UNIVERSIDADE ESTADUAL DA PARAÍBA-UEPB
CAMPUS VII - GOVERNADOR ANTÔNIO MARIZ
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

HALAN CAIO PEREIRA DO NASCIMENTO

INVESTIGANDO O IMPACTO DA REFATORAÇÃO NA COMPREENSÃO DE CÓDIGO: UM ESTUDO EMPÍRICO COM ESTUDANTES DE COMPUTAÇÃO

PATOS-PB
2025

HALAN CAIO PEREIRA DO NASCIMENTO

INVESTIGANDO O IMPACTO DA REFATORAÇÃO NA COMPREENSÃO DE CÓDIGO: UM ESTUDO EMPÍRICO COM ESTUDANTES DE COMPUTAÇÃO

Trabalho de Conclusão de Curso ou apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharelado em Ciência da Computação

Área de concentração: Engenharia de Software

Orientador: Prof. Dr. José Aldo Silva da Costa

PATOS-PB
2025

N244i Nascimento, Halan Caio Pereira do.
Investigando o impacto da refatoração na compreensão de código [manuscrito] : um estudo empírico com estudantes de computação / Halan Caio Pereira do Nascimento. - 2025.
39 f. : il. color.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Ciência da computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2025.

"Orientação : Prof. Dr. José Aldo Silva da Costa, Coordenação do Curso de Computação - CCEA".

1. Refatoração. 2. Compreensão de código. 3. Impactos da refatoração. I. Título

21. ed. CDD 005.1

Elaborada por Kaliane Eveny Martins de Oliveira - CRB - 15/986 BSC7

HALAN CAIO PEREIRA DO NASCIMENTO

INVESTIGANDO O IMPACTO DA REFATORAÇÃO NA COMPREENSÃO DE CÓDIGO: UM ESTUDO EMPÍRICO COM ESTUDANTES DE COMPUTAÇÃO

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação

Aprovada em: 06/06/2025.

BANCA EXAMINADORA

Documento assinado eletronicamente por:

- **Rosangela de Araujo Medeiros** (***.723.558-**), em **17/06/2025 10:25:22** com chave **8541830e4b7e11f098901a7cc27eb1f9**.
- **José Aldo Silva da Costa** (***.862.334-**), em **17/06/2025 09:55:34** com chave **5bb1b8e64b7a11f0affe06adb0a3afce**.
- **Vinícius Augustus Alves Gomes** (***.754.334-**), em **17/06/2025 16:52:51** com chave **a6f630404bb411f0a3c42618257239a1**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Folha de Aprovação do Projeto Final
Data da Emissão: 18/06/2025
Código de Autenticação: fcde9a

RESUMO

Atualmente o mercado de desenvolvimento de software vem se tornando cada vez mais competitivo e uma constante corrida contra o tempo. Por esse motivo, diversas organizações prezam pela entrega final de um produto e negligenciam o uso de padrões e técnicas de código limpo, sendo necessário refatoração posteriormente. A refatoração é aplicada para melhorar a legibilidade e manutenção de um código. Contudo, ainda há uma escassez de estudos que investiguem os efeitos da refatoração na compreensão de código fonte. Este trabalho tem como objetivo identificar o impacto das refatorações *Extract Method*, *Inline Method* e *Rename Method* na compreensão de código, no intuito de auxiliar equipes a adotarem métodos mais eficazes, e assim promover um ambiente de desenvolvimento de software mais eficiente. Para isso, foram realizados experimentos controlados com dois grupos de estudantes do curso de Ciência da Computação da Universidade Estadual da Paraíba (UEPB), visando investigar como essas abordagens afetam a legibilidade e entendimento, assim como as suas preferências de desenvolvimento. Como resultado, constatou-se que os alunos entrevistados sentem mais facilidade em compreender códigos mais enxutos e curtos, assim como mais descritivos, podendo chegar em até 38% de redução do tempo empregado durante essa compreensão. Este estudo contribui para a orientação de profissionais e estudantes a respeito dos efeitos da refatoração de código, assim como a utilização de dados empíricos dos seus benefícios e possíveis limitações.

Palavras-Chave: refatoração; compreensão de código; impactos.

ABSTRACT

The software development market is becoming increasingly competitive and a constant race against time. For this reason, many organizations value the final delivery of a product and neglect the use of clean code standards and techniques, requiring refactoring later. Refactoring is applied to improve the readability and maintainability of a code. However, there is still a lack of studies that investigate the effects of refactoring on source code comprehension. This work aims to identify the impact of the Extract Method, Inline Method and Rename Method refactorings on code comprehension, in order to help teams adopt more effective methods, and thus promote a more efficient software development environment. For this purpose, controlled experiments were carried out with two groups of Computer Science students at the State University of Paraíba (UEPB), aiming to investigate how these approaches affect readability and comprehension, as well as their development preferences. As a result, it was found that the students interviewed find it easier to understand leaner and shorter codes, as well as more descriptive ones, which can reduce the time spent on this understanding by up to 38%. This study contributes to the guidance of professionals and students regarding the effects of code refactoring, as well as the use of empirical data on its benefits and possible limitations.

Keywords: refactoring; code understanding; impacts.

LISTA DE FIGURAS

Figura 1 - Trecho de código antes de aplicar o *Extract Method* ……..…….... 14

Figura 2 - Trecho de código após aplicar o *Extract Method* …...…….. 14

Figura 3 - Trecho de código antes de aplicar o *Inline Method* ……..…..…… 16

Figura 4 - Trecho de código após aplicar o *Inline Method* ……..…….. 17

Figura 5 - Trecho de código antes de aplicar o *Refactor* ……..……..….. 17

Figura 6 - Trecho de código após aplicar o *Refactor* ……..….. 18

Figura 7 Exemplo de comparação de códigos ……. 33

LISTA DE QUADROS

Quadro 1 – Proposta de elaboração de tarefas 23

Quadro 2 – Distribuição das tarefas entre os participantes …..... 26

Quadro 3 – Proposta de roteiro de entrevista …….... 28

LISTA DE GRÁFICOS

Gráfico 1 - Média de tempo de resposta em segundos com *Extract Method* 30

Gráfico 2 - Média de tempo de resposta em segundos com *Inline Method* …….. 30

Gráfico 3 - Média de tempo de resposta em segundos com *Rename Method* … 31

Gráfico 4 Média da acurácia das respostas ……. 32

Gráfico 5 - Preferências dos alunos diante das refatorações apresentadas 34

LISTA DE TABELAS

Tabela 1 - Motivações para aplicação da Extração de Método ……..... 15

SUMÁRIO

1 INTRODUÇÃO..... 10
1.1 Problemática..... 11
1.2 Justificativa..... 11
1.3 Objetivos..... 12
2 REFERENCIAL TEÓRICO..... 13
2.1 Conceitos e fundamentos de refatoração..... 13
2.2 Técnicas de refatoração..... 13
2.2.1 Extract Method..... 13
2.2.2 Inline Method..... 16
2.2.3 Rename..... 17
2.3 Compreensão de Código..... 18
2.3.1 Definições..... 18
2.3.2 Fatores que influenciam a compreensão de código..... 19
2.3.3 Técnicas para medir a compreensão de código..... 20
3 METODOLOGIA..... 22
3.1 Seleção dos participantes..... 22
3.2 Experimento controlado..... 23
3.2.1 Objetivo do experimento..... 23
3.2.2 Elaboração das tarefas..... 23
3.2.3 Desenho experimental..... 25
3.2.4 Coleta de dados..... 26
3.2.5 Análise dos dados..... 26
3.2.6 Ferramentas e procedimentos..... 27
3.3 Entrevistas..... 27
3.3.1 Objetivos das entrevistas..... 27
3.3.2 Elaboração do roteiro de entrevista..... 27
3.3.3 Condução das entrevistas..... 28
3.3.4 Análise dos dados..... 28
4 RESULTADOS ENCONTRADOS..... 29
4.1 Impacto das refatorações no tempo de resolução das tarefas..... 29
4.2 Impacto na acurácia das respostas..... 32
4.3 Preferência e entrevistas..... 33
5. CONSIDERAÇÕES FINAIS..... 37
REFERÊNCIAS..... 38
APÊNCIDE 1 - Formulário de consentimento..... 39

1 INTRODUÇÃO

No cenário atual, marcado pelo constante surgimento de novas tecnologias em um ambiente altamente competitivo, o desenvolvimento de software vem se tornando uma contínua corrida contra o tempo. No entanto, essa luta contra o tempo tem efeitos negativos na sua qualidade. Devido a alta demanda, muitas organizações prezam inicialmente pela rápida entrega de um produto, o que pode comprometer significativamente a sua qualidade final. Isso se dá por razão de que, à medida que os desenvolvedores são pressionados e enfrentam prazos extremamente curtos, eles acabam descuidando de certos padrões e boas práticas essenciais de um software bem construído, resultando em uma entrega funcional, entretanto, comprometendo a qualidade do código.

