UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

DIEGO DEMÉTRIO SANTOS RODRIGUES

REFATORAÇÃO EM REALIDADE VIRTUAL: UM ESTUDO EMPÍRICO EM PROJETOS UNITY

Ouro Preto
2026

DIEGO DEMÉTRIO SANTOS RODRIGUES

REFATORAÇÃO EM REALIDADE VIRTUAL: UM ESTUDO EMPÍRICO EM PROJETOS UNITY

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Profa. Dra. Aline Norberta de Brito

Ouro Preto
2026

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Diego Demétrio Santos Rodrigues

Refatoração em Realidade Virtual: Um Estudo Empírico em Projetos Unity

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 23 de Julho de 2026.

Membros da banca

Aline Norberta de Brito (Orientadora) - Doutora - Universidade Federal de Ouro Preto
Saul Emanuel Delabrida Silva (Examinador) - Doutor - Universidade Federal de Ouro Preto
Rodrigo Ferreira de Brito (Examinador) - Mestre - Universidade Federal de Minas Gerais

Aline Norberta de Brito, Orientadora do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 23/07/2026.

Documento assinado eletronicamente por **Aline Norberta de Brito**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 26/07/2026, às 15:05, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1142653** e o código CRC **A4EB2C6B**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Resumo

A evolução sustentável de software impõe a necessidade de práticas constantes de melhoria da estrutura interna do código. Nesse cenário, a refatoração consolida-se como um mecanismo vital para controlar a dívida técnica e garantir a manutenibilidade dos projetos ao longo do tempo. Contudo, no contexto específico de aplicações de Realidade Virtual (VR) e Aumentada (AR) desenvolvidas sobre a plataforma Unity, a compreensão sobre como essas práticas são adotadas permanece limitada, carecendo de evidências empíricas que caracterizem as modificações no código-fonte frente às idiossincrasias de motores de jogos. Este estudo tem por objetivo identificar e documentar práticas de refatoração recorrentes e específicas nesse domínio. Para tanto, desenvolveu-se uma metodologia de Mineração de Repositórios de Software (MSR) que contempla a seleção de projetos relevantes no GitHub, a extração automatizada do histórico de versões e a aplicação de heurísticas de filtragem textual para distinguir refatorações genuínas de alterações em ativos de mídia ou configuração. Conclui-se que as atividades de refatoração representam uma parcela recorrente da evolução de projetos Unity, concentrando-se principalmente em alterações no código C#, frequentemente acompanhadas por modificações em artefatos específicos da *engine*. Além disso, foram identificados indícios de práticas de refatoração particulares desse domínio, contribuindo para a construção de um catálogo que auxilie na preservação da qualidade de código em sistemas interativos e imersivos.

Palavras-chave: Refatoração. Unity 3D. Realidade Virtual. Mineração de Repositórios de Software. Engenharia de Software.

Abstract

The sustainable evolution of software requires the continuous adoption of practices aimed at improving the internal structure of code. In this context, refactoring has become a vital mechanism for controlling technical debt and ensuring the long-term maintainability of projects. However, in the specific domain of Virtual Reality (VR) and Augmented Reality (AR) applications developed on the Unity platform, the understanding of how such practices are adopted remains limited, lacking empirical evidence that characterizes source code modifications in light of the idiosyncrasies of game engines. This study aims to identify and document recurring and domain-specific refactoring practices in this context. To this end, a Software Repository Mining (MSR) methodology was developed, encompassing the careful selection of relevant projects on GitHub, the automated extraction of version histories using the PyDriller library, and the application of textual filtering heuristics to distinguish genuine refactorings from changes in media assets or configuration files. It is concluded that refactoring activities represent a recurring aspect of the evolution of Unity projects, being primarily concentrated on C# code while frequently involving modifications to engine-specific artifacts. Furthermore, evidence of Unity-specific refactoring practices was identified, contributing to the construction of a refactoring catalog to support the preservation of code quality in interactive and immersive systems.

Keywords: Refactoring, Unity 3D, Virtual Reality, Software Repository Mining, Software Engineering.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABNT	Associação Brasileira de Normas Técnicas
DECOM	Departamento de Computação
UFOP	Universidade Federal de Ouro Preto
VR	Virtual Reality (Realidade Virtual)
AR	Augmented Reality (Realidade Aumentada)
XR	Extended Reality (Realidade Estendida)
HMD	Head-mounted display (Visor montado na cabeça)
LMS	Learning Management System (Sistema de Gestão de Aprendizagem)
xAPI	Experience API (API de Experiência)

Sumário

1 Introdução

A *engine* Unity consolidou-se como uma das principais plataformas para o desenvolvimento de aplicações imersivas, sendo amplamente adotada tanto na indústria quanto na academia (AL-ASWAD et al., 2022; GHRAIRI et al., 2018; COURA et al., 2025). Por exemplo, um estudo recente apontou um crescimento exponencial (RZIG et al., 2023; Fortune Business Insights, 2026), com um mercado global avaliado em cerca de 20 bilhões dólares em 2025.

Desenvolvedores utilizam a *engine* Unity para a implementação da lógica de aplicação, incluindo comportamentos, interações e sistemas de controle (Unity Technologies, 2025). Assim como qualquer sistema de *software*, estas aplicações estão em constante evolução, sendo passíveis de práticas fundamentais da engenharia de *software* (VALENTE, 2022). Dentre estas práticas, pode-se citar a refatoração, definida como o processo de modificar a estrutura interna de um sistema mantendo inalterado o seu comportamento observável (FOWLER, 2018; FOWLER, 1999).

Entretanto, existe uma lacuna na literatura referente a ferramentas e técnicas que deem suporte a esse ambiente, especialmente relacionadas à detecção de refatoração. Ferramentas como RefactoringMiner (TSANTALIS et al., 2018; TSANTALIS; KETKAR; DIG, 2020) e RefDiff (SILVA et al., 2021; SILVA; VALENTE, 2017; BRITO; VALENTE, 2020), estado da arte para detecção de refatorações, não detectam operações específicas do contexto Unity. Além disso, no melhor do nosso conhecimento, refatorações no ambiente Unity são pouco documentadas na literatura. Catálogos recentes discutem outras linguagens de programação (BRITO; HORA; VALENTE, 2023; VEGI; VALENTE, 2025).

A prática da refatoração pode assumir características particulares em domínios interativos e computacionalmente exigentes (RZIG et al., 2023). Aplicações de Realidade Virtual (VR) e Realidade Aumentada (AR) impõem requisitos rigorosos de desempenho, como baixas latências e taxas de quadros estáveis, que são essenciais para garantir a usabilidade e o conforto do usuário (LAVALLE, 2024; WANG et al., 2023).

Diante da escassez de ferramentas de apoio e das especificidades de desempenho impostas por aplicações imersivas, este trabalho tem por objetivo caracterizar práticas de refatorações no contexto de projetos desenvolvidos com a *engine* Unity.

1.1 Justificativa

A motivação central deste trabalho fundamenta-se na natureza crítica dos sistemas de Realidade Aumentada (AR) e Virtual (VR), definidos como aplicações em tempo real onde a performance impacta na usabilidade (LAVALLE, 2024; WANG et al., 2023). Nesse contexto, a

refatoração é uma prática importante para garantir a qualidade do sistema.

Entretanto, observa-se uma lacuna entre a importância dessa prática e o conhecimento empírico sobre a sua aplicação no ecossistema Unity/C#. Embora a ocorrência de problemas estruturais (*code smells*) em aplicações Unity/VR já tenha sido documentada na literatura (RZIG et al., 2023), existe uma escassez de estudos que caracterizem quais padrões de refatoração são efetivamente adotados por desenvolvedores para mitigar esses problemas.

Considerando esse cenário, este trabalho tem por objetivo caracterizar e documentar práticas de refatoração específicas e recorrentes em aplicações desenvolvidas em C# com foco na plataforma Unity. O objetivo é compreender como desenvolvedores lidam com a evolução de código nessa plataforma.

1.2 Objetivos

Esta seção descreve os objetivos geral e específicos necessários para a conclusão do trabalho proposto.

1.2.1 Objetivo Geral

Considerando os desafios de manutenção e evolução de sistemas baseados na *engine* Unity e a importância de práticas de refatoração nesse contexto, pretende-se, neste trabalho, atingir o seguinte objetivo geral:

> Caracterizar e documentar práticas de refatoração específicas e recorrentes em aplicações desenvolvidas em C#, com foco na *engine* Unity.

1.2.2 Objetivos Específicos

Para cumprir o objetivo principal, os seguintes objetivos específicos são propostos:

1. Analisar a frequência de práticas de refatoração em projetos C#, com foco principal no domínio Unity;
2. Identificar e caracterizar refatorações específicas do ecossistema C# e Unity;
3. Discutir técnicas e design de ferramentas capazes de dar suporte para a detecção de refatorações específicas para desenvolvimento de aplicações em C# e Unity.

1.3 Organização do Trabalho

O restante deste documento está organizado de forma a fornecer a fundamentação teórica necessária, detalhar a metodologia aplicada e apresentar os resultados obtidos. A subseção a seguir detalha cada um dos capítulos.

1.3.1 Estrutura da Monografia

A estrutura da monografia segue a organização descrita abaixo:

Capítulo 1: Introdução. Apresenta o contexto, a problematização, a justificativa e os objetivos que norteiam o trabalho.

Capítulo 2: Revisão Bibliográfica. Estabelece a fundamentação teórica necessária para o entendimento do estudo e discute os trabalhos relacionados que situam a pesquisa no estado da arte atual.

Capítulo 3: Metodologia. Descreve os procedimentos metodológicos adotados e a proposta técnica desenvolvida para viabilizar a investigação dos objetivos estabelecidos.

Capítulo 4: Resultados. Apresenta e analisa os resultados obtidos a partir da aplicação da metodologia, discutindo os achados em relação às questões de pesquisa propostas.

Capítulo 5: Considerações Finais. Conclui o estudo, apresentando a síntese dos resultados e as contribuições deste trabalho.

2 Revisão Bibliográfica

Nesta seção, são apresentados os conceitos fundamentais e os trabalhos relacionados à prática de refatoração, com foco na *engine* Unity. Especificamente, a Seção 2.1 inclui os conceitos teóricos relacionados a análise de código. Já a Seção 2.2, discutem-se as principais ferramentas, técnicas e desafios relacionados à detecção de refatoração no histórico dos projetos.

2.1 Fundamentação Teórica

Esta seção apresenta os conceitos teóricos fundamentais necessários para a compreensão da metodologia proposta neste trabalho, que se concentra em práticas de refatoração no domínio de AR/VR.

