# Xtract: Um Plug-in Baseado em IA para Recomendações Explicáveis da Refatoração Extract Method

Guisella Angulo Armijo
Departamento de Computação, UFSCar
São Carlos, Brasil
angulogc@gmail.com

Igor Ueno
Departamento de Computação, UFSCar
São Carlos, Brasil
igorueno@estudante.ufscar.br

Valter Vieira de Camargo
Departamento de Computação, UFSCar
São Carlos, Brasil
valtervcamargo@ufscar.br

## Resumo

A evolução dos sistemas de software aumenta gradativamente sua complexidade, tornando a refatoração uma tarefa essencial para manter a qualidade do código. Isso demanda a aplicação contínua de refatorações. Entretanto, a identificação manual de oportunidades de refatoração é uma tarefa complexa, e as ferramentas automáticas disponíveis costumam gerar extensas listas de recomendações incompletas ou pouco compreensíveis, resultando em baixa aceitação por parte dos desenvolvedores. Neste contexto, este trabalho introduz **Xtract**, um plug-in de recomendação integrado ao IntelliJ IDEA, projetado para gerar recomendações de *Extract Method* orientadas pelo critério W3B (Which, Where, Why e Benefits). O **Xtract** emprega um pipeline modular que combina a extração de métricas estruturais e de fragmentos de código, a classificação por meio do modelo Random Forest, a análise semântica utilizando o modelo pré-treinado CodeBERT, um Explicador Consensual baseado em IA Explicável (XAI) e o uso da LLM GPT-4 para estruturar a Recomendação. Em termos de uso, a ferramenta processa um método Java selecionado pelo desenvolvedor e apresenta uma recomendação contendo o fragmento de código sugerido para extração, o score de extração calculado pelo CodeBERT, acompanhado de uma explicação baseada em *features* (ranking de métricas), além de uma explicação em linguagem natural. Embora a abordagem subjacente contemple também a dimensão Benefits, esta não é apresentada na interface da versão do plug-in descrita neste artigo. Dessa forma, o **Xtract** busca aumentar a confiança e a aceitação dos desenvolvedores em relação às ferramentas de recomendação de refatoração.
**Demo de vídeo:** https://doi.org/10.5281/zenodo.21459888

## Palavras-chave

recomendações de refatoração, aprendizado de máquina, IA explicável

## 1 Introdução

À medida que os sistemas de software evoluem, eles inevitavelmente acumulam complexidade e sofrem degradação de qualidade [25]. De acordo com a primeira lei de Lehman, os sistemas devem ser continuamente adaptados, ou tornam-se progressivamente menos satisfatórios [23]. Para mitigar essa degradação, a refatoração é adotada como uma prática disciplinada para combater a erosão do design e apoiar a manutenção.

A refatoração, popularizada por Fowler em 1999 [15], consiste em modificar um sistema de software sem alterar seu comportamento externo, enquanto melhora sua estrutura interna. Ela ajuda a prevenir a degradação de design, reduz a propensão a erros e melhora a legibilidade e a manutenibilidade do código. Apesar desses benefícios, identificar quando e como refatorar ainda é um desafio, particularmente em sistemas grandes e complexos. Tradicionalmente, esse processo depende fortemente da experiência e intuição dos desenvolvedores, frequentemente apoiado por code smells como indicadores de possíveis problemas [15].

*Extract Method* é uma das refatorações amplamente adotadas no desenvolvimento de software, pois decompõe métodos grandes e complexos, abordando problemas como *Duplicated code*, *Long Method* e *Feature Envy* [4, 8, 20, 30, 39]. Em linhas gerais, essa refatoração envolve três etapas principais: i) identificar o método candidato, isto é, um método que apresenta algum sinal de problemas de manutenção; ii) identificar o trecho candidato a ser extraído do método original; iii) criar um novo método com o trecho extraído e substituir o trecho original por uma chamado ao novo método.

Existem diversas ferramentas automatizadas que oferecem a identificação de oportunidades para a aplicação da refatoração *Extract Method*, visto que a detecção manual é propensa a erros e é uma tarefa muito demorada. Essas ferramentas utilizam diversas técnicas, tais como: heurísticas e padrões estruturais [14, 35]; abordagens baseadas em busca, utilizando estratégias de otimização para equilibrar múltiplas métricas de qualidade [2, 28]; e mais recentemente, técnicas de aprendizado de máquina, explorando métricas de código-fonte e dados históricos para identificar oportunidades de refatoração [20, 22, 24, 40].

Apesar desses avanços, as recomendações frequentemente são incompletas; isto é, podem indicar a presença de um problema, mas falham em especificar qual refatoração aplicar, onde no código ela deve ser realizada, por que é recomendada ou quais benefícios traria. Como resultado, os desenvolvedores ficam com informações parciais, limitando sua capacidade de compreender, confiar e agir efetivamente sobre a recomendação. Além disso, um problema enfrentado pelas ferramentas que recomendam **Extract Method** é o fato de estarem atreladas a *code smells*, tornando a recomendação ainda mais restritiva.