Código de qualidade ruim afeta sua manutenção e evolução. No processo de codificação, os programadores devem levar em consideração que o sistema deverá ser mantido no decorrer do tempo, e que possivelmente esse código receberá a adição de novos recursos. No entanto, quando o código é mal estruturado, nota-se dificuldade na compreensão do código por parte de outros membros da equipe. Essa falta de clareza no código torna o processo de manutenção e evolução mais lento e propenso a erros, comprometendo a eficiência e a eficácia da equipe.

Martin (2009) afirma que escrever um código limpo exige o uso disciplinado de uma miríade de pequenas técnicas aplicadas por meio de uma sensibilidade meticulosamente adquirida sobre ‘limpeza’. Ou seja, além do conhecimento técnico da construção de um software, é imprescindível possuir também o conhecimento de padrões e técnicas de codificação limpa, para assim garantir a qualidade do código e facilitar a manutenção e colaboração ao longo de seu desenvolvimento.

Contudo, nem sempre todos os profissionais vinculados a uma equipe de desenvolvimento se atentam ao uso de padrões e técnicas de codificação necessárias, ocasionando conflitos. Isso porque quando outro membro precisa corrigir determinado *bug* gerado ou apenas entender o que se passa naquela sessão de código, o que aparenta uma atividade simples, acaba se tornando uma tarefa demorada e confusa de ser cumprida. Neste contexto, a refatoração do código surge como solução para

remediar código de qualidade ruim, objetivando aprimorar a estrutura do código e melhorar seu entendimento.

Fowler (2018) afirma que o propósito da refatoração é tornar o software mais fácil de entender e modificar, realizando mudanças no software que alterem pouca coisa ou nada no comportamento observável. Somente as mudanças feitas para tornar o software mais fácil de entender são consideradas refatorações. A técnica de refatoração tem sido amplamente reconhecida como uma prática benéfica para melhorar a estrutura interna do código. No entanto, faltam estudos sobre seu real impacto na compreensão dos desenvolvedores.

1.1 Problemática

A aplicação de técnicas de refatoração podem simplificar o código resultando em um número de linhas reduzido e estruturas mais claras e eficientes. No entanto, existem ocasiões em que refatorar o código pode torná-lo mais extenso. Por exemplo, quando um desenvolvedor extrai um método de um código curto. A extração do método resultará na criação de um novo método que inclusive pode conter variável de retorno.

Nesses casos, a refatoração é facilmente perceptível, aumentando o seu número de linhas, o que revela um impacto no código em si, porém, há uma necessidade de sabermos, por exemplo, como isso impacta a pessoa que lê ou desenvolve o código, ou seja, qual o impacto no entendimento do código por parte do desenvolvedor.

Entretanto, há uma escassez de estudos que investiguem a respeito do impacto da refatoração na compreensão do código. Em particular, estudos que investigam como a refatoração influencia na capacidade do desenvolvedor de ler e interpretar o código. Um estudo voltado para esse tema pode comparar códigos mal estruturados com suas versões pós-refatoração para demonstrar se houver melhorias de forma prática na compreensão.

1.2 Justificativa

Investigar o impacto de melhorias internas do código no entendimento dos desenvolvedores, em especial, desenvolvedores novatos, afeta em diversos aspectos importantes no desenvolvimento de software, principalmente no que concerne à

manutenção e adição de novos recursos. Quando se há uma boa clareza no código, os desenvolvedores são capazes de compreendê-lo e modificá-lo com mais agilidade. Com o código mais limpo e organizado, corrigir erros e implementar novas funcionalidades se torna uma tarefa mais agradável e simples de executar.

Por essas razões, é importante conduzir estudos que busquem evidenciar o real impacto de refatorações na compreensão de código, permitindo que sua adoção seja fundamentada em indicativos concretos. Além disso, resultados de pesquisas podem guiar a definição de melhores práticas de refatoração, auxiliando equipes a adotarem métodos mais eficazes e evitarem abordagens que não apresentam benefícios claros. Essa base sólida de conhecimentos contribui para uma melhor tomada de decisões que promovem um ambiente de desenvolvimento mais eficiente.

1.3 Objetivos

Geral

Este trabalho tem como objetivo investigar como a refatoração de código impacta o entendimento do código de alunos do curso de Ciência da Computação.

Específicos

- Realizar um experimento com as técnicas de refatoração *Extract Method*, *Inline Method* e *Rename Method*, envolvendo alunos de Ciência da Computação da Universidade Estadual da Paraíba (UEPB) para obter dados quantitativos;
- Realizar entrevistas almejando obter feedback qualitativo;
- Analisar os dados e discutir os resultados;
- Refletir sobre importantes lições aprendidas.

2 REFERENCIAL TEÓRICO

2.1 Conceitos e fundamentos de refatoração

O termo refatoração é proveniente do inglês ‘Refactoring’, que se baseia em realizar alterações em partes do código fonte de um software, sem alterar seu comportamento (Fowler, 2018), a fim de melhorar seu desempenho e evitar sua deterioração durante seu ciclo de vida.

Fowler (2018) afirma que

> “Ao longo dos anos, muitas pessoas no mercado passaram a usar ‘refatoração’ para se referir a qualquer tipo de limpeza no código [...] A refatoração está totalmente ligada à aplicação de pequenos passos que preservam o comportamento, efetuando uma grande mudança por meio do encadeamento de uma sequência desses passos”.

Esses passos se tratam de técnicas desenvolvidas ao longo dos anos que vem tornando o código mais limpo e menos propenso a falhas. Elas apresentam uma motivação para a refatoração de código, eliminando *bad smells* que se referem ao “mal cheiro” dentro dele. Neste contexto, mal cheiro refere-se a sinais de baixa qualidade, indicando potenciais ameaças à manutenção e evolução do software. Eles não apontam erros diretos, mas sim problemas de design, como duplicação de código ou métodos longos. Identificar e corrigir esses "cheiros ruins" é crucial para evitar dificuldades futuras e garantir a sustentabilidade do sistema, de acordo com a Suryanarayana *et al*. (2014).

2.2 Técnicas de refatoração

Existem diversas técnicas de refatoração que podem ser aplicadas durante o desenvolvimento de um software. Fowler (2018) apresenta um catálogo contendo diversas dessas abordagens, definindo e apresentando seu uso prático para melhor entendimento de cada uma. Dentre as principais, destacam-se:

2.2.1 Extract Method

A extração de métodos faz-se necessária quando um método específico do sistema possui uma grande quantidade de linhas em sua composição, dificultando seu

entendimento ou então se há blocos que podem ser extraídos sem comprometer o objetivo geral do método.

Para implementar o *Extract Method*, o desenvolvedor deve analisar o método e identificar se em seu meio há blocos que podem ser extraídos para métodos distintos, sem comprometer sua funcionalidade geral, visando melhorar a compreensão e manutenção do código. No exemplo apresentado na Figura 1, observa-se a implementação do método *printOwing*, que inicialmente chamava o método *printBanner* e, em seguida, imprimia informações adicionais.