2.1.1 Realidade Aumentada (AR) e Realidade Virtual (VR)

A Realidade Aumentada (AR) e a Realidade Virtual (VR) são tecnologias que se situam em um espectro contínuo que vai do ambiente puramente real ao puramente sintético, um conceito formalizado como o "*Continuum Realidade-Virtualidade*" (LAVALLE, 2024).

Na Realidade Virtual, o ambiente real do usuário é totalmente substituído por um ambiente computacional (sintético), interativo e tridimensional. A tecnologia visa a imersão sensorial completa, primariamente através de um *head-mounted display* (HMD) – do inglês, visor montado na cabeça – que isola o usuário do mundo físico (LAVALLE, 2024).

A Figura 2.1 mostra um exemplo, que se refere a um sistema para avaliação de interações de usuários em ambientes virtuais de aprendizagem personalizados. Denominado EduVR, o sistema integra uma plataforma *web* de gestão de aprendizagem (*Learning Management System* — LMS) a um ambiente de realidade virtual imersivo, permitindo que educadores gerenciem conteúdos multimídia e acompanhem o comportamento dos estudantes por meio da coleta automatizada de dados via *Experience API* (xAPI) (VIOL et al., 2024).

Figura 2.1 – Exemplo de sistema VR (VIOL et al., 2024)

Em contraste, a Realidade Aumentada não substitui completamente o ambiente. A solução complementa a percepção do ambiente real. Ela se caracteriza pela sobreposição de informações ou objetos virtuais ao mundo físico, exigindo um sistema capaz de realizar o registro (alinhamento) espacial 3D entre os elementos reais e virtuais em tempo real (LAVALLE, 2024).

A Figura 2.2 exemplifica essa capacidade técnica na aplicação móvel Pokémon GO, através do recurso de *Reality Blending* (Niantic, 2020). Neste sistema, algoritmos de visão computacional geram um mapa de profundidade para permitir a oclusão dinâmica. Note que o objeto virtual é renderizado parcialmente oculto pela vegetação real, demonstrando o alinhamento tridimensional preciso entre os domínios físico e digital, superando a simples sobreposição de imagens 2D.

Figura 2.2 – Exemplo de sistema AR (Niantic, 2020)

2.1.2 Refatoração de Código

A refatoração é uma prática formalmente definida por FOWLER (1999) como uma *"alteração na estrutura interna do software para torná-lo mais fácil de entender e mais barato de modificar, sem alterar seu comportamento observável"*. O objetivo principal é gerenciar a complexidade e o débito técnico, melhorando a estrutura interna do código. O autor define em seu livro-texto um vasto catálogo de operações de refatoração na linguagem Java, e atualizado para a linguagens JavaScript na segunda edição (FOWLER, 2018). O termo também é utilizado de forma ampla (VALENTE, 2022), referindo-se a *"melhoria de outros requisitos não-funcionais, que não estão relacionados com manutenibilidade"*.

Durante o ciclo de vida de um sistema de *software*, refatorações são realizadas por diversos motivos. Por exemplo, desenvolvedores refatoram o código para mitigar o acúmulo de dívida técnica (PERUMA et al., 2022), para correção de *bugs* e para facilitar a implementação de novas funcionalidades (SILVA; TSANTALIS; VALENTE, 2016)

A Figura 2.3 mostra um exemplo de refatoração denonimada *Extract Method*. Esta operação consiste em extrair um fragmento de código para um novo método cujo nome descreva claramente o seu propósito (FOWLER, 2018). O processo busca agrupar trechos logicamente relacio-

nados, exigindo o tratamento de variáveis locais e parâmetros para garantir que o comportamento observável do sistema permaneça inalterado (FOWLER, 2018), (FOWLER, 1999). No exemplo, os métodos `printContact()` e `printInfo()` são extraídos de `printDetails()` (BRITO, 2023).

Figura 2.3 – Exemplo de extração de método (BRITO, 2023)

2.2 Trabalhos Relacionados

Esta seção apresenta os principais trabalhos relacionados que fundamentam esta pesquisa. Inicialmente, são discutidos os conceitos e motivações da refatoração. Em seguida, são apresentadas as principais ferramentas e catálogos de refatoração da literatura. Por fim, são abordados estudos sobre relacionados na *engine* Unity, destacando as lacunas que motivam este trabalho.

2.2.1 Refatoração e Motivações

A refatoração é fundamental durante o desenvolvimento de software (FOWLER, 1999). Desenvolvedores movem, extraem, e renomeiam trechos de código constantemente, visando melhorar a manutenibilidade e facilitar a conclusão das tarefas de desenvolvimento (SILVA; TSANTALIS; VALENTE, 2016). Por esse motivo, diversos estudos propõem e desenvolvem técnicas modernas para auxiliar os desenvolvedores nestas tarefas.

Por exemplo, SILVA; TSANTALIS; VALENTE (2016) investigaram as motivações por trás de operações de refatoração, mostrando que a atividade de refatoração é motivada principalmente por mudanças nos requisitos — como a implementação de novas funcionalidades ou

correção de bugs — e com menos frequência pela simples resolução de *code smells*. O estudo catalogou 44 motivações distintas, destacando que operações como `Extract Method` são versáteis e frequentemente utilizadas para facilitar a reutilização de código e viabilizar extensões no sistema, e não apenas para melhorar a legibilidade.

Além disso, os autores observaram que a refatoração manual ainda é predominante, representando 55% dos casos analisados, muitas vezes devido à falta de confiança dos desenvolvedores no suporte automatizado para operações complexas. No entanto, o estudo encontrou evidências de que a IDE utilizada influencia essa adoção, notando que usuários do IntelliJ IDEA realizam significativamente mais refatorações automáticas do que usuários de outras ferramentas, como o Eclipse. O estudo concentra-se apenas em refatorações realizadas na linguagem Java.

2.2.2 Ferramentas e Catálogos de Refatoração

Considerando que a refatoração é uma atividade frequente no desenvolvimento de *software*, sendo realizada por diversos motivos, o uso de ferramentas e técnicas modernas que apoiem esse processo torna-se essencial. A refatoração pode, por exemplo, comprometer a comparação entre versões do código, prejudicando o *code review* (BRITO; VALENTE, 2021). Isso ocorre porque o *diff* textual ignora a estrutura sintática do código (MYERS, 1986), representando refatorações simples como conjuntos ruidosos de adições e remoções, o que dificulta a revisão de código e a compreensão da evolução do software (FALLERI et al., 2014; ALIKHANIFARD; TSANTALIS, 2025). Nesse contexto, ferramentas para detecção automática de refatorações podem minimizar limitações do *diff* textual padrão utilizado em sistemas de controle de versão como o *git* (GITHUB, 2025).

Especificamente, para mitigar esse problema, ferramentas sensíveis à semântica (*semantic-aware*) foram desenvolvidas. A ferramenta RAID (BRITO; VALENTE, 2021), por exemplo, melhora a visualização do diff através da indicação de operações de refatoração detectadas pelo RefDiff (SILVA; VALENTE, 2017; SILVA et al., 2021; BRITO; VALENTE, 2020). Já a ferramenta RefactoringMiner (ALIKHANIFARD; TSANTALIS, 2025) estabeleceu-se como o estado da arte para Java, utilizando análise de AST (árvores de sintaxe abstrata) para alcançar precisão superior a 99%. Entretanto, a sua arquitetura estendida suporta apenas as linguagens Kotlin, Python e C++.

Estas ferramentas baseiam-se em catálogos de operações de refatoração documentados na literatura, que não incluem a linguagem C# e consequentemente da *engine* Unity. Novos catálogos têm sido propostos, por exemplo, para compreender refatorações em Elixir (VEGI; VALENTE, 2025) e refatorações realizadas ao longo do tempo (BRITO; HORA; VALENTE, 2023). Refatorações no ambiente Unity são pouco exploradas na literatura.

De fato, a literatura recente sugere que **sistemas desenvolvidos na *engine* Unity podem ter alterações de código específicas do domínio** (RZIG et al., 2023).

2.2.3 Refatorações na *engine* Unity

Conforme mencionado anteriormente, apesar dos avanços significativos nas ferramentas de detecção de refatoração apresentadas anteriormente (como RefactoringMiner e RefDiff), observa-se uma lacuna importante: a linguagem C# não é suportada nativamente pelas abordagens.

Essa escassez de ferramentas reflete um desafio no desenvolvimento de jogos e aplicações imersivas. RZIG et al. (2023) apontam que a falta de suporte ferramental é uma barreira crítica para construtores de soluções em Realidade Virtual (VR). Os autores destacam que, embora a Unity forneça mecanismos para medição de cobertura de código, estes frequentemente se tornam inutilizáveis devido a problemas de compatibilidade (RZIG et al., 2023). Isso reforça a necessidade de ferramentas para análise estática, detecção e reparo de problemas específicos deste domínio.

Além das limitações técnicas das ferramentas, existe uma lacuna conceitual sobre a natureza das transformações de código neste ambiente. Não está claro na literatura se existem "refatorações específicas" para o domínio Unity AR/VR, ou se as práticas se limitam às refatorações tradicionais da orientação a objetos. A literatura recente demonstra que ecossistemas emergentes ou com paradigmas distintos demandam catálogos próprios, como observado em estudos que documentam refatorações específicas para linguagens como Elixir (VEGI; VALENTE, 2023). Estudos recentes exploram a evolução e manutenção de código em outros contextos, como otimização (NUSRAT et al., 2021).

O desenvolvimento na Unity envolve a manipulação de estruturas próprias da *engine*, como o gerenciamento de *GameObjects*, componentes (*MonoBehaviour*) e *scenes*. Dessa forma, antes de avançar para o desenvolvimento de ferramentas de detecção, faz-se necessário analisar empiricamente as práticas de refatoração reais que ocorrem neste ecossistema, mapeando como os desenvolvedores adaptam conceitos clássicos às restrições e arquiteturas de aplicações AR/VR.

3 Metodologia

Esta seção apresenta a metodologia adotada neste trabalho, desde a seleção dos repositórios e mineração dos dados até a análise dos resultados obtidos.

3.1 Visão Geral

Esta pesquisa adota uma abordagem exploratória e quantitativa, visando compreender as atividades de refatoração realizadas no contexto de projetos de Realidade Virtual (VR) desenvolvidos com a *engine* Unity. A Figura 3.1 mostra a visão geral da metodologia empregada neste estudo, que inclui quatro etapas principais.

