Pós-Graduação em Ciência da Computação

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TAYSE VIRGULINO RIBEIRO

Análise e Visualização de Dados aplicadas à Evolução de Projetos de Software

Recife

2021

TAYSE VIRGULINO RIBEIRO

Análise e Visualização de Dados aplicadas à Evolução de Projetos de Software

Dissertação apresentada como requisito parcial à obtenção do grau de Mestra em Ciência da Computação, área de concentração em Linguagens de Programação e Engenharia de Software, do Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco

Área de Concentração: Engenharia de Software e Linguagens de Programação

Orientador (a): Prof. Dr. Márcio Lopes Cornélio

Recife

2021

Catalogação na fonte
Bibliotecário Cristiano Cosme S. dos Anjos, CRB4-2290

R484a Ribeiro, Tayse Virgulino
Análise e visualização de dados aplicadas à evolução de projetos de software/ Tayse Virgulino Ribeiro. – 2021.
76 f.: il., fig., tab.

Orientador: Márcio Lopes Cornélio.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2021.
Inclui referências, apêndices e anexos.

1. Engenharia de Software e Linguagens de Programação. 2. Refatoração. 3. Medidas de software. 4. Engenharia de software. I. Cornélio, Márcio Lopes (orientador). II. Título.

005.1 CDD (23. ed.) UFPE - CCEN 2021 – 98

Tayse Virgulino Ribeiro

“Análise e Visualização de Dados aplicadas à Evolução de Projetos de Software”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 26/02/2021.

BANCA EXAMINADORA

_____

Prof. Dr. Henrique Emanuel Mostaert Rebêlo
Centro de Informática/ UFPE

_____

Prof. Dr. Everton Leandro Galdino Alves
Unidade Acadêmica de Sistemas e Computação / UFCG

_____

Prof. Dr. Unidade Acadêmica de Sistemas e Computação
Centro de Informática / UFPE
(Orientador)

Dedico este trabalho a minha família, em especial ao meu avô Rubens Pereira Virgulino (*in memória*)

AGRADECIMENTOS

Imersa em sentimentos, eu agradeço a todos que me acompanharam e apoiaram nesta minha caminhada.

Agradecer primeiramente a Deus, por teu amor e cuidado até aqui.

A minha mãe, Rosimar Virgulino, e minha irmã Tamirys Virgulino, por não desacreditarem das minhas escolhas. Aos meus avós, Luiza Severo e Rubens Pereira (*in memória*), que sempre foram minha inspiração, apoio e estiveram comigo até quando foi possível. A toda minha família, por mesmo de longe, se fazer presente. Obrigada por acreditarem no meu potencial.

A minha companheira Alessandra Araújo, por estar presente desde o início dessa caminhada, por todo apoio nestes últimos anos, por tanta troca de sentimento de amor e cuidado. Obrigada por permanecer comigo e me ajudar a não desistir! Eu te amo muito, muito obrigada.

Ao meu orientador Márcio Cornélio, por ter me acolhido no contexto acadêmico de forma tão empática, me proporcionando realizar este estudo num momento tão difícil da minha vida pessoal, profissional e acadêmica. Você esteve presente em boa parte deste caminho, e me auxiliou no processo de aprimoramento da minha vida profissional e, agora, como pesquisadora. Agradeço por ter persistido comigo.

A minha ex-orientadora da graduação e não obstante amiga Cristina Filipakis, que esteve presente desde o início, que me auxiliou e comemorou junto a entrada no mestrado e então ao término de mais uma etapa da minha vida. Obrigada por ter se disposto tanto, pelo apoio nos dias ruins e por toda alegria nas vitórias conquistadas. Sua participação neste processo foi enriquecedor e de grande valor para mim.

A minha ex-professora Heloise Acco, que me apoiou até aqui. Agradeço pelas conversas confortantes e de realidade (rsrs). Agradeço por acreditar no meu potencial e me compartilhar de tanta sabedoria.

Aos meus professores da graduação do Centro Universitário Luterano de Palmas. Que me auxiliaram também na minha vida acadêmica, e foram aqueles também que não mediram esforços para que eu pudesse então realizar minha sonhada Pós-Graduação.

A Thatiane Oliveira, que esteve sempre presente, que não poupou esforços para me ajudar e apoiar nesses mais de cinco anos de amizade. Agradeço imensamente pela colaboração para conclusão desta dissertação, pelas contribuições enriquecedoras. Eu te amo maninha, obrigada.

Como não agradecer as minhas amigas que mesmo a mais de 2.000 km fizeram questão de

estar, Fernanda Gomes, Rafaela Costa, Evelly Silva e Laryssa Resende. Obrigada meus amores, por se disporem tanto. Obrigada pelas conversas, força e estarem comigo em todas as minhas escolhas.

Hoje, não sei dizer como eu ficaria em Recife-PE sem essa rede de apoio linda que construí. Anny Caroliny, como sou grata por você ter se disposto a me receber desde o primeiro dia do mestrado, pelos finais de semana, por me tornar parte da sua família, porque eu me senti fazendo parte. Obrigada por isso, hoje eu só sei sentir gratidão e saudades!

A Sharla Gomes e Ana Carolina, pela apoio diário. Vocês foram parte da minha família em Recife-PE. Obrigada pelas recepções ao chegar em "casa". Pelo cuidado e amor. Obrigada meninas pela amizade que tanto tenho gratidão e amor.

A minha querida amiga Erika, a que me acolheu de forma imensurável durante toda minha caminhada no mestrado. Que compartilhou conhecimento, além das palavras de carinho e apoio. Obrigada minha querida, você se fez essencial neste processo.

Não poderia deixar de mencionar minha gratidão aos meus amigos do "Lab C011", por todo acolhimento, parceria e apoio em momentos tão delicados, e claro, por toda alegria nos momentos festivos. Sou grata por terem proporcionado viver momentos tão significativos com vocês e pelo todo conhecimento compartilhado durante a Pós-Graduação.

Por fim, ao Programa de Excelência Acadêmica da CAPES da Universidade Federal de Pernambuco, que me possibilitou apoio durante parte da minha Pós-Graduação.

RESUMO

O desenvolvimento de código com qualidade requer o uso de boas práticas de desenvolvimento de *software*. Com a evolução constante ao longo do ciclo de vida de um *software*, há preocupação com práticas para manutenção e evolução. Nesse cenário, temos o processo de refatoração, com a finalidade de promover a restruturação do código de modo a preservar seu comportamento. A partir da obtenção de dados de repositórios de *software* referentes a refatoração e medidas de *software*, neste trabalho acrescentamos visualização de dados com finalidade de explorar visualmente refatorações e medidas de *software* registradas em repositórios. O desenvolvimento deste trabalho se deu com auxílio das ferramentas voltadas para visualização de dados. A etapa de visualização dos dados, que consiste na análise dos dados, tratamento e apresentação de relatórios. Para tanto, foram obtidas as seguintes resoluções, primeiramente a tabulação dos dados, logo após, a compreensão e obtenção das características relacionais por meio de um Modelo de Entidade Relacionamento (MER). Das análises obtidas, tornou-se factível a apresentação de uma análise descritiva, por meio de análise quantitativa e qualitativa. Dispondo ainda de associação e relação dos projetos de *software* com refatorações e medidas existentes. O uso das ferramentas resultou em uma apresentação de correlações de ferramentas de visualização de dados, medidas de *software*, registros de versões, refatorações e projetos. Os resultados são apresentados por intermédio de grafos, gráficos e painéis interativos.

Palavras-chaves: Refatoração; Medidas de Software; Engenharia de *Software*; Visualização de Dados.

ABSTRACT

Developing quality code requires the use of good software development practices. The constant evolution throughout the software lifecycle brings concern with maintenance and evolution practices. In this scenario, we have the refactoring process to promote code restructuring and preserve its behavior. From obtaining data from software repositories, referring to refactoring and software measures, in this work, we added data visualization to explore refactorings and software measures registered in repositories visually. This work was developed with the help of tools aimed at data visualization. The data visualization stage consists of data analysis, processing, and reporting. The following resolutions were obtained, first the data tabulation, soon after, the understanding and obtaining of the relational characteristics through an Entity-Relational Model (ERM). From the analyzes obtained, it became feasible to present a descriptive analysis through quantitative and qualitative analysis. Furthermore, it was possible to obtain software project association and relation with refactorings and existing measures. The tools' use resulted in a presentation of correlations of data visualization tools, metrics, version records, refactorings, and projects. The results are presented using graphs, charts, and interactive panels.

Keywords: Refactoring; Software Metric; Software Engineering; Data Visualization.