Figura 1 - Trecho de código antes de aplicar o *Extract Method*

```
public void printOwing() {
    printBanner();

    // print details
    System.out.println("Name: " + name);
    System.out.println("Amount: " + getOutstanding());
}
```

Fonte: Adaptado de Fowler (2019)

Após a refatoração, conforme visto na Figura 2, foi criada uma nova função, *printDetails*, com a responsabilidade de exibir as informações anteriormente geradas por *printOwing*. Agora, o método *printOwing* limita-se a invocar o novo método *printDetails*, que por sua vez centraliza a impressão das informações.

Figura 2 - Trecho de código após aplicar o *Extract Method*

```
public void printOwing() {
    printBanner();

    // print details
    printDetails()
}

public void printDetails(){
    System.out.println("Name: " + name);
    System.out.println("Amount: " + getOutstanding());
}
```

Fonte: Adaptado de Fowler (2019)

Silva *et al*. (2016) explicita que as maiores motivações para adoção do *Extract Method* nos processos de refatoração, classificando-os por incidências de cada uma delas em ordem decrescente. Dentre as maiores ocorrências, têm-se as seguintes motivações:

Tabela 1 - Motivações para aplicação da Extração de Método

Tema	Descrição	Ocorrências
Extrair método reutilizável	Extraia um pedaço de código reutilizável de um único lugar e chame o método extraído em vários lugares.	43
Introduzir assinatura de método alternativo	Introduza uma assinatura alternativa para um método existente (por exemplo, com parâmetros adicionais ou diferentes) e faça o método original delegar ao extraído.	25
Método de decomposição para melhorar a legibilidade	Extraia um pedaço de código com uma funcionalidade distinta em um método separado para tornar o método original mais fácil de entender.	21
Facilitar a extensão	Extraia um pedaço de código em um novo método para facilitar a implementação de um recurso ou correção de bug, adicionando código extra no método extraído ou no método original.	15
Remover duplicação	Extraia um pedaço de código duplicado de vários lugares e substitua as instâncias de código duplicadas por chamadas para o método extraído.	14
Método de substituição preservando a compatibilidade com versões anteriores	Introduza um novo método que substitua um existente para melhorar seu nome ou remover parâmetros não utilizados. O método original é preservado para compatibilidade com versões anteriores, é marcado como obsoleto e delega para o extraído.	6
Melhore a testabilidade	Extraia um pedaço de código em um método separado para permitir seus testes unitários isoladamente do restante do método original.	6
Habilitar substituição	Extraia um pedaço de código em um método separado para permitir que subclasses substituam o comportamento extraído por um comportamento mais especializado.	4
Habilitar recursão	Extraia um pedaço de código para torná-lo um método recursivo.	2
Introduzir método de fábrica	Extrair uma chamada de construtor (criação de instância de classe) em um método separado. Extrair	1
Introduzir operação assíncrona	um pedaço de código em um método separado para fazê-lo executar em um thread.	1

Fonte: Silva *et al*. (2016)

Conforme apresentado na Tabela 1, a principal motivação é a extração de método reutilizável de um único lugar e que chame o método extraído em vários lugares. Geralmente essa abordagem é aplicada quando vemos um bloco de código dentro de um método que poderá ser usado mais vezes, de forma individual, em mais seções do software, então ele é retirado de dentro do método e criado em um método separado;

Em segunda posição em quantidade de ocorrências, destaca-se a extração de um bloco de método para uma delegação posterior dentro do método original. Diferente da motivação anterior, essa abordagem é utilizada quando queremos dividir um método extenso sem objetivar o reuso do bloco extraído em outro local; Na sequência, observa-se a extração para simplificar a interpretação. Essa técnica nos permite extrair um pequeno trecho de código para favorecer uma compreensão geral do código.

2.2.2 Inline Method

De forma contrária ao *Extract Method*, a internalização de métodos refere-se à ‘mesclagem’ de métodos. Frequentemente é utilizado quando um método é extremamente simples e não agrega valor significativo para reutilização. A ideia principal é reduzir a complexidade desnecessária do código desenvolvido.

Para a aplicação do *Inline Method*, é recomendado analisar previamente se este método não é sobrescrito por subclasses. Caso não haja sobrescrita, o desenvolvedor deve encontrar os demais métodos chamados em sua composição e substituir cada chamada pelo corpo do método e posteriormente excluir a definição do método internalizado.

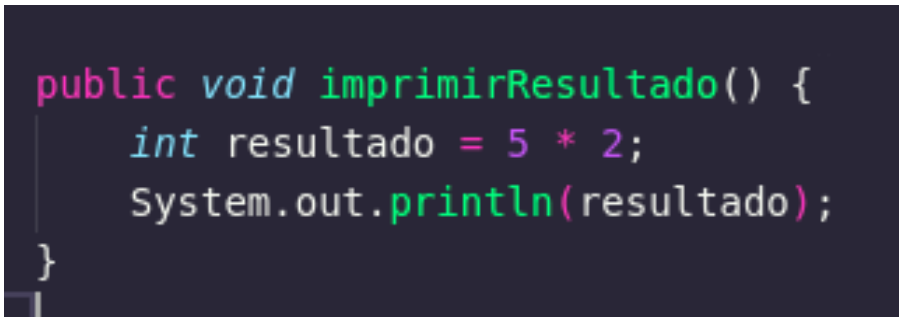
No trecho de código mostrado na Figura 4, vemos a execução dessa técnica. Observa-se que inicialmente temos dois métodos diferentes, sendo que o primeiro método *calcularDobro* é acionado pelo segundo método *imprimirResultado*. Após a aplicação do *Inline Method*, o método resultante se torna único, tornando o corpo do *calcularDobro* integrado ao método principal *imprimirResultado*, conforme mostrado na figura 5.

Figura 3 - Trecho de código antes de aplicar o *Inline Method*

```java
public int calcularDobro(int numero) {
    return numero * 2;
}

public void imprimirResultado() {
    int resultado = calcularDobro(5);
    System.out.println(resultado);
}
```

Fonte: Elaborado pelo autor (2025).

Figura 4 - Trecho de código após aplicar o *Inline Method*



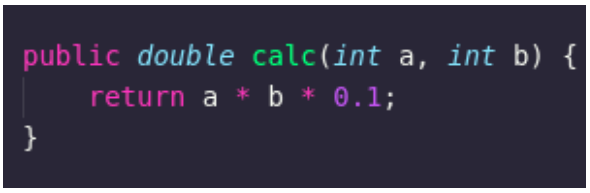
Fonte: Elaborado pelo autor (2025).

2.2.3 Rename

A refatoração *Rename* envolve alterar nomes de variáveis, campos, definições de classes ou de métodos para torná-lo mais significativo e descritivo, tornando mais simples de entender e manter. Nomes significativos são importantes quando essas estruturas são usadas frequentemente em todo o código do programa.

A utilização do *Rename* é bem intuitiva. Sempre que observado que um identificador implementado não condiz com o que ele está de fato cumprindo, se faz necessário que ele seja renomeado para que não haja conflitos de interpretação e aplicação da estrutura. No código evidenciado vemos a aplicação da renomeação do identificador de um método que anteriormente chamado *calc,* se passou a ser chamado *calcularDesconto*. Da mesma forma seus parâmetros, que também foram refatorados, anteriormente com os nomes ‘a’ e ‘b’, agora com nomes mais descritivos para sua utilidade.

Figura 5 -Trecho de código antes de aplicar o *Rename*



Fonte: Elaborado pelo autor (2025).

Figura 6 - Trecho de código após aplicar o *Rename*

```
public double calcularDesconto(int preco, int quantidade) {
    return preco * quantidade * 0.1;
}
```

Fonte: Elaborado pelo autor (2025).