A primeira etapa consiste na identificação dos repositórios GitHub de interesse. Em seguida, na etapa de mineração de repositórios de software, realiza-se uma análise automática e em larga escala através de um conjunto de *scripts*, visando identificar atividades de refatoração no histórico dos projetos. A terceira etapa envolve a interpretação dos resultados e classificação das operações de refatoração específicas da *engine* Unity. Por fim, a quarta etapa corresponde à análise e síntese dos resultados obtidos. Nas próximas subseções, detalha-se a metodologia empregada em cada uma das etapas.

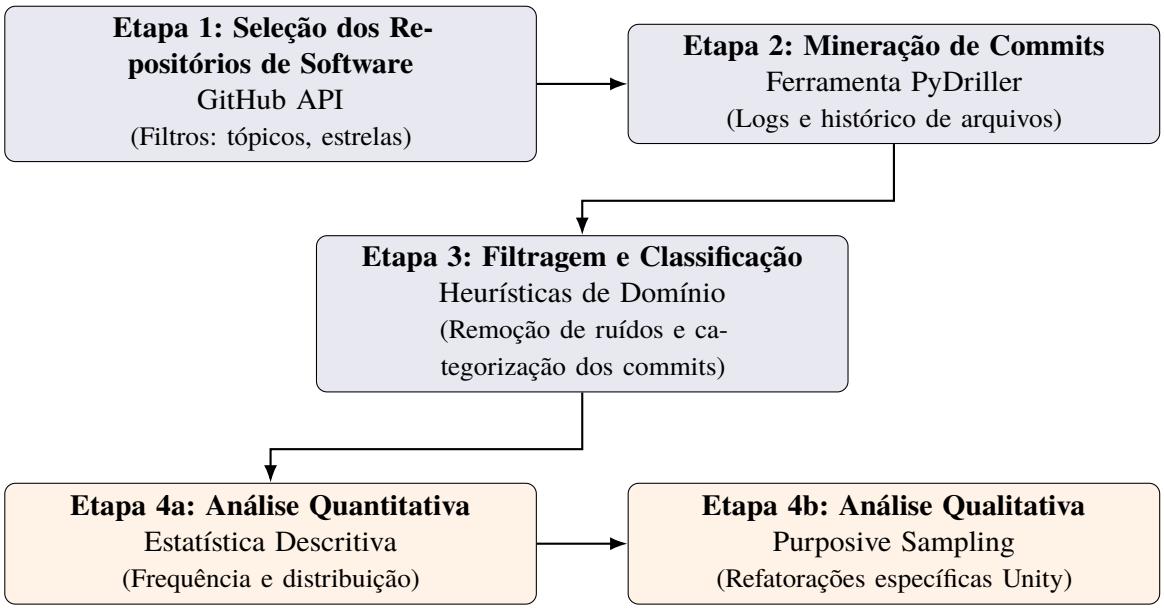


Figura 3.1 – Visão geral da metodologia

3.2 Seleção dos Repositórios de Software

Foi selecionado como *dataset* inicial o conjunto de repositórios extraído do trabalho de RZIG et al. (2023), composto por 43 repositórios. Este conjunto compreende uma base previamente validada de projetos hospedados no GitHub e desenvolvidos no motor Unity para realidade virtual. A partir dessa base, estende-se o *dataset* com a extração de novos repositórios focados em realidade virtual, aumentada e estendida (VR, AR e XR).

Extensão do conjunto de dados. Para expandir o conjunto de dados, extraem-se novos repositórios por meio da `API REST` do GitHub. A busca é direcionada aos projetos mais populares que utilizam C# como linguagem principal e que já estão categorizados com o motor gráfico Unity. Para isso, realizam-se duas consultas distintas na API combinando os parâmetros de busca: `language:C# topic:unity` e `language:C# topic:unity3d`. Os resultados de cada consulta são ordenados de forma decrescente pelo número de estrelas, métrica utilizada para indicar a relevância do sistema (BORGES; HORA; VALENTE, 2016). Para identificar os projetos de tecnologias imersivas, utiliza-se a classificação por tópicos definida pelos próprios desenvolvedores nos repositórios. Com essa abordagem, mitiga-se a inclusão de projetos que apenas mencionam os termos incidentalmente na documentação. A Tabela 3.1 mostra os tópicos considerados neste estudo.

Tabela 3.1 – Tópicos utilizados para filtragem de projetos de tecnologias imersivas

Tópicos			
`vr`	`virtual-reality`	`virtualreality`	`virtual_reality`
`ar`	`augmented-reality`	`augmentedreality`	`augmented_reality`
`xr`	`extended-reality`	`extendedreality`	`extended_reality`
`openxr`	`open-xr`	`open_xr`	

Seleção de repositórios de interesse. Após a etapa de mineração, os projetos da base conhecida e da busca ampla são unificados em um conjunto único, com a remoção de duplicatas. Em seguida, todos os repositórios são avaliados sob os mesmos critérios de filtragem. A aplicação desses critérios tem o objetivo de selecionar projetos mantidos e adequados para a análise:

- *C1 - Forks:* Não ser um *fork*, a fim de evitar duplicidade de código-fonte (BRITO; HORA; VALENTE, 2021).
- *C2 - Disponibilidade:* Exclusão de repositórios deletados da plataforma ou inacessíveis.
- *C3 - Manutenção:* Exclusão de repositórios arquivados, visando a análise de projetos ativos na comunidade (COELHO et al., 2020).
- *C4 - Atividade Recente:* Exclusão de projetos inativos, definidos como aqueles sem novos *commits* nos últimos 5 anos.

3.3 Mineração de Commits

Após seleção dos repositórios GitHub, inicia-se a etapa de identificação de commits que remetem a atividades de refatoração. Para tanto, analisa-se as mensagens dos *commits*, seguindo a estratégia

de *log-based detection* utilizada por RATZINGER et al. (2007). Os autores demonstram que a análise das mensagens de *commit* permite identificar atividades de refatoração diretamente no histórico de versionamento. Em seguida, os commits são filtrados, visando a remoção de modificações que não envolvem alterações de código.

Seleção de commits candidatos. Utiliza-se a biblioteca `PyDriller` para a análise de repositórios Git, permitindo a extração de informações detalhadas sobre *commits* e modificações de arquivos (SPADINI; ANICHE; BACCHELLI, 2018). O processo de mineração é realizado através de um motor de busca customizado que executa as seguintes operações:

1. *Clonagem local dos repositórios.* Cada repositório é clonado para um diretório temporário localmente. Esta abordagem é adotada para evitar o consumo excessivo de limites de requisição (*rate limits*) e *tokens* da API do GitHub, permitindo que a análise de milhares de *commits* ocorra diretamente em disco, sem as restrições impostas por acessos remotos frequentes.

2. *Análise do histórico de commits:* O motor de mineração percorre sequencialmente os *commits* de cada projeto, utilizando a classe `Repository` do `PyDriller` para acessar o histórico de modificações.

3. *Identificação de atividades de refatoração:* Para identificar *commits* envolvendo atividades de refatoração, realiza-se uma análise textual no título e descrição. Especificamente, busca-se por palavras-chave que remetem à atividades de refatoração, definidas na literatura recente (ALOMAR et al., 2022). A Tabela 3.2 apresenta os termos selecionados que abrangem diferentes contextos de atividades de refatoração, tais como: a redistribuição de comportamento entre classes e módulos (*Move*, *Split*), a redução da complexidade e dívida técnica (*Simplify*, *Redesign*) e a melhoria da legibilidade e estrutura do sistema (*Rename*, *Repackage*, *Restructure*).

Tabela 3.2 – Palavras-chave utilizadas relacionadas à refatoração

Termos de Busca		
Extract*	Inlin*	Mov*
Pull* up	Push* down	Repackag*
Redesign*	Refactor*	Renam*
Reorganiz*	Restructur*	Rewrit*
Simplify* code	Split*	

Filtragem de commits. A última etapa de mineração consiste em filtros visando identificar arquivos de interesse nos *commits*. Dado que a mineração baseada em palavras-chave pode retornar um número expressivo de falsos positivos, aplica-se um conjunto de heurísticas de filtragem para refinar o *dataset*. O objetivo desta etapa é isolar modificações que representam refatorações genuínas de lógica e arquitetura, descartando ruídos inerentes ao desenvolvimento com a *engine* Unity e práticas de controle de versão. As regras de exclusão são definidas com base nas seguintes justificativas:

1. *Remoção de artefatos que não são código:* Conforme definido por FOWLER (2018), a refatoração visa a melhoria da estrutura interna do *software*. Portanto, *commits* que alteram exclusivamente

arquivos que não representam código-fonte ou estruturação lógica do projeto são descartados, pois não alteram a lógica do sistema.

2. *Remoção de artefatos e metadados da engine Unity:* A arquitetura do Unity utiliza formatos proprietários para serializar propriedades visuais, rastrear importações e configurar instâncias de conteúdo. São excluídas modificações restritas a metadados (`.meta`) e artefatos de apresentação visual ou áudio, como materiais (`.mat`), clipes e grafos de animação (`.anim`, `.controller`), sequências temporais (`.playable`) e roteamento de som (`.mixer`). Segundo a documentação oficial da ferramenta (Unity Technologies, 2023), estes arquivos descrevem configurações de estado e arte, não constituindo refatoração de código estrutural.

3. *Dependências e SDKs:* São excluídas modificações localizadas em diretórios específicos de bibliotecas externas, *plugins* e SDKs de Realidade Virtual. Esta filtragem é necessária para distinguir a evolução do código proprietário do projeto de atualizações de manutenção em dependências de terceiros. A inclusão de arquivos de bibliotecas copiadas para o repositório distorce significativamente as métricas de atividade (KALLIAMVAKOU et al., 2014), criando a falsa impressão de desenvolvimento produtivo quando ocorrem apenas atualizações de código externo.

4. *Commits sem arquivos modificados:* Essas ocorrências refletem *commits* vazios ou operações exclusivas do sistema de controle de versão que não afetam os artefatos do projeto. Como a identificação de práticas de refatoração exige a análise das modificações em arquivos, esses registros não possuem valor analítico para o escopo do estudo.

5. *Commits de integração (merges):* Mensagens de *commit* automáticas geradas por fusões de ramificações (*merge branches*) são excluídas. Esses registros são removidos pois contêm mensagens geradas automaticamente pelo sistema de controle de versão (Git) (SANTOS; HINDLE, 2016), não fornecendo, portanto, uma descrição explícita sobre a intenção ou o conteúdo da alteração.

As extensões e os caminhos referentes às regras 1, 2 e 3 estão detalhados na Tabela 3.3.