Este trabalho apresenta um plug-in baseado em IA, denominado **Xtract**, projetado para gerar recomendações de refatoração *Extract Method*. O **Xtract**, implementado como um plug-in para IntelliJ, orientado pelo critério **W3B** (Which, Where, Why e Benefits) [7] para definir a completude da recomendação em termos de: qual refatoração aplicar, onde no código ela deve ser realizada, por que ela é sugerida e quais benefícios ela proporciona. Seu núcleo integra técnicas de aprendizado de máquina a mecanismos de explicabilidade baseados em *features* e em linguagem natural, utilizando o modelo GPT-4, com o objetivo de produzir recomendações que vão além da simples detecção de oportunidades de refatoração, apoiando explicitamente a compreensão e a tomada de decisão dos desenvolvedores.

A abordagem subjacente ao Xtract contempla as quatro dimensões do critério W3B. Entretanto, por questões de espaço e de projeto da interface do usuário, a versão do plug-in apresentada neste artigo exibe apenas as dimensões *Which*, *Where* e *Why*.

## 2 Fundamentação Teórica

### 2.1 Refatoração de Software e Code Smell

Refatoração é o processo de modificar um sistema de software sem alterar seu comportamento externo, visando melhorar sua estrutura interna [15, 29]. As técnicas de refatoração podem ser manuais, semi-automáticas ou automatizadas [3, 25]. Entretanto, a identificação manual de oportunidades de refatoração é uma tarefa trabalhosa e dependente da experiência do desenvolvedor [33].

Nesse contexto, *code smells* têm sido amplamente utilizados como indicadores de possíveis problemas de design e oportunidades de refatoração. Segundo Fowler, um *code smell* representa um indício superficial de problemas mais profundos no software [15]. Apesar de sua relevância, os *code smells* representam apenas parte das motivações para refatoração [10]. Estudos mostram que refatorações como *Extract Method* também podem ser motivadas por fatores como melhoria de legibilidade, modularização, remoção de duplicação, correção de bugs e otimização de desempenho [19].

### 2.2 Machine Learning

Aprendizado de Máquina permite que sistemas aprendam padrões a partir de dados e melhorem seu desempenho ao longo do tempo [26]. Entre os principais paradigmas estão o aprendizado supervisionado, utilizado em tarefas de classificação e regressão, e o aprendizado não supervisionado, voltado para a descoberta de padrões e agrupamentos [21, 27]. Redes neurais artificiais e técnicas de *Deep Learning* ampliaram a capacidade dos modelos em capturar relações complexas e não lineares [34].

Mais recentemente, Modelos de Linguagem de Grande Porte (LLMs) e Modelos de Linguagem Pré-treinados (PLMs) passaram a desempenhar papel importante em tarefas de engenharia de software. Baseados na arquitetura Transformer [38], esses modelos aprendem representações semânticas e sintáticas a partir de grandes volumes de dados. Modelos como BERT [12] e CodeBERT [13] demonstraram resultados promissores em tarefas relacionadas a código-fonte, incluindo classificação, sumarização, busca de código e recomendação de refatorações. Em especial, o CodeBERT combina código-fonte e linguagem natural durante o pré-treinamento, permitindo análises mais contextualizadas e eficazes para aplicações em engenharia de software.

### 2.3 Explainable Machine Learning

Com o aumento da complexidade dos modelos de aprendizado de máquina, surge o desafio de equilibrar desempenho e interpretabilidade [18]. Nesse contexto, a Inteligência Artificial Explicável (*Explainable Artificial Intelligence*-XAI) busca tornar as predições dos modelos mais compreensíveis e confiáveis para os usuários [17].

As técnicas de XAI fornecem explicações sobre o comportamento dos modelos, auxiliando na transparência, confiança e interpretabilidade das decisões automatizadas [16]. Essas abordagens podem envolver modelos inerentemente interpretáveis ou métodos *post-hoc*, aplicados após o treinamento do modelo [5] como LIME, SHAP e Anchors, que explicam predições individuais por meio da relevância de atributos ou regras locais. Além de melhorar a interpretabilidade, XAI também contribui para depuração, validação e identificação de vieses em sistemas de ML, promovendo maior confiabilidade e adoção responsável dessas tecnologias [31].

## 3 Critério W3B

O critério **W3B** (**W**hich, **W**here, **W**hy e **B**enefits) surge como resultado de uma revisão sistemática conduzida com o objetivo de compreender o uso de aprendizado de máquina em recomendações de refatoração [7]. O critério visa avaliar a completude de uma recomendação, considerando como completa aquela que fornece aos desenvolvedores informações suficientes para apoiar a tomada de decisão. Nesse contexto, uma recomendação é considerada completa quando informa: (i) qual refatoração deve ser aplicada, (ii) onde ela deve ser realizada no código-fonte, (iii) por que ela foi recomendada e (iv) quais benefícios sua aplicação pode trazer. A seguir detalhamos cada item do criterio.

**WHICH:** define qual refatoração deve ser aplicada, especificando explicitamente o tipo de refatoração, como *Extract Method*, *Move Method* ou *Rename Method*.

**WHERE:** identifica onde a refatoração deve ser aplicada, indicando os elementos do código afetados, como classes, métodos, atributos ou trechos específicos de código. Dependendo do tipo de refatoração, o nível de granularidade pode variar. Por exemplo, em *Extract Method*, é necessário identificar tanto o método original quanto o trecho exato de código que deve ser extraído.

**WHY:** fornece uma explicação sobre por que a refatoração foi recomendada, apresentando os fatores que motivaram a decisão do modelo. Esse aspecto é importante para aumentar a transparência e a confiança nas recomendações, podendo ser apoiado por técnicas de Inteligência Artificial Explicável (XAI) [11].

