relacionam com o presente trabalho são descritos.

Bogart *et al.* (2020b) realizaram a extensão de uma ferramenta já existente para encontrar oportunidades de refatorações, buscando a diminuição do espaço existentes entre os desenvolvedores e ação de refatoração de código, através da visualização das mesmas. A extensão mostra a visualização em uma única tela, dos múltiplos resultados de refatorações sugeridas pelas ferramentas. Os resultados encontrados foram promissores em relação ao aumento da produtividade e de tempo ganho através do uso das visualizações das refatorações. Neste trabalho, diferentemente do proposto por Bogart *et al.* (2020b), usaremos o contexto histórico das refatorações implementadas, para mostrar derivados tipos de visualizações da mesma.

Brito e Valente (2021) propuseram a RAID, uma ferramenta que automatiza em partes o processo de revisão de código em *Pull Requests* dentro da ferramenta Github. Com esta ferramenta é possível identificar operações de refatorações presentes nestas *Pull Requests* e aliviar o esforço cognitivo necessário na ação da revisão de código. A ferramenta utiliza-se de outra ferramenta, a RefDiff (SILVA *et al.*, 2021) para encontrar as refatorações. Neste trabalho, diferentemente do utilizado por Brito e Valente (2021), iremos identificar as refatorações já implementados no projeto e mostrar numa plataforma própria, ao invés de mostrar os dados na própria ferramenta do Github.

Tsantalis *et al.* (2022b) propôs a RefactoringMiner, que em suma, captura todas as refatorações implementadas entre um *commit* filho e um *commit* pai conforme o fluxo do controle de repositório do GitHub. A ferramenta criada suporta cerca de 90 tipos de refatoração e é amplamente uma referência no contexto de ferramentas para detecção de refatorações. RefactoringMiner se difere das outras ferramentas por sua implementação inovadora, que lhe proporciona não só uma maior eficiência, mas também uma maior acurácia. Neste trabalho utilizaremos esta ferramenta como provedora dos dados sobre refatorações nos projetos analisados. Porém, pelo suporte a muitos tipos de refatorações, neste trabalho também faremos uma categorização desses tipos suportados, para facilitar a visualização em tipos de gráficos mais compactos como o gráfico do tipo *donut*.

O Quadro 1 mostra a comparação realizada do presente trabalho e os trabalhos relacionados. A grande diferença entre os trabalhos relacionados com a ferramenta proposta neste trabalho está no tratamento das refatorações encontradas. A ferramenta proposta por, Brito e Valente (2021) apesar de também buscar refatorações já executadas em projetos, não representa essas refatorações por meio de visualizações. O trabalho de Tsantalis *et al.* (2022b), em suma, irá fazer parte da solução proposta como peça fundamental, assim como será detalhada na Seção 5.2, enquanto no trabalho de Bogart *et al.* (2020b) a ferramenta proposta mostra os dados encontrados sobre refatorações por visualizações. Assim como, possuem uma própria ferramenta para demonstrar essa funcionalidade. Porém, a ferramenta proposta só busca por oportunidades de refatorações possíveis de serem executadas, enquanto a ferramenta proposta neste trabalho busca todas as refatorações executadas dentro de um projeto.

Quadro 1 – Comparativo entre os trabalhos relacionados e o trabalho proposto

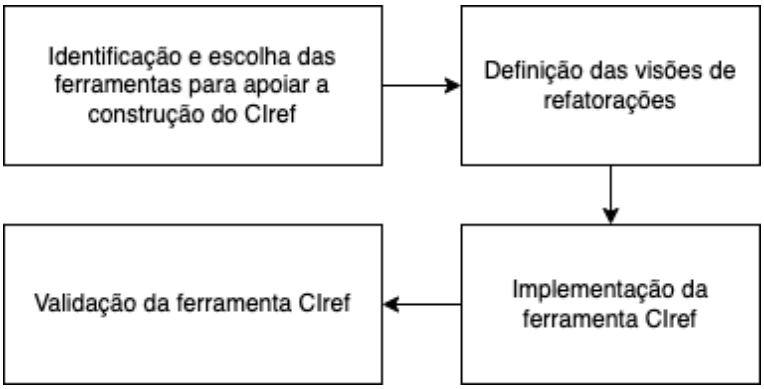
Trabalhos	**Apresenta os dados através de visualizações**	**Identifica/Utiliza refatorações já executadas**	**Possui uma plataforma própria**
Presente Trabalho	Sim	Sim	Sim
Bogart *et al.* (2020b)	Sim	Não	Sim
Brito e Valente (2021)	Não	Sim	Não
Tsantalis *et al.* (2022b)	Não	Sim	Não

Fonte: Elaborado pelo autor

4 PROCEDIMENTOS METODOLÓGICOS

Nesta seção serão apresentadas as etapas necessárias para a realização do presente trabalho. A Figura 2 apresenta uma visualização geral do encadeamento entre as etapas e os capítulos a seguir descrevem mais detalhadamente cada uma delas.

Figura 2 – Passos para a realização do trabalho



Fonte: Elaborado pelo autor.

4.1 Identificação e escolha das ferramentas para apoiar a construção do CIref

4.1.1 Ferramentas para identificação de refatorações

Para o desenvolvimento da ferramenta CIref foi necessária a identificação e escolha de uma ferramenta para coleta das refatorações do projeto. Foram selecionadas as ferramentas JDeodorant (FOKAEFS *et al.*, 2011) e RefactoringMiner (TSANTALIS *et al.*, 2022a) detalhadas no Capítulo 2.

Para determinar a escolha da ferramenta que seria utilizada como a fonte de dados sobre refatorações para a ferramenta proposta neste presente trabalho, elaboramos o Quadro 2, comparando entre as duas opções discorridas nos tópicos anteriores para determinarmos a que mais se adéque a arquitetura, definida na Seção 5.2.

Analisando o Quadro 2, pode-se afirmar que a ferramenta que mais se encaixa nos critérios analisados para a escolha da ferramenta que servirá como extrator de dados sobre refatorações em projeto JAVA é a RefactoringMiner. Dessa forma, a ferramenta CIref utilizará os dados gerados pela RefactoringMiner para propor suas visualizações propostas na Seção 4.2.

Quadro 2 – Comparativo entre as ferramentas possíveis para o fornecimento de dados sobre as refatorações em projetos JAVA

Critérios	Jdeodorant	RefactoringMiner
Possui facilidade para a exportação de resultados	Não	Sim
Disponibiliza API para consulta de dados	Não	Sim
Mostra oportunidades de refatorações	Sim	Não
Mostra refatorações executadas	Não	Sim
É de fácil *deploy*	Não	Sim

Fonte: Elaborado pelo autor

4.1.2 Ferramenta para obtenção dos dados sobre as modificações executadas nos projetos

Neste Capítulo 4.1.2, iremos analisar qual ferramenta iremos utilizar para conseguir as informações de quem/quando realizaram mudanças dentro dos projetos JAVA. Antes disso, precisamos entender onde os projetos são salvos e por onde o controle de mudança é realizado.

GitHub [1] é a principal ferramenta disponível como sistema de versionamento de código (VCS, em inglês) hoje. Isso se deve em parte ao seu brutal domínio em relação a outras ferramentas do gênero. O Github já conta com cerca de 94 milhões de usuários ao redor do mundo. E veio para solucionar diversos problemas cruciais, que programadores mais antigos enfrentavam com VSCs anteriores. O Git, como é popularmente conhecido, consegue nos trazer diversos dados sobre os colaboradores dos projetos, e em que momento na linha do tempo do mesmo, estes fizeram alterações. Por conta disso, focaremos neste presente trabalho, na análise sobre as refatorações em projetos que estão dentro desta plataforma.

4.1.2.1 API Github

A primeira opção para nos fornecer os dados do projeto, assim como de seus contribuidores, é um recurso disponibilizado pela própria empresa Github, a qual é sua API de acesso aos dados. Com essa API, conseguimos obter todos os recursos que serão utilizados na CIref, como:

- Dia e horário em que uma modificação foi realizada. Com essa informação, será possível traçar toda a trajetória de alterações que um projeto sofreu, e se tiveram, por exemplo, picos de alterações próximas a datas específicas - Que servirá para embasar uma das visualizações descritas na Seção 4.2.
- Autor das modificações. Com o autor das modificações, conseguiremos assinalar alguns

[1] https://github.com/

comportamentos recorrentes a um dos contribuidores do projeto. Excelente para conhecer o perfil de desenvolvedores que desenvolvem um projeto.

– Identificação única para cada conjunto de modificações realizadas. Com essa identificação única, conhecida como SHA - que nada mais é do que um *hash* gerado automaticamente, ou seja, uma sequência de caracteres e dígitos distribuídos de forma aleatória - conseguimos agrupar as modificações em grupos maiores, facilitando o entendimento e visualização das mesmas.

4.1.2.2 API Github Integrada ao RefactoringMiner

Como segunda opção para conseguirmos os dados do projeto hospedado no Github, temos um serviço interno da ferramenta. Esta ferramenta interna foi considerada uma opção por conta da praticidade que ela proporciona. A ferramenta já internamente utilizada pelo RefactoringMiner para conseguir os dados sobre o repositório, então, se tornaria muito prático, se as informações sobre os autores dos *commits* - trechos de modificações, analisados pela ferramenta - viessem também como recurso. Porém, durante o estudo sobre a ferramenta escolhida para obter os dados sobre as refatorações, ficou claro a inutilização dessa API interna para conseguir os dados sobre os autores das modificações.

4.1.2.3 Utilização das duas opções

O que foi escolhido como melhor opção para a continuação desse presente trabalho, foi a utilização dessas duas ferramentas citadas nos Capítulos 4.1.1 e 4.1.2 em conjunto. A API Interna continuará sendo utilizada em conjunto com o RefactoringMiner para a obtenção dos dados sobre refatoração, porém, inseriremos no retorno dos resultados trazidos pela ferramenta, o SHA de cada extração realizada pela mesma. Dessa forma, como será descrito melhor na Seção 5.2, esse SHA será utilizado em chamadas paralelas à API do Github original, disponibilizada pela própria empresa. Assim, Teremos ambos os dados, das refatorações executadas, e os dados de que colaboradores as realizaram.

4.2 Definição das visualizações de refatorações

As visualizações, termo escolhido para demonstrar quais as visualizações estarão presentes na ferramenta proposta neste trabalho, foram pensadas juntando as duas fontes de

dados sobre os projetos, obtidas através das ferramentas escolhidas na Seção 4.1. Foi considerado uma visualização, apenas as informações que combinadas fornecessem *insights* sobre o projeto analisado.

4.2.1 Janelas de tempos

Antes mesmo de falar das visualizações em si, é preciso falar sobre os intervalos de tempos definidos, no qual, irão moldar todas as outras visualizações definidas na Seção 4.2. A ideia, primeiramente, partia do pressuposto que só existiria um período: o projeto inteiro. Porém, analisando melhor, seria mais interessante fornecer a escolha de analisar outras janelas de tempo também. Dessa forma, caso o usuário estivesse a procura de alguma informação mais recente, a teria. Então, as janelas de tempo selecionadas foram:

- **Todo o Projeto**: como o próprio nome já diz, o período analisado nesta janela de tempo é do primeiro *commit* realizado no projeto até o mais recente *commit*.
- **Últimos 3 meses**: essa janela de tempo foi escolhida pensando em servir como retrospectiva geral, de algum marco dentro do projeto. Ex: Retrospectiva - ato da metodologia de desenvolvimento de software ágil SCRUM - de uma ou mais *sprints*.
- **Esse mês**: a visualização mensal das visualizações definidas na seção 4.2, servem como uma retrospectiva mais recente das refatorações executadas, porém com a especificidade de ser o mês atual analisado e não os últimos 3 meses - ou 90 dias.
- **Essa semana**: a visualização semanal, está presente para servir como uma classificação atual de como aquele projeto está sendo refatorado, com o período mais curto, as visualizações se tornam mais específicas, permitindo ver detalhes mais aguçados sobre as refatorações do projeto.

4.2.2 Linha do tempo do número de refatorações

Esta visualização, tem em suma a responsabilidade de mostrar através de um gráfico de linha o progresso do número de refatorações em números absolutos ao decorrer do tempo nos projetos. Com os dados das duas ferramentas escolhidas na seção 4.1 em mãos, os dados sobre as refatorações sofrem algumas mudanças na sua estrutura, para facilitar a organização das chamadas a API do Github que serão feitas paralelamente. Os passos em ordem para montar a estrutura dos dados necessária para esta visualização, são:

1. Identificar a lista por todos os SHA's distintos, vindos diretamente dos dados sobre as

refatorações obtidas pela ferramenta RefactoringMiner.