2.3 Compreensão de Código

2.3.1 Definições

A compreensão de código é a atividade pela qual engenheiros de software chegam a um entendimento do comportamento de um sistema de software, utilizando o código-fonte como principal referência (Bennett *et al*., 2002). De forma semelhante, para Xia *et al*. (2018), a compreensão de programas é um processo em que desenvolvedores adquirem ativamente conhecimento sobre um sistema de software ao explorar e pesquisar artefatos de software, além de ler o código-fonte e/ou documentação relevante.

Boehm *et al.* (1976) observam que

> "O código possui a propriedade de compreensibilidade na medida em que seu propósito é claro para o revisor. Isso implica que nomes de variáveis ou símbolos são usados consistentemente, módulos de código são auto descritivos, estrutura de controle é simples ou está em conformidade com um padrão prescrito, etc".

Deste modo, a compreensão envolve reconhecer os elementos sintáticos do código, como variáveis, funções e estruturas de controle, assim como também entender a lógica subjacente e as intenções de design que conduzem o comportamento do sistema.

É importante considerar que os desenvolvedores analisam e entendem um trecho de código-fonte para realizar tarefas específicas, como manutenção, depuração, ou melhoria de funcionalidades. Para Xia *et al*. (2018), o conhecimento adquirido do

código no processo de entendimento auxilia em outras atividades de engenharia de software, como correção de erros, aprimoramento, reutilização e documentação.

Compreender a lógica do código permite que os desenvolvedores diagnostiquem as áreas nas quais ocorrem falhas, reduzindo o risco de introduzir novos erros. O entendimento detalhado de como o software funciona possibilita que desenvolvedores escrevam documentações mais claras e úteis, o que facilita a transferência de conhecimento e reduz a curva de aprendizado para novos integrantes da equipe.

2.3.2 Fatores que influenciam a compreensão de código

Entender código pode tornar-se uma tarefa confusa e complicada e diversos fatores podem influenciar na compreensão de código. Um desses fatores consiste na complexidade do código. Feitelson (2023) declara que “é difícil entender o código escrito por outros. Isso é comumente atribuído, pelo menos em parte, à complexidade do código: quanto mais complexo o código, mais difícil é entendê-lo e, por implicação, trabalhar com ele.”

Desta forma, o software deve ser desenvolvido seguindo padrões e técnicas de codificação, que contribuirá para que outros desenvolvedores compreendam sua lógica e execução mais facilmente. De acordo com o crescimento do software, sua complexidade também se intensifica, por esse motivo, seguir padrões de programação permitirá que o desenvolvedor compreenda o objetivo do software que está manipulando.

Outro fator importante consiste na experiência do desenvolvedor, que deve possuir conhecimentos prévios das linguagens de programação, *frameworks* e bibliotecas utilizadas naquele software. De acordo com Littman *et al.* (1987), “entender um programa compreende conhecer os objetos que o programa manipula e as ações que ele executa, bem como seus componentes funcionais e as interações causais entre eles”.

Por esse motivo, deve-se levar em consideração que desenvolver um software requer mais do que um conhecimento de uma linguagem de programação ou *framework*, depende principalmente de conhecer técnicas de desenvolvimento limpo, que deixará o código mais simples e intuitivo, tornando seu entendimento mais claro.

Um ponto adicional consiste na legibilidade do código. Martin (2009) enfatiza que “desenvolvedores leem 10 vezes mais código do que escrevem,” e, por isso, práticas de desenvolvimento limpo são essenciais. Padrões de nomenclatura, estrutura modular e comentários claros melhoram a legibilidade e permitem uma compreensão mais intuitiva do código.

Além disso, a presença de uma documentação bem estruturada também facilita o entendimento do código, especialmente para novos membros da equipe. Ao compreender o contexto e os objetivos do software, os desenvolvedores conseguem identificar mais facilmente a lógica de cada módulo, resultando em uma curva de aprendizado reduzida.

2.3.3 Técnicas para medir a compreensão de código

Duas principais abordagens têm sido utilizadas para investigar compreensão de código que são abordagens qualitativas e quantitativas. As abordagens qualitativas podem ser entendidas como entrevistas qualitativas e questionários estruturados fornecem dados valiosos sobre as dificuldades e estratégias utilizadas pelos desenvolvedores para entender o código.

Busetto (2020) confirma que as entrevistas são usadas para obter *insights* sobre experiências subjetivas, opiniões e motivações de uma pessoa, podendo ser distinguidas como estruturadas (questionários), abertas (entrevista) ou semiestruturadas. Essa abordagem qualitativa permite coletar percepções pessoais e identificar os principais desafios relatados.

Nas abordagens quantitativas, os desenvolvedores são convidados a realizar tarefas específicas. Os tempos de execução e a precisão das respostas são analisados para medir a eficiência na compreensão. Feitelson (2023) indica que essa abordagem permite obter dados quantitativos sobre o desempenho e a eficácia da compreensão de código em cenários práticos.

Dentro dos estudos qualitativos, diversas técnicas têm sido exploradas, incluindo opiniões para coleta de percepções sobre os desafios enfrentados pelos desenvolvedores ao trabalhar com código, questionários para obter informações sobre as estratégias de compreensão e os obstáculos enfrentados e entrevistas, que

permitem explorar as experiências e os métodos que os desenvolvedores utilizam para interpretar o código.

Na investigação experimental, várias métricas têm sido utilizadas para avaliar a compreensão do código, como Feitelson *et al*., (2021) e Oliveira *et al*., (2020) definem, tais como o tempo de execução, em que se é medido o tempo necessário para completar tarefas específicas, fornecendo uma indicação da eficiência na compreensão, a acuracidade, que avalia a precisão das respostas dadas pelos desenvolvedores em relação às tarefas propostas, o esforço visual, utilizando o eye tracking para mapear onde os desenvolvedores focam sua atenção durante a leitura do código, a atividade cerebral, observando a atividade cerebral durante a compreensão do código, ajudando a identificar os processos cognitivos envolvidos e biossensores, utilizando sensores biométricos para medir respostas fisiológicas que podem indicar níveis de estresse ou carga cognitiva durante a tarefa de compreensão.

3 METODOLOGIA

Este trabalho caracteriza-se no que se refere à natureza como pesquisa primária, caracterização essa definida por Wazlawick (2021), que busca realizar observações por meio de experimentos e investigações para criar uma nova teoria.

Atentando a tal natureza, foi empregado o método de pesquisa exploratória, buscando obter-se dados consistentes sobre as implicações das refatorações na compreensão de código, e descritivo através de estudo semiestruturado na UEPB, especificamente na cidade de Patos-PB. Esta conta com dois principais métodos de coleta de dados: um experimento para dados quantitativos (Seção 3.2) e uma entrevista para dados qualitativos (seção 3.3).

3.1 Seleção dos participantes

Para participação neste estudo, foram selecionados 19 alunos do curso de Ciência da Computação com faixa etária de 21 a 30 anos, que se encontram cursando a partir do 5º (quinto) período. Esses alunos já possuem uma boa experiência com desenvolvimento de software através de estudos durante sua trajetória no curso, com as linguagens de programação *Python, C, C#, JavaScript e Java*, e já carregam um bom conhecimento prévio da linguagem e a dominam suficientemente para resolver as tarefas propostas pelo experimento, uma vez que as disciplinas de programação são disponibilizadas entre o 1º (primeiro) e 4º (quarto) semestres.

Para a seleção desses alunos, foram enviados convites por meio de mensagens de texto e de forma presencial. Durante o experimento, foi designada a eles a tarefa de analisar uma série de trechos de códigos completos que aplicam diferentes técnicas de refatoração. Além disso, também foram realizadas entrevistas de natureza semi-estruturada para coletar respostas que foram analisadas e comparadas posteriormente. O roteiro de entrevista é apresentado na Seção 3.3.2.

3.2 Experimento controlado

3.2.1 Objetivo do experimento

O experimento utilizado para levantar dados foi um estudo comparativo controlado que visa investigar como as técnicas de refatoração *Extract Method*, *Inline Method* e *Rename* afetam a compreensão e a clareza do código para os alunos da UEPB, objetivando medir se estas abordagens de fato contribuem positivamente para a compreensão do código. Após o experimento, será possível chegar a conclusão de como a escolha destas técnicas impactam na legibilidade e entendimento do código analisado.