**BENEFITS:** descreve os benefícios esperados com a aplicação da refatoração, como melhorias em legibilidade, manutenção, modularidade ou qualidade do software.

## 4 Xtract Plug-in

### 4.1 Caso de Uso

A Figura 1 apresenta o principal caso de uso da ferramenta proposta. O desenvolvedor interage diretamente com o plug-in integrado à IDE IntelliJ para identificar oportunidades de refatoração em métodos do projeto local.

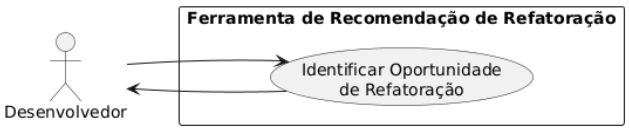


**Figure 1: Caso de Uso**

**Nome do Caso de Uso:** Identificar Oportunidade de Refatoração.
**Objetivo:** Auxiliar o desenvolvedor na identificação de oportunidades de refatoração do tipo *Extract Method*, fornecendo recomendações acompanhadas de explicações baseadas no critério W3B diretamente na IDE.

- **Ator Principal:** Desenvolvedor
- **Pré-condição:** O projeto Java deve estar aberto na IDE e conter métodos implementados.
- **Pós-condição:** A ferramenta apresenta uma recomendação de refatoração acompanhada de explicações que justificam a decisão produzida pela abordagem.

## 4.2 Fluxo sob o Ponto de Vista do Usuário

A utilização da ferramenta ocorre diretamente na IDE IntelliJ, permitindo que o desenvolvedor realize a identificação de oportunidades de refatoração de maneira integrada ao ambiente de desenvolvimento. Figura 2 mostra o fluxo de interação, o qual inicia quando o usuário seleciona o método que deseja analisar no editor de código da IDE. Em seguida, por meio do menu contextual acionado com o clique do botão direito, o usuário executa a opção *"Analysis for Refactoring Opportunities"* e seleciona, no submenu, a opção *"Extract Method"* disponibilizada pelo plug-in.

Após a solicitação, a ferramenta realiza automaticamente o processamento interno do método selecionado. Durante essa etapa, métricas estruturais são extraídas, fragmentos candidatos são identificados e os modelos de aprendizado de máquina analisam o método para verificar sua elegibilidade para refatoração e identificar os trechos mais adequados para extração.

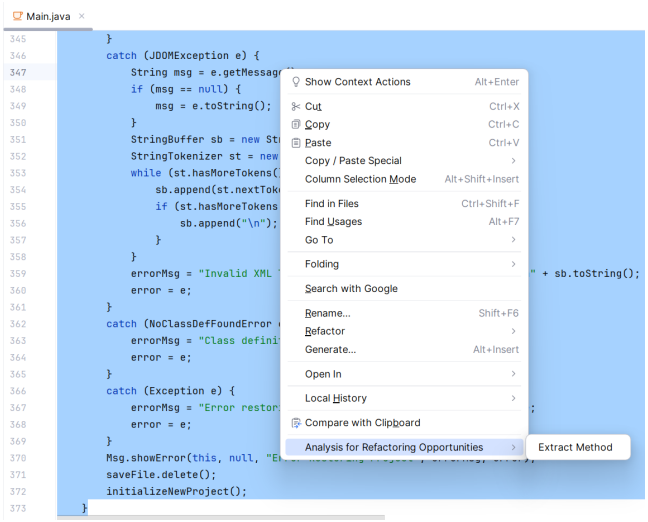


**Figure 2: IU- Análise de oportunidades de refatoração**

Concluído o processamento, a ferramenta exibe, em uma janela *popup*, o resultado da análise realizada sobre o método, indicando se há necessidade de refatoração e apresentando a probabilidade prevista pelo modelo Random Forest. As Figura 3 ilustra o resultado obtido.

Após a exibição do *popup*, o plug-in destaca o trecho de código recomendado para extração e apresenta os detalhes da recomendação em uma janela lateral expansível. A Figura 4 ilustra uma recomendação gerada para a aplicação da refatoração *Extract Method*.

A recomendação exibida ao desenvolvedor é baseada no critério *W3B* e reúne informações que auxiliam na compreensão do fragmento sugerido para extração, bem como do escore calculado pelo

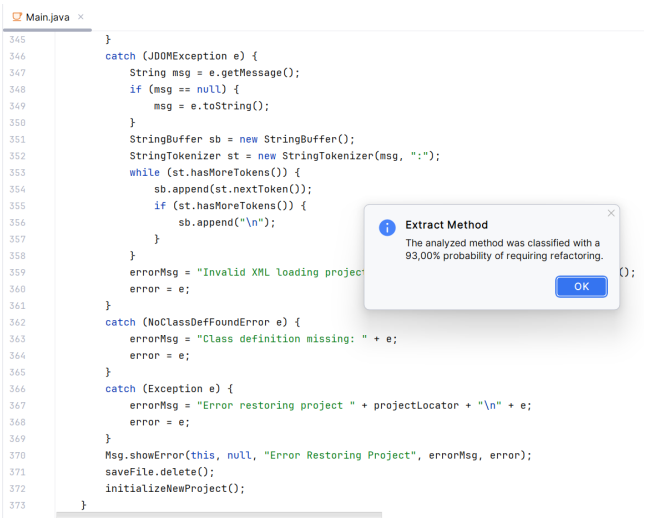


**Figure 3: Xtract - Popup - Resultado da análise do Método**

modelo CodeBERT. Dessa forma, o desenvolvedor pode entender não apenas qual fragmento foi selecionado, mas também os fatores que influenciaram a decisão do modelo.