2. Realizar chamadas paralelas à API do Github, para que as datas de quando cada *commit* foi realizado sejam conseguidas.
3. Fundir as informações de datas conseguidas pela API do Github, com a listagem das refatorações já obtidas anteriormente
4. Classificar a listagem por dias distintos, assim conseguimos contar os registros de refatorações executadas naquele mesmo dia.
5. Ordenar a listagem de classificação de forma ascendente, vindo dos registros de refatoração mais antigos até os mais novos.

Dessa forma, temos como resultado, um objeto ordenado com as datas e o total de refatorações executadas naquele projeto no determinado dia.

4.2.3 Refatorações por desenvolvedor em números absolutos

Esta visualização irá representar os números absolutos de refatorações executadas pelos colaboradores do projeto, não importando o tipo de refatoração, apenas a quantidade. Essa visualização é interessante, pois mostra o colaborador mais ativo no quesito refatoração de código. E paralelamente, essa visualização não é suficiente para indicar que um desenvolvedor é mais eficiente do que outro. Pois, os números absolutos de refatoração podem mascarar o verdadeiro esforço aplicado em, por exemplo, refatorações mais complexas, mas que não eram tão recorrentes. Por esse motivo, esta visualização é o par perfeito para a visualização que será descrita no Capítulo 4.2.5. Para obtermos os dados necessários para formular esta visualização são:

1. Classificar todas as refatorações obtidas através do RefactoringMiner por SHA distintos. Dessa forma temos uma listagem de SHA - que pode ser interpretado como *commit*, já que representa um identificador único do mesmo - contendo todas as refatorações que contenham o mesmo SHA.
2. Realizar chamadas paralelas à API do Github, para que as informações dos autores de cada *commit* sejam identificadas
3. Fundir as informações dos autores dos *commits*, conseguidas pela API do Github, com a listagem das refatorações, já classificadas por SHA.
4. Realizar a contagem dos registros com autores distintos.

Com esses passos, conseguiremos uma listagem de itens contendo o nome dos

colaboradores e também quantas refatorações cada um destes efetuaram durante o projeto.

4.2.4 Refatorações por tipo de refatoração

Sendo uma visualização mais simples, que não precisa da utilização da API do Github, a visualização por tipo de refatoração utiliza-se apenas dos dados obtidos do RefactoringMiner. E para conseguirmos construir uma visualização desses tipos, o único passo de tratamento de dados é a classificação da listagem trazida pelo RefactoringMiner pela propriedade "tipo".

Com os dados já classificados por tipo, frequentemente a listagem estaria com muitos itens distintos, o que poderia prejudicar uma visualização com proporções limitadas, como um gráfico do tipo *donut*. Isso se deve principalmente ao grande número de tipos de refatoração com a identificação suportada pelo RefactoringMiner. Portanto, foi necessário organizar os 90 tipos de refatorações, de uma forma que a visualização não fosse prejudicada. O processo para essa organização foi baseado na similaridade dos tipos de refatoração, já que a maioria deles têm parte dos nomes em comum com outros tipos. O resultado dessa organização, foi a seguinte lista:

- **Refactorings Move,** que dentre as refatorações abordadas por essa categoria, estão: *Move Method, Move Attribute, Move Class, Extract and Move Method, Move and Rename Class*. Pode-se notar, que existem alguns tipos que contêm o nome de duas categorias, esses itens são contados normalmente, visto que, para que uma refatoração composta seja realizada, ambos os tipos em questão precisam ser implementados.
- Refactorings Rename, que dentre as refatorações abordadas por essa categoria, estão: *Rename Parameter*, *Rename Variable* e *Rename Package*.
- Refactorings Modify, que dentre as refatorações abordadas por essa categoria, estão: *Modify Class Annotation* e *Modify Attribute Annotation*.
- Refactorings Extract, que dentre as refatorações abordadas por essa categoria, estão: *Extract Methods*, *Extract Superclass* e *Extract Interface*.
- Refactorings Change, que dentre as refatorações abordadas por essa categoria, estão: *Change Variable Type* e *Change Return Type*.
- Refactorings Add, que dentre as refatorações abordadas por essa categoria, estão: *Add Parameter*, *Add Variable Annotation*.
- Refactorings Remove, que dentre as refatorações abordadas por essa categoria, estão: *Remove Variable Modifier*, *Remove Thrown Exception Type* e *Remove Parameter Annotation*.
- Others Refactorings, são os tipos de refatoração que não se encaixam nos tipos descritos.

Estão nessas categorias tipos como: *Split Package*, *Collapse Hierarchy* e *Inline Attribute*.

Organizado dessa forma, a visualização das refatorações por tipo, será condensada de 90 tipos para apenas 8. Melhorando os resultados para uma visualização sem elementos que possam distorcê-la.

4.2.5 Refatorações por pontos

Esta visualização é o principal diferencial de outras ferramentas sobre refatorações. Com esta visualização, é possível ordenar a prioridade dos tipos de refatoração de acordo com cada projeto. A ordenação é feita pelos tipos de refatoração encontrados no Capítulo 4.2.4, em que o usuário final determina que tipo de refatoração tem prioridade de importância dentro de cada determinado projeto.

Esta foi uma estratégia pensada para não haver confusão de pensamentos quando se está analisando os números de refatorações executadas por colaboradores dos projetos. Isso se faz necessário, pois nem todos os tipos de refatoração tem o mesmo impacto para o projeto. E analisar somente os números absolutos de refatorações por desenvolvedor, pode, pode exemplo, mascarar as poucas refatorações realizadas por um desenvolvedor, mas que foram de grande impacto para o projeto.

Definindo a estratégia melhor, cada tipo de refatoração receberia uma pontuação conforme a ordem de prioridade definida pelo usuário final. Sendo, por padrão, peso 1 para todos os tipos de refatoração. Quando cadastrado uma nova ordem de prioridade pelo usuário, essa visualização trará uma média ponderada entre o peso de cada tipo de refatoração e a quantidade de refatorações desses determinados tipos, resultando em um número final de pontos.

4.2.6 Refatorações por caminho do arquivo

Partindo para especificidade, esta visualização é a responsável por fornecer a visualização dos arquivos mais refatorados conforme a janela de tempo selecionada. Esta visualização será formada por dois tipos de gráficos em conjunto, para mostrar de forma geral os arquivos mais refatorados dentro do projeto e também os detalhes das refatorações de cada arquivo, possibilitando a inspeção desses tipos ainda mais detalhados. Para conseguir essas informações, iremos transformar os dados recebidos diretamente pelo RefactoringMiner, realizando os seguintes passos:

– Extrair os caminhos absolutos das refatorações antes das alterações feitas pelas mesmas.

- Identificar os caminhos absolutos que são iguais, removendo os itens duplicados.
- Agrupar os itens das refatorações que possuem o caminho absoluto igual aos itens da lista de caminhos extraídos.

4.3 Implementação da Ferramenta CIref

A implementação da ferramenta será feita conforme as camadas definidas na Seção 5.2. A abordagem utilizada será começar a implementação com as API que servirá os dados sobre as refatorações. Esta API é construída utilizando a metodologia REST, que permite com que os recursos desta sejam consumidas por regras e protocolos bem definidos. A API em questão, terá uma autenticação básica de token único que será compartilhada com as outras partes da aplicação. Este token é uma sequência de caracteres geradas através da criptografia de uma frase simples. Esta API construída terá ao todo 3 rotas principais, que fornecerão variação da coleta dos dados sobre refatorações. São elas:

- A rota principal que será consumida pelo *frontend*, detalhadas na Seção 5.2, é a rota que identificará as refatorações executadas em todo o tempo do projeto.
- Outra rota que estará disponível nesta API é a de extrair as refatorações entre dois *commits*. Essa rota será utilizada principalmente quando alguma janela de tempo for selecionada pelo usuário.
- A última rota construída nesta API é a responsável por extrair as refatorações de uma *tag* dentro do repositório do Github. Esta rota foi construída visando os trabalhos futuros com esta ferramenta, para evitar retrabalho com a manutenção desta API.

Partindo para a construção do *backend* da ferramenta, muitas responsabilidades foram atribuídas a esta aplicação, as que a fazem ser a aplicação central da ferramenta proposta neste trabalho. Toda a construção de rotas que servirão o *frontend* e a manipulação dos dados para a construção das visualizações detalhadas na Seção 4.2, estão entre as principais responsabilidades desta ferramenta. Todo o controle de banco de dados também é feito nesta aplicação, desta forma, os registros só podem ser inseridos por uma única aplicação dentre as três camadas presentes na arquitetura da ferramenta detalhadas na Seção 5.2.

Sendo a única fonte direta de contato com o usuário final, o *frontend* foi pensado para ter um nível de segurança elevado, pois a ferramenta terá acesso aos repositórios permitidos pelos usuários finais e dessa forma terá acesso ao código-fonte das aplicações. Então, faz-se necessário garantir que nenhuma informação sobre repositórios alheios sejam visualizados indevidamente.

Por conta disso, o *frontend* da aplicação será construído utilizando o *framework* para a biblioteca ReactJs, denominado NextJs. Esse *framework*, garante que a autenticação seja feita pelo lado do servidor de uma aplicação *frontend*. Assim, nem a árvore de elementos da aplicação é mostrada sem que a autenticação seja concluída com sucesso.

4.4 Validação da Ferramenta CIref

Para a validação da ferramenta, foi planejado além da escolha do método avaliativo, também critérios para a seleção dos participantes deste método avaliativo. O modelo de aceitação tecnológica - em inglês, TAM - foi o método que mais se encaixou nos critérios que foram planejados. Os critérios escolhidos para a adesão do método avaliativo foram estes: (i) um método capaz de avaliar as intenções de uso futuro da ferramenta; (ii) um método capaz de conseguir capturar tanto as expectativas do usuário ao executar a ferramenta; (iii) Um método capaz de distinguir, na sua avaliação, as vantagens da ferramenta proposta neste trabalho, em relação as outras existentes.

O método avaliativo TAM, em tese, se exemplifica por suas duas principais vertentes: a da utilidade percebida e a da facilidade de uso. Sendo a primeira como a análise da probabilidade subjetiva dos usuários para o uso de uma ferramenta proposta aumentar seu desempenho em relação às ferramentas já existentes no mercado (KARAHANNA *et al.*, 1999). Já a facilidade de uso visa a análise do esforço cognitivo e mental para a utilização de um determinado *software* ou tecnologia.

5 FERRAMENTA CIREF

A CIref é uma ferramenta que visa fornecer visualizações de informações sobre o contexto histórico de refatorações em projetos JAVA, disponível em uma plataforma WEB. Ajudando os desenvolvedores e gestores de projetos a conhecer o perfil de refatorações executadas pelos colaboradores de um projeto, o número de refatorações ao decorrer do tempo e a comparar refatorações executadas por esses desenvolvedores.

5.1 Requisitos do CIref

A partir da análise dos objetivos da ferramenta, os Quadros 3 e 4 mostram o levantamento dos requisitos funcionais e não funcionais da ferramenta.

Quadro 3 – Requisitos funcionais da ferramenta CIref

Código	Identificação
RF1	A ferramenta deve permitir o login social com a ferramenta Github
RF2	A ferramenta deve permitir com que o usuário selecione os repositórios para a extração das refatorações
RF3	A ferramenta deve permitir com que outros repositórios sejam adicionados após o primeiro acesso
RF4	A ferramenta deve mostrar os dados de um repositório por vez
RF5	O usuário deve poder selecionar a janela de tempo em que ocorreram as refatorações
RF6	O usuário deve poder cadastrar a prioridade dos tipos de refatorações na plataforma
RF7	O usuário deve poder selecionar dois desenvolvedores para comparar as refatorações executadas por eles
RF8	A ferramenta deve sincronizar os dados sempre que um projeto for selecionado
RF9	O usuário deve poder visualizar os arquivos mais refatorados em um projeto
RF10	O usuário deve poder visualizar todos os tipos e subtipos de refatorações executadas em um projeto