3.2.2 Elaboração das tarefas

Cada participante foi submetido a um conjunto de tarefas de código curto, entre 9 a 18 linhas, que consistiam em resolver um problema em programação na linguagem Java. As tarefas envolvem níveis Médio e Difícil de dificuldade, conforme exibido no Quadro 1. Para resolver a tarefa, o participante precisou ler e compreender o código e informar sua saída final em voz alta, para que fosse verificada a acurácia e o tempo usado de sua resposta.

Quadro 1 - Proposta de elaboração de tarefas

Nome da Tarefa	Nível	Descrição	Refatoração proposta
Verificar sequências	Médio	O método contém três variáveis com valores distintos e logo após uma condicional (IF) que verifica se obedecem uma sequência específica, retornando true ou false.	*Extract Method*
Obter Palavras curtas	Difícil	O método verifica uma variável do tipo String, divide	*Extract Method*

		ela em uma lista, separada pelos espaços contidos no texto e verifica quais palavras possuem três letras ou menos, inserindo-as em outra variável String e retornando-a.	
Verificar múltiplo de cinco	Médio	Inicia com uma variável do tipo Int e verifica se esse valor é um múltiplo de cinco, retornando true ou false como resultado.	*Inline Method*
Calcular preço final	Difícil	O método inicia com uma variável de preço original de um produto do tipo double e calcula o desconto final de 15% de desconto. Retorna um valor do tipo double.	*Inline Method*
Verificar Aprovação	Médio	O método recebe um vetor de notas, calcula a média das notas e informa se o aluno foi aprovado ou reprovado.	*Rename*
Verificar elegibilidade de promoção	Difícil	O método possui duas variáveis inicialmente, informando o tempo de trabalho em anos, do tipo Int, e a avaliação	*Rename*

		de desempenho, do tipo double. Posteriormente valida se o funcionário é elegível para promoção, verificando se possui 7 anos ou mais de serviço e avaliação superior ou igual a 4.0. Retorna true ou false.	

Fonte: Elaborada pelo autor, (2025).

3.2.3 Desenho experimental

No decorrer do experimento, cada participante foi direcionado a responder um conjunto de seis tarefas distintas, sendo elas três não refatoradas e três refatoradas. Entretanto, não foram apresentados trechos iguais de código ao mesmo participante, evitando que descubram com mais facilidade seu resultado, confundindo os resultados do experimento. Ou seja, se a tarefa 1 lhe foi apresentada de forma não refatorada, a tarefa 2, que é a sua versão refatorada, não lhe foi apresentada posteriormente, sendo substituída por outra tarefa diferente. O Quadro 2 a seguir mostra como foram distribuídas as tarefas de acordo com cada participante.

Quadro 2 - Distribuição das tarefas entre os participantes

	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
Participante Ímpar	Verificar sequências	Obter Palavras curtas	Verificar múltiplo de cinco	Calcular preço final	Verificar elegibilidade de promoção	Verificar aprovação
	Extract	*Extract*	*Inline*	*Inline*	*Rename*	*Rename*
	Não Refatorado	Refatorado	Não Refatorado	Refatorado	Não Refatorado	Refatorado
Participante par	Verificar sequências	Obter Palavras curtas	Verificar múltiplo de cinco	Calcular preço final	Verificar elegibilidade de promoção	Verificar aprovação
	Extract	*Extract*	*Inline*	*Inline*	*Rename*	*Rename*
	Refatorado	Não Refatorado	Refatorado	Não Refatorado	Refatorado	Não Refatorado

Fonte: Elaborada pelo autor, (2025).

3.2.4 Coleta de dados

Durante o experimento, foram observadas variáveis como o tempo de resolução, ou seja, quanto tempo foi necessário para a resolução do desafio proposto e a acurácia, verificando se a resolução feita pelo aluno está correta.

Para medir o tempo de resposta do participante em cada desafio, foi utilizado um cronômetro, que foi ligado ao iniciar o desafio e desligado ao momento em que o candidato pronunciava sua resposta em voz alta. Para avaliar a acurácia da resposta, utilizou-se um gabarito previamente preenchido com as resoluções do desafio, informando ao candidato se sua solução estava correta ou incorreta.

3.2.5 Análise dos dados

A etapa de análise de dados do experimento sucedeu em uma análise quantitativa, investigando uma comparação entre os tempos e acurácia das respostas aos desafios propostos, comparando as resoluções que foram mais eficazes entre tempo e exatidão, levantando quais foram os métodos mais eficazes em compreensão entre os alunos.

3.2.6 Ferramentas e procedimentos

A apresentação das versões dos códigos ocorreu por meio de arquivos PDF, contendo os códigos, possibilitando a sua visualização sem contratempos de IDEs ou arquivos ilegíveis. Durante o experimento, o participante foi informado que suas resoluções estariam sendo gravadas e seu tempo de cada desafio seria cronometrado para posterior análise. Para isso, ele assinou um termo de consentimento disponível em apêndice, permitindo assim suas respostas gravadas.

3.3 Entrevistas

3.3.1 Objetivos das entrevistas

A etapa de entrevista visa analisar como as técnicas de refatoração influenciam na compreensão dos códigos apresentados anteriormente no experimento. Seu objetivo foi entender a preferência entre um código mais enxuto ou mais modularizado por parte dos alunos da instituição de ensino, por meio das técnicas de refatoração apresentadas. Da mesma forma, será possível também obter suas opiniões a respeito das dificuldades encontradas por eles durante o processo de compreensão desses códigos.

As entrevistas foram realizadas com os mesmos participantes, sempre ao final do experimento. Elas tiveram um tempo médio de 5 a 10 minutos, conforme o participante respondia as perguntas. Para que suas respostas fossem utilizadas neste estudo, o termo de consentimento também abordava a respeito das entrevistas, permitindo que os participantes tivessem ciência de que suas falas seriam gravadas e armazenadas exclusivamente para uso desta pesquisa.

3.3.2 Elaboração do roteiro de entrevista

Com o objetivo de estruturar a coleta de dados qualitativos do estudo, o Quadro 3 mostra o roteiro de entrevista que guia as interações com os participantes do estudo. As perguntas foram projetadas para capturar dados detalhados que pudessem subsidiar a análise qualitativa da pesquisa e discussão dos resultados.

Quadro 3 - Proposta de roteiro de entrevista

Qual dos dois códigos apresentados você preferiu? Por quê?
Qual código foi mais fácil de compreender? O que contribuiu para isso?
Em qual dos códigos você encontrou mais dificuldade? O que exatamente foi difícil?
A organização do código influenciou sua compreensão? De que forma?
As escolhas dos nomes de variáveis e métodos influenciaram sua compreensão? Como?
Você sabe o que é refatoração de códigos?
Em que momento devemos aplicá-la? Por quê?
Você vê vantagens no uso da refatoração de código? Quais?

Fonte: Elaborada pelo autor, (2025).

3.3.3 Condução das entrevistas

Após a resolução dos desafios propostos, o participante foi perguntado sobre sua preferência em relação aos códigos apresentados, informando qual método ele considera mais simples de entender e o porquê da sua preferência e dificuldades em comparação ao outro exemplo proposto. Essas respostas foram de utilidade para observar se as técnicas de refatoração abordadas neste estudo verdadeiramente melhoram a legibilidade e compreensão do código para desenvolvedores.

3.3.4 Análise dos dados

Na análise de dados qualitativos, foram explorados quais desafios os participantes se sentiram mais confiantes em termos de compreensão e por qual razão eles o definiram como mais legível e mais simples de compreender, observando suas preferências e dificuldades entre os mesmos desafios. As respostas das entrevistas foram gravadas, conforme consentimento dos participantes e posteriormente transcrevidas para planilhas Excel ou Google Sheets, onde serão analisadas minuciosamente para obter-se dados concretos delas.