A interface de recomendação é composta pelos seguintes elementos:

(i) **Fragmento recomendado**: o trecho de código sugerido para extração é destacado diretamente no editor. Esse destaque pode ser ativado ou desativado por meio do botão *Hide Recommended Fragment*;
(ii) **Escore do fragmento**: a janela lateral apresenta o escore atribuído ao fragmento. O fragmento apresentado é aquele com o maior escore, isto é, é a melhor opção para ser extraído como parte da refatoração *Extract Method*;
(iii) ***Key Features Influencing the Prediction***: a interface apresenta um ranking das características mais relevantes que influenciaram a predição do modelo, permitindo ao desenvolvedor compreender os principais fatores considerados durante a análise;
(iv) ***Model Decision Explanation***: a recomendação inclui uma explicação em linguagem natural sobre o raciocínio adotado pelo modelo para gerar a recomendação. Essa explicação é construída a partir das informações extraídas durante a análise, buscando tornar a decisão do modelo mais compreensível e interpretável para o desenvolvedor.

## 5 Xtract Plug-in Processo Interno

Figura 5 apresenta o plug-in, que está estruturado em duas fases de análise compostas por cinco etapas principais. Primeiramente, dado como entrada o código do método alvo, o componente *Fragment and Metrics Extractor* (FME) obtém métricas de código-fonte e segmenta o método em fragmentos. Em seguida, essas métricas extraídas são utilizadas como entrada para um modelo Random Forest, responsável por prever se o método necessita de refatoração *Extract Method*.