LISTA DE FIGURAS

Figura 1 – Etapas do processo de refatoração de *software*. 26
Figura 2 – Metodologia proposta 38
Figura 3 – Análise visual utilizando Tableau Desktop 40
Figura 4 – Serviços Neo4j 41
Figura 5 – Propriedades do Neo4j 42
Figura 6 – MER da base de dados da pesquisa (OLIVEIRA, 2020) 43
Figura 7 – Procedimento metodológico 44
Figura 8 – Relação da Ferramenta SmartSmell e tipos de Smells 48
Figura 9 – Relação da Ferramenta Mauricioaniche/CK e medidas suportadas 49
Figura 10 – Relação da Ferramenta RefactoringMiner e os tipos de Refatoração 50
Figura 11 – Relação dos projetos com a medida CBO 51
Figura 12 – Relação dos projetos com a medida RFC 51
Figura 13 – Relação dos projetos com a medida LCOM 52
Figura 14 – Relação dos projetos com a medida DIT 52
Figura 15 – Relação dos projetos com a medida WMC 53
Figura 16 – Relação dos projetos com a medida LOC 53
Figura 17 – Relação dos projetos com a medida NumbersQTY 54
Figura 18 – Relação das medidas WMC e CBO 55
Figura 19 – Relação das medidas WMC e DIT 55
Figura 20 – Relação das medidas LCOM e DIT 56
Figura 21 – Relação das medidas LCOM e CBO 57
Figura 22 – Relação dos projetos e versões 57
Figura 23 – Ilustração das relações entre os nós 59
Figura 24 – Quantidade de Refactorings Identificados 60
Figura 25 – Quantidade de Code Smells Identificados nos Projetos 61
Figura 26 – Ranking de Projetos com mais Code Smells Identificados 62
Figura 27 – Ranking de Projetos com menos Smells Identificados 62
Figura 28 – Projeto com a cadeia de relação dos *refactorings* 63
Figura 29 – Relação de Project, Resource, Refactoring, Project Refactoring, Tag, Project Metric e Metric 63

Figura 30 – Seleção do Project Lawnchair 64
Figura 31 – Relação de Project e Tag 64
Figura 32 – Relação dos nós Project, Refactoring, Resource, Tag e Metric 65
Figura 33 – Relação dos nós Project, Refactoring, Resource, Tag e Metric 66
Figura 34 – Relação dos nós Project, Refactoring, Resource, Tag e Metric 66
Figura 35 – Fluxo realizado para alcançar a proposta desta pesquisa 68

LISTA DE TABELAS

Tabela 1 – Classificação dos padrões de projeto (GAMMA et al., 2000). 19
Tabela 2 – Classificação de refatorações por objetivos 20
Tabela 3 – Atividades necessárias no processo de refatoração (TOURWE; MENS, 2003; MENS; TOURWE, 2004). 25
Tabela 4 – Medidas utilizadas na literatura para o contexto de refatoração (CARNEIRO, 2003). 30
Tabela 5 – Relacionamento entre as tabelas da base de dados 58
Tabela 6 – Refatorações identificadas na base de dados 75
Tabela 7 – Code Smells Identificadas nos Projetos 76

SUMÁRIO

1 INTRODUÇÃO 14
1.1 CONTEXTUALIZAÇÃO E JUSTIFICATIVA 14
1.2 OBJETIVOS 15
1.3 QUESTÕES DE PESQUISA 15
1.4 ORGANIZAÇÃO DO TRABALHO 16
2 REVISÃO DA LITERATURA 17
2.1 REFATORAÇÃO 17
2.1.1 Refatoração para padrões de projetos 18
2.1.2 Classificação e catálogo de refatoração 19
2.1.3 Composição dos métodos 19
2.1.4 Organização de dados 22
2.1.5 Simplificando expressões condicionais 22
2.1.6 Tornando a chamada de métodos mais simples 23
2.1.7 Trabalhando com generalizações 24
2.1.8 Como aplicar a refatoração 25
2.2 TRABALHOS RELACIONADOS A REFATORAÇÃO 26
2.3 MEDIDAS DE SOFTWARE 28
2.3.1 Medidas no processo de refatoração 29
2.3.2 Trabalhos relacionados a medidas de *software* no processo de refatoração 31
2.4 ANÁLISE E VISUALIZAÇÃO DE DADOS 33
2.4.1 Big Data 33
2.4.2 Business Intelligence 34
2.4.3 Ferramentas de BI 35
3 METOLOGIA 38
3.1 FERRAMENTAS 39
3.1.1 Tableau: Ferramenta de Análise e Visualização de Dados 39
3.1.2 Neo4J: Ferramenta de Análise e Visualização de Dados 40
3.2 INSTRUMENTOS E ANÁLISE DE DADOS 42
4 RESULTADOS 45

4.1 ANÁLISE E VISUALIZAÇÃO DOS DADOS 45
4.1.1 Análise quantitativa 45
4.1.2 Análise qualitativa 47
5 CONCLUSÃO 67
5.1 AMEAÇAS À VALIDADE 69
5.1.1 Critérios de seleção 69
5.1.2 Representatividade da amostra 69
5.2 LIMITAÇÕES 70
5.3 TRABALHOS FUTUROS 71
REFERÊNCIAS 72
APÊNDICE A – REFATORAÇÕES REALIZADAS PELO REFACTORINGMINER 74
ANEXO A – REFATORAÇÕES IDENTIFICADAS NA BASE DE DADOS 75
ANEXO B – CODE SMELLS IDENTIFICADAS NOS PROJETOS 76

1 INTRODUÇÃO

Neste capítulo, constam as principais questões que motivaram a realização desta dissertação, introduzindo conceitos acerca de refatorações e medidas de *softwares* no processo de visualização de dados. Apresentamos também o objetivo da pesquisa e, finalmente, a estrutura desta dissertação.

1.1 CONTEXTUALIZAÇÃO E JUSTIFICATIVA

A aplicação dos métodos para o desenvolvimento de projetos de *software* tem por consequência influenciar nos resultados, no que tange à disponibilidade e confiabilidade do produto (FORNO; MULLER, 2017).

A evolução contínua de sistemas traz consigo a preocupação de manutenção constante. Normalmente, correções e evoluções são realizadas sem planejamento, com possibilidade de provocar reações não desejáveis no *software* futuramente. Embora com o uso de métodos existentes, ou até mesmo por um método criado pela própria organização, muitos deles ainda resultam em falhas e refatorações (FORNO; MULLER, 2017).

Na refatoração, uma alteração feita no código fonte, pode modificar sua estrutura interna, buscando facilitar a compreensão do código e permitir modificações com um custo mais baixo sem alterar o comportamento observável do *software* FOWLER (1999) e ainda assim, deve funcionar perfeitamente antes de refatorar, só assim você deverá refatorar ou realizar qualquer adequação.

Diante desse cenário, a realização desta pesquisa incorporou a necessidade exploratória do contexto de refatoração de *software*, como também, o processo de visualização dos dados relativos à medidas de *software*. A fim de proporcionar uma compreensão do conjunto de dados expostos para estabelecer relações e avaliações das informações.

Baseado nisso, temos a refatoração com o objetivo de permitir a reestruturação do código de modo a preservar seu funcionamento (OPDYKE, 1992). Assim como, prover a qualidade do código, sem que seu comportamento seja diretamente afetado (OLIVEIRA, 2020).

Com base na pesquisa de OLIVEIRA (2020), é possível estabelecer uma relação entre *code smells* e refatorações, observando o reflexo em medidas de *software*?

1.2 OBJETIVOS

O objetivo geral deste trabalho é realizar análise descritiva de visualização de medidas de *software*, identificando possíveis refatorações registradas em repositórios em projetos de *software*. Ou seja, descrever as principais características dos dados, além de observar as situações que levam a novos fatos. Portanto, para alcançar tal objetivo, temos os seguintes objetivos específicos:

- Identificar as relações de entidades existentes na base de dados, como por exemplo: hierarquia de classes e dependências entre as classes;
- Auxiliar, por meio de visualização de dados, a relação entre medidas de *software* e refatoração em projetos de *software*;
- Analisar o conjunto de objetos denominados vértices (ou nós) nos relacionamentos atribuídos;
- Determinar as medidas de *software* que serão analisadas;
- Realizar análises descritivas das relações obtidas na base de dados;
- Apresentar os relatórios gerados por meio das ferramentas *Tableau*[1] e *Neo4j*[2].

1.3 QUESTÕES DE PESQUISA

O processo de refatoração não pode ser considerado um problema, pois o mesmo tem por objetivo propor medidas de *software* para verificar o código durante seu desenvolvimento e ciclo de vida sem ocasionar problemas futuros de comportamento no código. Para auxiliar na compreensão e execução desta pesquisa, estabelecemos duas questões de pesquisa:

1. Como visualizar, a partir da análise e classificação dos dados referentes à medidas de *software*, as características e projetos necessários para estabelecer relação entre refatorações e medidas de *software*?
2. Como avaliar visualmente a evolução e mudanças nos projetos após sofrerem processo de refatoração?

[1] https://www.tableau.com
[2] https://neo4j.com/

Além disso, com a classificação e análise dos dados será possível resolver as questões de pesquisa e avaliar as medidas se *software* que foram tratadas no projeto. Ao longo do estudo, técnicas e impactos serão apresentados.

1.4 ORGANIZAÇÃO DO TRABALHO

A dissertação está dividida em mais quatro capítulos, como segue:

O **Capítulo 2** apresenta os principais conceitos dos assuntos e trabalhos relacionados ao tema desta dissertação. Ao qual aborda sobre refatorações, suas características e catálogo de refatorações. Além disso, as medidas de *softwares* para aplicar aos processos de *softwares*. Como também abordaremos sobre análise e visualização de dados.