Fonte: Elaborado pelo autor

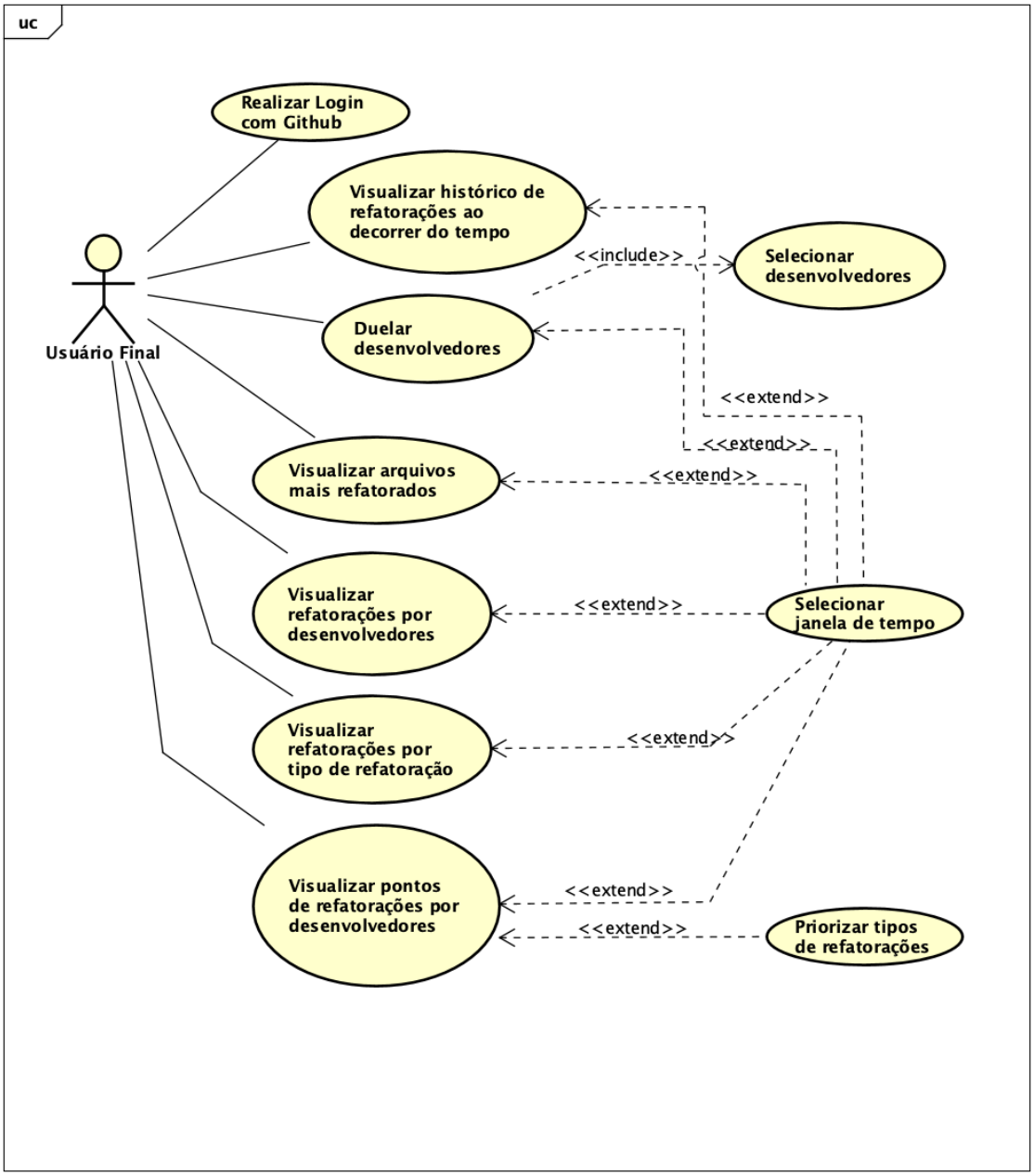
Quadro 4 – Requisitos não funcionais da ferramenta CIref

Código	Atributo	Identificação
RNF1	Disponibilidade	A ferramenta deve está disponível por 90% do tempo
RNF2	Maturidade	A ferramenta não deve falhar na visualização dos dados e nem nas suas atualizações
RNF3	Integridade	A ferramenta não deve permitir com que os dados sobre projetos de outras pessoas sejam visualizados
RNF4	Usabilidade	A ferramenta deve possuir uma boa distribuição dos elementos visuais em tela

Fonte: Elaborado pelo autor

A partir dos requisitos funcionais e não funcionais listados nos Quadros 3 e 4, foi construído o diagrama de casos de uso representado pela Figura 3. Neste diagrama está presente as principais funcionalidades que o usuário final pode realizar dentro da ferramenta proposta neste presente trabalho.

Figura 3 – Diagrama de casos de uso da ferramenta

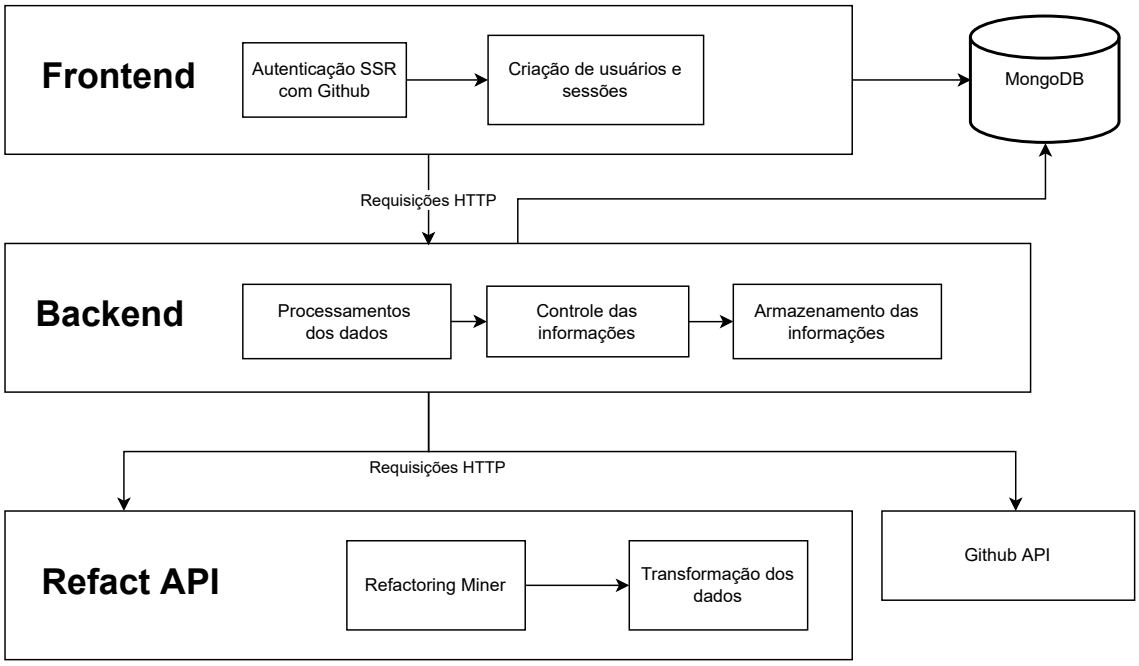


Fonte: Elaborado pelo autor.

5.2 Arquitetura do CIref

A arquitetura proposta na Figura 4, ressalta os três principais componentes para que a ferramenta consiga ser utilizada em sua integralidade. O diagrama está organizado do mais alto nível, sendo esse o *frontend*, onde o usuário final terá o contato direto com a ferramenta, até o mais baixo nível, onde estão presentes a API do Github e a API de extrações das refatorações.

Figura 4 – Diagrama de casos de uso da ferramenta



Fonte: Elaborado pelo autor.

5.2.1 Refact API

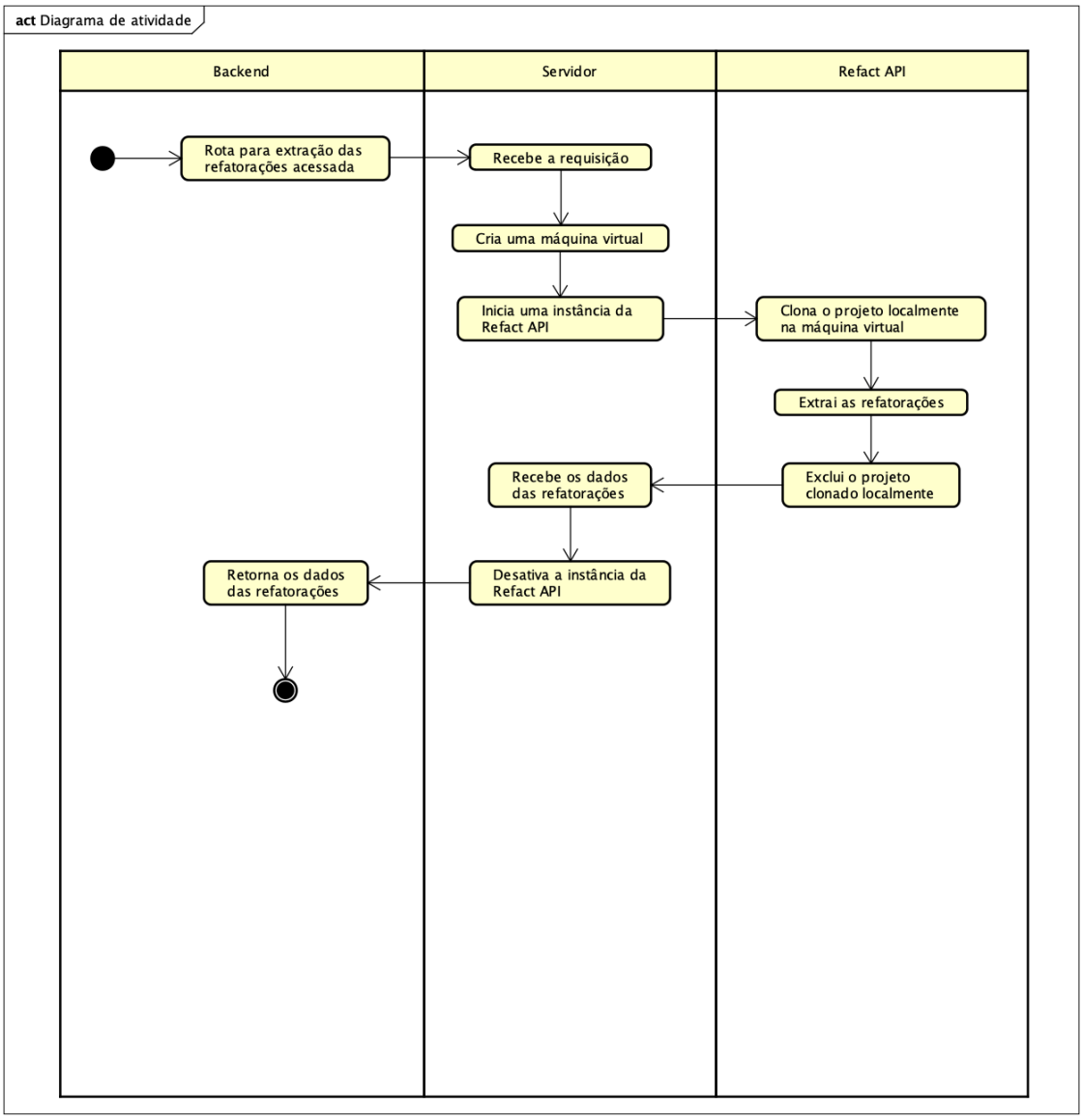
Para começar o detalhamento dos principais componentes da ferramenta, temos a que está no mais baixo nível em relação a utilização pelo usuário final. Essa API, denominada **Refact API**, foi criada do zero, mas conta com um papel fundamental para o funcionamento da CIref. Sua principal funcionalidade é servir como um *wrapper* - um embrulho, em português - para a utilização do RefactoringMiner.

A denominação *wrapper* se dá pela estratégia de decomposição escolhida para esta camada. Esta API está disponível através de um microsserviço ligado a camada de *backend* da ferramenta. Por adotar esse tipo de separação entre as camadas de *backend* e o *wrapper* da ferramenta RefactoringMiner, esta API pode ter sua manutenção feita, novas funcionalidades implementadas e ser escalada para atender altas demandas de requisições, de forma independente da camada que ela está ligada. A escolha por essa estratégia de decomposição se deu por este serviço ser a principal camada da ferramenta que está sendo proposta neste trabalho, portanto, as falhas neste serviço tem que serem mínimas, garantindo que o serviço fique de forma estável e disponível para cumprir os requisitos não funcionais RNF1 e RNF3 definidos na Seção 5.1.

Esta API foi construída utilizando a linguagem de programação Kotlin, com o

framework Spring Boot e o gerenciador de pacotes para aplicações JAVA, Maven [1]. Essas escolhas se justificam por possuírem um grau de facilidade maior para a realização do *deploy* do que uma aplicação comum JAVA.

Figura 5 – Diagrama de atividade representando a ativação da Refact API



Fonte: Elaborado pelo autor.

A Figura 5, contém o diagrama, que representa as atividades realizadas pelos atores presente no fluxo de uso da ferramenta CIref. Fica claro, como mostrado no diagrama, a vantagem de se adotar a estratégia de microsserviço para esta camada da arquitetura da ferramenta.

Importante, também, salientar a exclusão dos projetos clonados localmente nas máquinas virtuais. Essa ação reduz possíveis custos operacionais que a API poderia ter por conta

[1] https://maven.apache.org/

de alguns projetos grandes, que quando clonados, aumentaria significativamente os recursos computacionais requeridos para executar a Refact API.

5.2.2 Backend

O cérebro da ferramenta proposta neste trabalho, está nesta camada da arquitetura. O *backend* da aplicação terá a responsabilidade central de trabalhar com os dados vindos tanto da Refact API, quanto com os dados vindos da API do Github. Além de gerenciar todo o manejamento de rotas que servirão os dados diretamente para o *frontend*, esta camada da aplicação irá lidar com o gerenciamento do banco de dados, evitando a duplicidade dos registros das refatorações encontradas e as inserindo também na base de dados.