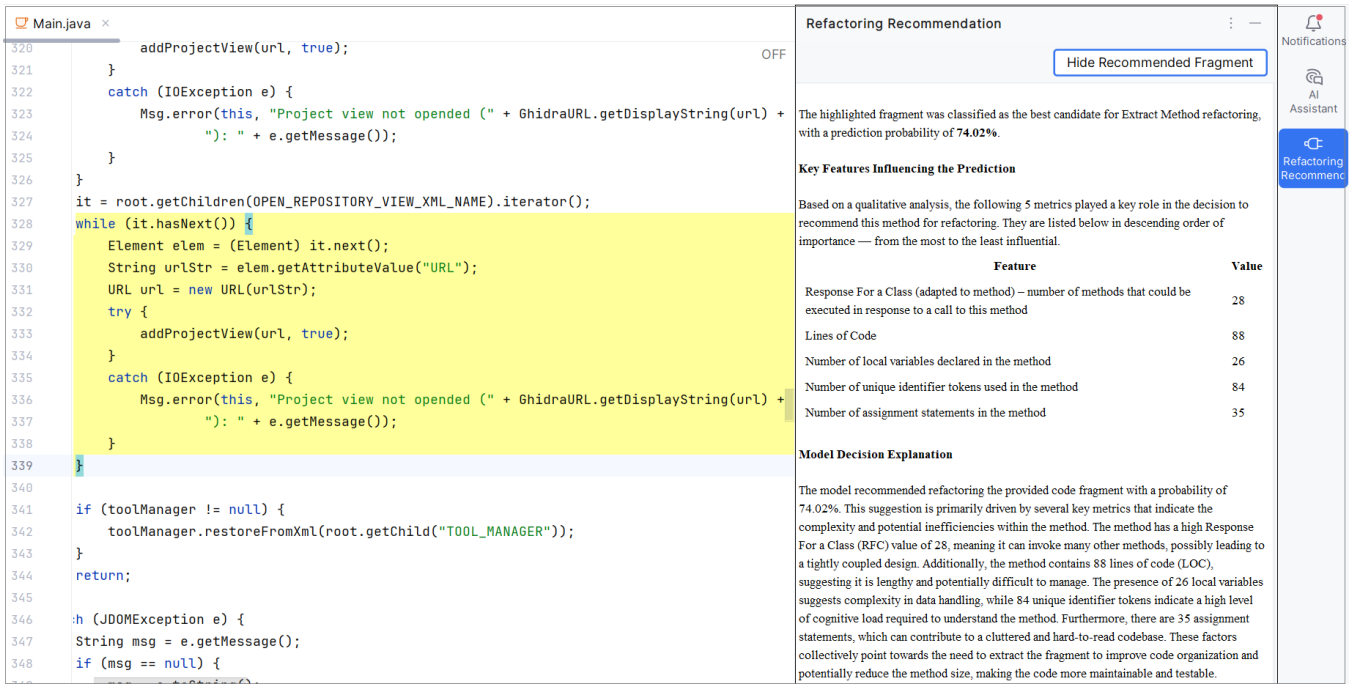


**Figure 4: Xtract - Pipeline**

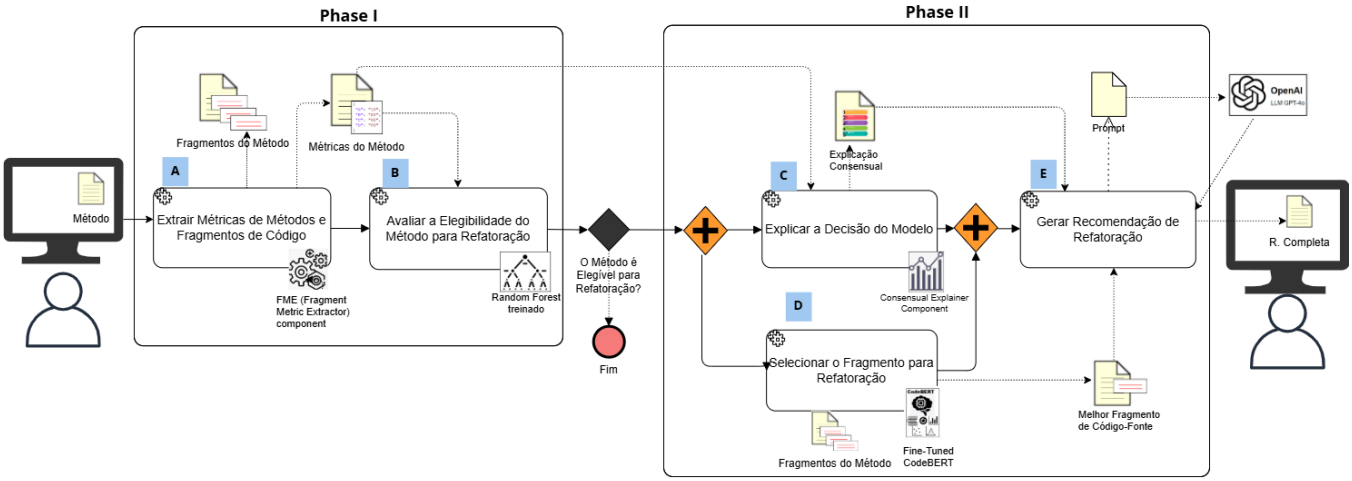


**Figure 5: Xtract - Pipeline**

Caso a previsão seja positiva, um componente de explicabilidade gera justificativas baseadas nas características mais relevantes para a decisão do modelo. Paralelamente, os fragmentos extraídos são analisados por um modelo CodeBERT treinado (*fine-tuned*), responsável por identificar o fragmento de código mais adequado para extração. Por fim, a recomendação completa é gerada combinando o fragmento selecionado, as explicações produzidas e os elementos do critério W3B. A divisão em duas camadas busca reduzir o custo computacional da análise. A primeira camada avalia apenas características estruturais do método para decidir se a refatoração deve ser realizada. Já a segunda camada executa uma análise mais detalhada dos fragmentos internos do método, permitindo identificar o trecho mais apropriado para extração e gerar uma recomendação mais precisa e interpretável.

## 5.1 Fase I

Esta fase corresponde à primeira camada de análise da abordagem e é composta pelas Etapas **A** e **B**. Ela concentra-se exclusivamente na análise em nível de método, considerando apenas métricas estruturais e comportamentais, com o objetivo de determinar se o método deve passar pela refatoração *Extract Method*.

**Etapa A: Extração de Métricas e Fragmentos de Código**.

Esta etapa é responsável por preparar todas as informações estruturais necessárias para as fases seguintes da abordagem. Como entrada, o componente **FME (Fragment and Metrics Extractor)** recebe o código do método selecionado.

O FME produz dois artefatos: (i) um arquivo CSV contendo 20 métricas estruturais e comportamentais do método e (ii) um arquivo JSON contendo fragmentos candidatos extraídos do método. As métricas são utilizadas posteriormente pela Etapa **B** para avaliar se o método é candidato à refatoração, enquanto os fragmentos extraídos são armazenados para análise posterior na Etapa **D**.

**Etapa B: Avaliar a Elegibilidade do Método para Refatoração**.

Nesta etapa, um modelo Random Forest (RF) treinado determina se o método analisado é candidato à refatoração *Extract Method*. O modelo Random Forest foi escolhido devido à sua robustez, interpretabilidade e eficiência no tratamento de dados de alta dimensionalidade, características amplamente exploradas em estudos de recomendação de refatoração [8, 9, 30, 37, 40]. O treinamento foi feito utilizando um conjunto de dados (150 mil amostras) contendo métricas estruturais e comportamentais extraídas de métodos que passaram por esse tipo de refatoração.

O modelo RF atua como um classificador binário que recebe como entrada o arquivo CSV contendo métricas estruturais e comportamentais do método analisado. Um limiar de probabilidade de 0.5 é utilizado para determinar se o método é candidato à refatoração: métodos classificados positivamente seguem para as próximas etapas da abordagem, enquanto os demais encerram o processo. Além da classificação, o mesmo modelo é reutilizado na Etapa **C**, em conjunto com técnicas de explicabilidade, para justificar as predições realizadas. A identificação do método candidato também contribui diretamente para o elemento *Where* do critério *W3B*, utilizado na geração de recomendações completas de refatoração.

## 5.2 Fase II

Esta fase realiza uma análise mais detalhada dos métodos identificados como candidatos à refatoração na Fase I. Essa etapa tem como objetivo gerar recomendações completas, selecionando o fragmento mais adequado para extração e fornecendo explicações sobre a decisão do modelo.

**Etapa C: Explicar a Decisão do Modelo**.

Nesta etapa, o foco está na interpretabilidade da recomendação. Após a predição realizada pelo modelo Random Forest, as métricas do método analisado são enviadas para o componente *Consensual Explainer*, responsável por gerar explicações sobre os fatores que influenciaram a decisão do modelo.