No **Capítulo 3** apresentamos a metodologia de pesquisa adotada para a realização do estudo. Primeiramente é explicado o processo de confecção do fluxo do processo, passando pelas etapas do método desenvolvimento para analisar e classificar os dados. Neste Capítulo ainda abordamos o método para confecção das etapas de visualização de dados, identificação de refatoração de *software* obtenção de medidas de *software*. Por fim, realizamos as devidas considerações finais acerca dos critérios para o método proposto.

Além disso, no **Capítulo 4** é apresentada uma análise de aspectos que verificam a abordagem de otimização do fluxo de trabalho e de visualização de dados para validar a obtenção de medidas de *softwares* e classificação dos dados coletados; apresentando o confronto dessas visões em comparação a literatura e com os dados que foram coletados e mensurados.

Finalmente, o **Capítulo 5** apresenta as considerações finais desta pesquisa, assim como evidencia as principais contribuições e limitações. Além disso, elenca possibilidades de trabalhos futuros.

2 REVISÃO DA LITERATURA

Neste capítulo apresentamos os principais conceitos dos assuntos e trabalhos relacionados ao tema desta dissertação. Este capítulo está dividido em quatro seções. A Seção 2.1 discorre sobre Refatoração e suas características. A Seção 2.2 aborda Medidas de *Software*, com o intuito de apresentar aplicações no contexto de refatoração. A Seção 2.3 aborda Análise e Visualização de Dados, esta que traz consigo conceitos relacionados a tratamento, análise e obtenção de dados. Por fim, a Seção 2.4 traz alguns trabalhos relacionados acerca da revisão realizada e faz considerações finais a respeito dos assuntos discutidos neste capítulo.

2.1 REFATORAÇÃO

É uma técnica que visa organizar o código, e por consequência minimizar a chance de introduzir novas falhas FOWLER (1999). Permite a reestruturação do código de modo a preservar seu comportamento e a permitir que mudanças sejam realizadas facilmente (OPDYKE, 1992). A refatoração é um processo de alteração do código-fonte de uma maneira que não altere seu comportamento externo e ainda melhore a sua estrutura interna.

O procedimento de refatoração proposto por FOWLER (1999) consiste em uma etapa inicial de testes, que define-se uma suíte de testes à qual o código alvo da refatoração será submetido. Se os testes passarem com sucesso, a suíte será utilizada em um teste de regressão posterior à modificação do código. Em seguida, faz-se uma pequena modificação no código como um passo visando chegar à refatoração. O código deve ser, então, compilado e submetido à suíte de testes utilizada anteriormente. O procedimento continua com ciclo de modificações pequenas no código, compilação e teste até se obter a refatoração desejada.

A técnica de refatoração de *software* pode, e deve, ser utilizada por diversas razões, o que resulta em uma série de vantagens, como: melhorias no projeto, coesão de código-fonte, torna o *software* fácil de entender, *design* apropriado e diminuição do tempo de manutenção (OPDYKE, 1992).

A refatoração de *software* é vista como uma abordagem padrão que permite que o código seja aperfeiçoado sem que novos erros sejam aplicados ou que o comportamento seja alterado. Além disso, a refatoração constante raramente é possível sem uma cobertura de testes automatizados. Os testes devem compor como parte do processo de refatoração, pois existe

a possibilidade de interferir nos componentes em funcionamento em uma extremidade do sistema ao refatorar trechos de códigos com anomalias. Dessa forma, garantindo um alerta se um determinado código ou componente teve seu comportamento alterado (CARNEIRO, 2003; OLIVEIRA, 2020).

Logo, deve-se **manter o código limpo**; eliminando códigos duplicados, métodos longos e muitos parâmetros . Não seguir essa recomendação afeta a legibilidade do código, o que pode levar a um tempo de depuração mais longo. Além disso, pode-se refatorar com vistas a melhorar o desempenho de um produto. A refatoração do código contribui para redução do débito técnico. Por fim, deve-se manter o código legível e sempre atualizado, evoluindo o código conforme a atualização das tecnologias aplicadas ao produto final (BARROS, 2015). Outra recomendação é reduzir o débito técnico, pois o custo de um projeto de *software* não é finalizado depois que se lança sua primeira versão.

O processo de refatoração pode ser realizado manualmente ou de forma automatizada. As refatorações manuais são constantemente realizadas, porém podem introduzir mais erros ao código-fonte do *software*. Isso ocorre pelo fato que os desenvolvedores não estarem preparados tecnicamente para realizar processos padrões como é exigido ao aplicar refatoração no produto final (Ge; DuBose; Murphy-Hill, 2012; NAIRNEI, 2018).

O processo de refatoração consiste compreender não somente os fatores de padrões e classificações de código, mas também em sua composição, organização na classificação de refatoração, conforme será abordado nas subseções seguintes.

2.1.1 Refatoração para padrões de projetos

Um padrão pode ser considerado como um recurso reutilizável para um problema que advém de uma determinada circunstância (BUSCHMANN et al., 1996). Os padrões de projetos são estabelecidos de acordo com entidades primitivas como objetos e classes. Assim, um padrão deve abstrair, nomear e identificar pontos importantes de uma estrutura de projeto para torná-la útil na criação de um projeto reutilizável (GAMMA et al., 2000).

Além disso, os padrões de desenvolvimento de projetos afetam a forma como o produto é feito, gerando um dicionário comum e que aumenta o nível qualidade do produto final para aperfeiçoar a comunicação e documentação (NAIRNEI, 2018).

Os padrões de criação se referem ao processo de criação de objetos. Os padrões estruturais lidam com a composição de classes ou de objetos. Os padrões comportamentais referem-se

Tabela 1 – Classificação dos padrões de projeto (GAMMA et al., 2000).

		Finalidade		
		De criação	Estrutural	Comportamental
Escopo	Classe	Factory Methody	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

como as classes e objetos interagem e distribuem suas responsabilidades (GAMMA et al., 2000).

A refatoração é uma atividade de aperfeiçoamento que mantém a qualidade do produto final por meio da padronização dos projetos, aplicando essa técnica na manutenção do *software* e repassando a importância da técnica para os profissionais da indústria.

2.1.2 Classificação e catálogo de refatoração

As refatorações podem ser classificadas por objetivo e nível de abstração. As classificações por objetivos podem ter seis classificações com suas respectivas motivações (CARNEIRO, 2003). Também, podem ser denominadas como "catálogo de refatoração"(FOWLER, 1999), com a possibilidade de acréscimo de novos componentes, ilustrado na Tabela 2. Para melhor compreensão do Catálogo de Refatoração, na Tabela 2 será discriminado cada classificação.

2.1.3 Composição dos métodos

A composição dos métodos conta com conjuntos característicos de refatorações com o objetivo de posicionar o código dos métodos de forma apropriada. Com isso, frequentemente ocorre o uso de refatorações associadas à composição de métodos para que o comportamento dos objetos seja expresso de forma apropriada. Baseado nisso, os métodos de tamanho excessivo não são recomendados, pois contêm grande número de informações, geralmente acompanhados de lógicas complexas. A refatoração de migração de trecho de código para um novo método,

Tabela 2 – Classificação de refatorações por objetivos

Classificação de Refatorações	Relação de Refatorações
Composição dos Métodos	Migração de trecho de código para um novo método, Dissolução de método, Dissolução de variável temporária, Substituir variável temporária por consulta, Introdução de variável esclarecedora, Substituir variável temporária por duas ou mais, Uso de variável temporária, Substituição de método por objeto, Substituição de algoritmo
Movendo funcionalidades entre objetos	Migração de método para outra classe, Migração de variável para outra classe, Migração de trecho de código para uma nova classe, Dissolução de classe, Delegação oculta, Remoção de delegação, Introdução de método estrangeiro, Introdução de extensão local
Organização de Dados	Encapsulamento de informações, Criação de objeto para tratamento de Informações, Modificação para objeto do tipo referência, Modificação para objeto do tipo valor, Substituição de arranjo para objeto, Separação da interface da lógica do negócio, Transformação de uma associação unidirecional para bidirecional, Transformação de uma associação bidirecional para unidirecional, Inclusão de uma constante para representar um número, Transformação de uma variável pública em privada com mecanismos de acesso, Encapsulamento de uma coleção, Substituição de registro por classe de dados, Substituição de tipo de código por classe, Substituição de tipo de código por subclasses, Substituição de tipo de código por objeto do tipo estado, Substituição de subclasses por variável
Simplificando Expressões Condicionais	Decomposição de condicional, Consolidação de expressão condicional, Consolidação de trechos duplicados em expressão condicional, Remoção de variável de controle, Substituição de condicionais encadeadas por cláusulas, Substituição de condicional por polimorfismo, Uso de objeto nulo, Introdução de afirmação
Tornando a chamada de métodos mais simples	Renomeando métodos, Adição de parâmetros Remoção de parâmetros, Separação de consulta por modificador, Parametrização de método , Substituição de parâmetros por métodos, Uso de objeto para obtenção de informações, Substituição de parâmetro por método, Substituição de parâmetro por objeto, Remoção de método de atribuição de valores, Ocultação de método em relação a outras classes, Substituição de construtor por subclasses, Encapsulando projeção de tipo, Substituição de código de erro por exceção, Substituição de exceção por teste
Trabalhando com Generalizações	Migração de variável para superclasse, Migração de método para superclasse, migração de construtor para superclasse, Migração do método para subclasse, Migração de variável para subclasse, Migração de trecho de código para uma nova subclasse, Migração de trecho de código para uma nova superclasse, Migração de trecho de código para uma nova interface, Junção de superclasse e subclasse, Migração de trecho de código para um novo método modelo, Substituição de herança por delegação, Substituição de delegação por herança

retira uma parte do código de um método já existente e importa para um novo método criado exclusivamente para tal fim. Todavia, na dissolução de método é exatamente o inverso, antes de remover o método, o seu código é migrado para um método destino.