Tabela 3.3 – Artefatos, metadados e diretórios descartados na mineração

Categoria	**Extensões / Caminhos**
Documentação e Web	`.md`, `.txt`, `.gitignore`, `.license`, `.html`, `.css`, `.scss`, `.pdf`, `.rb`, `.gem`
Configuração e Projeto	`.json`, `.yml`, `.yaml`, `.csproj`, `.sln`, `.bat`, `.ps1`, `.sh`, `Makefile`, `.dll`, `.so`, `.dylib`, `.jar`, `.aar`, `.bundle`, `.nuspec`, `.props`
Mídia Genérica Estática	`.png`, `.jpg`, `.tga`, `.psd`, `.fbx`, `.obj`, `.wav`, `.mp3`, `.ogg`
Conteúdo e Apresentação (Unity)	`.mat`, `.anim`, `.controller`, `.playable`, `.mixer`
Metadados (Unity)	`.meta`
Dependências e SDKs (Diretórios)	`Library/`, `Packages/`, `Assets/Plugins/`, `Assets/Oculus/`, `Assets/SteamVR/`

Classificação de commits. Após a filtragem, os *commits* são classificados em quatro categorias baseadas na arquitetura da *engine* (Unity Technologies, 2023). O objetivo é isolar as modificações em código-fonte C#

das demais estruturas lógicas, visuais e de dados de um projeto Unity. A Tabela 3.4 mostra o mapeamento das extensões dos arquivos por categoria.

Tabela 3.4 – Extensões correspondentes às categorias arquiteturais

Categoria	**Extensões**
Código C#	`.cs`
Código Específico de Domínio	`.compute`, `.hlsl`, `.cginc`, `.jslib`, `.shader`
Estrutura Visual e Interface	`.unity`, `.prefab`, `.uxml`
Ativos e Dados	`.asset`, `.asmdef`, `.inputactions`

As categorias são definidas da seguinte forma:

- *Código C#:* Modificações na camada de lógica da aplicação. Abrange alterações em classes e *scripts* que definem o comportamento da aplicação, compreendendo a manipulação de métodos, classes e estruturas de dados.
- *Código Específico de Domínio:* Código executado fora do *runtime* padrão do C#. Envolve processamento paralelo via *Shaders* (GPU) e *scripts* de interoperabilidade para *WebAssembly*, destinados à otimização de performance gráfica ou à adaptação entre plataformas.
- *Estrutura Visual e Interface:* Alterações na composição da hierarquia de objetos (*Scene Graph*) e na estrutura declarativa da interface. A categoria abrange a organização de hierarquias de cena e a estruturação de elementos de interface, sem que haja modificação direta na lógica de controle.
- *Ativos e Dados:* Modificações em ativos de dados serializados e definições de arquitetura. Inclui parâmetros configuráveis (como *ScriptableObjects*), definições de compilação (*Assembly Definitions*) e mapeamentos de entrada.

Para classificar *commits* com múltiplos tipos de arquivos, o script adota uma classificação não exclusiva. O *commit* é contabilizado em todas as categorias correspondentes às extensões modificadas, refletindo o acoplamento arquitetural do Unity (Unity Technologies, 2023). Extensões não mapeadas são classificadas como *Outros*.

3.4 Análise e Síntese dos Resultados

Análise Quantitativa. Realiza-se a análise quantitativa dos dados minerados para responder às duas primeiras questões de pesquisa relacionadas com a frequência de atividades de refatoração e tipos de arquivos alterados, conforme detalhado a seguir.

(QP1) Qual é a frequência de commits relacionados a atividades de refatoração em projetos Unity? Mapeia-se a frequência absoluta e relativa dos termos de busca (Tabela 3.2) identificados nas mensagens de *commit*. Essa análise quantifica e identifica quais operações de refatoração são mais recorrentes no histórico de versionamento dos repositórios que compõem o conjunto de dados.

(QP2) Quais categorias de arquivos são modificadas em commits de refatoração em projetos Unity? Determina-se a proporção de *commits* que modificam arquivos pertencentes às quatro categorias arquiteturais da *engine* (Seção 3.3). Para avaliar o acoplamento entre as categorias, quantifica-se a ocorrência de commits híbridos por meio de uma matriz de co-ocorrência, evidenciando o nível de acoplamento estrutural durante as tarefas de refatoração.

Para o processamento dos dados e geração dos gráficos, utilizam-se as bibliotecas `Pandas` e `Seaborn` na linguagem Python. Os dados são sumarizados por meio de contagens absolutas, distribuições percentuais e frequências de interceptação de categorias, descrevendo de forma direta o comportamento do conjunto de dados minerado.

Análise Qualitativa. Nesta etapa, realiza-se uma análise qualitativa e exploratória dos *commits* previamente classificados como refatoração, visando responder a terceira questão de pesquisa:

(QP3) Existem indícios de refatorações específicas da engine Unity? Em um nível mais alto de abstração, busca-se identificar indícios de padrões e práticas recorrentes no desenvolvimento de jogos e aplicações de Realidade Virtual com a *engine* Unity, com ênfase em possíveis refatorações específicas desse contexto.

Dada a impossibilidade de analisar manualmente todo o volume de commits minerados, para responder à terceira questão de pesquisa, adota-se uma amostragem intencional (*purposive sampling*) de caráter exploratório (PATTON, 2007). Uma amostra aleatória de commits é selecionada com base nos seguintes critérios: (i) existe uma mensagem de commit clara e descritiva, que sugere a intenção do desenvolvedor; (ii) existem modificações que envolvem artefatos da arquitetura Unity (commits classificados com outras categorias além de Código C#); e (iii) existem evidências de alterações estruturais com potencial impacto na manutenibilidade, organização ou performance do projeto no contexto do Unity.

4 Resultados

Este capítulo apresenta os resultados obtidos neste estudo, sua discussão e as principais ameaças à validade da pesquisa.

4.1 Caracterização do Conjunto de Dados

Nesta seção, apresenta-se o conjunto de dados utilizado neste estudo, seguindo as etapas definidas na metodologia. Primeiramente, os repositórios são caracterizados quanto à sua popularidade. Na sequência, descrevem-se os *commits* minerados e os filtros aplicados para a construção do conjunto de dados final.

4.1.1 Caracterização dos Repositórios de Software

A partir do processo de unificação entre a base de projetos da literatura e a busca ampla (3.2), o conjunto de repositórios inicial totalizou 94 repositórios. Após a aplicação dos critérios de filtragem estabelecidos, 42 repositórios foram removidos. Dessa forma, o conjunto de repositórios final, utilizado para as análises deste trabalho, é composto por 52 repositórios.

A Figura 4.1 mostra um exemplo de repositório que atende aos critérios de seleção. Trata-se do `Open Brush`,[1] um jogo popular de realidade virtual voltado à produção artística em ambientes tridimensionais, com aproximadamente 1.100 estrelas e 850 commits.

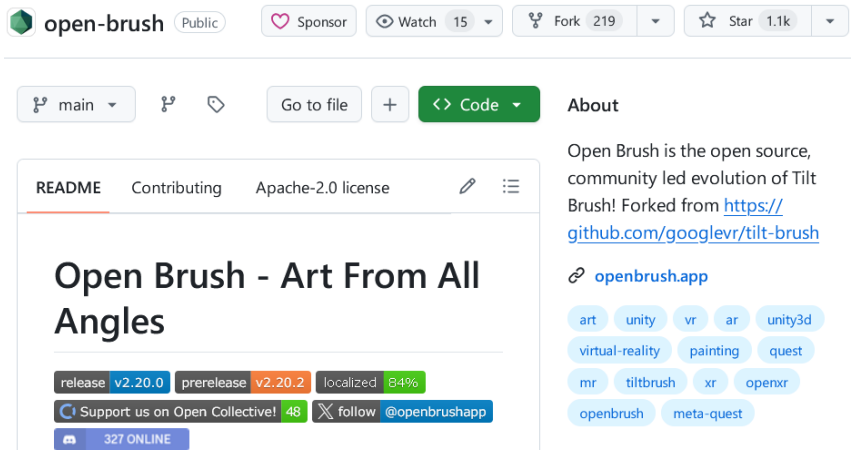
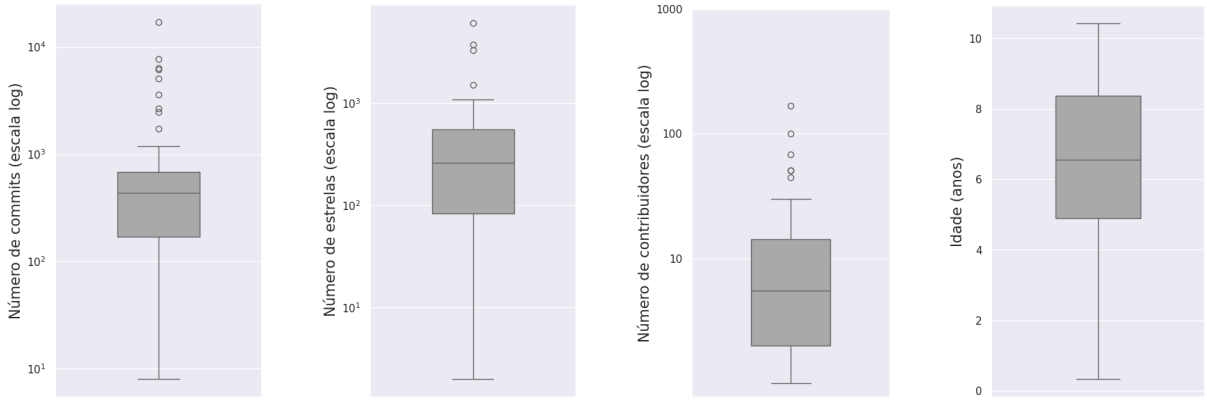


Figura 4.1 – Exemplo de repositório Unity - Open Brush

Conforme apresentado na Figura 4.2a, o número de *commits* por projeto varia entre 8 e 17.118, sendo a mediana igual a 433,5. Em relação ao número de estrelas, os valores variam entre 2 e 6.069, sendo a mediana igual a 261 (Figura 4.2b). Aproximadamente 75% dos projetos possuem até 551 estrelas (terceiro quartil). A Figura 4.2c mostra a distribuição do número de contribuidores, cujos valores variam entre 1 e 168, com mediana igual a 5,5, primeiro quartil igual a 2 e o terceiro quartil com 14,25. Por fim, a Figura 4.2d apresenta a distribuição da idade dos projetos, com valores variando entre 0,33 e 10,42 anos.