O componente utiliza uma estratégia baseada em consenso entre dois técnicas de explicabilidade amplamente utilizadas: SHAP e LIME, o componente é customizável e pode incluir outros modelos de explicabilidade. Como essas técnicas podem gerar interpretações diferentes para a mesma predição, o componente combina os resultados para identificar padrões de concordância e produzir explicações mais estáveis e confiáveis.

Como saída, o sistema gera uma lista ranqueada com as cinco características mais relevantes para a decisão do modelo. Essa etapa atende diretamente ao elemento *Why* do critério W3B, fornecendo justificativas para a recomendação e aumentando a transparência e a confiança do usuário.

**Etapa D: Seleção do Fragmento para Refatoração**.

Nesta etapa, os fragmentos candidatos extraídos na Etapa **A** são avaliados por um modelo CodeBERT ajustado (*fine-tuned*). Esse modelo atribui um score de afinidade a cada fragmento, indicando quão adequado ele é para ser extraído com base nas relações semânticas entre o fragmento e o contexto do método original.

O modelo foi treinado utilizando um conjunto de dados especializado composto por pares método-fragmento extraídos de refatorações reais obtidas inicialmente do dataset de Aniche [6] e identificadas com o RefactoringMiner [36]. Diferentemente de abordagens baseadas apenas em métricas, o conjunto de dados considera relações semânticas do código-fonte, permitindo identificar fragmentos mais representativos para extração.

Como resultado, a etapa seleciona o fragmento considerado mais adequado para extração. Esse fragmento, combinado ao método compõe o elemento *Where* do critério W3B.

**Etapa E: Gerar Recomendação de Refatoração**.

Na etapa final, o objetivo é sintetizar todos os artefatos gerados anteriormente em uma recomendação de refatoração completa e estruturada. Para isso, um prompt cuidadosamente elaborado é enviado ao *GPT-4* por meio da API da OpenAI. O prompt incorpora os artefatos gerados ao longo do pipeline, incluindo o fragmento selecionado e sua pontuação de afinidade associada (Etapa **D**), bem como as 5 características (features) mais bem classificadas pelo *Consensual Explainer* (Etapa **C**).

Para a formulação do prompt, adotou-se uma estratégia de *Zero-Shot prompting*, na qual a tarefa é especificada diretamente, sem o fornecimento de exemplos prévios [1]. Como resultado, esta etapa consolida todos os elementos do pipeline em uma saída final que atende os elementos *Which*, *Where* e *Why* do critério W3B, produzindo uma recomendação de refatoração explicável e acionável.

## 5.3 Arquitetura

Figura 6 mostra a arquitetura estruturada em 3 camadas, como detalhada a continuação.

**(I) Camada de Interface do Usuário**. Esta camada é responsável pela interação entre o desenvolvedor e a ferramenta de recomendação de refatoração dentro da IDE IntelliJ. Essa camada permite a seleção do método a ser analisado e a visualização das recomendações geradas pela abordagem. O componente *Method Selection Manager* captura o método selecionado pelo usuário no projeto local e encaminha a solicitação para o motor de recomendação. Após o processamento, o componente *Recommendation Viewer* apresenta as recomendações de refatoração e as explicações produzidas pela ferramenta diretamente na interface da IDE.

**(II) Motor de Recomendação**. Esta camada concentra a lógica principal da abordagem e é responsável pela análise dos métodos

selecionados, identificação de candidatos à refatoração e geração de explicações consensuais. Inicialmente, o componente *Fragment Metric Extractor (FME)* extrai métricas estruturais e comportamentais do método, além de identificar fragmentos candidatos à extração. Na sequência, o *Random Forest Classifier* verifica se o método é elegível para refatoração, enquanto o *Fine-tuned CodeBERT Model* realiza a análise semântica dos fragmentos candidatos. Por fim, o componente *Consensus Explainer* gera uma explicação consensual que subsidia a recomendação final apresentada ao usuário.

**(III) Serviço de Explicação baseado em LLM**. A camada *LLM-based Explanation Service* é responsável por enriquecer as explicações geradas pela abordagem utilizando modelos de linguagem de grande porte (LLMs). O componente *GPT Explanation Generator* recebe as informações produzidas pelos outros componentes e gera explicações em linguagem natural sobre a recomendação de refatoração, auxiliando o desenvolvedor na compreensão da recomendação fornecida pela ferramenta.

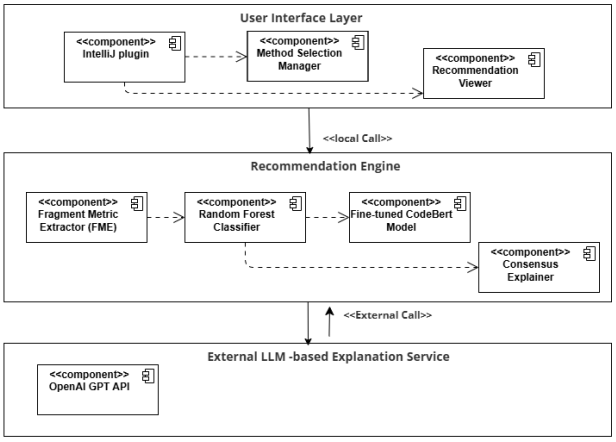


**Figure 6: Xtract - Arquitetura**

## 6 Trabalhos Relacionados

Diversas abordagens propõem ferramentas e plug-ins para recomendação automática de refatorações, principalmente voltadas para *Extract Method* e *Move Method*.