O problema na migração de trecho de código para um novo método ocorre com as variáveis locais, sendo as variáveis temporárias as maiores responsáveis por isso. Assim, quando tal caso ocorrer, deverá ser substituído o uso da variável temporária por uma consulta para que sejam removidas as variáveis temporárias. Caso a variável temporária seja usada para várias finalidades, deve-se substituir a variável temporária por duas ou mais para facilitar a substituição. Em alguns momentos, pode ser complexo remover uma variável temporária, podendo ser necessário substituir o método por um objeto, possibilitando dividir um método com o mesmo custo de adicionar uma nova classe para tal finalidade.

Os parâmetros não representam problemas, desde que não sejam usados no âmbito do método para atribuição de valores. Nesse caso, deve-se utilizar uma variável temporária. Uma vez que o método tenha sido dividido, pode-se compreendê-lo bem melhor. O algoritmo também pode ser melhorado para que se torne mais claro através da refatoração de substituição de algoritmo (FOWLER, 1999).

Movendo funcionalidades entre objetos, pode-se escolher de forma apropriada o posicionamento dos membros (métodos e atributos). Para se encontrar uma nova posição para as funcionalidades pode-se migrar de método e variável para outra classe.

Quando uma classe estiver com muitos métodos, pode-se migrar um trecho de código para uma nova classe, para conduzir parte desses para uma nova classe. Caso uma classe esteja com pouca ou praticamente nenhuma responsabilidade, deve-se retirar de uma classe para mover seus atributos para outra classe já existente. Se outra classe está sendo utilizada, é aconselhável realizar a delegação oculta. Às vezes, ocultar a classe delegada resulta em modificações na interface do proprietário, sendo necessário utilizar o processo de remoção da situação delegada.

As duas últimas refatorações deste grupo são caracterizadas pela introdução de método estrangeiro e de extensão local. Eles são usados sempre que não for possível o acesso ao código-fonte da classe, ainda que seja necessário mover responsabilidades para tal classe. Caso seja apenas um ou dois métodos, deve-se introduzir um método estrangeiro, para mais de dois métodos deve-se realizar a extensão local (FOWLER, 1999).

2.1.4 Organização de dados

Neste grupo estão refatorações que contribuem para melhorar o tratamento de dados. Muitas vezes é necessária a criação de um dispositivo para possibilitar acesso aos dados armazenados por um objeto, para tal fim desempenhar o encapsulamento de informações.

A possibilidade de se definir novos tipos de dados amplia o que poderia ser feito somente com os tipos de dados de linguagens tradicionais. Pode-se iniciar com um valor de dados simples e depois verificar que um objeto pode ser mais adequado. Ao criar um objeto para tratamento é possível que os dados sejam convertidos em objetos. Quando tais objetos forem instâncias necessárias em várias partes de um programa, deve-se modificar para um objeto do tipo referência. Nesse caso, um vetor pode ser substituído para objeto. Em todos esses casos, o objeto é o primeiro passo. O segundo passo é a migração do método para outra classe, para se possibilitar adicionar comportamento aos novos objetos.

Os números com significados especiais representam um problema. Para substituí-los é necessário incluir uma constante para representar um número. Já os relacionamentos entre objetos podem ser uni e bidirecionais. Assim, a depender do caso, pode-se transformar em uma associação unidirecional para bidirecional ou então o inverso, a fim de remover complexidade desnecessária quando não for necessária comunicação bidirecional. São comuns os casos em que classes de interfaces gráficas de usuário também ofereçam funcionalidades associadas à lógica do negócio. Para remover esse comportamento para uma classe apropriada, deve-se separar a interface da lógica do negócio.

Em relação ao uso do encapsulamento, quando os dados estiverem com acesso público, pode-se transformar um atributo público em privado com mecanismos de controle de visibilidade. Caso seja uma coleção, deve-se empregar o encapsulamento de uma coleção. Caso um registro inteiro esteja acessível, deve-se substituir um registro por classe de dados.

Logo, se os códigos não alteram o comportamento da classe, é destinado a substituição do tipo de código por classe ou de estado (FOWLER, 1999).

2.1.5 Simplificando expressões condicionais

O uso indevido de lógica condicional pode tornar a compreensão do código um processo não trivial, por isso existe um conjunto de refatorações que podem ser aplicadas com o objetivo de simplificar o código. A principal refatoração desse grupo é a atividade de decomposição de

um condicional, que partilha uma variável condicional em várias partes, sendo importante pelo fato de separar a lógica do chaveamento dos detalhes do que ocorrerá.

Ainda assim, aplica-se a consolidação de expressão, quando tem-se vários testes e todos têm o mesmo resultado. Os programas orientados a objeto geralmente têm menor incidência de código condicional que programas procedurais, pois muito do comportamento condicional pode ser tratado através de polimorfismo. Logo, o polimorfismo torna-se mais adequado pelo fato de quem atribui não precisar ter conhecimento do comportamento condicional, sendo também mais fácil a extensão das condições.

Por fim, pode-se também substituir um condicional de polimorfismo, quando identificado. Portanto, o polimorfismo também pode ser usado como um objeto nulo, para remover verificações a respeito de valores nulos (FOWLER, 1999).

Por exemplo o uso de refatorações como: *move method, extract class, extract method, inline method, pull up method e push down method*. A aplicação de refatorações consistentes, a fim de garantir manutenção do comportamento durante o ciclo de vida do código, conforme apresenta no catálogo de refatorações de (FOWLER, 1999). Portanto, para exemplificar o esquema do método analisaremos a refatoração *Extract Method*. A refatoração deve ser aplicada quando o método se caracteriza muito longo. A execução pelos seguintes passos por (FOWLER, 1999):

- Criar novo método e nomeá-lo conforme o caracterize;
- Alimente o novo método com as informações do código extraído, conforme a necessidade de referenciar as variáveis locais;
- Transfira para o novo método como parâmetros as variáveis local para que sejam encorporadas e lidas pelo código extraído;
- Compile e teste o código após tratar todas as variáveis necessárias;

O processo de execução da refatoração se da no formato *adhoc*, mas pode-se ser manipulado por ferramentas de detecção de refatoração.

2.1.6 Tornando a chamada de métodos mais simples

O objetos estão diretamente relacionados a interfaces. Através de interfaces fáceis de compreender pode-se atingir bons resultados em programação orientada a objetos. Na maioria

das vezes, a forma mais simples e importante na atividade de avaliação dos métodos é a modificação do seu nome.

Dispondo do conhecimento do que um determinado programa realiza, a renomeação de métodos pode ser indicada para que o próprio nome do método sirva como mecanismo para transmitir seus objetivos. Abrangendo também em relação a modificação dos nomes, o mesmo pode ser considerado para variáveis e classes, sendo nesse caso mecanismos simples, não havendo portanto refatorações associadas.

Tendo em vista que o uso de objetos permite que sejam usadas listas de parâmetros menores, existe uma relação de refatorações cujo objetivo é a redução da quantidade de parâmetros. Se diversos valores de objetos estiverem sendo utilizados, pode ser aplicado o uso de objeto para obtenção de informações para se reduzir tais valores a um único objeto.

Vários métodos similares podem ser combinados, adicionando-se um parâmetro por meio da parametrização do método. Manifestando uma separação clara entre métodos que podem modificar o estado de objetos, daqueles que vão somente realizar consultas sobre o estado dos objetos. Caso essa separação não esteja sendo atendida, deve-se consultar o modificador. Boas interfaces devem deixar visível e disponível somente o necessário. Tanto as informações expressas através dos atributos como os métodos devem obedecer essa orientação. Nesse caso, deve-se ocultar o método em relação a outras classes e remover os métodos de atribuição de valores.

Já os construtores pressupõem o conhecimento da classe de um objeto que será instanciado. Dessa maneira, pode-se eliminar a necessidade desse conhecimento através da substituição de construtor por subclasses. É possível ainda evitar encapsulamento de recursos. Entretanto, aqueles que não têm familiaridade com tal recurso, geralmente usam códigos de erro para sinalizar a existência de problemas. Nesse caso, pode-se substituir o código com erro por uma exceção (FOWLER, 1999).

2.1.7 Trabalhando com generalizações

Associado a generalizações tem-se um conjunto de refatorações, em que a maioria trata da realocação de métodos ao longo da hierarquia. Nesse contexto, a migração de variável e método para uma superclasse proporciona a ascensão de membros das classes na hierarquia.

No caso de construtores a situação não é comum, sendo necessária a migração de construtor para superclasse. Caso sejam detectados métodos que tenham estrutura parecidas, mas alguns

detalhes diferentes, deve-se usar migração de trecho de código para um novo método modelo, para que sejam separadas as diferenças. Além de promover a mudança de membros ao longo da hierarquia, a própria hierarquia pode ser modificada através da criação de novas classes. Essa atividade pode ser realizada também pela migração para um superclasse, subclasse e interface. Caso seja verificado que existem classes desnecessárias na hierarquia, pode-se unir a superclasse e subclasse para removê-las. Algumas vezes, percebe-se que herança não é a melhor forma de tratar uma determinada situação, sendo necessário o uso de delegação (FOWLER, 1999).