4 RESULTADOS ENCONTRADOS

Nesta seção, são apresentados e discutidos os resultados. O principal objetivo desta pesquisa foi investigar como a refatoração de código, especificamente as técnicas de *Extract Method, Inline Method* e *Rename*, impactam o entendimento de código dos alunos do curso de Ciência da Computação, permitindo que sua adoção seja fundamentada em indicativos concretos.

A coleta de dados foi realizada por meio de aplicação de experimento controlado envolvendo 19 estudantes de Computação, para dados quantitativos, e entrevistas com esse mesmo grupo, para dados qualitativos. Após a análise destas respostas, os dados foram organizados e interpretados com base em métricas como tempo e acurácia de acertos, assim como preferências dos participantes sobre os códigos em suas versões antes e depois da aplicação da refatoração.

Para isso, foram analisadas variáveis como média, mediana e desvio padrão do tempo de resposta de seis diferentes códigos, assim como a média da acurácia - quantidade de tentativas - para estas mesmas tarefas. Posteriormente, realizou-se a comparação entre as versões refatoradas e não refatoradas dos desafios expostos na qual permitiu avaliar o impacto de cada técnica sobre a performance dos participantes.

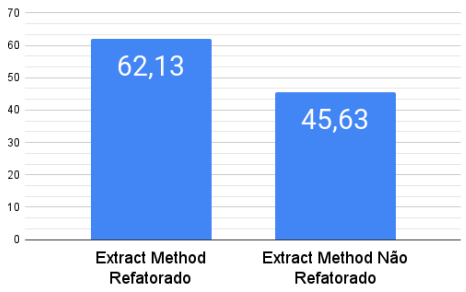
4.1 Impacto das refatorações no tempo de resolução das tarefas

Com respeito ao impacto do *Extract Method*, conforme exibido no Gráfico 1, os participantes que visualizaram e resolveram as tarefas na forma refatorada obtiveram um custo de tempo médio de 62 segundos, ou seja, 1 minuto e 2 minutos por resposta, enquanto que os outros participantes que resolveram as mesmas tarefas em sua forma não refatorada obtiveram um custo médio de 45 segundos por resposta, o que significa um impacto de 27,4% de aumento no tempo ao aplicar a extração do método.

Com respeito ao *Inline Method*, resultados semelhantes também foram encontrados na resolução das tarefas evidenciados no Gráfico 2. Nele pode-se notar que os participantes que solucionaram as tarefas refatoradas com esta técnica alcançaram uma média de tempo de 30 segundos em cada desafio proposto, em oposição aos participantes que executaram as mesmas tarefas, porém implementadas

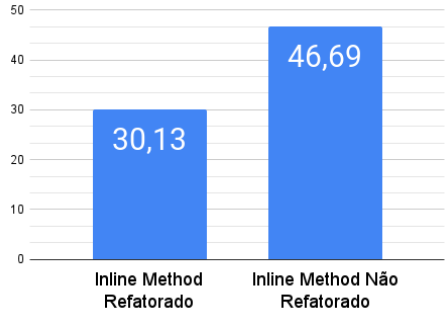
sem a aplicação da técnica proposta, que conseguiram uma média de 46 segundos por tarefa. Isso representa um impacto de uma redução de 35,2% ao aplicar a refatoração.

Gráfico 1 - Média de tempo de resposta em segundos com *Extract Method*



Fonte: Elaborado pelo autor, (2025).

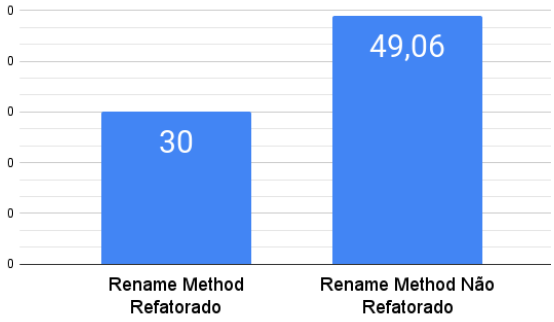
Gráfico 2 - Média de tempo de resposta em segundos com *Inline Method*



Fonte: Elaborado pelo autor, (2025).

Com o *Rename Method*, ao se analisar os tempos de resposta, pode-se constatar que a aplicação da técnica de refatoração causa um impacto positivo na compreensão do código, conforme apresentado no Gráfico 3. Ao realizarem as tarefas com implementação do *Rename Method*, os participantes sentiram mais facilidade de chegar ao resultado, conseguindo o tempo médio de 30 segundos, enquanto que ao realizar os mesmos desafios, sem a refatoração aplicada, os participantes alcançaram o tempo médio de 49 segundos para cada questão. Ou seja, ao aplicar a refatoração, houve uma redução de 38,7% no tempo de resolução.

Gráfico 3 - Média de tempo de resposta em segundos com *Rename Method*



Fonte: Elaborado pelo autor, (2025).

Também foram calculadas a mediana dos tempos de resposta dos participantes, possibilitando encontrar o valor central, quando ordenados, do conjunto de dados obtidos ao medir o tempo da resposta. Com o *Extract Method*, os participantes obtiveram a mediana de 95,5 segundos em sua versão refatorada, enquanto que, na versão não refatorada, chegaram a mediana de 80 segundos, resultando numa redução de 19,3% na mediana.

No *Inline Method* os resultados confirmam o que foi obtido com o *Extract*. Ao realizarem as tarefas refatoradas com a técnica, os participantes alcançaram a mediana de 52,5 segundos, diferentemente dos participantes que resolveram a tarefa não refatorada, que chegaram a mediana de 81,25 segundos, resultando num aumento de 35,3% na mediana.

Do mesmo modo, foram calculadas as medianas de tempo nas tarefas contidas com o *Rename Method*. Ao solucionarem as tarefas com o *Rename* refatorado, os participantes obtiveram a mediana de 53 segundos, dissemelhantemente dos participantes que as resolveram sem a refatoração implementada, que obtiveram a mediana de 86,5 segundos, ou seja, um aumento de 38,7%.

Em complemento a análise de dados quantitativos da pesquisa, também foi calculado o desvio padrão desses dados, a fim de encontrar a medida que expressa o grau de dispersão dos tempos de resposta dos participantes, resultando nos seguintes dados.

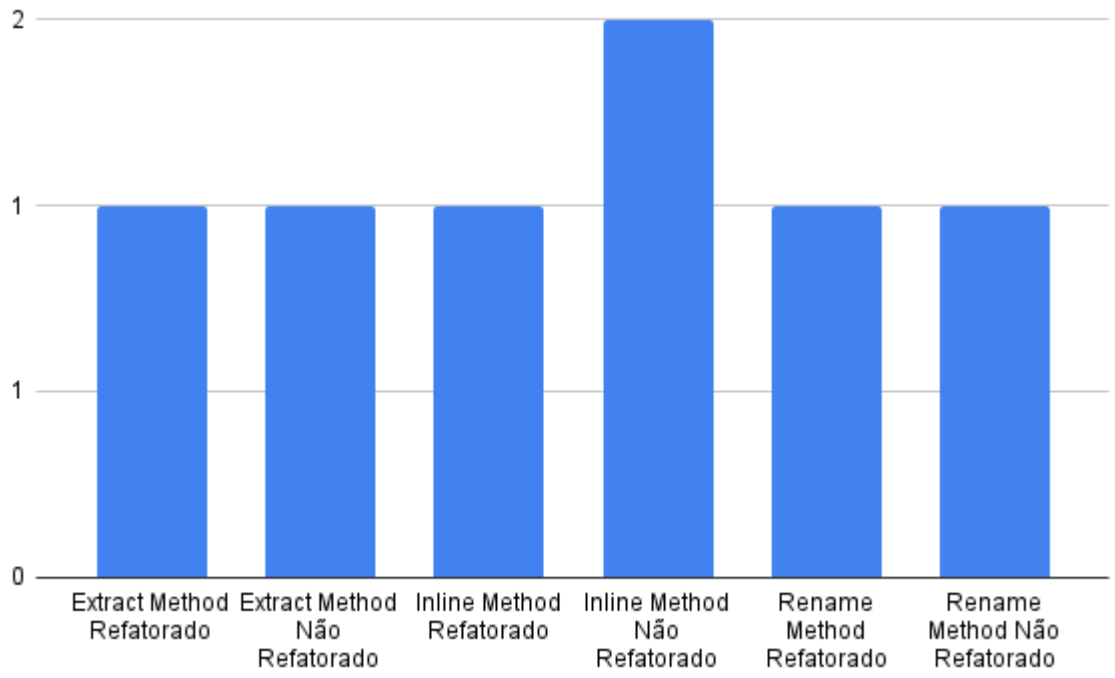
Nas tarefas com o *Extract Method*, a variação de tempo calculada é de 23,88 segundos para versão refatorada e 27 segundos para a versão não refatorada.