[1] <https://github.com/icosa-foundation/open-brush>



(a) *Commits* (b) Estrelas (c) Contribuidores (d) Idade

Figura 4.2 – Caracterização dos projetos GitHub

4.1.2 Caracterização dos Commits

O conjunto de repositórios selecionados totaliza 68.317 *commits* em seu histórico de versionamento. A partir desse volume geral, a etapa de mineração (Seção 3.3) recuperou um conjunto inicial de 4.674 *commits* candidatos a atividades de refatoração.

Filtragem dos Commits. A aplicação dos critérios de filtragem (Seção 3.3) resultou na remoção de 807 *commits* (17,27% do volume minerado inicial). Dentre as exclusões, 532 (11,38%) correspondem a registros sem arquivos alterados, 273 (5,84%) a modificações exclusivas em artefatos não relacionados a código-fonte e 2 (0,04%) a mensagens de *merge*. Após a limpeza, restaram 3.867 *commits* no *dataset*, totalizando 82,73% dos registros minerados originais que atendem aos critérios de inclusão definidos para o estudo. A Tabela 4.1 resume a distribuição da filtragem.

Tabela 4.1 – Resumo da filtragem de *commits* minerados

Métrica / Motivo	**Quantidade**	**Percentual %**
Total de *commits* minerados	4.674	100,00
Total de *commits* removidos	807	17,27
↪ Sem arquivos alterados	532	11,38
↪ Apenas artefatos não-código	273	5,84
↪ *Merge commit*	2	0,04
Total de *commits* retidos	**3.867**	**82,73**

Categorização dos Commits. Após a etapa de filtragem, os 3.867 *commits* retidos foram classificados de forma não exclusiva, permitindo capturar o acoplamento entre as categorias. A Tabela 4.2 apresenta a frequência absoluta e a proporção de *commits* que contêm arquivos de cada categoria. Os dados indicam uma predominância da categoria *Código C#*, presente em 91,85% dos *commits* analisados. Em seguida, a categoria *Estrutura Visual e Interface* figura como a segunda mais frequente, ocorrendo em 27,00% das operações, enquanto as modificações em *Ativos e Dados* estão presentes em 15,65%. As categorias *Código Específico de Domínio* e *Outros* apresentam as menores incidências no *dataset*, representando 5,09% e 1,16% do total, respectivamente.

Tabela 4.2 – Distribuição dos *commits* por categoria de classificação

Categoria	Ocorrências	Percentual (%)
Código C#	3.552	91,85
Estrutura Visual e Interface	1.044	27,00
Ativos e Dados	605	15,65
Código Específico de Domínio	197	5,09
Outros	45	1,16

4.2 Questões de Pesquisa

Nesta seção, são apresentados e discutidos os resultados obtidos ao longo da investigação, organizados de forma a responder a cada uma das questões de pesquisa propostas.

4.2.1 (QP1) Qual é a frequência de *commits* relacionados a atividades de refatoração em projetos Unity?

Para responder a esta questão de pesquisa, avaliou-se a proporção de commits que incluem palavras-chaves que remetem a atividades de refatoração. Dentre os 68.317 commits analisados, constata-se que 3.867 (5,66%) deles estão associados a práticas de refatoração, evidenciando a ocorrência contínua dessa atividade no ciclo de desenvolvimento.

A Tabela 4.3 mostra os resultados, detalhando a frequência de cada palavra-chave. Como pode-se observar, as tarefas de refatoração estão majoritariamente concentradas na reorganização e mudanças na nomenclatura dos artefatos. Por exemplo, o termo *Mov** é o mais frequente, sendo encontrado em 1.841 *commits* (47,61%) de 42 projetos distintos. Em seguida, destacam-se as palavra-chave *Renam** com 994 ocorrências (25,70%) e o termo geral *Refactor** com 925 ocorrências (23,92%).

Tabela 4.3 – Distribuição dos termos de busca nos *commits* de refatoração

Termo de busca	Projetos distintos	Commits	%
*Mov**	42	1.841	47,61%
*Renam**	36	994	25,70%
*Refactor**	39	925	23,92%
*Split**	26	127	3,28%
*Extract**	17	123	3,18%
*Reorganiz**	16	51	1,32%
*Rewrit**	8	43	1,11%
Outros	26	84	2,17%

Práticas que indicam extração ou divisão de responsabilidades (*Split** e *Extract**) apresentaram frequências menores, 3,28% e 3,18%, respectivamente. O termo *Reorganiz** registrou 51 ocorrências (1,32%). Um exemplo desta categoria pode ser observado em um commit do projeto `MixedRealityToolkit-Unity` da Microsoft, onde um desenvolvedor reorganiza uma cena para agrupar elementos soltos dentro de contêineres parentais.[2]

[2] <https://github.com/microsoft/MixedRealityToolkit-Unity/commit/8640cb4>

Os demais termos pesquisados (*inlin**, *restructur**, *redesign**, *simplify* code*, *pull* up*, *repackag** e *push* down*) apresentaram baixas ocorrências, sendo agrupados na categoria *outros*.

A Figura 4.3 mostra um exemplo de commit com a palavra-chave *refactor*. Neste caso, o desenvolvedor realizou mudanças para desacoplar funcionalidades dentro de um *Prefab*. [3]

Figura 4.3 – Exemplo de commit com palavra-chave *refactor*

Resumo: Na *engine* Unity, *commits* que remetem a atividades de refatoração estão relacionados principalmente aos termos `move` (1.841 ocorrências, 47.61%) e `rename` (994 ocorrências, 25.7%). Existem também uma taxa significativa de *commits* que mencionam explicitamente atividades de refatoração (925 ocorrências, 23.92%).

4.2.2 (QP2) Quais categorias de arquivos são modificadas em *commits* de refatoração em projetos Unity?

Para responder a esta questão, analisou-se a distribuição e a interdependência dos arquivos modificados durante as operações de refatoração. Os resultados revelam que, embora as alterações ocorram predominantemente na lógica do sistema (*Código C#*), elas frequentemente exigem modificações nas estruturas da *engine* (*Estrutura Visual e Interface*, *Ativos e Dados*, e *Código Específico de Domínio*).

Para quantificar esse acoplamento arquitetural, foi gerada uma matriz de co-ocorrência (Figura 4.4), que detalha os *commits* híbridos, ou seja, que alteram arquivos de múltiplas categorias simultaneamente. A análise quantitativa demonstra que das 1.044 ocorrências envolvendo a *Estrutura Visual e Interface* (como *Scenes* e *Prefabs*), 850 coincidem com *Código C#*, indicando que em 81,4% dos casos as refatorações visuais são acompanhadas de alterações em *scripts*. Dinâmica semelhante ocorre em *Ativos e Dados* (como *ScriptableObjects* e *Assembly Definitions*), onde em 529 das 605 ocorrências (87,4%) também ocorrem modificações no código. Esses percentuais evidenciam que a manutenção de artefatos declarativos e serializados exige, em sua maioria, a modificação da lógica C#.

A categoria *Código Específico de Domínio*, que engloba *shaders* e *scripts* de interoperabilidade externa, apresenta um padrão similar de interdependência. Das 197 ocorrências registradas, 163 (82,7%) intersectam com *Código C#* e 136 (69,0%) com componentes visuais. Esse acoplamento reflete a necessi-

[3] <https://github.com/ExtendRealityLtd/Zinnia.Unity/commit/fd9a165>

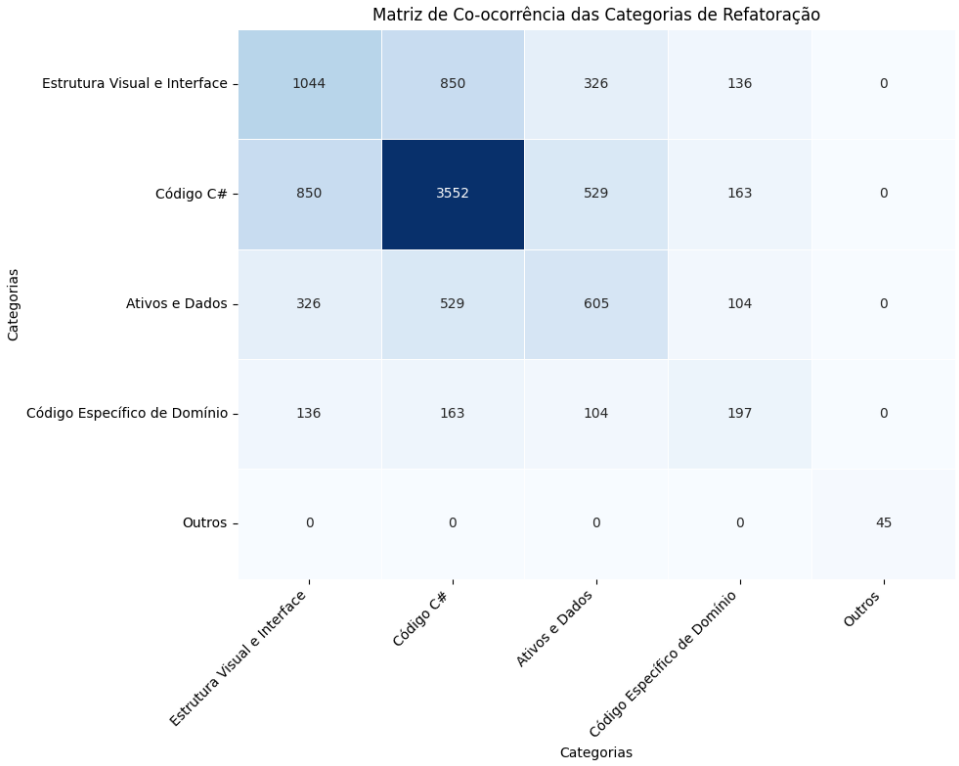


Figura 4.4 – Matriz de co-ocorrência das categorias de refatoração

dade técnica de atualizar as classes C# que invocam o processamento externo e as hierarquias visuais que o renderizam sempre que o código de domínio é refatorado.

O acoplamento fora do código C# é evidenciado na interseção de 326 ocorrências entre *Estrutura Visual e Interface* e *Ativos e Dados*. Esse cruzamento demonstra que a reestruturação de parâmetros globais ou de mapeamentos (*Input Actions*) ocorre de forma simultânea à alteração das hierarquias na cena em mais da metade (53,8%) das operações que envolvem ativos de dados.

Em contrapartida, a categoria *Código C#* concentra o maior volume absoluto de modificações isoladas. Com um total de 3.552 ocorrências e interseções proporcionalmente baixas com os artefatos da *engine*, os dados comprovam que a maioria das refatorações aplicadas puramente à lógica é executada de maneira estrita, sem propagar alterações para as estruturas visuais, de dados ou de domínio mapeadas.