2.1.8 Como aplicar a refatoração

Além de contextualizar o conceito de refatoração e sua classificação FOWLER (1999) e CARNEIRO (2003), falam da importância do processo de aplicação. Em 2003, TOURWE; MENS (2003) identificaram três passos necessários para o processo de refatoração. No ano seguinte, em 2004, os mesmos autores evoluíram o método, a fim de contemplar um contexto de qualidade e preservação de comportamento MENS; TOURWE (2004). A Tabela 3 ilustra os resultados desses dois trabalhos.

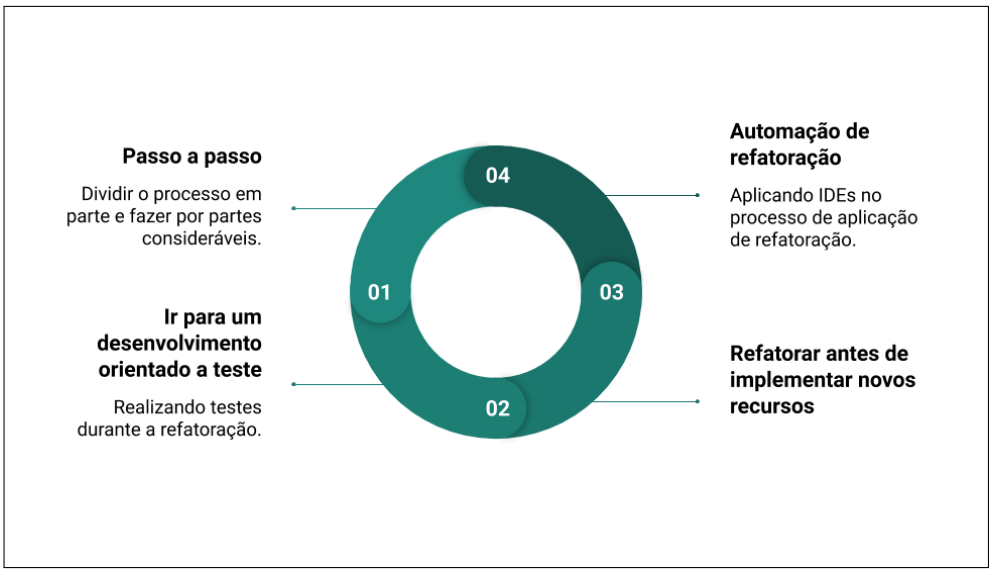
Tabela 3 – Atividades necessárias no processo de refatoração (TOURWE; MENS, 2003; MENS; TOURWE, 2004).

2003	**2004**
Identificar quando um código deve ser refatorado	Identificar quando um código deve ser refatorado
Identificar quais transformações deverão ser utilizadas no código	Identificar quais transformações deverão ser utilizadas no código
Executar as transformações, se possível de maneira automatizada e que possa ser verificada	Garantir que a refatoração preserva o comportamento do código
	Aplicar a refatoração
	Avaliar o impacto da refatoração na qualidade do software (complexidade, compreensibilidade, manutenibilidade)
	Manter a consistência entre o código refatorado e os artefatos de software (documentação, requisitos, testes, etc.)

Portanto, o processo de refatoração é uma série de pequenas etapas direcionadas a transformar o código-fonte do *software* em algo mais fácil de reutilizar e manter, tal como ilustrado

na Figura 1.

Figura 1 – Etapas do processo de refatoração de *software*.



Fonte: CARNEIRO (2003), FOWLER (1999), OPDYKE (1992)

Após compreender o conceito de refatoração e seus diferentes tipos, a próxima seção busca explorar o uso de medidas de *software*, a fim de identificar e medicar códigos que podem indicar necessidade de refatoração.

2.2 TRABALHOS RELACIONADOS A REFATORAÇÃO

Foram identificados na literatura alguns trabalhos relacionados ao contexto de refatoração. Um trabalho relacionado foi escrito por PAIXãO et al. (2020) e está relacionado ao processo de refatoração de código, que é uma etapa importante para manutenibilidade do *software*. O objetivo central do trabalho se deu em identificar se a alteração no código realizado foi de forma adequada. Por meio do estudo realizado, é descrito uma mineração de 1.780 alterações de código revisadas de seis sistemas pertencentes a duas grandes comunidades de código aberto, relatadas por um estudo empírico aprofundado sobre refatoração de *software* durante a revisão de código. Portanto, inspecionando e classificando as intenções dos desenvolvedores no processo de alteração de código em sete categorias distintas.

Com a abordagem de diversos estudos empíricos que exploram a intenção do desenvolvedor no que tange à decisão do processo de refatoração. A pesquisa tem como contribuições: (i) descobertas sobre como as refatorações são realizadas ao longo de revisões de código com distintos objetivos, (ii) uma discussão sobre as implicações dessas descobertas e (iii) motiva um

novo conjunto de dados que permite aos pesquisadores investigar o contexto e as motivações por trás das operações de refatoração durante a revisão de código (PAIXãO et al., 2020).

O estudo foi dividido em quatro etapas, que se resumem em, primeiramente: selecionar os sistemas de *software* que adotam a revisão de código moderna de PAIXãO et al. (2020). Esse tem por objetivo agrupar um conjunto de dados de código aberto que vincula revisões de código com suas respectivas alterações. A segunda etapa diz respeito a identificar as operações de refatoração durante a revisão do código. De acordo com PAIXãO et al. (2020), foi utilizada a ferramenta *RefMiner* para identificar operações de refatoração de acordo com treze tipos de refatoração, que são comumente empregados por desenvolvedores. Consequentemente, a terceira etapa realiza o processo de inspecionar e classificar manualmente as intenções dos desenvolvedores por traz das discussões de revisão de código.

Foram consideradas as 1.780 avaliações que empregaram operações de refatoração para realizar uma inspeção e classificação manuais das intenções dos desenvolvedores, segundo PAIXãO et al. (2020). Isso posto, a última etapa que diz respeito a validação de revisões de código que apresentam uma intenção de refatoração. Essa que resulta nas análises do códigos, dentre as limitações de fatores de ferramenta e tomada de decisão.

O trabalho foi resumido da seguinte forma para os pesquisadores. Primeiro, apresenta: (i) incluem refatorações complexas, como alterar membros da classe, e (ii) são adicionadas ou desfeitas ao longo de uma mudança relacionada a um recurso. Ações desse tipo influenciam no ato de incentivo dos desenvolvedores nas revisões durante o processo (PAIXãO et al., 2020). Alterações explícitas na refatoração geralmente envolvem operações complexas de refatoração, como mover ou extrair classes. Portanto, foi possível notar que (i) essas alterações tiveram uma quantia menor de revisões do que as outras e (ii) as refatorações realizadas raramente foram refinadas ou desfeitas durante a revisão do código (PAIXãO et al., 2020).

Por fim, foi observado que as intenções dos desenvolvedores surgem após uma análise a longo prazo de uma revisão. Isso demonstra que é necessário revisar, refinar e quando necessário desfazer o processo de refatoração com base nos *feedbacks*. Como também, demonstra a importância da refatoração ser realizada de forma iterativa.

Além disso, no trabalho de SHEN et al. (2019) foi apresentado um algoritmo de fusão com reconhecimento de refatoração, baseado em gráfico para programação orientada a objeto (JAVA), a fim de aprimorar explicitamente a capacidade do algoritmo em detectar e resolver conflitos relacionados à refatoração. A técnica foi nomeada de *IntelliMerge*, que tinha capacidade de evitar e resolver divergências relacionadas à refatoração, sem prejudicar sua eficácia. O

IntelliMerge possui um cenário de entrada de mesclagem com três vias de um projeto Git. Um cenário com duas ramificações a serem mescladas (*left* e *right*) e o *nearest common ancestor* (NCA) na sua versão base. A saída do *IntelliMerge* contém a versão mesclada e um conjunto de divergências que não podem ser resolvidos automaticamente (SHEN et al., 2019).

Foram coletados 1.070 cenários de mesclagem contendo refatoração de 10 projetos Java de código aberto no *Github* que foram reproduzidos com as seguintes ferramentas: *IntelliMerge*, *GitMerge* 4 e *jFSTMerge*, para avaliar a eficácia e eficiência do *IntelliMerge*. Os resultados do experimento mostraram que: (1) em termos de resultados mesclados automaticamente, o *IntelliMerge* atinge uma boa precisão de 88,48% e um *recall* de 90,22% com a versão resolvida manualmente como a verdade fundamental, (2) em termos de conflitos de mesclagem, o *IntelliMerge* produz menos blocos de conflito (58,90% a menos que o *GitMerge* e 11,84% a menos que o jFSTMerge) e reduz as linhas de código conflitante (em 90,98% comparado ao *GitMerge* e em 78,38% comparado ao *jFSTMerge*), (3) em mais de 90 % dos cenários de mesclagem examinados, o *IntelliMerge* conclui todo o processo de mesclagem em 5 segundos, o que é aceitável para ser aplicado em aplicativos reais (SHEN et al., 2019).