O **AntiCopyPaster** [4] é um plug-in para IntelliJ IDEA que recomenda a refatoração *Extract Method* para evitar duplicação de código. A abordagem detecta fragmentos duplicados, extrai métricas do código e utiliza uma rede neural convolucional (CNN) para decidir se a refatoração deve ser aplicada. Como saída, a ferramenta apresenta notificações diretamente na IDE. Entre as limitações estão possíveis sobreposições entre exemplos positivos e negativos, o que pode afetar a qualidade do modelo de aprendizado de máquina.

O **DOIMR** [32] é uma abordagem interativa que permite aos desenvolvedores explorar métricas de qualidade e localização do código para gerar recomendações de refatoração personalizadas. A abordagem combina busca multiobjetivo, agrupamento de soluções e feedback iterativo do usuário para melhorar atributos de qualidade. Como limitação, a ferramenta exige usuários experientes, e pode tornar o processo cansativo devido à interação contínua.

O **RMove** [9] é uma ferramenta para recomendação da refatoração *Move Method* para a remoção do *code smell Feature Envy*. A abordagem utiliza grafos de dependência de métodos e árvores sintáticas abstratas (AST) para representar características estruturais e semânticas do código, treinando classificadores para sugerir o *Move Method*. Sua principal limitação está na dependência de bases de dados de projetos open source, que podem conter ruídos.

## 7 Conclusão

Este artigo apresentou a **Xtract**, um plug-in integrado à IDE IntelliJ desenvolvido para auxiliar desenvolvedores na identificação de oportunidades de refatoração do tipo *Extract Method*, por meio de recomendações explicáveis. Diferentemente das abordagens existentes, que frequentemente fornecem recomendações parciais, a ferramenta proposta é orientada pelo critério W3B (*Which*, *Where*, *Why* e *Benefits*) para gerar recomendações mais abrangentes e compreensíveis diretamente no ambiente de desenvolvimento. Entretanto, devido às limitações de espaço do artigo, esta versão priorizou os elementos do critério W3B relacionados à identificação do trecho candidato (*Which*), sua localização (*Where*) e à explicação da recomendação (*Why*).

A abordagem combina métricas estruturais, análise semântica e técnicas de Inteligência Artificial Explicável para melhorar tanto a qualidade das recomendações quanto a confiança do desenvolvedor nas decisões produzidas pela ferramenta. A arquitetura proposta integra o componente Fragment Metric Extractor (FME), um classificador Random Forest, um modelo Fine-tuned CodeBERT e o módulo Consensual Explainer para gerar recomendações fundamentadas em práticas reais de refatoração. Além disso, a integração de um LLM possibilita a geração de explicações contextualizadas em linguagem natural, apresentadas diretamente na IDE.

Sob a perspectiva do usuário, a ferramenta foi projetada para minimizar interrupções no fluxo de desenvolvimento, integrando o processo de recomendação diretamente à interface da IDE IntelliJ. Dessa forma, o desenvolvedor pode analisar métodos diretamente no editor de código-fonte e receber recomendações em tempo real por meio de uma janela lateral dedicada.

De forma geral, este trabalho contribui para reduzir a distância entre técnicas automatizadas de recomendação de refatoração e sua adoção prática em ambientes reais de desenvolvimento de software.

## Disponibilidade de Artefatos

O código-fonte do **Xtract** está licenciado sob a licença MIT e disponível no GitHub em https://github.com/Advanse-Lab/xtract-plugin-intellij.

## Agradecimentos

Este estudo foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código Financeiro 001. Este trabalho recebeu apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), por meio do processo nº 2026/06067-0, vinculado ao processo nº 2025/07160-0.

## Referências


[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Vahid Alizadeh, Houcem Fehri, and Marouane Kessentini. 2019. Less is More: From Multi-objective to Mono-objective Refactoring via Developer's Knowledge Extraction. In *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 181–192.

[3] Vahid Alizadeh, Mohamed Amine Ouali, Marouane Kessentini, and Meriem Chater. 2019. RefBot: Intelligent software refactoring bot. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 823–834.

[4] Eman Abdullah AlOmar, Anton Ivanov, Zarina Kurbatova, Yaroslav Golubev, Mohamed Wiem Mkaouer, Ali Ouni, Timofey Bryksin, Le Nguyen, Amit Kini, and Aditya Thakur. 2023. Just-in-time code duplicates extraction. *Information and Software Technology* 158 (2023).

[5] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. 2021. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 11, 5 (2021), e1424.

[6] Mauricio Aniche, Erick Maziero, Rafael Durelli, and Vinicius HS Durelli. 2020. The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Transactions on Software Engineering* 48, 4 (2020), 1432–1450.

[7] Guisella Armijo, Daniel Santibañez, Rafael Durelli, and Valter Camargo. 2024. On the Employment of Machine Learning for Recommending Refactorings: A Systematic Literature Review. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 334–345. doi:10.5753/sbes.2024.3436

[8] Di Cui, Qiangqiang Wang, Siqi Wang, Jianlei Chi, Jianan Li, Lu Wang, and Qingshan Li. 2023. REMS: Recommending Extract Method Refactoring Opportunities via Multi-view Representation of Code Property Graph. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE.