O *backend* da CIref é construído com o *framework* NestJS [2], que trabalha internamente com estratégias que garantem mais segurança e escalabilidade para a aplicação. Isso se deve, graças a sua estrutura em módulos, onde cada módulo é construído independentemente de outros módulos e pode ser inclusive reaproveitado em outras aplicações NestJs.

5.2.3 Frontend

O *frontend* desta ferramenta é desenvolvido utilizando a biblioteca ReactJs, que têm grande popularidade entre a comunidade de desenvolvimento de aplicações WEB. Aliado com essa biblioteca, a CIref é construída utilizando o framework NextJs que nos oferece funcionalidades essenciais quando se trata de informações delicadas, como um repositório de código do Github. Por este motivo, na ferramenta proposta neste trabalho toda a autenticação da aplicação é trabalhada com um conceito chamado *Server Side Rendering* (SSR), que em português é traduzida como Renderização pelo servidor.

Esse conceito se resume em executar algumas regras de negócio, antes mesmo da árvore de elementos da aplicação WEB ser montada. Permitindo com que validações sejam realizadas, chamadas a recursos externos como APIs - por exemplo, o *backend* detalhado na Seção 5.2.2 - sejam executadas e muito mais. Nesta aplicação, utilizamos deste recurso para tratar a autenticação dos usuários na plataforma, desse modo, quando o usuário está com o *login* inválido na plataforma, todas as informações serão mantidas em total segurança, visto que, sem esta autenticação, nenhuma informação sensível é mostrada em tela.

O *frontend*, também, será o ponto de contato mais próximo que o usuário terá da

[2] https://nestjs.com/

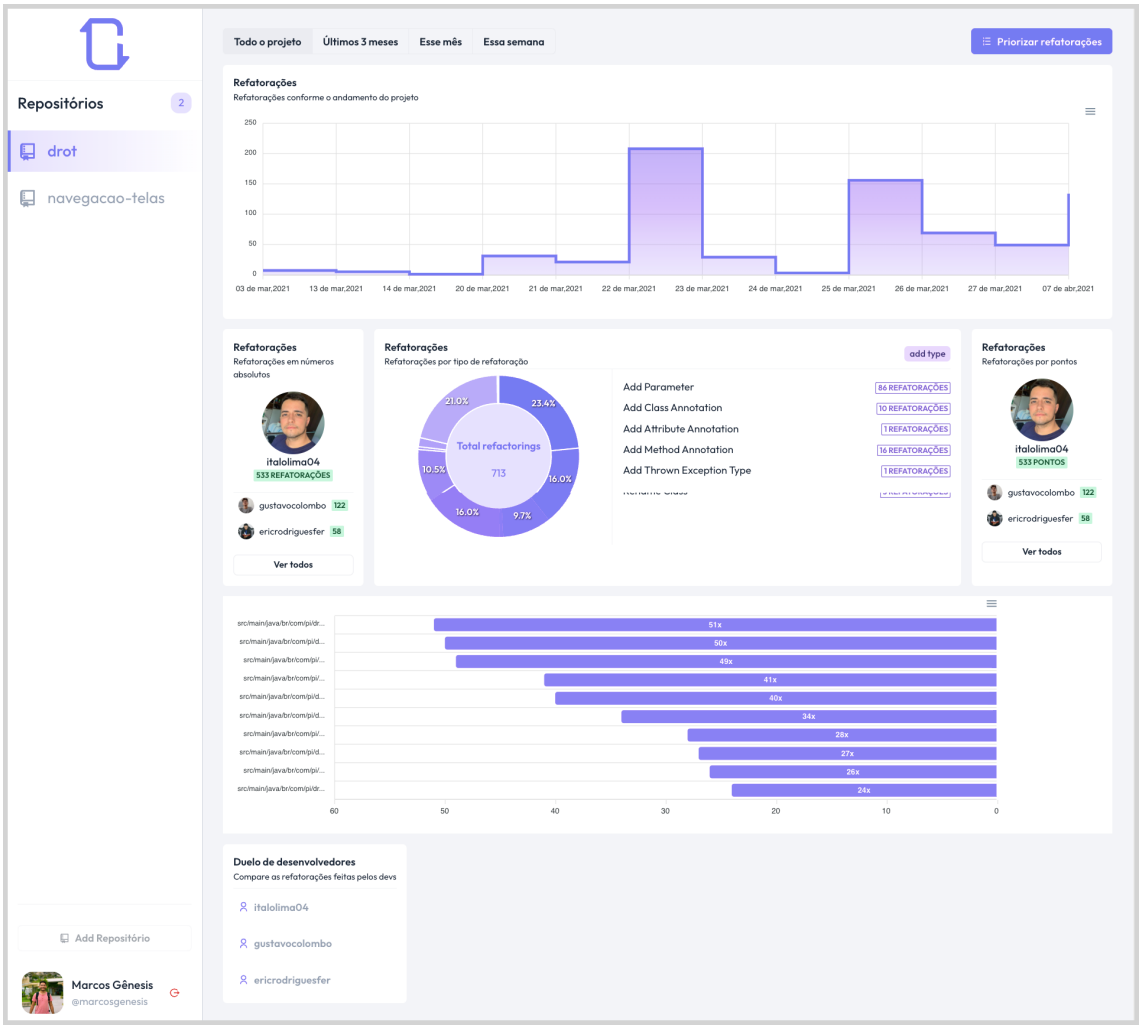
aplicação. Será através desta plataforma WEB, que o mesmo conseguirá utilizar todos os recursos disponíveis na plataforma e detalhados na seção a seguir.

5.3 Funcionalidades do CIref

5.3.1 Visão Geral da ferramenta

A Figura 6 representa a visão geral de todas as funcionalidades presentes na ferramenta proposta neste presente trabalho. Todas as funcionalidades mostradas nesta figura, serão descritas nas seguintes subseções.

Figura 6 – Visão geral da ferramenta

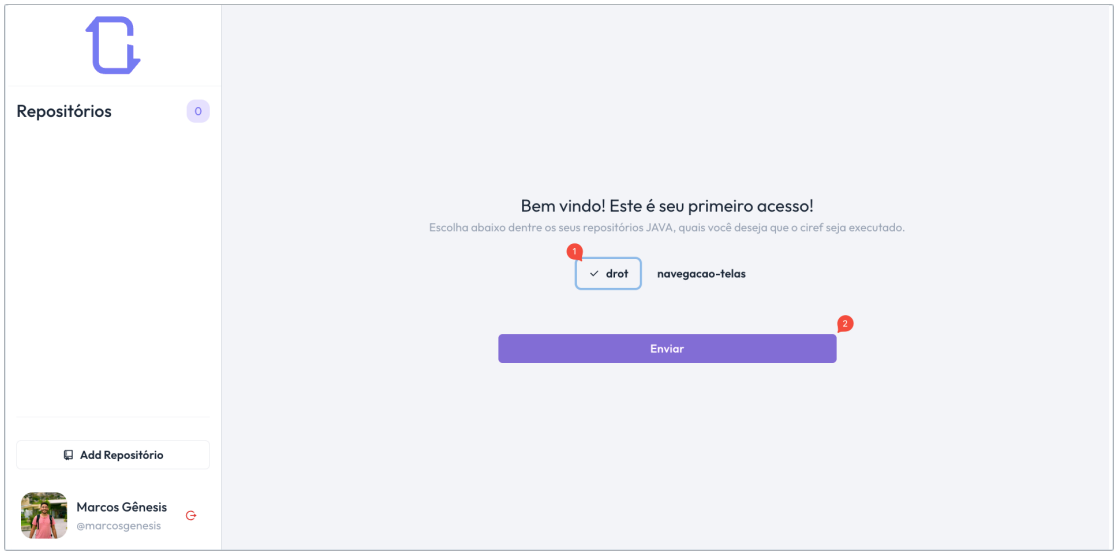


Fonte: Elaborado pelo autor.

5.3.2 Primeiro Acesso

Sendo a segunda ação dos usuários, após a realização do login com o Github, a tela mostrada na Figura 7 é apresentada já com a primeira utilização dos dados provindos diretamente da API do Github e filtrados para serem mostrados apenas projetos que a linguagem principal seja JAVA, sendo até o momento a única linguagem suportada para a extração das refatorações executadas na ferramenta proposta neste trabalho. O usuário, nesta tela, pode realizar duas ações principais. São elas: (1) selecionar um ou mais repositórios para que a ferramenta tenha acesso; (2) Clicar no botão Enviar para que essas preferências fiquem salvas no banco de dados.

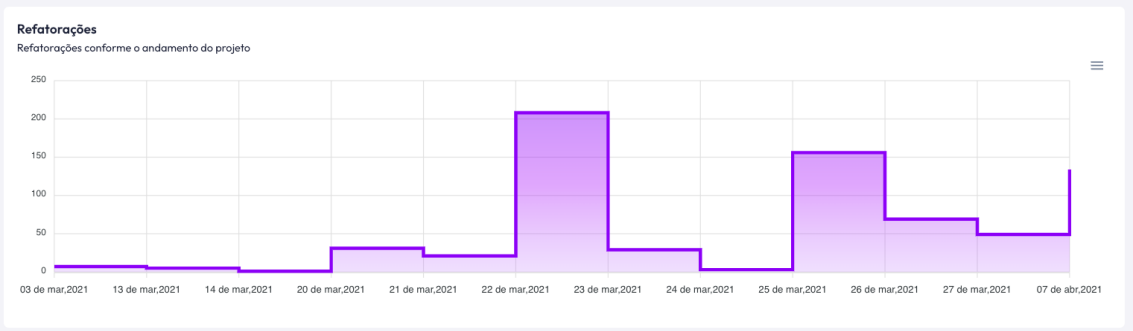
Figura 7 – Primeiro acesso do usuário na plataforma



Fonte: Elaborado pelo autor.

5.3.3 Visualização do número de refatorações ao decorrer do tempo

Figura 8 – Número de refatorações ao decorrer do tempo em um gráfico de linhas em um projeto real



Fonte: Elaborado pelo autor.

Esta funcionalidade, refere-se a visualização de todas as refatorações executadas dentro de um período - que, por padrão, é toda a duração do projeto. Na Figura 8, é possível observar a presença de um gráfico de linhas, com as linhas adaptadas para mostrarem os passos corretos da progressão dos números. Isso quer dizer que, o gráfico de linhas nesta funcionalidade não irá apresentar angulações acentuadas, como é o padrão deste tipo de gráfico.

Esta funcionalidade é muito interessante para identificar padrões incomuns no fluxo de refatorações executadas nos projetos. Na Figura 8, é possível reconhecer certos picos no número de refatorações perto de datas específicas. Se essas datas, fossem, por acaso, datas de fim de *sprints*, teríamos a informação de que os desenvolvedores estão efetuando mais refatorações perto dos fins de ciclo. Essa informação poderia servir como embasamento para alguma tomada de decisão dentro do projeto, e com o auxílio de outras funcionalidades descritas mais a frente, até mesmo os desenvolvedores que mais realizaram essas refatorações poderia ser identificado.

5.3.4 Visualização do número absoluto de refatorações por colaborador

Figura 9 – Número de refatorações absolutos por colaborador

Fonte: Elaborado pelo autor.

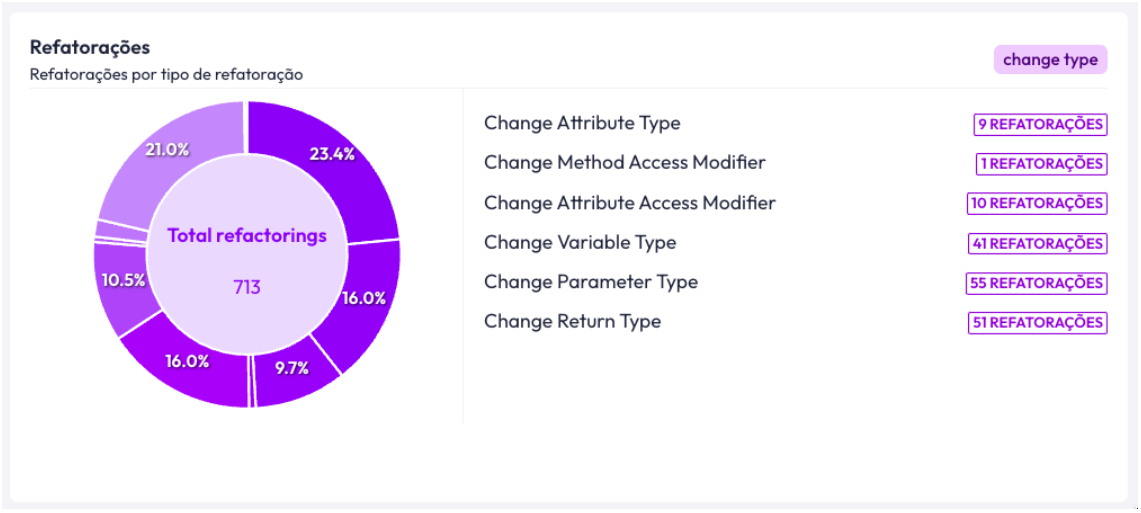
A principal informação conseguida através dessa funcionalidade é o *ranking* com os desenvolvedores que mais realizaram refatorações em um projeto de acordo com uma janela de tempo, que como já supracitado, por padrão esta janela refere-se ao tempo integral do projeto. Nesta visualização, não é considerado nenhum tipo de critério para a contabilização dos números

de refatorações, apenas os números absolutos das mesmas.