2.3 MEDIDAS DE SOFTWARE

As medidas de *software* são utilizadas para estimar o custo, esforço e tempo que são aplicados em desenvolvimento e manutenção do *software*. A medição nos permite realizar previsões gerais de um *software* e identificar componentes como "maus cheiros de código". Nesse contexto, as medidas correspondem a uma forma de determinar se um *software* possui um determinado atributo, permitindo a comparação com atributos da mesma natureza (SOMMERVILLE, 2007).

Além disso, existem as medidas para análise de modelos *Unified Modeling Language* (UML), que é uma família de notações gráficas, apoiada por um metamodelo único, que auxilia na descrição e no projeto de sistemas de *softwares* orientados por objetos (FOWLER et al., 1999). Essas medidas, tratam de métodos, atributos, herança, relacionamentos, dentre outros elementos da orientação a objetos (YI; WU; GAN, 2004).

Para apoiar o assunto explorado nesta seção, será tratado sobre as medidas de *software* para auxiliar no processo de refatoração e mensurar *software*.

2.3.1 Medidas no processo de refatoração

Para avaliar a qualidade de um sistema, é necessário aplicar medidas de *software* que permitem mensurar suas características. As medidas usam classificações numéricas para quantificar as características e atributos de uma entidade de *software* (CHIDAMBER; KEMERER, 1994; MACHADO, 2017). Além do mais, as medidas podem fornecer previsões sobre as características do produto final ou do custo total do seu desenvolvimento.

OLIVEIRA (2020) aponta que as medidas de *software* podem funcionar para as equipes, assim como, no aprimoramento tanto do produto quanto dos desenvolvedores. OLIVEIRA 2020, reforça que medir e analisar *software* não precisa ser um exercício oneroso ou que atrapalha a criação e manutenibilidade do código.

A mensuração de código auxilia no entendimento sobre as características de uma dada aplicação, sendo que o seu resultado, ainda que incompleto, proporciona informações de grande potencial para a avaliação das características do *software* (CARNEIRO, 2003).

De acordo com CHIDAMBER; KEMERER (1994) e MACHADO (2017), algumas das medidas usadas para mensurar o software no contexto de refatoração são:

- *Lack of Cohesion of Methods* (LCOM):
  É uma das principais medidas utilizadas para mensurar a coesão, que é a característica que indica o grau de especialização de uma classe no sistema (PRESSMAN, 2010). Tal medida estima os conjuntos de métodos em uma classe que não estão relacionados entre si através do compartilhamento de atributos acessados internamente. Ao calcular o LCOM de uma classe quanto maior for o valor obtido, mais baixa será a sua coesão (CHIDAMBER; KEMERER, 1994).

- *Cohesion Among Methods of Class* (CAM):
  Outra medida em sistemas orientados a objetos para mensurar a coesão, que calcula a relação entre os métodos de uma classe com base na lista de parâmetros passadas para os métodos (BANSIYA et al., 1999).

- *Weighted Methods Per Class* (WMC):
  É a soma das complexidades dos métodos de uma classe. A complexidade pode ser medida pela análise da quantidade de métodos, suas respectivas complexidades e linhas de código (CHIDAMBER; KEMERER, 1994).

- *Number of Public Methods* (NPM):
  Outra medida utilizada para medir a complexidade de uma classe que faz contagem dos métodos públicos declarados na classe (CHIDAMBER; KEMERER, 1994).

- *Lines of Code* (LOC):
  A medida que representa a quantidade de linhas de código (CHIDAMBER; KEMERER, 1994).

- *Depth of Inheritance Tree* (DIT):
  Define a quantidade de classes que estão em níveis acima na hierarquia e que podem afetar diretamente a classe. Logo, quanto mais baixo a classe está na hierarquia, mais complexo será o comportamento de seus métodos (CHIDAMBER E KEMERER, 1994).

- *Number of Children* (NOC):
  Essa medida calcula o número de sub-classes subordinadas a uma classe na hierarquia, fazendo uma estimativa da quantidade de sub-classes que vão herdar os métodos da super-classe (CHIDAMBER; KEMERER, 1994).

Na Tabela 4 é apresentado um resumo com algumas medidas utilizadas no contexto de refatoração e citadas na literatura consultada para esta pesquisa.

Tabela 4 – Medidas utilizadas na literatura para o contexto de refatoração (CARNEIRO, 2003).

Medida	**Aplicação**	**Atributo**	**Medição**
Cyclomatic complexity - CC	Classes e métodos	Complexidade	Complexidade e alternativas possíveis no controle de fluxo
Lines of Code – LOC	Classes e métodos	Tamanho	Obter o tamanho das classes e métodos
Depth of Inheritance Tree - DIT	Classes e interfaces	Herança	Reuso, compreensão e teste
Number of Children - NOC	Classes e interfaces	Herança	Reuso
Response for Class - RFC	Classes	Comunicação	Acoplamento, complexidade e pré-requisitos para teste
Coupling between object class - CBO	Classes	Comunicação	Coesão e reuso
Lack of Cohesion in Methods - LCOM	Classes	Comunicação	Coesão, complexidade, encapsulamento e uso de variáveis
Weighted Methods Class - WMC	Classes	Complexidade	Complexidade, tamanho, esforço para manutenção e reuso
Number of Methods - NOM	Métodos	Tamanho	Corresponde a WMC onde o peso de cada método é 1
Number of Instance Variables – NIV	Classes	Tamanho	Obter o número de variáveis de instância
Number of Class Variables – NCV	Classes	Tamanho	Obter o número de variáveis de classe
Number of Inherited Methods – NMI	Métodos	Herança	Obter o número de métodos herdados e definidos em uma superclasse
Number of Overriden Methods – NMO	Métodos	Herança	Obter o número de métodos definidos em uma superclasse e redefinidos na subclasse

2.3.2 Trabalhos relacionados a medidas de *software* no processo de refatoração

O estudo realizado por OLIVEIRA (2020) tem como objetivo identificar e correlacionar refatorações que resultam na eliminação de *code smells* e aferição de medidas de *software*. Para alcançar este objetivo foi adotado um método com as etapas de Construção, Inicialização, *Refactoring*, *Code Smell*, Medidas de *Software*, Processamento e Sumarização. Foi realizada mineração de repositórios de *software* para o desenvolvimento do trabalho.

Os pontos abaixo descrevem o método apresentado, que é focado em entender os seguintes fatores:

- Construção: verifica a lista e disponibilidade dos repositórios;
- Inicialização: obtém as informações do repositório GIT, armazenando em um repositório local;
- *Refactoring*: investiga as refatorações realizadas entre as versões e cria um catálogo com base no item de recurso (classe) alterado;
- *Code Smells*: inspeciona as versões em busca de problemas estruturais (*code smells*) com base no uso de ferramentas com este fim;
- Medidas de *Software*: coleta medidas de *software* em todas as classes do projeto, ou seja, em toda versão (*tag*) avaliada;
- Processamento: segundo OLIVEIRA (2020), realiza o cruzamento dos dados coletados nas etapas anteriores e busca relacionar os acontecimentos de um recurso;
- Sumarização: é realizado o agrupamento dos dados e sumarização no formato de dados relacional para extração e análise.

Essas foram as características propostas para construir a solução a fim de possibilitar a identificação no processo de refatoração que resultam em eliminação de *code smells* e obtenção de medidas de *software*.

No que tange aos resultados de OLIVEIRA (2020), foi realizado com apoio das ferramentas selecionadas *JQuality/SmartSmell, aDoctor, CK* e Ferramentas de Identificação de Refatoração (*RefactoringCrawler*, *RefactoringMiner* e *Ref-Finder*) e foi dividido em quatro etapas. A primeira etapa consistia em abranger as soluções sobre identificação de refatorações, *code*

smells e medidas de *software*. Segundo OLIVEIRA (2020), essa etapa contém os critérios práticos para que no processo de seleção apenas ferramentas alinhadas no processo do trabalho fossem utilizadas. Já na segunda etapa, desenvolver uma ferramenta com o objetivo de minerar dados em repositórios públicos de *software*, promover a integração e execução das ferramentas selecionadas na primeira fase. A terceira etapa consolida todos os dados obtidos da fase anterior, construindo um conjunto de dados que fosse passível de análise e um *dataset* estruturado com as informações correlacionadas. Por fim, a quarta etapa foi a análise dos dados, propondo identificar os reflexos que as refatorações e os *code smells* causam nas medidas de *software* (OLIVEIRA, 2020). Finalmente, o trabalho teve por consequência a validação da qualidade do código que foi orientado pelas medidas de *software*. Como também, possibilitando investigar uma solução que indicasse se a refatoração, realmente, provocou a melhoria no código.

Já no trabalho de dissertação de CARNEIRO (2003), foram estabelecidos relacionamentos entre alguns dos principais tipos de refatoração propostos porFOWLER (1999) e medidas obtidas através do código-fonte do *software*. A utilização de medidas tem potencial para auxílio na execução do processo de refatoração de *software*. Assim como, tornar seus resultados analisáveis quantitativamente (CARNEIRO, 2003).