Por fim, a categoria *Outros* contabilizou 45 ocorrências isoladas. Esse isolamento se deve à própria definição da categoria na metodologia, que engloba apenas *commits* formados exclusivamente por extensões que não se enquadram nas categorias principais da *engine*.

A Figura 4.5 mostra um exemplo, onde um desenvolvedor modifica arquivos de três categorias simultaneamente: *Código Específico de Domínio* (`.shader`), *Estrutura Visual e Interface* (`.unity`) e *Ativos e Dados* (`.asset`). [4]

[4] <https://github.com/Unity-Technologies/InputSystem/commit/f57279b>

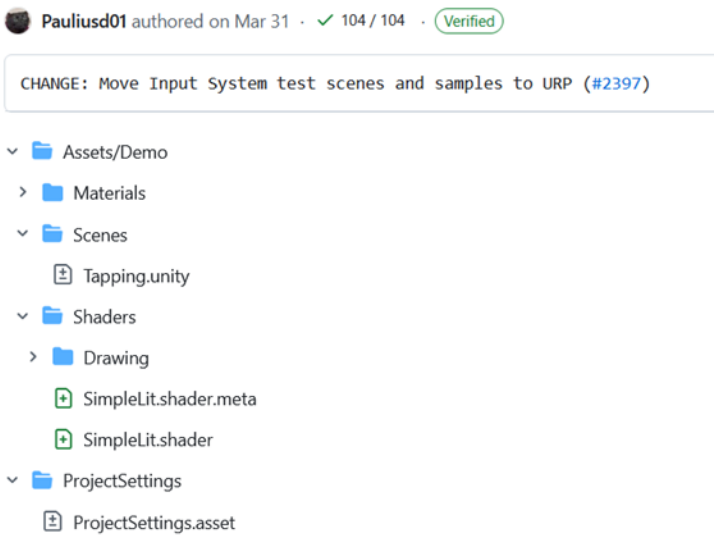


Figura 4.5 – Exemplo de *commit* com modificações em múltiplas categorias arquiteturais

Resumo: As refatorações em projetos Unity concentram-se principalmente em *Código C#* (3.552 ocorrências), porém frequentemente exigem modificações simultâneas em artefatos da *engine*. Destacam-se as co-ocorrências entre *Código C#* e *Estrutura Visual e Interface* (850 ocorrências, 81,4%) e entre *Código C#* e *Ativos e Dados* (529 ocorrências, 87,4%), evidenciando o forte acoplamento entre a lógica da aplicação e os componentes estruturais do Unity durante atividades de refatoração.

4.2.3 (QP3) Existem indícios de refatorações específicas da *engine* Unity?

Para responder a esta questão, uma amostra aleatória de *commits* foi selecionada para inspeção manual. O objetivo é verificar a ocorrência refatorações clássicas e mudanças específicas do domínio Unity. As mensagens de *commit* e as modificações nos arquivos (*diffs*) foram analisadas para identificar a intenção das alterações e seu papel na arquitetura dos projetos. Nos parágrafos a seguir, discutem-se os casos encontrados que sugerem a existência de atividades de refatoração.

Refatoração de Cena e Prefabs. Esta categoria envolve alterações na organização de *GameObjects* e *Prefabs* no Unity Editor. Observa-se a divisão de responsabilidades entre diferentes *GameObjects*. Um exemplo dessa prática foi identificado no repositório `Zinnia.Unity`[5], onde três *scripts* de modificação (posição, rotação e escala) foram removidos de um nó raiz e transferidos para três novos *GameObjects* filhos. Essa reestruturação permite a ativação e desativação de comportamentos isoladamente pelo Editor.

Outras modificações nesta categoria sugerem a padronização e a legibilidade da cena. No projeto `Mixed RealityToolkit-Unity` (MRTK)[6], por exemplo, identificou-se o agrupamento de elementos soltos na raiz da cena para dentro de contêineres parentais lógicos. No mesmo *commit*, nomes gerados automaticamente, como `SectionTitle (2)`, foram renomeados para identificadores descritivos, como `InteractableSectionTitle`, aplicando o princípio de renomeação para revelar intenção. Adicionalmente, identificou-se a movimentação interna de nós filhos entre diferentes objetos pais no Zinnia.Unity[7]

[5] <https://github.com/ExtendRealityLtd/Zinnia.Unity/commit/fd9a165>
[6] <https://github.com/microsoft/MixedRealityToolkit-Unity/commit/8640cb4>
[7] <https://github.com/ExtendRealityLtd/Zinnia.Unity/commit/1fbaf93>

para garantir a padronização de múltiplos *Prefabs*.

Refatoração de Shaders. Esta categoria reúne alterações na organização do código de *shaders*. Os exemplos encontrados sugerem mudanças para reduzir a complexidade estrutural por meio da eliminação de código duplicado no código de renderização. A inspeção manual desses commits indicou a extração de blocos lógicos específicos para novos arquivos dedicados. No repositório `UniVRM`[8], por exemplo, a lógica de *normal mapping* foi removida do corpo principal de um *shader* e isolada em uma nova função dedicada dentro de um arquivo `.hlsl` independente. Em outro caso, no repositório `EditorXR`[9], cálculos matemáticos duplicados de vértices e recortes (*clipping*) foram extraídos de três *shaders* distintos e centralizados em um único arquivo de inclusão compartilhado (`.cginc`). Ambas as práticas reduzem a duplicação de código e concentram a manutenção em módulos compartilhados (Unity Technologies, 2025).

Arquitetura de Compilação e Dependências. Esta categoria abrange ações envolvendo o compilador do Unity para otimizar os tempos de compilação e reduzir o acoplamento. A gestão de dependências exige modificações em arquivos de definição (`.asmdef`). Observou-se a limpeza de referências não utilizadas nestes arquivos, como evidenciado no UniVRM[10], onde dependências obsoletas foram removidas para desacoplar módulos em nível de compilação. Em outro exemplo, no ViveInputUtility[11], um arquivo `.asmdef` isolado foi deletado e substituído por uma referência (`.asmref`). Isso instrui o Unity a fundir a unidade fragmentada ao *assembly* primário, reduzindo o número de unidades de compilação e simplificando o grafo de dependências (Unity Technologies, 2025).

Arquitetura de Dados. Esta categoria reúne alterações na organização de configurações e metadados serializados do projeto. Identificou-se a separação de dados genéricos e específicos em diferentes arquivos *ScriptableObject*. No MRTK[12], campos de configuração serializados (como modelos de mãos padrão e *render controllers*) foram removidos de um perfil de configuração global e movidos para um perfil específico de mapeamento de controles. Essa prática corresponde à refatoração de *Move Field* na arquitetura de dados do Unity.

Organização do Editor. Esta categoria reúne alterações voltadas à organização da interface do Unity Editor. Foi identificado um indício dessa prática na alteração de *strings* dentro de arquivos `.shader`.[13] No MRTK, foram adicionados espaços à *string* que define o caminho do *shader* para reorganizar os materiais nos menus do *Inspector*, sem alterar o comportamento da aplicação em tempo de execução.

Resumo: A inspeção manual dos *commits* identificou indícios de refatorações específicas da *engine* Unity, agrupadas em cinco categorias: *Refatoração de Cena e Prefabs*, *Refatoração de Shaders*, *Arquitetura de Compilação e Dependências*, *Arquitetura de Dados* e *Organização do Editor*. Essas práticas evidenciam que atividades de refatoração na Unity abrangem não apenas o código-fonte, mas também artefatos e mecanismos próprios da *engine*.

[8] <https://github.com/vrm-c/UniVRM/commit/fd52ae2>
[9] <https://github.com/Unity-Technologies/EditorXR/commit/f979044>
[10] <https://github.com/vrm-c/UniVRM/commit/71fb834>
[11] <https://github.com/ViveSoftware/ViveInputUtility-Unity/commit/40274b5>
[12] <https://github.com/microsoft/MixedRealityToolkit-Unity/commit/867f63c>
[13] <https://github.com/microsoft/MixedRealityToolkit-Unity/commit/80031d9>

4.3 Discussões e Implicações

Os resultados apresentados nas seções anteriores permitem discutir os principais desafios para o desenvolvimento de ferramentas de detecção de refatorações voltadas ao ecossistema Unity. Por exemplo, as análises de acoplamento arquitetural ($QP2$) e das práticas específicas de domínio ($QP3$) sugerem que abordagens baseadas exclusivamente na análise de código-fonte tendem a ser insuficientes para detectar refatorações realizadas em projetos Unity, uma vez que parte das mudanças relevantes envolve artefatos e mecanismos próprios da *engine*.

Com base nessas evidências, discute-se o projeto de uma ferramenta de detecção de refatorações para Unity, inspirada no RefactoringMiner (TSANTALIS et al., 2018; TSANTALIS; KETKAR; DIG, 2020; ALIKHANIFARD; TSANTALIS, 2025), composta pelos seguintes componentes:

Extração de árvore de sintaxe abstrata (AST) via Roslyn. Ferramentas consolidadas na literatura utilizam a comparação de ASTs para detectar alterações estruturais (TSANTALIS; KETKAR; DIG, 2020). Para projetos Unity, o suporte à linguagem C# pode ser implementado por meio da plataforma de compilação .NET, o Roslyn (MICROSOFT, 2025). Suas APIs permitem extrair a AST e o modelo semântico do código, possibilitando a identificação de refatorações tradicionais da orientação a objetos, como extração de métodos e movimentação de atributos, realizadas nos arquivos `.cs`.

Análise de arquivos declarativos e serializados (YAML). Conforme demonstrado pela matriz de co-ocorrência, alterações visuais acompanham modificações em scripts em 81,4% dos casos. Como os artefatos da *engine* (`.prefab`, `.unity` e `.asset`) utilizam uma representação textual baseada em YAML (Unity Technologies, 2025), a ferramenta deve incluir um *parser* para esses arquivos. Esse componente deve extrair os Identificadores Únicos Globais (*GUIDs*) e reconstruir a hierarquia de *GameObjects* e seus componentes.

Correlação entre código e artefatos da *engine*. Este componente relaciona as informações obtidas pelo Roslyn com a estrutura reconstruída a partir dos arquivos YAML. Dessa forma, torna-se possível detectar refatorações específicas do domínio. Por exemplo, se a análise da AST indicar a extração de lógica de uma classe para novas classes e, simultaneamente, a análise dos arquivos YAML identificar novos *GameObjects* associados a essas classes, a ferramenta poderá classificar essa alteração como uma refatoração de cena e *prefabs*, em vez de apenas uma movimentação de código.