Com essa informação, é possível descobrir qual desenvolver tem maiores número de refatorações executas no projeto, porém, a informação trazida por esta funcionalidade precisa ter sua análise com cuidado. Isso porque, um número grande de refatorações implementadas nem sempre quer dizer que tal desenvolvedor é melhor refatorando do que outro desenvolvedor. Pois, refatorações mais simples de se implementar e refatorações complexas, nesta visualização, possuem o mesmo peso, o que pode causar uma má interpretação. Contudo, esta funcionalidade possui um ótimo complemento para o usuário final quando anexada as informações trazidas pela funcionalidade descrita no Capítulo 5.3.6.

5.3.5 Visualização dos tipos de refatoração mais recorrentes no projeto analisado

Figura 10 – Número de refatorações ao decorrer do tempo em um gráfico de linhas



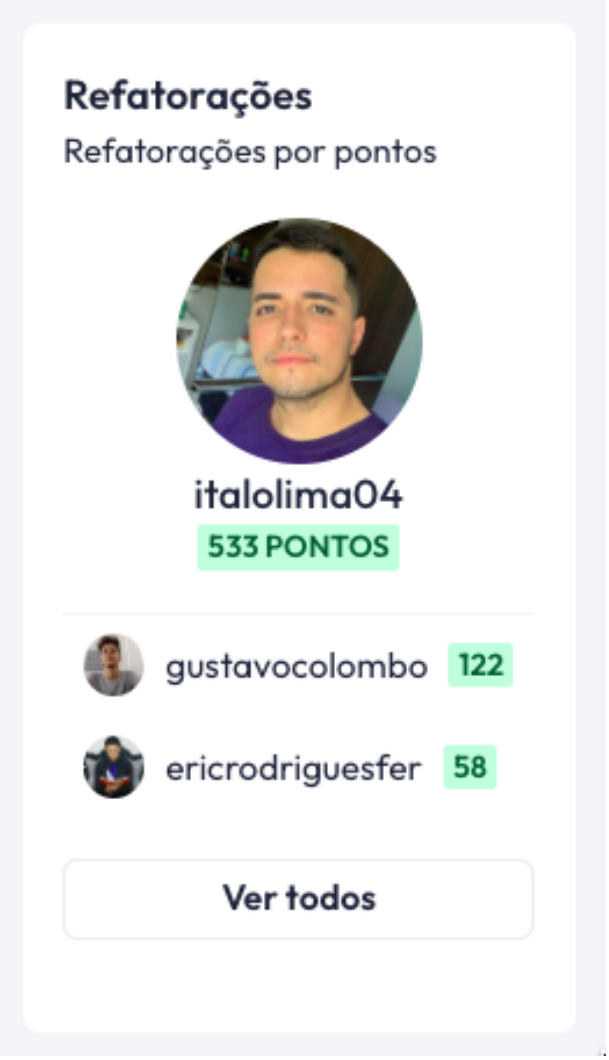
Fonte: Elaborado pelo autor.

Nesta funcionalidade, é possível visualizar a porcentagem de todos os tipos de refatoração executadas dentro de um projeto. A Figura 10 mostra como esta visualização é concretizada dentro da plataforma. Optamos por fazer essa representação através de um gráfico do tipo *donut*. E como explicado na Seção 4.2, foi preciso agrupar as refatorações em subtipos para que esta visualização em *donut* não fosse prejudicada. Porém, os dados originais sobre os tipos de refatorações não foram perdidos, e ao passar o mouse por cima das partições do gráfico *donut*, é possível visualizar ao lado os subtipos de refatoração presentes em cada categoria representada no gráfico.

5.3.6 Visualização da pontuação dos desenvolvedores baseado na priorização das refatorações

Esta funcionalidade é a mais complexa em relação as demais, isto o porquê, ela é a responsável por atribuir pesos diferentes para cada tipo de refatoração, ajustados ao contexto do repositório. Isso significa, que o usuário final, pode priorizar os tipos de refatoração que para determinado projeto são mais importantes em relação as outras.

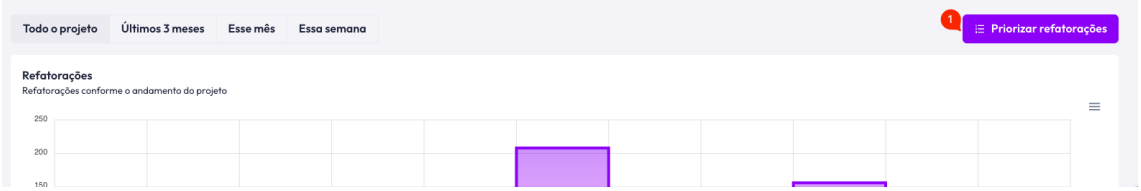
Figura 11 – Visualização padrão por pontuação



Fonte: Elaborado pelo autor.

Como a Figura 11 mostra, na visão padrão da ferramenta, todas as refatorações têm pesos iguais. Isso significa, que a Figura 11 tem os mesmos valores apresentados pela Figura 9, exceto pela denominação apresentada. Porém, esta visualização pode ser incrementada com a utilização da priorização dos tipos de refatoração, acionado ao clicar no botão indicado na Figura 12.

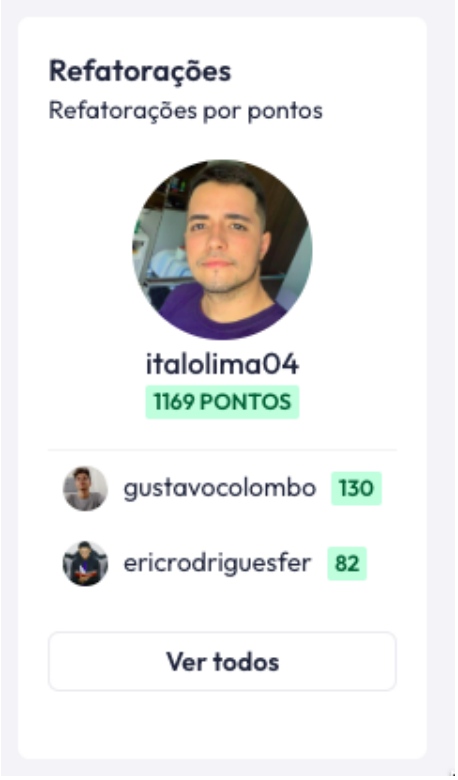
Figura 12 – Botão que aciona a priorização dos tipos de refatoração



Fonte: Elaborado pelo autor.

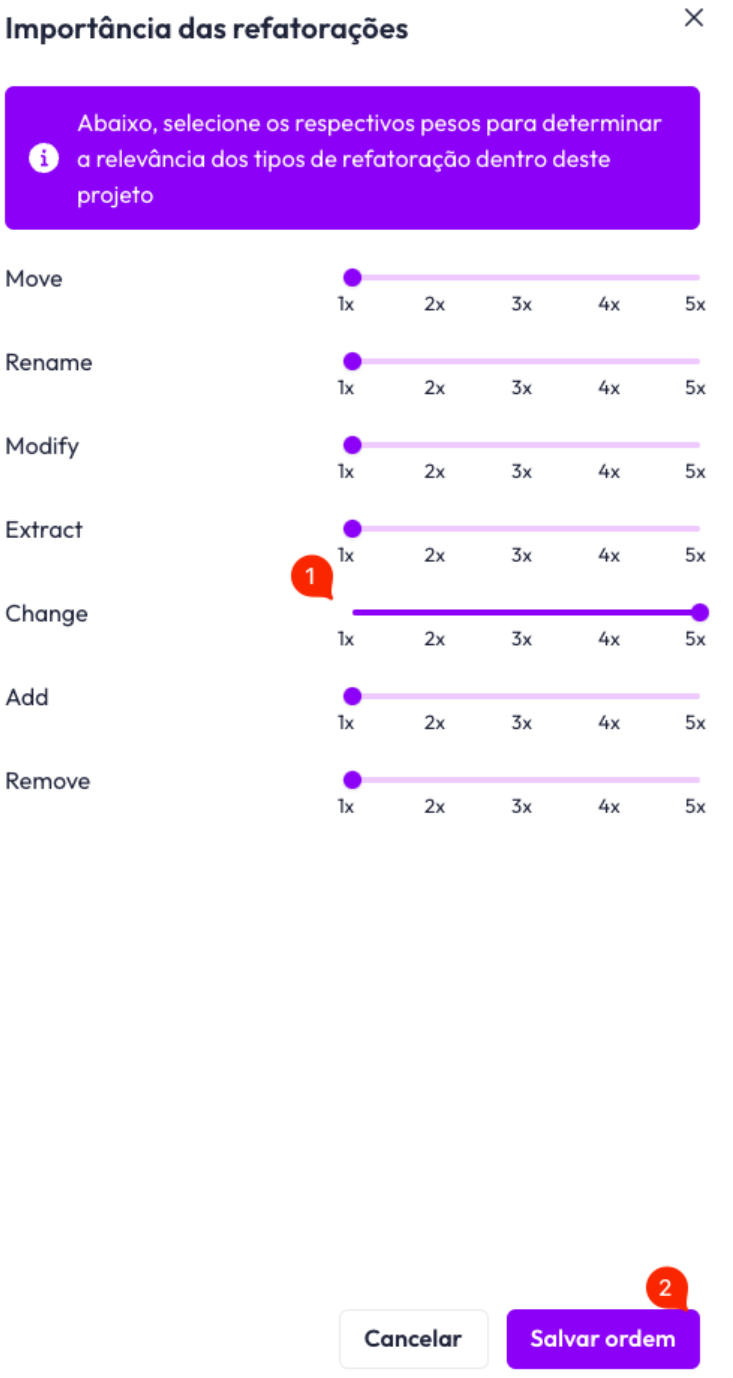
Quando clicado, o botão abre um *drawer*, uma seção em apenas parte da tela, onde é mostrado todas as categorias de refatorações, inicialmente com o peso equivalente a 1x - isso quer dizer que 1 refatoração tem equivalentemente 1 ponto. Nesta hora, o usuário poderá livremente priorizar as refatorações conforme o que for mais importante para o projeto. Na Figura 15, utilizada como demonstração, apenas o tipo *change* (1) teve seu valor modificado. Após determinar os pesos para essas categorias de refatoração, ao clicar no botão (2), a visualização previamente com os valores padrão mostrados na Figura 11 se transformará para a visão mostrado conforme a Figura 13. Desta forma, ficará mais fácil identificar quais desenvolvedores realmente estão realizando refatorações impactantes para o projeto.

Figura 13 – Número de pontos por refatorações após a priorização das refatorações



Fonte: Elaborado pelo autor.

Figura 14 – *Drawer* para realizar a priorização das refatorações



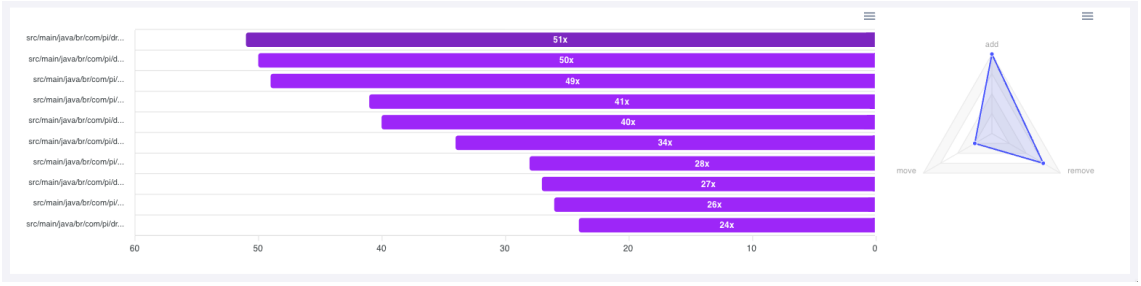
Fonte: Elaborado pelo autor.

5.3.7 Visualização dos arquivos mais refatorados

Esta funcionalidade traz a visualização dos 10 arquivos mais refatorados em uma janela de tempo de um projeto. Dentro dessa visão, é possível identificar além do caminho relativo dos arquivos mais modificados, também identificar quais os tipos de refatorações mais executadas dentro desse arquivo. Desta forma, é possível inferir algumas informações sobre a

visualização dos dados, como saber se os desenvolvedores estão enfrentando dificuldades em alguma implementação em um determinado arquivo.