[9] Di Cui, Siqi Wang, Yong Luo, Xingyu Li, Jie Dai, Lu Wang, and Qingshan Li. 2022. RMove: Recommending Move Method Refactoring Opportunities using Structural and Semantic Representations of Code. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 281–292.

[10] Warteruzannan Soyer Cunha, Guisella Angulo Armijo, and Valter Vieira de Camargo. 2020. InSet: A Tool to Identify Architecture Smells Using Machine Learning. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 760–765. doi:10.1145/3422392.3422507

[11] Arun Das and Paul Rad. 2020. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371* (2020).

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 4171–4186.

[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[14] Marios Fokaefs, Nikolaos Tsantalis, and Alexander Chatzigeorgiou. 2007. Jdeodorant: Identification and removal of feature envy bad smells. In *2007 ieee international conference on software maintenance*. IEEE, 519–520.

[15] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Boston, MA, USA.

[16] Prashant Gohel, Priyanka Singh, and Manoranjan Mohanty. 2021. Explainable AI: current status and future directions. *arXiv preprint arXiv:2107.07045* (2021).

[17] David Gunning. 2017. Explainable artificial intelligence (xai). *Defense advanced research projects agency (DARPA), nd Web* 2, 2 (2017), 1.

[18] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. 2019. XAI—Explainable artificial intelligence. *Science Robotics* 4, 37 (2019), eaay7120.

[19] Jalisson Henrique, Marcos Dósea, and Cláudio Sant'Anna. 2021. Minerando Motivações para Aplicação de Extract Method: Um Estudo Preliminar. In *Anais do IX Workshop de Visualização, Evolução e Manutenção de Software*. SBC, 21–25.

[20] Ayaka Imazato, Yoshiki Higo, Keisuke Hotta, and Shinji Kusumoto. 2017. Finding extract method refactoring opportunities by analyzing development history. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 190–195.

[21] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. 2007. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering* 160 (2007), 3–24.

[22] Zarina Kurbatova, Ivan Veselov, Yaroslav Golubev, and Timofey Bryksin. 2020. Recommendation of Move Method Refactoring Using Path-Based Representation of Code. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 315–322.

[23] Meir M Lehman. 1980. Programs, life cycles, and laws of software evolution. *Proc. IEEE* 68, 9 (1980), 1060–1076.

[24] Hui Liu, Zhifeng Xu, and Yanzhen Zou. 2018. Deep learning based feature envy detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 385–396.

[25] Tom Mens and Tom Tourwé. 2004. A survey of software refactoring. *IEEE Transactions on software engineering* 30, 2 (2004), 126–139.

[26] R. Michalski, J. Carbonell, and T. Mitchell. 1985. Machine Learning: Artificial Intelligence Approach.

[27] Kevin P. Murphy. 2023. *Probabilistic Machine Learning: Advanced Topics*. The MIT Press, Cambridge, MA. 1360 pages.

[28] Ally S Nyamawe, Hui Liu, Nan Niu, Qasim Umer, and Zhendong Niu. 2020. Feature requests-based recommendation of software refactorings. *Empirical Software Engineering* 25, 5 (2020), 4315–4347.

[29] William F Opdyke. 1992. Refactoring object-oriented frameworks. (1992).

[30] Indranil Palit, Gautam Shetty, Hera Arif, and Tushar Sharma. 2023. Automatic refactoring candidate identification leveraging effective code representation. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 369–374.

[31] P Jonathon Phillips, Carina A Hahn, Peter C Fontana, David A Broniatowski, and Mark A Przybocki. 2020. Four principles of explainable artificial intelligence. *Gaithersburg, Maryland* (2020).

[32] Soumaya Rebai, Vahid Alizadeh, Marouane Kessentini, Houcem Fehri, and Rick Kazman. 2020. Enabling decision and objective space exploration for interactive multi-objective refactoring. *IEEE Transactions on Software Engineering* (2020).

[33] Cleiton Silva, Amanda Santana, Eduardo Figueiredo, and Mariza AS Bigonha. [n. d.]. Revisiting the Bad Smell and Refactoring Relationship: A Systematic Literature Review. ([n. d.]).

[34] Haykin Symon. 1999. Neural networks: a comprehensive foundation. *Prentice-Hall* (1999).

[35] Ricardo Terra, Marco Tulio Valente, Sergio Miranda, and Vitor Sales. 2018. JMove: A novel heuristic and tool to detect move method refactoring opportunities. *Journal of Systems and Software* 138 (2018), 19–36.

[36] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. 2022. RefactoringMiner 2.0. *IEEE Transactions on Software Engineering* 48, 3 (2022), 930–950. doi:10.1109/TSE.2020.3007722

[37] David van der Leij, Jasper Binda, Robbert van Dalen, Pieter Vallen, Yaping Luo, and Maurício Aniche. 2021. Data-driven extract method recommendations: a study at ING. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1337–1347.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[39] Sihan Xu, Chenkai Guo, Lei Liu, and Jing Xu. 2017. A log-linear probabilistic model for prioritizing extract method refactorings. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2503–2507.

[40] Ruru Yue, Zhe Gao, Na Meng, Yingfei Xiong, Xiaoyin Wang, and J David Morgenthaler. 2018. Automatic clone recommendation for refactoring based on the present and the past. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 115–126.