Regras de detecção específicas do domínio. A ferramenta deve incorporar regras adicionais para identificar as demais categorias observadas na análise qualitativa. Para tanto, sugere-se também a criação de catálogos de refatoração específicos do domínio, semelhante a estudos envolvendo tecnologias emergentes (VEGI; VALENTE, 2023; VEGI; VALENTE, 2025).

Arquivos de dados e compilação. Detectar alterações em classes derivadas de *ScriptableObject* associadas à reorganização de parâmetros serializados nos arquivos `.asset` correspondentes. Da mesma forma, sugere-se que a ferramenta seja capaz de analisar modificações em arquivos de definição (`.asmdef` e `.asmref`) para identificar alterações nas dependências entre módulos (Unity Technologies, 2025).

Refatoração de shaders. Por fim, sugere-se também a utilização de *parsers* específicos ou análise textual para detectar a extração de blocos de processamento de arquivos `.shader` para arquivos `.cginc`.

Em síntese, os resultados desta pesquisa indicam que os padrões de refatoração observados podem orientar o desenvolvimento de ferramentas de detecção voltadas ao ecossistema Unity. Para isso, essas ferramentas devem combinar a análise sintática e semântica do código C# com a análise dos artefatos específicos da *engine*, permitindo identificar refatorações que envolvem múltiplos tipos de arquivos e que não seriam detectadas por abordagens baseadas exclusivamente no código-fonte.

4.4 Ameaças à Validade

Os parágrafos a seguir descrevem as principais ameaças à validade deste estudo, destacando as estratégias utilizadas para mitigá-las.

Generalização dos resultados. Como é usual em estudos empíricos em Engenharia de Software, os resultados deste trabalho não podem ser generalizados para outros contextos. O estudo analisou exclusivamente repositórios públicos hospedados no GitHub, desenvolvidos com Unity e classificados com tópicos relacionados à Realidade Virtual, Realidade Aumentada e Realidade Estendida (VR/AR/XR). Dessa forma, os resultados não podem ser generalizados para projetos voltados a outras plataformas (ex., aplicações móveis sem suporte a VR ou jogos para consoles), projetos de código fechado ou para projetos desenvolvidos em outras *engines* (ex., Unreal Engine e Godot). Entretanto, o conjunto de dados é composto por projetos populares, selecionados com base no número de estrelas do GitHub, uma métrica amplamente utilizada como indicador de relevância e reconhecimento pela comunidade (BORGES; VALENTE, 2018). A mediana dos projetos selecionados é de 261 estrelas, mantidos por equipes com uma mediana de 5,5 contribuidores. Além disso, o trabalho reutiliza e expande conjuntos de dados existentes. Especificamente, reutilizam-se 14 projetos selecionados em estudos anteriores (RZIG et al., 2023), e 38 novos projetos foram adicionados. Finalmente, a inspeção manual de commits para a terceira questão de pesquisa foi realizada por amostragem e consiste em uma análise preliminar, realizada pelo primeiro autor deste trabalho. Entretanto, os resultados iniciais mostram a necessidade de ferramentas para detecção de refatoração, bem como catálogos específicos de domínio (VEGI; VALENTE, 2025; TSANTALIS et al., 2018)

Seleção de repositórios. Para selecionar os repositórios candidatos, utilizou-se como critério os tópicos definidos pelos próprios desenvolvedores no GitHub. Entretanto, a liberdade de anotação pode resultar em uma taxonomia não padronizada, com rótulos ambíguos, excessivamente específicos ou inconsistentes (SAS et al., 2023). Dessa forma, a ausência de padronização e a baixa adoção dos tópicos implicam perda de cobertura, uma vez que projetos relevantes podem ser excluídos caso não utilizem as marcações exigidas. Neste estudo, a seleção baseia-se em termos populares (como `vr`, `virtual-reality` e `ar`), alguns deles previamente utilizados na literatura para identificação de projetos relacionados a tecnologias imersivas (RZIG et al., 2023).

Identificação de refatorações em commits. Neste estudo, buscou-se termos que remetem à atividades de refatoração nas mensagens dos commits. Entretanto, as refatorações podem não ser documentadas de forma explícita nos commits (WEISSGERBER; DIEHL, 2006; PARNIN; BLACK; MURPHY-HILL, 2012). Além disso, o termo pode ser usado para atividades de manutenção gerais (VALENTE, 2022), o que pode reduzir a cobertura da abordagem e introduzir falsos positivos e negativos no conjunto de dados. Para mitigar esta ameaça, aplicou-se uma etapa de filtragem estrutural (Seção 3.3), removendo artefatos

não relacionados a código-fonte, arquivos de metadados (`.meta`) e diretórios de dependências externas. Adicionalmente, o conjunto de termos utilizados foram reutilizados de pesquisas anteriores (ALOMAR et al., 2022).

Classificação das categorias arquiteturais. A classificação de commits nas quatro categorias arquiteturais (Tabela 3.4) utiliza apenas a extensão do arquivo como critério. A extensão `.asset`, por exemplo, é empregada pela *engine* Unity para serializar tanto *ScriptableObjects* (que encapsulam dados e regras de negócio) quanto *Render Pipeline Assets* (que definem configurações gráficas) (Unity Technologies, 2025). O *script* classificador atribui ambas as ocorrências à categoria Ativos e Dados. Como consequência, as proporções de commits por categoria (Tabela 4.2) e a matriz de co-ocorrência (Figura 4.4) devem ser interpretadas como aproximações, não podendo ser generalizadas para outros contextos. Entretanto, a análise considera um conjunto de dados significativo, com centenas de commits, e filtros baseados em estudos anteriores.

5 Considerações Finais

Este trabalho teve como objetivo caracterizar as práticas de refatoração em aplicações desenvolvidas em C# utilizando a *engine* Unity. A metodologia combinou a mineração de repositórios de *software*, a análise quantitativa do histórico de versionamento e a análise qualitativa de *commits* em projetos Unity, visando identificar e documentar refatorações específicas desse domínio.

Os *scripts* desenvolvidos e os artefatos obtidos neste trabalho, incluindo os dados brutos e processados em formato `.csv`, encontram-se publicamente disponíveis na plataforma GitHub: <https://github.com/Quorthon13/unity-refactoring-empirical-study>

5.1 Síntese dos Resultados

A análise quantitativa ($QP1$) demonstrou a frequência das práticas de refatoração em projetos Unity. As refatorações corresponderam a 5,66% das modificações analisadas e, após a etapa de filtragem, obteve-se um conjunto final de 3.867 *commits* associados a essas operações.

A análise das categorias de arquivos ($QP2$) revelou o acoplamento entre o código-fonte e os artefatos da *engine* por meio de uma matriz de co-ocorrência. Em 81,4% das refatorações envolvendo componentes visuais e de interface, também ocorreram alterações em arquivos C#, indicando que mudanças na estrutura visual geralmente são acompanhadas por alterações na lógica da aplicação.

A análise qualitativa ($QP3$) identificou cinco categorias de refatorações específicas do domínio Unity: Refatoração de Cena e *Prefabs*, Refatoração de *Shaders*, Arquitetura de Compilação, Arquitetura de Dados e Organização do Editor.

Com base nesses resultados, foram discutidos os requisitos para uma ferramenta de detecção automatizada de refatorações em projetos Unity. Os resultados indicam que a análise apenas do código-fonte é insuficiente para detectar todas as refatorações observadas. Para isso, a ferramenta deve combinar a análise da árvore de sintaxe abstrata (AST) de C# por meio do Roslyn com a análise dos arquivos declarativos baseados em YAML, permitindo relacionar alterações no código-fonte às modificações nos artefatos da *engine*.

5.2 Contribuições do Trabalho

Esta pesquisa contribui para a área de Engenharia de Software ao ampliar o conhecimento sobre práticas de refatoração em projetos Unity e habilitar o desenvolvimento de ferramentas de apoio à manutenção de software. As principais contribuições são.

- **Caracterização de refatorações específicas do domínio Unity.** O trabalho identifica e caracteriza práticas de refatoração específicas do ecossistema Unity, um aspecto ainda pouco explorado na

literatura. A análise resultou em cinco categorias de refatorações relacionadas à organização de cenas e *prefabs*, *shaders*, arquitetura de compilação, arquitetura de dados e organização do editor.

- **Caracterização do acoplamento entre código e artefatos da *engine*.** Evidências de que refatorações em projetos Unity frequentemente envolvem alterações coordenadas entre o código-fonte e os artefatos específicos da plataforma.
- **Arquitetura para detecção automatizada de refatorações em Unity.** Proposta dos requisitos e dos componentes necessários para uma ferramenta capaz de detectar refatorações considerando tanto o código C# quanto os arquivos declarativos da *engine*.
- **Metodologia de mineração para projetos Unity.** Definição de um processo de mineração de repositórios adaptado às características do ecossistema Unity, incluindo critérios para seleção de projetos e filtragem de alterações não relacionadas à refatoração.
- **Conjunto de projetos para pesquisa.** Atualização de um conjunto de repositórios públicos de aplicações Unity voltadas à Realidade Virtual, Realidade Aumentada e Realidade Estendida (VR, AR e XR) (RZIG et al., 2023), passível de reutilização em estudos futuros.
- **Caracterização quantitativa das práticas de refatoração.** Quantificação da frequência de refatorações em projetos Unity a partir da análise de um conjunto de repositórios públicos.

5.3 Trabalhos Futuros

Como trabalhos futuros, propõem-se as seguintes atividades:

- **Catálogo de refatorações:** Documentar formalmente as refatorações específicas do ecossistema Unity, descrevendo a mecânica e a motivação de cada operação identificada.
- **Implementação de ferramenta de detecção:** Desenvolver o protótipo da ferramenta discutida neste trabalho, integrando o compilador Roslyn e um *parser* YAML.
- **Análise em outras *engines*:** Replicar a metodologia em projetos de outros motores gráficos, como a Unreal Engine, para comparar os padrões de evolução de *software*.

Declaração de Uso de Inteligência Artificial Generativa

Durante a preparação deste documento, eu, Diego Demétrio Santos Rodrigues, estudante de graduação em Ciência da Computação da Universidade Federal de Ouro Preto, declaro o uso de IAGen Gemini versão 3.1 Pro para revisão textual e geração de códigos LaTeX.

Após o uso desta ferramenta, eu revisei e editei o conteúdo em conformidade com os princípios éticos para uso de IAGen (Resolução CONPEP nº 144) e com os acordos estabelecidos com a orientadora desta pesquisa. Dessa forma, assumo total responsabilidade pelo conteúdo da publicação.