Figura 15 – Visualização de arquivos mais refatorados



Fonte: Elaborado pelo autor.

5.3.8 Duelo entre desenvolvedores

Encerrando as funcionalidades da ferramenta proposta neste trabalho, A Figura 16 representa uma funcionalidade que pode ser bastante explorada em trabalho futuros. O seu objetivo é realizar uma comparação - que na ferramenta, foi chamado de duelo - entre os desenvolvedores do projeto. Com essa comparação, é possível identificar o perfil de refatoração que é mais executada pelos desenvolvedores. Esta funcionalidade também serve como complemento para a funcionalidade descrita na Seção 5.3.6, onde é possível visualizar a distribuição dos tipos de refatoração executadas por determinado desenvolvedor, que somados com o peso atribuído para os tipos de refatoração, pode ajudar a encontrar os melhores perfis de desenvolvedores que aplicam refatorações nos projetos.

Figura 16 – Duelo de desenvolvedores em relação a refatorações

Fonte: Elaborado pelo autor.

5.4 Impactos da Ferramenta

A ferramenta proposta neste trabalho, pode impactar diretamente algumas áreas da literatura de engenharia de software e outros campos científicos. A CIref pode ajudar na educação universitária de cursos voltados a tecnologia, por exemplo, ao fornecer visualizações capazes de identificar quais desenvolvedores de um projeto mais realizaram refatorações, neste caso os alunos seriam os desenvolvedores, e o(a) professor(a) seria o dono do projeto em que seria atribuído a ferramenta proposta neste trabalho. Além deste recurso, o responsável pelo projeto poderia priorizar os tipos de refatorações que seriam mais importantes em determinado contexto e assim, conseguir distinguir entre os participantes do projeto, aqueles que mais implementaram tal tipo específico de refatoração.

Outra área em que a ferramenta também pode impactar é na área de pesquisa sobre mineração de dados, sendo fundamentalmente uma ferramenta para visualizar quantidades relativamente grandes de insumos sobre refatorações. A CIref pode ajudar os pesquisadores a identificar mais facilmente os registros de refatoração que mais aparecem nos projetos, assim como a visualização de todas as outras funcionalidades da ferramenta proposta neste trabalho em relação aos projetos que podem passar por tais pesquisas sobre mineração de dados.

E o principal impacto, que já se pode observar sobre a utilização da ferramenta, é referente ao uso pelos desenvolvedores em seus projetos pessoais ou das empresas onde trabalham. Ao fornecer diferentes visualizações sobre as refatorações, os responsáveis pelos projetos podem analisar de forma mais detalhada o comportamento dos desenvolvedores em relação as refatorações de código e também observar o comportamento geral dos desenvolvedores em diferentes janelas de tempo.

6 VALIDAÇÃO DA FERRAMENTA

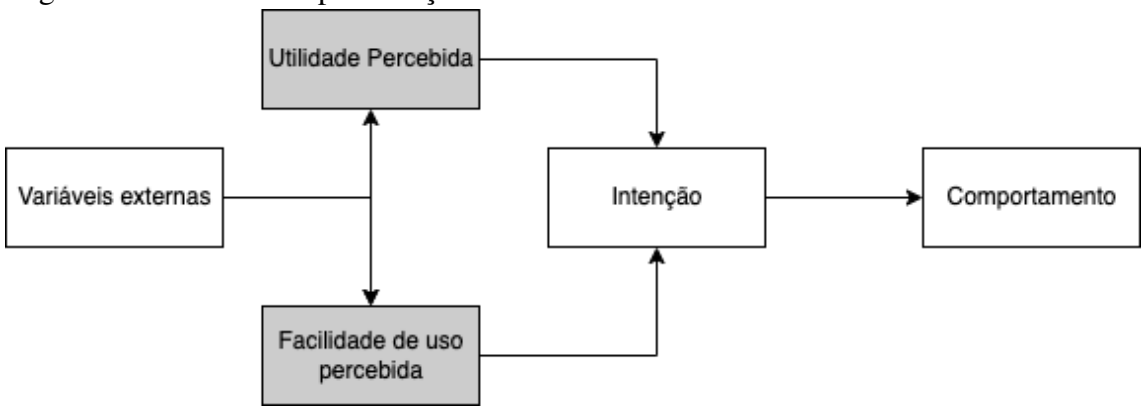
A ferramenta CIref foi construída conforme detalhado na seção 5.2, resultando em uma ferramenta capaz de exibir visualizações sobre o contexto histórico sobre refatorações em projetos JAVA. Esta seção, avaliará a aceitação da ferramenta pelos desenvolvedores. Na Seção 6.1 será apresentado o embasamento teórico para a escolha do método avaliativo. A Seção 6.2 mostrará todas as sentenças criadas para a avaliação da ferramenta. Finalizando, a Seção 6.3 mostrará o processo seletivo dos participantes

6.1 Modelo TAM e escala *Likert*

Inicialmente criado para focar no porquê usuários tendem a aceitar ou rejeitar a tecnologia da informação, o *TECHNOLOGY ACCEPTANCE MODEL* (TAM) possui uma base teórica forte e um vasto apoio acadêmico (DAVIS *et al.*, 1989). Este modelo se baseia em dois principais tópicos, ressaltados na Figura 17.

Como determinado por Davis *et al.* (1989) O primeiro determinante para a utilização correta deste modelo é a utilidade percebida, que em outras palavras busca coletar o grau em que um sistema pode melhorar o seu desempenho. E em sequência, o segundo determinante é o chamado Facilidade de uso percebida, onde se busca coletar qual o grau que o uso de um sistema pode ser livre de esforços. Porém, neste presente trabalho, também iremos considerar a intenção de uso futuro, para a verificação da consolidação das respostas obtidas nos dois itens principais do fluxo.

Figura 17 – Fluxo de representação do Modelo TAM



Fonte: (DAVIS *et al.*, 1989)

O modelo TAM foi escolhido para este presente trabalho por conta da sua capacidade de identificação da não aceitação - neste caso, de um sistema - pelos usuários e assim adotar as

medidas corretivas para este sistema (DAVIS *et al.*, 1989).

Associado ao modelo TAM, os construtos - aqui, chamados de sentenças - virão acompanhado dos itens presentes na Escala *Likert*. Esta escala foi fundamentalmente fundada para mensurar atitudes no contexto das ciências comportamentais, sendo uma listagem de sentenças em que os respondentes indicarão seu nível de concordância com as tais (LIKERT, 1932). A Figura 18 representa um exemplo desta escala, e seus graus de concordância que variam de (1) representando a total discordância com a sentença à (5) concordância total com a sentença.

Figura 18 – Exemplo de escala de Likert

A ferramenta proposta aumenta minha produtividade				
Discordo Totalmente	Discordo	Neutro	Concordo	Concordo Totalmente
1	2	3	4	5

Fonte: Elaborado pelo autor

6.2 Sentenças para o formulário

Para a formulação das sentenças utilizadas para avaliar a aceitação da ferramenta proposta neste presente trabalho, foi criado a Tabela 1 que mostra as sentenças criadas para cada uma das áreas citadas na Seção 6.1

6.3 Seleção dos participantes

Após formulado o questionário disponibilizado no Apêndice A com as sentenças para a avaliação da ferramenta descrita na Seção 6.2, e a escala de graus de concordância com as mesmas definida, como detalhado na Seção 6.1. A escolha dos participantes para a aplicação do formulário de avaliação foi pautada em alguns critérios:

- Ter no mínimo 2 anos de experiência com desenvolvimento de software. Esse critério foi selecionado para filtrar candidatos a participar da avaliação da ferramenta que possuem uma experiência consolidada, com a participação em projetos reais e entende como funciona a demanda da refatoração em projetos comerciais.
- Trabalhar ou já ter trabalhado em projetos JAVA. Por se tratar de uma linguagem com um nicho muito específico, é preciso selecionar corretamente os participantes que já tenham trabalhado com a linguagem JAVA.

Tabela 1 – Sentenças formuladas para a aplicação do questionário

Sentenças sobre Utilidade Percebida (UP)
UP1. Utilizar a ferramenta proposta me ajudará a ter mais controle sobre meu código.
UP2. Utilizar a ferramenta proposta reduzirá o tempo gasto com refatorações menos importantes.
UP3. Utilizar a ferramenta proposta ajudará a melhorar a qualidade do meu trabalho/projetos.
UP4. Utilizar a ferramenta proposta aumentará a minha produtividade.
UP5. Utilizar a ferramenta proposta me fornecerá mais conhecimentos sobre refatorações.
UP6. Utilizar a ferramenta proposta permitirá com que eu ganhe tempo no trabalho/projetos.
UP7. As visualizações trazidas pela ferramenta proposta são úteis no dia-a-dia do meu trabalho.
UP8. Verificar as visualizações trazidas pela ferramenta proposta me permite comparar meu desempenho com outros desenvolvedores.
UP9. A visualização trazida pela plataforma sobre o número de refatorações executadas ao decorrer do tempo é um diferencial em relação as outras ferramentas existentes.
UP10. No geral, achei a ferramenta proposta fácil de se utilizar.
Sentenças sobre Facilidade de Uso Percebida (FUP)
FUP1. Minha interação com a ferramenta proposta é fácil de se compreender.
FUP2. A ferramenta proposta mostra informações importantes sobre refatoração.
FUP3. Ao utilizar a ferramenta consigo informações para implementar outros tipos de refatoração.
FUP4: No geral, acho a ferramenta proposta, útil para o meu trabalho.
Sentenças sobre Autoprevisão de Uso Futuro (AUF)
AUF1: Caso disponível, utilizaria a ferramenta proposta em meu trabalho/projetos.
AUF2: Acho a visão trazida pela ferramenta sobre refatorações mais interessante do que as outras ferramentas/plugins trazem.

Fonte: Elaborado pelo autor

O resultado da seleção foi um total de 8 participantes, sendo 50% com mais de 4 anos de experiência em desenvolvimento de software e em projetos JAVA, 37,5% dos demais possuíam 3 anos de experiência e apenas 12,5% tinham apenas o requisito mínimo exigido de dois anos de experiência.

O procedimento para a aplicação deste formulário foi realizado de forma presencial, devido a todos os participantes serem da mesma cidade de origem. A aplicação do formulário foi o último passo realizado para a validação da utilização da ferramenta proposta neste trabalho. O primeiro passo realizado foi a apresentação da ferramenta, assim como a apresentação de todas as funcionalidades, descritas neste documento na Seção 5.3. Tais funcionalidades foram mostradas com projetos reais do autor deste trabalho, em que já é simulado o uso real da ferramenta. Logo após, foi realizado um sorteio para definir a ordem de utilização da ferramenta entre os 8 participantes, visto que, a ferramenta só estava disponível em um único computador para a validação. Cada um dos oito participantes foi instruído apenas para a configuração inicial da ferramenta e a conexão com seus repositórios. Logo após isso, toda a utilização ficou por conta de cada participante, recomendado apenas a exploração de todas as funcionalidades propostas pela ferramenta, não tendo um tempo estimado para essa exploração. Após o término da utilização de

cada um dos participantes, foi solicitado o preenchimento do formulário contendo as sentenças formuladas como mostrado na Tabela 1.

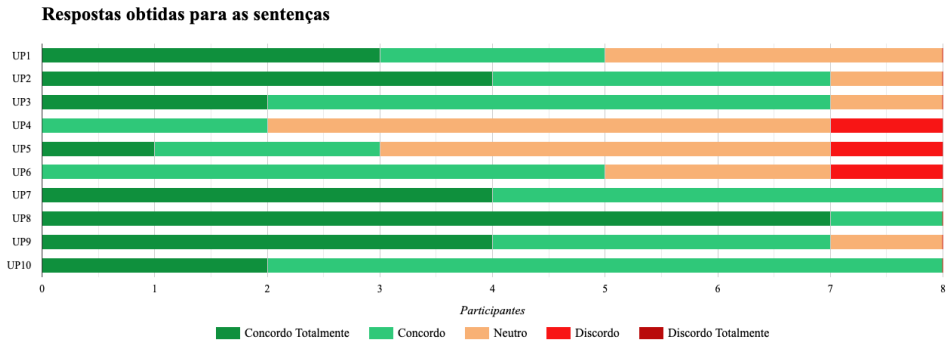
6.4 Resultados Encontrados

Nesta seção, serão apresentados os resultados encontrados após a aplicação do formulário com as sentenças definidas durante a Seção 6.2. Os resultados apresentados nesta seção, conforme as seções definidas no modelo de avaliação descritos na Seção 6.1.

6.4.1 Utilidade Percebida

A Figura 19 mostra a distribuição das respostas coletadas no formulário respondido pelos participantes selecionados, em relação à utilidade percebida da ferramenta. Todas as identificações apresentadas na figura, representam uma sentença escolhida na Seção 6.2.