A refatoração foi bem sucedida nessa abordagem, quando obtém a seleção e a aplicação efetiva de medidas baseadas nos objetivos de refatoração da equipe de desenvolvimento do *software* (CARNEIRO, 2003). O objetivo do trabalho foi introduzir medidas no uso de refatoração. Sendo uma fase primordial para o estabelecimento futuro de uma metodologia para auxílio na detecção de oportunidades de refatoração usando medidas.

No trabalho de CARNEIRO (2003) foram estabelecidos relacionamentos entre medidas, refatorações e *bad smells* usando a abordagem de FOWLER (1999) como estrutura de partida e posteriormente uso das seguintes abordagens: uma abordagem *Top Down* usando o paradigma Meta Pergunta Métrica - MPM; uma abordagem *Bottom Up* usando análise empírica, utilizando um grande conjunto de medidas e verificando o seu relacionamento com refatoração e *bad smells*. Os resultados das abordagens, comprovou que as medidas podem auxiliar no uso da refatoração. A abordagem *Top Down* demonstrou, que as medidas refletem 25% do conjunto de medidas, já as medidas não disponíveis refletem os restantes 75% do conjunto. Logo, a abordagem *Bottom Up*, demonstrou que, serão necessários mais estudos para expor o grau de relacionamento entre *bad smells* e medidas, pois, devido ao número pequeno de ocorrências de *bad smells* reportados no estudo de caso, não foi possível a obtenção de dados representativos para tal finalidade (CARNEIRO, 2003).

Portanto, para se compreender a mensuração realizada nos códigos com refatorações, se faz possível conhecer, analisar e visualizar de forma mais objetiva o contexto ao qual a refatoração foi aplicada, os meios para se obter esse tipo de informação serão apresentados na próxima seção.

2.4 ANÁLISE E VISUALIZAÇÃO DE DADOS

Devido ao crescente avanço tecnológico, ocorreu um crescimento no volume de informações e consequentemente surgiram desafios relacionados à manipulação, extração de valor e conhecimento. Diante disso, torna-se necessário realizar estudos em busca de soluções que mitiguem os desafios apresentados.

Algumas das técnicas utilizadas para lidar com os desafios apresentados são *Big Data* e *Business Intelligence*, as quais serão apresentada nesta seção.

2.4.1 Big Data

O termo *Big Data* é atualmente empregado a um conjunto de dados que crescem rapidamente. É um conjunto de dados cujo tamanho está além da capacidade das ferramentas de *software* capturar, armazenar, gerenciar e processar os dados em um ambiente (ELGENDY; ELRAGAL, 2014).

Big data são dados cuja escala, distribuição, diversidade e/ou pontualidade exigem o uso de novas arquiteturas, análises e ferramentas técnicas para permitir *insights* que revelam novas fontes de valor comercial. Há cinco características aplicadas ao contexto do *big data*: volume, variedade, velocidade, veracidade e valor (5Vs) (ELGENDY; ELRAGAL, 2014).

- Volume: o volume dos dados é caracterizado pelo seu tamanho;
- Velocidade: indica a taxa ou a frequência com que os dados estão sendo alterados;
- Variedade: refere-se aos diferentes tipos e formatos de dados;
- Veracidade: concerne a verificação dos dados coletados para adequação e relevância ao seu contexto e/ou objetivo;
- Valor: quanto mais relevante e importante os dados coletados, mais significativo é explorar e questionar a cerca do processo de análise.

O *Big data* atua diretamente no processo de análise de dados, que é onde as técnicas analíticas avançadas são utilizadas. A análise baseada em grandes amostras de dados indica e promove as mudanças nos negócios. No entanto, quanto maior o conjunto de dados, mais difícil se torna seu gerenciamento.

2.4.2 Business Intelligence

As tecnologias de *Business Intelligence e Analytics* - BI tem o objetivo de facilitar a coleta de dados, análise e entrega de informações e são projetadas para apoiar a tomada de decisões (RIKHARDSSON; YIGITBASIOGLU, 2018). Logo, deve auxiliar a encontrar as causas de problemas e as melhores práticas estratégias de decisão. O método de BI pode utilizar diversos sistemas gerenciais, ferramentas de *Data Mining* para auxiliar no processo de análise. Com suporte aos eixos de metodologia, arquitetura, ferramentas, banco de dados e aplicações para análise de dados.

A entrada do processo de BI é um banco de dados que trata da saída de um conjunto de conhecimentos. A etapa principal é a mineração e análise de dados, com o processo de análise realizado a partir das amostras.

Segundo LOH (2014), o processo de descoberta do conhecimento pode ser realizado várias vezes, com diferentes amostras, técnicas e ferramentas. Como também, pode ter intervenção humana. Para realizar a preparação dos dados e depois a sua interpretação. Apoiado nisso, LOH (2014) apresenta algumas técnicas de BI:

- *Data Warehouse* (DW):

  Se trata de uma combinação de conceitos e tecnologias que facilita na etapa de integração dos dados corporativos, além de gerenciar e manter históricos obtidos das operações realizadas, a fim de gerar relatórios para análises (SANTOSO et al., 2017).

- OLAP:

  De acordo com LOH (2014), OLAP (*on line analytical processing*) se caracteriza como uma análise multidimensional cuja processo de construção permite análise de negócio e visualização de dados corporativos em tempo hábil.

- *Data Mining*:

  Mineração de dados é uma expressão utilizada quando os dados se apresentam de forma

estruturada, ou seja, seu objetivo principal é identificar conhecimento a partir de grande volume de dados (GOLDSCHMIDT; PASSOS; BEZERRA, 2015)

- ETL:

  Extract, Transform And Load (ETL) processo de retirada dos dados de fontes externas para o DW, se caracterizando também como etapa crítica e demorada do DW para implementação.

2.4.3 Ferramentas de BI

Alguns dos métodos avançados de análise de dados mais comuns são: regras de associação, *clustering*, regressão, árvores de classificação e decisão. Dessa forma, com o aumento do volume de dados, tornou-se necessário uma forma rápida para analisar esses dados. Portanto, surge a necessidade de novas ferramentas e métodos especializados para análise de *big data*, bem como das arquiteturas necessárias para armazenar e gerenciar esses dados. Dessa maneira, afetando, desde os dados em si e sua coleta, até o processamento e as decisões finais extraídas (ELGENDY; ELRAGAL, 2014).

Logo, foi proposto o *Big Data Analytics* e Decisão, que incorpora as ferramentas e métodos de análise ao processo de tomada de decisão. O *framework* esquematiza diferentes ferramentas de armazenamento, gerenciamento e processamento para a tomada de decisão. Dessa forma, as alterações refletem nas áreas de armazenamento e arquitetura de *big data*, processamento de dados e, por fim, as análises de *big data*.

De acordo com GALDINO (2016), alguns dos principais ambientes e ferramentas de *Big Data* são:

- Ambientes em nuvem:

  A computação em nuvem é uma grande adepta no uso de ferramentas de *big data*. A baixa do preço de armazenamento nos últimos anos, juntamente com à elasticidade que ambientes oferecem facilitam o acesso a esses serviços para as empresas que não têm o orçamento suficiente. Os ambientes em nuvem permitem pagamento por hora e cobram somente pela quantidade de informação utilizada.

- HDFS:

  O *Hadoop Distributed File System* ou Sistema de arquivos distribuídos surge com a

necessidade de se trabalhar com arquivos grandes. O HDFS faz a divisão em blocos dos arquivos e os distribui em vários nós, com replicação para atender nível de segurança.

- *Map Reduce*:

  É o sistema analítico do *Hadoop* desenvolvido para operar com grandes volumes de dados que realiza o processamento do código onde estão sendo processados. O processamento analítico é distribuído em vários servidores, cujo se deseja capturar informação. Assim, a ferramenta separa os dados em partições, mapeia as atividades em cada local, duplica em ambientes e depois faz as reduções. Sendo assim, são formados pares com chaves para serem enviados e juntando pares com as mesmas características. Sendo elas três fases, *Map*, onde os dados são agrupados; *Shuffle*, o qual além de reunir, organiza. Por fim, a *Reduce*, onde os dados são associados.

- Tableau:

  Através de nós de *clusters* usa computação distribuída com alta escalabilidade, tolerância a falhas e confiabilidade. Sendo voltada para *clusters* e processamento de grande volume de dados. O objetivo principal é tratar esse grande volume de dados sem ter a necessidade de copiá-los em outro servidor. Os dados são tratados dentro do servidor em tempo real, gerando mais praticidade, assim como mantém a tolerância a falhas através da replicação dos dados. Além disso, a ferramenta *Tableau* é responsável por compartilhar os dados, gerar relatórios e painéis por meio dos dados analisados.

- MPP:

  O *Massively Parallel Processing* ou Processamento Massivo Paralelo é um paradigma de *Big Data*, feito para processar grandes quantidades de dados, é escalável em relação à quantidade de dados, suporta linguagem SQL e tabelas relacionais. Além disso, pode trabalhar em conjunto com *Data Warehouse*, realizando ações paralelas.

- HBase:

  É um banco de dados *Nosql* que processa grande volume de dados de maneira rápida e em tempo real. Trabalha com o conceito chave, de que cada dado é associado a outro trazendo uma característica similar ao modelo relacional.

- Spark:

  O *Spark* atua sendo mais abrangente na questão de diferentes tipos de processamento. A principal diferença em relação ao *Map Reduce* é o fato de persistir em disco. O *Spark*

trabalha em memória, faz encadeamento de funções e só apresenta o resultado no fim do processamento. O *driver*, aplicação principal do *Spark*, faz alocação das máquinas no *cluster* para processamento de funções.