Referências

AL-ASWAD, A.; IANNANTUONI, A.; ZHANG, Y.; ZHU, Y. *Unity Engine: A Unicorn Powering the Video Game and VR/AR Economy*. 2022. <https://d3.harvard.edu/platform-digit/submission/unity-engine-a-unicorn-powering-the-video-game-and-vr-ar-economy/>. Online: acesso em 19 Jun. 2025.

ALIKHANIFARD, P.; TSANTALIS, N. A novel refactoring and semantic aware abstract syntax tree differencing tool and a benchmark for evaluating the accuracy of diff tools. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 34, n. 2, jan. 2025. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3696002>.

ALOMAR, E. A.; PERUMA, A.; MKAOUER, M. W.; NEWMAN, C. D.; OUNI, A. An exploratory study on refactoring documentation in issues handling. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2022. (MSR '22), p. 107–111. ISBN 9781450393034. Disponível em: <https://doi.org/10.1145/3524842.3528525>.

BORGES, H.; HORA, A.; VALENTE, M. T. Understanding the factors that impact the popularity of github repositories. In: IEEE. *2016 IEEE international conference on software maintenance and evolution (ICSME)*. [S.l.], 2016. p. 334–344.

BORGES, H.; VALENTE, M. T. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, Elsevier, v. 146, p. 112–129, 2018.

BRITO, A.; HORA, A.; VALENTE, M. T. Characterizing refactoring graphs in Java and JavaScript projects. *Empirical Software Engineering*, v. 26, 2021.

BRITO, A.; HORA, A.; VALENTE, M. T. Towards a catalog of composite refactorings. *Journal of Software: Evolution and Process*, v. 1, p. 1–43, 2023.

BRITO, A. N. de. *Refactoring graphs: reasoning about refactoring over time*. Tese (Doctoral thesis) — Universidade Federal de Minas Gerais, Brasil, 03 2023. Advisor: Marco Tulio de Oliveira Valente. Co-advisor: André Cavalcante Hora. Disponível em: <https://hdl.handle.net/1843/56371>.

BRITO, R.; VALENTE, M. T. Refdiff4go: Detecting refactorings in go. In: *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse*. New York, NY, USA: Association for Computing Machinery, 2020. (SBCARS '20), p. 101–110. ISBN 9781450387545. Disponível em: <https://doi.org/10.1145/3425269.3425274>.

BRITO, R.; VALENTE, M. T. RAID - Refactoring aware and intelligent diffs. In: *29th International Conference on Program Comprehension (ICPC)*. [S.l.: s.n.], 2021. p. 265–275.

COELHO, J.; VALENTE, M. T.; MILEN, L.; SILVA, L. L. Is this GitHub project maintained? measuring the level of maintenance activity of open-source projects. *Information and Software Technology*, v. 1, p. 1–35, 2020.

COURA, L.; GONZAGA, J.; BIANCHI, A. G. C.; SILVA, R. I.; GERTRUDES, J. C.; FORTES, R.; DELABRIDA, S. Treinando professores em design de jogos em realidade virtual usando design participatório: um relato de experiência. In: *XXIV Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais (IHC)*. [S.l.: s.n.], 2025. p. 106–129.

FALLERI, J.-R.; MORANDAT, F.; BLANC, X.; MARTINEZ, M.; MONPERRUS, M. Fine-grained and accurate source code differencing. In: *Proceedings of the 29th ACM/IEEE International Conference on*

Automated Software Engineering. New York, NY, USA: Association for Computing Machinery, 2014. (ASE '14), p. 313–324. ISBN 9781450330138. Disponível em: <https://doi.org/10.1145/2642937.2642982>.

Fortune Business Insights. *Virtual Reality (VR) Market Size, Growth, Share | Report, 2034*. 2026. Report ID: FBI101378. Disponível em: <https://www.fortunebusinessinsights.com/industry-reports/virtual-reality-market-101378>.

FOWLER, M. *Refactoring: Improving the Design of Existing Code*. [S.l.]: Addison-Wesley, 1999.

FOWLER, M. *Refactoring: Improving the Design of Existing Code*. 2nd. ed. Boston, MA: Addison-Wesley Professional, 2018. (Addison-Wesley Signature Series (Fowler)). ISBN 978-0134757599.

GHRAIRI, N.; KPODJEDO, S.; BARRAK, A.; PETRILLO, F.; KHOMH, F. The state of practice on virtual reality (vr) applications: An exploratory study on github and stack overflow. In: *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. [S.l.: s.n.], 2018. p. 356–366.

GITHUB. *Pull Requests*. 2025. <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-comparing-branches-in-pull-requests#diff-view-options>. Online: acesso em 25 Out. 2025.

KALLIAMVAKOU, E.; GOUSIOS, G.; BLINCOE, K.; SINGER, L.; GERMAN, D. M.; DAMIAN, D. The promises and perils of mining github. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2014. (MSR 2014), p. 92–101. ISBN 9781450328630. Disponível em: <https://doi.org/10.1145/2597073.2597074>.

LAVALLE, S. M. *Virtual Reality*. Hardcover. Cambridge: Cambridge University Press, 2024. 392 p. ISBN 978-1107198937.

MICROSOFT. *The .NET Compiler Platform SDK (Roslyn)*. 2025. <https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/>. Online: acesso em 01 Nov. 2025.

MYERS, E. W. Ano(nd) difference algorithm and its variations. *Algorithmica*, Springer-Verlag, Berlin, Heidelberg, v. 1, n. 1–4, p. 251–266, nov. 1986. ISSN 0178-4617. Disponível em: <https://doi.org/10.1007/BF01840446>.

Niantic. *Reality Blending and PokéStop Scanning Come to Pokémon GO*. 2020. Disponível em: <https://nianticlabs.com/news/realityblending-announcement>. Acesso em: 7 fev. 2026.

NUSRAT, F.; HASSAN, F.; ZHONG, H.; WANG, X. How developers optimize virtual reality applications: A study of optimization commits in open source unity projects. In: *43rd International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2021. p. 473–485.

PARNIN, C.; BLACK, A. P.; MURPHY-HILL, E. How We Refactor, and How We Know It . *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 38, n. 01, p. 5–18, jan. 2012. ISSN 1939-3520. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/TSE.2011.41>.

PATTON, M. Q. Sampling, qualitative (purposeful). *The Blackwell encyclopedia of sociology*, Wiley Online Library, 2007.

PERUMA, A.; ALOMAR, E. A.; NEWMAN, C. D.; MKAOUER, M. W.; OUNI, A. Refactoring debt: myth or reality? an exploratory study on the relationship between technical debt and refactoring. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2022. (MSR '22), p. 127–131. ISBN 9781450393034. Disponível em: <https://doi.org/10.1145/3524842.3528527>.

RATZINGER, J.; SIGMUND, T.; VORBURGER, P.; GALL, H. Mining software evolution to predict refactoring. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. USA: IEEE Computer Society, 2007. (ESEM '07), p. 354–363. ISBN 0769528864. Disponível em: <https://doi.org/10.1109/ESEM.2007.63>.

RZIG, D. E.; IQBAL, N.; ATTISANO, I.; QIN, X.; HASSAN, F. Virtual reality (vr) automated testing in the wild: A case study on unity-based vr applications. In: *32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. [S.l.: s.n.], 2023. p. 1269–1281.

SANTOS, E. A.; HINDLE, A. Judging a commit by its cover: correlating commit message entropy with build status on travis-ci. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2016. (MSR '16), p. 504–507. ISBN 9781450341868. Disponível em: <https://doi.org/10.1145/2901739.2903493>.

SAS, C.; CAPILUPPI, A.; SIPIO, C. D.; ROCCO, J. D.; RUSCIO, D. D. Gitranking: A ranking of github topics for software classification using active sampling. *Software: Practice and Experience*, Wiley Online Library, v. 53, n. 10, p. 1982–2006, 2023.

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

SILVA, D.; TSANTALIS, N.; VALENTE, M. T. Why we refactor? confessions of github contributors. In: *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*. [S.l.: s.n.], 2016. p. 858–870.

SILVA, D.; VALENTE, M. T. RefDiff: Detecting refactorings in version histories. In: *14th International Conference on Mining Software Repositories (MSR)*. [S.l.: s.n.], 2017. p. 269–279.

SPADINI, D.; ANICHE, M.; BACCHELLI, A. Pydriller: Python framework for mining software repositories. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ESEC/FSE 2018), p. 908–911. ISBN 9781450355735. Disponível em: <https://doi.org/10.1145/3236024.3264598>.

TSANTALIS, N.; KETKAR, A.; DIG, D. RefactoringMiner 2.0. *IEEE Transactions on Software Engineering (TSE)*, 2020.

TSANTALIS, N.; MANSOURI, M.; ESHKEVARI, L. M.; MAZINANIAN, D.; DIG, D. Accurate and efficient refactoring detection in commit history. In: *40th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2018. p. 483–494.

Unity Technologies. *Unity User Manual: Asset Workflow - Meta Files*. 2023. <https://docs.unity3d.com/Manual/AssetWorkflow.html>. Acessado em: 2024-05-20.

Unity Technologies. *Manual: Programming in Unity*. 2025. <https://docs.unity3d.com/Manual/>. Online: acesso em 03 Nov. 2025.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Independente, 2022.

VEGI, L. F. da M.; VALENTE, M. T. Towards a catalog of refactorings for elixir. In: *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.: s.n.], 2023. p. 358–362.

VEGI, L. F. da M.; VALENTE, M. T. Understanding refactorings in elixir functional language. *Empirical Software Engineering*, maio 2025.

VIOL, R. S.; FERNANDES, P. C.; LACERDA, I. I.; MELO, K. G. F.; NUNES, A. J. D. A.; PIMENTA, A. S. G. a. M.; CARNEIRO, C. M.; SILVA, M. C.; DELABRIDA, S. Eduvr: Towards an evaluation platform for user interactions in personalized virtual reality learning environments. In: *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2024. (IHC '23). ISBN 9798400717154. Disponível em: <https://doi.org/10.1145/3638067.3638131>.

WANG, J.; SHI, R.; ZHENG, W.; XIE, W.; KAO, D.; LIANG, H.-N. Effect of frame rate on user experience, performance, and simulator sickness in virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, v. 29, n. 5, p. 2478–2488, 2023.

WEISSGERBER, P.; DIEHL, S. Identifying refactorings from source-code changes. In: *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. [S.l.: s.n.], 2006. p. 231–240.