Figura 19 – Visão geral das respostas obtidas sobre utilidade percebida.



Fonte: Elaborado pelo autor

As respostas coletadas no formulário de avaliação da ferramenta, confirmam a intenção positiva no uso posterior da ferramenta proposta, com nenhuma discordância dos participantes selecionados no quesito facilidade da utilização da mesma. 100% dos participantes demonstraram concordância com tal fator, sendo destes, 75% concordaram que a ferramenta proposta é de fácil utilização, e os outros 25% concordaram com certeza sobre tal sentença. Isso indica que as funcionalidades propostas para a ferramenta foram bem planejadas e são de fácil execução.

Algo presente nos resultados obtidos foi o considerado grau de neutralidade em relação aos itens **UP4** e **UP5**. Isso indica que apesar da ferramenta proporcionar visualizações sobre o contexto histórico de refatorações dos projetos, 62,5% dos participantes acreditam que

essas visualizações não irão aumentar e nem diminuir sua produtividade em um ambiente de trabalho ou projetos pessoais. Enquanto 25% dos participantes acreditam que haverá um ganho de produtividade. Podemos inferir destes resultados que a ferramenta proposta pode apenas servir como uma ferramenta para visualização de dados, e não uma ferramenta diferencial, que irá aumentar a produtividade dos desenvolvedores nos projetos. Com essa, informação é possível observar a oportunidade de trabalhos futuros visando o incremento das funcionalidades desta ferramenta para a obtenção de resultados considerados positivos para a mesma.

As sentenças **UP4, UP5** e **UP6** foram as únicas sentenças de todo o formulário que continham desconcordâncias com as sentenças apresentadas. Apesar do percentual de discordância desses 3 itens serem igualmente de 12,5% - um valor baixo - podemos afirmar que há uma preocupação dos desenvolvedores em relação ao ganho de tempo, produtividade e conhecimento sobre refatorações ao se utilizar a ferramenta proposta neste trabalho.

O grande ponto positivo retirado dos dados coletados, foram referentes as respostas obtidas para as sentenças **UP7**, **UP8** e **UP9**, onde 95,8% dos participantes do formulário avaliativo apresentaram respostas com grau de concordância positivo em relação à utilidade das visualizações trazidas pela plataforma no dia-a-dia de trabalho e também na possibilidade de comparação de desempenho entre os próprios desenvolvedores no quesito refatoração, com o salientamento de que as visões presentes na ferramenta são um diferencial quando comparadas a outras ferramentas que abordam o tema refatorações.

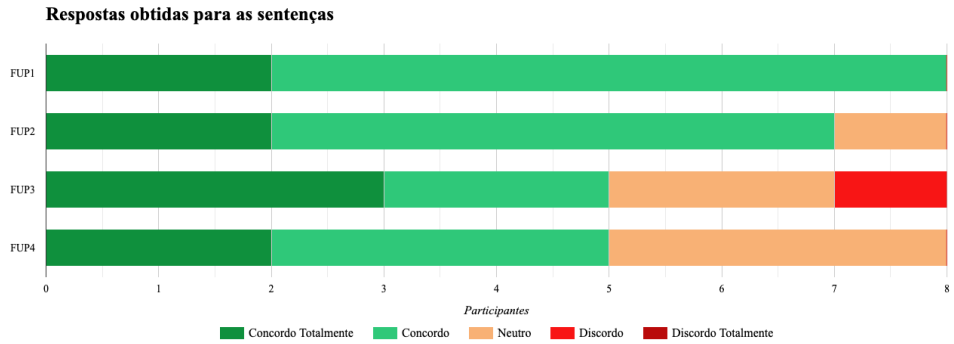
6.4.2 Facilidade de Uso Percebida

A Figura 21 mostra a distribuição das respostas coletadas no formulário respondido pelos participantes selecionados, em relação à facilidade de uso percebida da ferramenta. Todas as identificações apresentadas na figura, representam uma sentença escolhida na Seção 6.2.

Observando os resultados obtidos pelo formulário avaliativo da ferramenta pelos participantes, conseguimos identificar o grau de concordância de 100% dos participantes no que se refere a sentença **FUP1**. Da porcentagem total refente as repostas obtidas, 75% concorda com a afirmação apresentada que a interação usuário-ferramenta é fácil de se compreender, enquanto os outros 25% concordam com a afirmação integralmente. Ou seja, somado com os resultados obtidos na Seção 6.4.1, podemos comprovar a eficácia da ferramenta no quesito interação com o usuário final e facilidade na execução das funcionalidades planejadas.

87,5% das respostas obtidas para a sentença **FUP2** foram com graus de concordância

Figura 20 – Visão geral das respostas obtidas sobre facilidade de uso percebida.



Fonte: Elaborado pelo autor

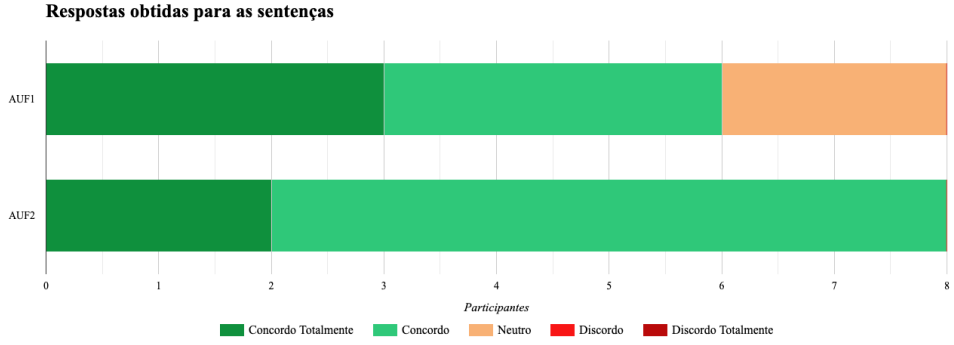
positiva. Esse dado, mostra a eficácia das visualizações das refatorações aliadas ao contexto histórico do projeto fornecido pela ferramenta. Além disso, com a validação da importância de tais informações, oportunidades de trabalhos futuros surgem como opções para a exploração dessas visualizações trazidas pela plataforma.

Porém, como os dados obtidos para a sentença **FUP3** mostram, a tendência apesar de ser positiva para que os desenvolvedores consigam implementar outros tipos de refatorações baseado nas informações apresentadas na ferramenta, isto não é um consenso dos participantes do formulário avaliativo, onde cerca de 12,5% dos mesmos discordam que a ferramenta disponibiliza informações para que outros tipos de refatorações sejam implementadas.

Em resumo, 62,5% dos participantes do formulário consideraram a ferramenta útil para o dia-a-dia de trabalho desses desenvolvedores. Ou seja, podemos confirmar que a ferramenta proposta teve sua utilidade comprovada para o trabalho diário dos desenvolvedores.

6.4.3 Autoprevisão de Uso Futuro

Figura 21 – Visão geral das respostas obtidas sobre autoprevisão de uso futuro.



Fonte: Elaborado pelo autor

75% dos participantes do formulário de avaliação da ferramenta demonstraram nas respostas do formulário graus de concordâncias positivas para a utilização da ferramenta em seus projetos. E 100% dos participantes concordaram que as visualizações trazidas pela ferramenta é mais interessante do que as visualizações existentes em outras plataformas. Com esses resultados, podemos afirmar que os desenvolvedores se interessam em utilizar a ferramenta proposta neste trabalho em seus projetos.

6.5 Limitações da ferramenta

Algumas limitações foram encontradas em relação à construção da ferramenta e o possível uso futuro desta. A primeira limitação encontrada foi em relação a frequente utilização das chamadas a API do Github, onde o *token*, responsável por garantir a autenticação da ferramenta para com o Github, se expira facilmente por conta da sobrecarga de requisições. Sendo necessário, modificar a autenticação da ferramenta proposta neste trabalho com a API do Github. Outra limitação, também encontrada, foi observada durante a validação feitas com os participantes, onde cerca de 37,5% dos participantes tiveram dificuldades de entender o estado da aplicação, enquanto esta, estava sincronizando os dados das refatorações dos projetos selecionados, sendo necessário o planejamento de como ficará a plataforma quando os dados estiverem vazios ou desatualizados.

6.6 Ameaças à validade

Dentre as ameaças a validade mais impactantes para este trabalho, está a utilização da ferramenta RefactoringMiner como fonte única dos dados para a busca do histórico de refatorações dentro dos projetos escritos na linguagem Java. As refatorações trazidas pela ferramenta não são filtradas para selecionar apenas os tipos de refatorações que preservam o comportamento do código. Uma ameaça a validade detectada também é a escolha das tecnologias utilizadas para a construção da ferramenta proposta neste trabalho, sendo de suma importância a validação futura das mesmas para garantir a segurança das informações aliadas com um bom desempenho, verificando se as opções tecnológicas adotadas neste trabalho foram assertivas. Outra ameaça a validade deste trabalho está na não validação específica de cada visualização para confirmar a eficácia do tipo de gráfico utilizado em cada uma destas. Por último, uma das ameaças a validade deste trabalho se dar ao tamanho do espaço amostral dos participantes

da etapa de validação da ferramenta, sendo oito um número consideravelmente pequeno para confirmar a eficácia e usabilidade da ferramenta.

7 CONCLUSÃO

Neste trabalho foi proposto a criação da ferramenta CIref, permitindo a visualização do contexto histórico de refatorações em projetos JAVA. Para construção da CIref, foi utilizada a ferramenta RefactoringMiner para a extração das refatorações executadas em um projeto, devido a sua fácil implementação via API e sua vasta quantidade de tipos de refatorações suportadas. Foi escolhida também a API do Github para obtenção dos dados sobre os autores dos *commits* e também quando os mesmos foram realizados.

Após a definição das ferramentas, montamos os requisitos necessários para a ferramenta e também definimos um diagrama de casos de uso para representar as funcionalidades dentro do sistema, dentre as principais, estão as: (i) visualização da linha do tempo do número de refatorações executadas, que permite a identificação de períodos recorrentes em que o número de refatorações aumenta ou diminui mais do que o normal; (ii) duelo entre os desenvolvedores, permitindo a comparação do desempenho de dois desenvolvedores em simultâneo, para permitir a identificação de que tipo de perfil, ambos os desenvolvedores possuem para as refatorações;(iii) priorização dos tipos de refatoração, permitindo com que para cada projeto analisado refatorações tenham pesos diferentes e assim, sendo possível a identificação dos desenvolvedores que mais implementaram as refatorações mais importantes definidas para cada projeto, representadas por pontuação; e, (iv) visualização de arquivos mais modificados dentro de um projeto, o que permite identificar possíveis dificuldades enfrentadas pelos desenvolvedores.

Foi criado, também, toda a estruturação da arquitetura da ferramenta e o detalhamento das partes fundamentais da mesma, como a Refact API - API criada para prover os dados sobre as refatorações, o *backend* da aplicação, responsável por fundir os dados, controlar as ações da ferramenta e evitar duplicidade de dados, e também o *frontend* da ferramenta, sendo o principal ponto de acesso para o usuário final, é neste que ele irá conseguir usufruir de todas as funcionalidades planejadas.

Neste trabalho, também definimos o TAM como modelo de avaliação que seria utilizado para avaliar a ferramenta proposta, assim com também foi descrito todas as sentenças utilizadas pelo mesmo em seus 3 principais níveis: (i) utilidade percebida, (ii) facilidade de uso percebida e (iii) autoprevisão de uso futuro. Além disso, foi decidido todos os critérios necessários para a escolha dos participantes do formulário de avaliação da ferramenta, que resultou em 8 desenvolvedores com 2 anos ou mais de experiência em desenvolvimento de software. A avaliação da ferramenta foi feita de forma presencial, com todos os 8 participantes

após os mesmos receberem uma aula sobre como utilizar a ferramenta, e adicionarem os próprios repositórios a mesma.

Os resultados encontrados, apontaram uma necessidade pela ferramenta por parte dos desenvolvedores. Os pontos fortes da ferramenta foram as visualizações únicas comparadas a outras ferramentas existentes e também a facilidade de uso da ferramenta, tendo um alto grau de aceitação por parte dos participantes da avaliação. Como pontos negativos, foi observado a falta de incentivos a prática de refatorações com base nas informações apresentadas em tela e também a discordância de parte dos desenvolvedores com a afirmação de que ganhariam tempo adotando o uso da ferramenta.

REFERÊNCIAS

ALOMAR, E. A.; PERUMA, A.; MKAOUER, M. W.; NEWMAN, C. D.; OUNI, A. Behind the scenes: On the relationship between developer experience and refactoring. **Journal of Software: Evolution and Process**, Wiley Online Library, p. e2395, 2021.

ALSHAYEB, M.; LI, W.; GRAVES, S. **An empirical study of refactoring, new design, and error-fix efforts in extreme programming**. [s.l.]. 2001.