Enquanto que, nas tarefas com o *Inline Method*, as variações são de 39 segundos em seus métodos refatorados e 12 segundos nos métodos não refatorados. As variações do *Rename Method* são bem próximas em ambos os cenários. Os resultados alcançados foram uma variação de 18 segundos nas tarefas refatoradas e 21 segundos nas tarefas não refatoradas.

4.2 Impacto na acurácia das respostas

Em termos de acurácia das respostas, pode-se notar que não houve muitas variações na quantidade de tentativas dos participantes para chegar a um resultado correto da tarefa. Grande parte dos participantes optou por dar uma resposta concreta apenas quando tinha total certeza de que sua resposta estaria correta, ocasionando uma média de acurácia entre uma e duas tentativas, conforme exibido no gráfico 4.

Gráfico 4 - Média da acurácia das respostas



Fonte: Elaborado pelo autor, (2025).

Essa média também se deu por razão de que as tarefas apresentadas aos participantes eram tarefas curtas e simples de se resolver. A adoção dessas tarefas mais simples teve o objetivo de não deixar o experimento difícil e cansativo,

possibilitando assim a sua conclusão com o maior aproveitamento das respostas dos participantes.

4.3 Preferência e entrevistas

No que diz respeito às preferências dos participantes, foi realizada também uma comparação entre duas versões dos códigos resolvidos na etapa experimental da pesquisa, sendo uma contendo a aplicação da respectiva refatoração e uma sem qualquer modificação refatorada, conforme apresentado um exemplo na Figura 7.

Figura 7 - Exemplo de comparação de códigos

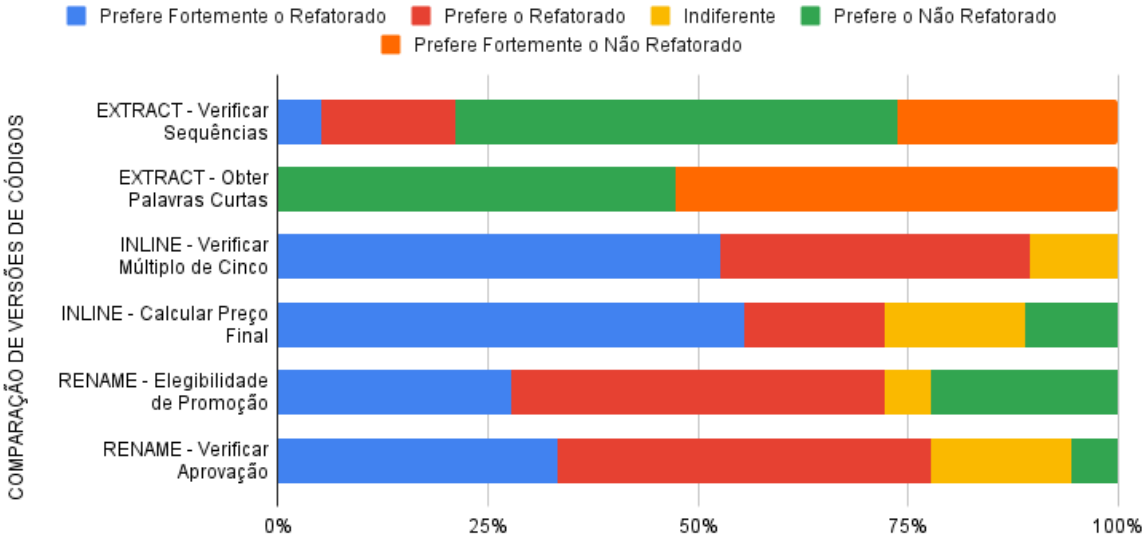
```
public static void main(String[] args) {
    double precoOriginal = 100.0;

    System.out.println(calcularPrecoFinal(precoOriginal));
}

public static double calcularPrecoFinal(double precoOriginal) {
    return precoOriginal - (precoOriginal * 0.15);
}
```

```
public static void main(String[] args) {
    double precoOriginal = 100.0;

    System.out.println(calcularPrecoFinal(precoOriginal));
}

public static double calcularPrecoFinal(double precoOriginal) {
    double desconto = obterDesconto(precoOriginal);
    return precoOriginal - desconto;
}

private static double obterDesconto(double preco) {
    return preco * 0.15;
}
```

Fonte: Elaborado pelo autor, (2025).

O objetivo dessa comparação foi identificar qual das duas versões os alunos consideraram mais compreensível, possibilitando a obtenção de dados sobre sua percepção em relação à facilidade de entendimento dos códigos apresentados.

Gráfico 5 - Preferências dos alunos diante das refatorações apresentadas



Fonte: Elaborado pelo autor, (2025).

Conforme evidenciado no Gráfico 5, observando as métricas dos métodos que aplicam o *Extract Method* e o *Inline Method*, pode-se atentar uma tendência nas preferências dos participantes. Os dados obtidos indicam que os alunos possuem afinidade com os códigos escritos utilizando a técnica *Inline Method* em sua estrutura, expressando que para esse grupo, códigos mais curtos e diretos, com uma quantidade menor de métodos são tidos como mais simples de compreender.

Ao analisar também as métricas das tarefas que incluem o *Rename Method*, é possível reparar que há uma preferência maior dos alunos para os métodos onde essa refatoração se faz presente. Entre as duas opções apresentadas, a maioria dos participantes apontou a versão refatorada como a mais fácil de entender. Isso indica que nomes mais descritivos e claros ajudam bastante na compreensão da lógica por trás do que foi implementado. Esse resultado destaca a importância de dar nomes apropriados aos métodos, pois isso é fundamental para a legibilidade e manutenção do código.

Nas entrevistas, os participantes de forma geral concordaram que códigos escritos de forma mais curta e enxuta são mais simples de compreender do que os códigos mais extensos. Em suas resposta à entrevista um deles expressou essa preferência alegando: “*em ambos os códigos eu não encontrei tanta dificuldade, porém*

eu achei o código REFATORADO mais extenso, chama muitas funções entre si e aparentemente ele é mais custoso e eu preciso ficar mais atento aos retornos de diferentes funções entre si, diferentemente do NÃO REFATORADO que tem apenas uma única chamada e eu preciso ficar atento apenas aos comandos dessa única função". Essa resposta do participante refere-se a aplicação do *Extract Method.*

Também a respeito da técnica do *Extract Method* e em concordância ao comentário acima, outro participante relatou: "*Bom, ele acabou sendo um código mais limpo, com menos linhas, e faz basicamente a mesma coisa do código REFATORADO, só que o código REFATORADO acaba se tornando mais verboso, adicionando coisas que não é necessário para ter o mesmo resultado.*"

Observando esses comentários, pode-se notar que ao utilizar códigos mais extensos e com muitas funções, os participantes sentem uma maior dificuldade em entender o princípio do código,resultando em um tempo maior até sua compreensão. É possível evidenciar esse aumento no tempo de compreensão através do Gráfico 1, que aponta que os códigos refatorados com o *Extract Method* necessitam de mais tempo para serem compreendidos.