ARCHAMBAULT, S. G.; HELOUVRY, J.; STROHL, B.; WILLIAMS, G. Data visualization as a communication tool. **Library Hi Tech News**, v. 32, p. 1–9, 2015.

BAVOTA, G.; LUCIA, A.; PENTA, M. D.; OLIVETO, R.; PALOMBA, F. An experimental investigation on the innate relationship between quality and refactoring. **Journal of Systems and Software**, v. 107, 05 2015.

BOGART, A.; ALOMAR, E. A.; MKAOUER, M. W.; OUNI, A. Increasing the trust in refactoring through visualization. In: **PROCEEDINGS OF THE IEEE/ACM 42ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING WORKSHOPS**. New York, NY, USA: Association for Computing Machinery, 2020. (ICSEW'20), p. 334–341. ISBN 9781450379632. Disponível em: https://doi.org/10.1145/3387940.3392190.

BOGART, A.; ALOMAR, E. A.; MKAOUER, M. W.; OUNI, A. Increasing the trust in refactoring through visualization. In: **PROCEEDINGS OF THE IEEE/ACM 42ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING WORKSHOPS**. New York, NY, USA: Association for Computing Machinery, 2020. (ICSEW'20), p. 334–341. ISBN 9781450379632. Disponível em: https://doi.org/10.1145/3387940.3392190.

BRITO, R.; VALENTE, M. T. Raid: Tool support for refactoring-aware code reviews. In: **2021 IEEE/ACM 29TH INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION (ICPC)**. [*S. l.*: *s. n.*], 2021. p. 265–275.

CARPENDALE, S.; GHANAM, Y. **A Survey Paper on Software Architecture Visualization**. PRISM, 2008. Disponível em: https://prism.ucalgary.ca/handle/1880/46648.

CASERTA, P.; ZENDRA, O. Visualization of the static aspects of software: A survey. **IEEE Transactions on Visualization and Computer Graphics**, v. 17, n. 7, p. 913–933, July 2011. ISSN 1941-0506.

DAVIS, F. D.; BAGOZZI, R. P.; WARSHAW, P. R. User acceptance of computer technology: A comparison of two theoretical models. **Management Science**, INFORMS, v. 35, n. 8, p. 982–1003, 1989. ISSN 00251909, 15265501. Disponível em: http://www.jstor.org/stable/2632151.

DIEHL, S. **Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software**. Springer Berlin Heidelberg, 2007. ISBN 9783540465058. Disponível em: https://books.google.com.br/books?id=rToBgkAuzN4C.

FOKAEFS, M.; TSANTALIS, N.; STROULIA, E.; CHATZIGEORGIOU, A. Jdeodorant: identification and application of extract class refactorings. In: **2011 33RD INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE)**. [*S. l.*: *s. n.*], 2011. p. 1037–1039.

FONTANA, F. A.; BRAIONE, P.; ZANONI, M. Automatic detection of bad smells in code: An experimental assessment. **J. Object Technol.**, v. 11, n. 2, p. 5–1, 2012.

FOWLER, M. **CodeSmell**. 2006. Disponível em: https://martinfowler.com/bliki/CodeSmell.html.

FOWLER, M. **Refactoring**: improving the design of existing code. [*S. l.*]: Addison-Wesley Professional, 2018.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. **Refactoring**: improving the design of existing code. Addison-Wesley Professional, 1999. Hardcover. Acesso em: 10 dez. 2022. ISBN 0201485672. Disponível em: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{&}path=ASIN/0201485.

GLASS, R. Maintenance: less is not more. **IEEE Software**, v. 15, n. 4, p. 67–68, 1998.

KARAHANNA, E.; STRAUB, D. W.; CHERVANY, N. L. Information technology adoption across time: A cross-sectional comparison of pre-adoption and post-adoption beliefs. **MIS Q.**, Society for Information Management and The Management Information Systems Research Center, USA, v. 23, n. 2, p. 183–213, jun 1999. ISSN 0276-7783. Disponível em: https://doi.org/10.2307/249751.

KIM, M.; ZIMMERMANN, T.; NAGAPPAN, N. A field study of refactoring challenges and benefits. In: **PROCEEDINGS OF THE ACM SIGSOFT 20TH INTERNATIONAL SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING**. New York, NY, USA: Association for Computing Machinery, 2012. (FSE '12). ISBN 9781450316149. Disponível em: https://doi.org/10.1145/2393596.2393655.

KIM, M.; ZIMMERMANN, T.; NAGAPPAN, N. An empirical study of refactoringchallenges and benefits at microsoft. **IEEE Transactions on Software Engineering**, IEEE, v. 40, n. 7, p. 633–649, 2014.

LACERDA, G.; PETRILLO, F.; PIMENTA, M.; GUÉHÉNEUC, Y. Code smells and refactoring: A tertiary systematic review of challenges and observations. **CoRR**, abs/2004.10777, 2020. Disponível em: https://arxiv.org/abs/2004.10777.

LANZA, M.; MARINESCU, R.; DUCASSE, S. **Object-Oriented Metrics in Practice**. Berlin, Heidelberg: Springer-Verlag, 2005. ISBN 3540244298.

LIKERT, R. **A Technique for the Measurement of Attitudes**. Archives of Psychology, 1932. (A Technique for the Measurement of Attitudes, Nº 136-165). Disponível em: https://books.google.com.br/books?id=9rotAAAAYAAJ.

LIN, Y.; PENG, X.; CAI, Y.; DIG, D.; ZHENG, D.; ZHAO, W. Interactive and guided architectural refactoring with search-based recommendation. In: **PROCEEDINGS OF THE 2016 24TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING**. [*S. l.*: *s. n.*], 2016. p. 535–546.

MEALY, E.; CARRINGTON, D.; STROOPER, P.; WYETH, P. Improving usability of software refactoring tools. In: **2007 AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE2007 AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE (ASWEC'07)**. [*S. l.*: *s. n.*], 2007. p. 307–318.

MENS, T.; TOURWÉ, T. A survey of software refactoring. **IEEE Transactions on software engineering**, IEEE, v. 30, n. 2, p. 126–139, 2004.

NUCCI, D. D.; PALOMBA, F.; TAMBURRI, D. A.; SEREBRENIK, A.; LUCIA, A. D. Detecting code smells using machine learning techniques: Are we there yet? In: **2018 IEEE 25TH INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION AND REENGINEERING (SANER)**. [*S. l.*: *s. n.*], 2018. p. 612–621.

O'KEEFFE, M.; CINNEIDE, M. Search-based software maintenance. In: **CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING (CSMR'06)**. [*S. l.*: *s. n.*], 2006. p. 10 pp.–260.

PALOMBA, F.; BAVOTA, G.; PENTA, M. D.; OLIVETO, R.; POSHYVANYK, D.; LUCIA, A. D. Mining version histories for detecting code smells. **IEEE Transactions on Software Engineering**, v. 41, n. 5, p. 462–489, 2015.

PALOMBA, F.; BAVOTA, G.; PENTA, M. D.; OLIVETO, R.; POSHYVANYK, D.; LUCIA, A. D. Mining version histories for detecting code smells. **IEEE Transactions on Software Engineering**, v. 41, n. 5, p. 462–489, 2015.

PRICE, B. A.; BAECKER, R. M.; SMALL, I. S. A principled taxonomy of software visualization. **J. Vis. Lang. Comput.**, v. 4, p. 211–266, 1993.

REFACTORING: a program restructuring aid in designing object-oriented application frameworks. Tese (Doutorado).

SHATNAWI, R.; LI, W. An investigation of bad smells in object-oriented design. In: **2018 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION2018 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ITNG'06)**. [*S. l.*: *s. n.*], 2006. p. 161–165.

SILVA, D.; SILVA, J. P. da; SANTOS, G.; TERRA, R.; VALENTE, M. T. Refdiff 2.0: A multi-language refactoring detection tool. **IEEE Transactions on Software Engineering**, v. 47, n. 12, p. 2786–2802, 2021.

SPENCE, I. No humble pie: The origins and usage of a statistical chart. **Journal of Educational and Behavioral Statistics**, v. 30, n. 4, p. 353–368, 2005. Disponível em: https://doi.org/10.3102/10769986030004353.

STOREY, M.-A.; WONG, K.; MüLLER, H. How do program understanding tools affect how programmers understand programs? **Science of Computer Programming**, v. 36, n. 2, p. 183–207, 2000. ISSN 0167-6423. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167642399000362.

TSANTALIS, N.; KETKAR, A.; DIG, D. Refactoringminer 2.0. **IEEE Transactions on Software Engineering**, v. 48, n. 3, p. 930–950, 2022.

TSANTALIS, N.; KETKAR, A.; DIG, D. Refactoringminer 2.0. **IEEE Transactions on Software Engineering**, v. 48, n. 3, p. 930–950, 2022.

VASSALLO, C.; PALOMBA, F.; GALL, H. C. Continuous refactoring in ci: A preliminary study on the perceived advantages and barriers. In: **2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [*S. l.*: *s. n.*], 2018. p. 564–568.

VIDAL, S.; VAZQUEZ, H.; DIAZ-PACE, J. A.; MARCOS, C.; GARCIA, A.; OIZUMI, W. Jspirit: a flexible tool for the analysis of code smells. In: **2015 34TH INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY (SCCC)**. [*S. l.*: *s. n.*], 2015. p. 1–6.

ZHANG, M.; HALL, T.; BADDOO, N. Code bad smells: a review of current knowledge. **Journal of Software Maintenance and Evolution: research and practice**, Wiley Online Library, v. 23, n. 3, p. 179–202, 2011.

APÊNDICE A – FORMULÁRIO DE IDENTIFICAÇÃO E AVALIAÇÃO DA FERRAMENTA

Questão 1. Qual seu nome?

Questão 2. Qual o seu tempo de experiência em desenvolvimento? (em anos)

(a) 1 ano

(b) 2 anos

(c) 3 anos

(d) 4 anos

(e) 5 anos

Questão 3. Qual o seu curso?

(a) Engenharia de Software

(b) Ciência da computação

(c) Engenharia da computação

(d) Redes de computadores

(e) Design Digital

(f) Sistemas de Informação

(g) Outros

Questão 4. Para as sentenças sobre utilidade percebida, assinale

– Utilizar a ferramenta proposta me ajudará a ter mais controle sobre meu código

  (a) Discordo Totalmente

  (b) Discordo

  (c) Neutro

  (d) Concordo

  (e) Concordo Totalmente

– Utilizar a ferramenta proposta reduzirá o tempo gasto com refatorações menos importantes

  (a) Discordo Totalmente

  (b) Discordo

  (c) Neutro

  (d) Concordo

  (e) Concordo Totalmente

– Utilizar a ferramenta proposta ajudará a melhorar a qualidade do meu trabalho

   (a) Discordo Totalmente
   (b) Discordo
   (c) Neutro
   (d) Concordo
   (e) Concordo Totalmente

– Utilizar a ferramenta proposta aumentará a minha produtividade
   (a) Discordo Totalmente
   (b) Discordo
   (c) Neutro
   (d) Concordo
   (e) Concordo Totalmente

– Utilizar a ferramenta proposta me fornecerá mais conhecimentos sobre refatorações
   (a) Discordo Totalmente
   (b) Discordo
   (c) Neutro
   (d) Concordo
   (e) Concordo Totalmente

– Utilizar a ferramenta proposta permitirá com que eu ganhe tempo no trabalho/projetos
   (a) Discordo Totalmente
   (b) Discordo
   (c) Neutro
   (d) Concordo
   (e) Concordo Totalmente

– As visualizações trazidas pela ferramenta proposta são úteis no dia-a-dia do meu trabalho
   (a) Discordo Totalmente
   (b) Discordo
   (c) Neutro
   (d) Concordo
   (e) Concordo Totalmente

– Verificar as visualizações trazidas pela ferramenta proposta me permite comparar meu desempenho com outros desenvolvedores
   (a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

– A visualização trazida pela plataforma sobre o número de refatorações executadas ao decorrer do tempo é um diferencial em relação as outras ferramentas existentes.

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

– No geral, achei a ferramenta proposta fácil de se utilizar

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

Questão 5. Para as sentenças sobre percepção de facilidade de uso, assinale

– Minha interação com a ferramenta proposta é fácil de se compreender

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

– A ferramenta proposta mostra informações importantes sobre refatoração

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

– Ao utilizar a ferramenta consigo informações para implementar outros tipos de refatoração

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

– No geral, acho a ferramenta proposta, útil para o meu trabalho/projetos

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

Questão 6. Sobre as sentenças sobre autoprevisão de uso futuro, assinale

– Caso disponível, utilizaria a ferramenta proposta em meu trabalho/projetos

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente

– Acho a visão trazida pela ferramenta sobre refatorações mais interessante do que as outras ferramentas/plugins trazem

(a) Discordo Totalmente

(b) Discordo

(c) Neutro

(d) Concordo

(e) Concordo Totalmente