Ao ser apresentado uma das tarefas com o *Inline Method* e questionado sobre como a organização do código influenciou na sua compreensão, um dos participantes afirmou: "*influenciou porque o REFATORADO ficou mais enxuto que o NÃO REFATORADO. Ficou mais fácil de ler e compreender.*"

Outro participante foi questionado a respeito de como a organização do código aplicando o *Inline Method* influenciou na sua compreensão, em resposta ele afirmou "*De forma positiva. Porque só uma função, ela faz o que tem que ser feito e ela já retorna o nosso resultado para a nossa main principal.*"

Essas respostas e opiniões dos participantes reforçaram os resultados encontrados no tempo de resposta das tarefas mostrados no Gráfico 2, nas quais a implementação mais enxuta dos códigos resulta em uma maior agilidade para compreensão total de sua funcionalidade, chegando a uma redução de mais de 30% no tempo total de compreensão..

Do mesmo modo, os participantes também foram questionados a respeito da aplicação do *Rename Method* em meio às tarefas a fim de identificar suas preferências e dificuldades ao adotar essa técnica da refatoração.

Ao ser apresentado a comparação dos códigos implementados com o *Rename Method* e questionado sobre como a organização do código influenciou sua compreensão, ele informou que "*a única coisa que influenciou minha compreensão foi a nomenclatura da função.*". Ele enfatiza o motivo dessa resposta ao ser indagado qual código achou mais fácil de compreender, relatando: "*Eu continuo preferindo o código REFATORADO, porque o NÃO REFATORADO me deixou um pouco confuso na questão do 'isElegível', que normalmente a gente usa pra uma constante ou função que retorna pra gente um valor true ou falso, e não cálculo, o que a função tá retornando*."

Identicamente, outro participante destacou: "*Influenciou no quesito de declaração de variáveis, né? Que dependendo ali, nesse caso específico, calculaMédia se encaixa melhor.*". O participante fez essa alegação porque no exemplo em questão o método dispõe de um retorno do tipo *Double* e, na sua versão não refatorada, tinha o nome de 'IsAprovado', enquanto que a sua versão refatorada tinha 'CalculaMedia' como identificador.

Esses feedbacks validam o que também foi constatado na fase experimental, na qual os participantes obtiveram um custo de tempo menor para compreender os códigos refatorados em comparação aos exemplos sem refatoração propostos.

5. CONSIDERAÇÕES FINAIS

O propósito central deste estudo foi examinar o impacto da refatoração de código, em particular as técnicas de *Extract Method*, *Inline Method* e *Rename*, no entendimento de código dos estudantes de Ciência da Computação, possibilitando que sua implementação seja baseada em indicativos tangíveis e concretos.

Os resultados obtidos sugerem que as técnicas de refatoração que priorizem a clareza e simplicidade, como o *Inline Method* e *Rename Method,* contribuem positivamente para uma maior facilidade de compreensão pelos alunos da UEPB. Com essas refatorações, observou-se que a técnica de *Inline* apresentou uma redução de 35,2% no tempo de resposta, enquanto que a técnica de *Rename* que proporcionou uma redução de 38,7% nesse tempo . Isso reforça a relevância da aplicação dessas técnicas como parte de estratégias mais eficazes para um aprendizado mais efetivo destes alunos.

A principal limitação desta pesquisa foi a restrição da disponibilidade do pesquisador para realizar as coletas de dados na universidade, em função de compromissos profissionais em outra cidade, resultando em uma amostra pequena dos alunos desta instituição, de modo a restringir a generalização dos resultados. Mesmo com essa limitação, os dados coletados no experimento forneceram resultados relevantes sobre os impactos da implementação das técnicas de refatoração na compreensão de código pelos alunos.

Trabalhos futuros podem buscar por ampliar o tamanho do grupo investigado, envolvendo estudantes de diferentes níveis e instituições, objetivando uma maior generalização dos resultados obtidos, entregando resultados mais promissores e mais concretos, permitindo uma análise mais robusta dos dados.

De igual modo, recomenda-se a aplicação do estudo utilizando outras técnicas de refatoração para investigar seus reais impactos na clareza e compreensão do código, resultando em uma documentação mais robusta de quais técnicas são eficazes para melhor legibilidade dos códigos entre os desenvolvedores.

REFERÊNCIAS

BENNETT, K.; RAJLICH, V.; WILDE, N. Software evolution and the staged model of the software lifecycle. In: ZELKOWITZ, M. V. (Ed.). *Advances in Computers*. Elsevier, 2002. v. 56.

BOEHM, B. W.; BROWN, J. R.; LIPOW, M.. Quantitative evaluation of software quality. In: *Proceedings of the 2nd International Conference on Software Engineering (ICSE '76)*, San Francisco, California, USA. IEEE Computer Society Press, 1976. p. 592–605.

BUSETTO, L., Wick, W. & Gumbinger, C. How to use and assess qualitative research methods. *Neurol. Res. Pract.* **2**, 14 (2020).

FEITELSON, D. G. From code complexity metrics to program comprehension. Communications of the ACM, v. 66, n. 5, p. 52–61, 2023.

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

LITTMAN, D. C.; PINTO, J.; LETOVSKY, S.; SOLOWAY, E. Mental models and software maintenance. *Journal of Systems and Software*, v. 7, n. 4, p. 341–355, dez. 1987.

MARTIN, R. C. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, 2009.

OLIVEIRA, D.; BRUNO, R.; MADEIRAL, F.; CASTOR, F. Evaluating code readability and legibility: an examination of human-centric studies. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME), 2020, Adelaide, SA, Australia. Proceedings [...]. IEEE, 2020.

SILVA, D.; TSANTALIS, N.; VALENTE, M. T. Why We Refactor? Confessions of GitHub Contributors. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016). New York: ACM, 2016.

SURYANARAYANA, G.; SAMARTHYAM, G.; SHARMA, T. *Refactoring for software design smells: managing technical debt*. Elsevier Science, 2014.

WAZLAWICK, R. S. *Metodologia de pesquisa para ciência da computação*. 3. ed. Rio de Janeiro: LTC, 2021.

XIA, X.; BAO, L.; LO, D.; XING, Z.; HASSAN, A. E.; LI, S.. Measuring program comprehension: a large-scale field study with professionals. *IEEE Transactions on Software Engineering*, v. 44, n. 19, p. 951–976, 2018.

ZANETTE, A.. *Clean code: boas práticas para manter seu código limpo!*. BeCode, 2017.

APÊNCIDE 1 - Formulário de consentimento

UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS V
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE COMPUTAÇÃO/ PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Título da Pesquisa: INVESTIGANDO O IMPACTO DA REFATORAÇÃO NA COMPREENSÃO DE CÓDIGO: UM ESTUDO EMPÍRICO COM ESTUDANTES DE COMPUTAÇÃO

Pesquisador Responsável: Halan Caio Pereira do Nascimento - halan.nascimento@aluno.uepb.edu.br
Orientador: José Aldo Silva da Costa - jose.aldo@servidor.uepb.edu.br

Eu declaro que fui informado(a) sobre os objetivos da pesquisa mencionada acima e autorizo a realização da entrevista, bem como a utilização dos dados coletados (Respostas às tarefas de programação, tempo de completude das tarefas e respostas à entrevista) para fins exclusivos do Trabalho de Conclusão de Curso (TCC) do pesquisador responsável ou de artigos que possam vir posteriormente a este trabalho.

Estou ciente de que minha participação é voluntária, podendo desistir a qualquer momento sem qualquer prejuízo. Também compreendo que meus dados serão tratados de forma a manter meu anonimato e utilizados apenas para os fins acadêmicos mencionados. Também entendo que o objeto de estudo avaliado não sou eu, nem a minha performance mas, as tarefas em questão.

Patos - PB, ___ / ___ / _____

_____

Nome completo do(a) Entrevistado(a)

_____

Assinatura do(a) Entrevistado(a)