- Pentaho:

  O *Pentaho Corporation* é uma iniciativa da comunidade de desenvolvimento *Open Source* para oferecer ferramentas de *Business Intelligence* para que as organizações aprimorarem sua performance, eficiência e efetividade na gestão da informação.

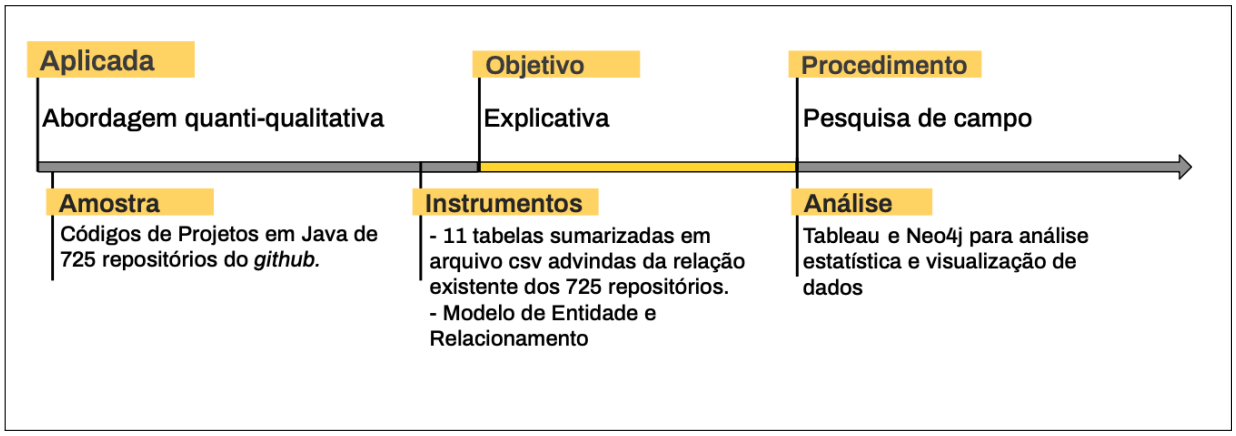
Diante dos trabalhos relacionados nesta seção, foi possível evidenciar que o principal diferencial se deu pelo apoio aos códigos identificados e classificados com refatoração que passaram pelo processo de classificação de medidas de *software*, a fim de analisar a causa e histórico de desenvolvimento do projeto. Ou seja, a proposta dessa dissertação visa utilizar visualização de dados com vistas a facilitar o acompanhamento de projetos para tomada de decisão sobre melhoria de código.

3 METOLOGIA

Nesse capítulo descrevemos o método utilizado para o desenvolvimento deste trabalho. Dessa forma, a pesquisa foi de natureza aplicada em campo, com abordagem quanti-qualitativa e de carácter explicativo. Com base no trabalho de dissertação de OLIVEIRA (2020), esta pesquisa utilizou-se de uma amostra de 725 projetos em linguagem Java retirados de repositórios do *github*. O trabalho de OLIVEIRA (2020) foi utilizado de base para esta pesquisa a fim de fornecer meios e justificativas para proporcionar uma análise e visualização de dados de forma mais explicativa e exploratória. Como também, auxiliar no processo de tomada de decisão, por meio das análises obtidas.

Os instrumentos utilizados para coleta nesta pesquisa foi um conjunto de 11 tabelas sumarizadas em formato de arquivo *csv*, advindas da relação existente dos 725 repositórios, conforme disponibilizados pelo OLIVEIRA (2020)[1]. Portanto, as ferramentas *Tableau* e *Neo4j* foram adotadas para explorar os dados advindos da relação desse MER, resultando em análises e visualizações dos dados, que são detalhadas no **capítulo 4**. O processo de análise e visualização dos dados teve a finalidade de organizar, resumir e descrever os aspectos relevantes de um conjunto de características de forma quantitativa e qualitativa. Assim como, analisar o estado e históricos de relação dos dados. O processo supracitado pode ser visualizado na Figura 35.

Figura 2 – Metodologia proposta



Fonte: A autora (2021)

Diante da metodologia apresentada, a próxima seção aborda passo-a-passo do procedimento metodológico realizado para se alcançar a proposta levanta para essa pesquisa.

[1] https://cutt.ly/BvItU3T

3.1 FERRAMENTAS

A utilização das ferramentas expostas nesta seção têm por objetivo auxiliar na agilidade de respostas de análise de dados. Se fez necessário validar o uso de ferramentas de análise e visualização de dados como o *Tableau*[2] e o *Neo4j*, para construção de análises explicativas. A fim de auxiliar na tomada de decisão e dispor de interface intuitiva e interativa.

3.1.1 Tableau: Ferramenta de Análise e Visualização de Dados

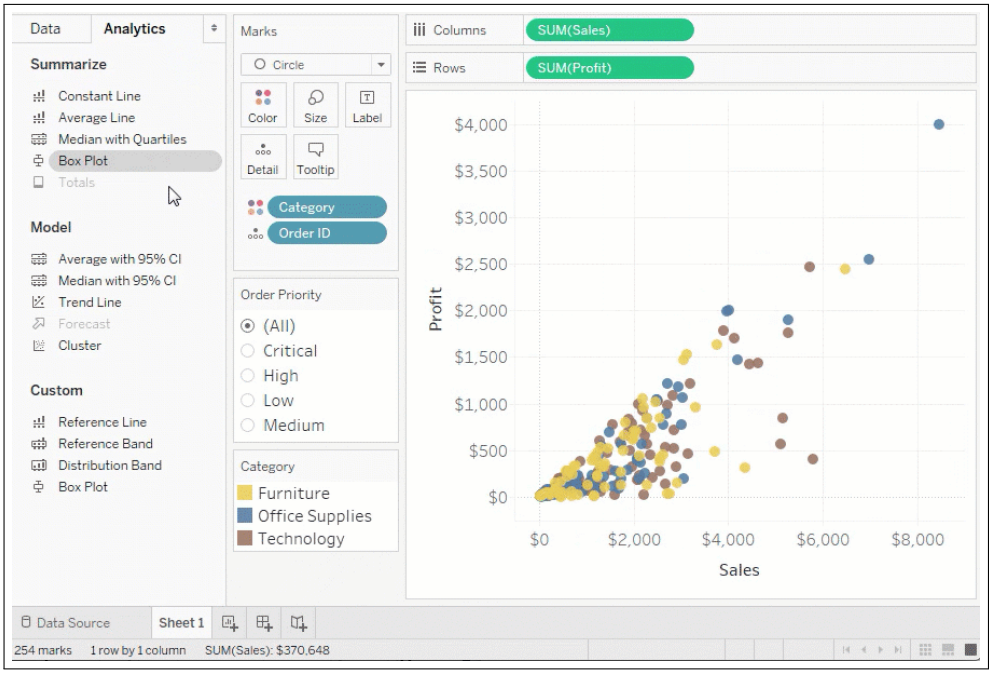
A ferramenta *Tableau* trata-se de um *software* de exploração de fonte de dados que tem por objetivo permitir obter respostas em frações de segundos. Essa ferramenta é uma das líderes em BI e análise de dados no mercado. Portanto, é uma das ferramentas para auxílio no processo de análise e visualização dos dados desta pesquisa.

O *Tableau* fornece análises avançadas, capaz de auxiliar em resolução de problemas. Com o *Tableau*, é possível consumir os dados com segurança de forma integrada em dispositivos ou aplicações. Com segurança e conformidade, a ferramenta *Tableau* permite colaboração no processo de decisões e interfaces interativas pra compartilhar informações. Outro fato, é que proporciona questionar e responder conforme for a necessidade e contexto do pesquisador. Baseado nisso, os recursos são de fácil consulta e possibilitam liberar histórico de versionamento. Bem como, tem que por garantir a segurança e confidencialidade dos dados, de forma centralizada. Além disso, como enfatizado no início, permite a preparação dos dados de maneira interativa e todo acesso ao dados pode ser realizado localmente ou na nuvem. Na Figura 3 é apresentado um dos produtos da ferramenta *Tableau*, o *Tableau Desktop*.

O *Tableau Desktop* é uma das ferramenta de *business intelligence* responsável pelas análises de interface visual. Essa ferramenta trabalha com questionamento, agrupamento de banco de dados, cálculos e diversos outros recursos. Esse é um dos produtos que o *Tableau* fornece para trabalhar análise e manipulação avançada de dados.

[2] https://www.tableau.com

Figura 3 – Análise visual utilizando Tableau Desktop



Fonte: SOFTWARE (2003-2020)

3.1.2 Neo4J: Ferramenta de Análise e Visualização de Dados

Da mesma forma, outra ferramenta apoiada no objetivo de auxiliar no propósito de analisar e proporcionar visualização dos dados, o *Neo4j*[3] é uma plataforma de banco de dados em grafo, tem como abordagem a inteligência artificial, detecção de fraude e recomendações em tempo real. É um produto que impulsiona o valor nas conexões e relacionamentos em dados por meio de aplicações que capturam a constante mudança e escalonamento de regras de negócios. Como também explorar como as pessoas, processos e sistemas estão relacionados.

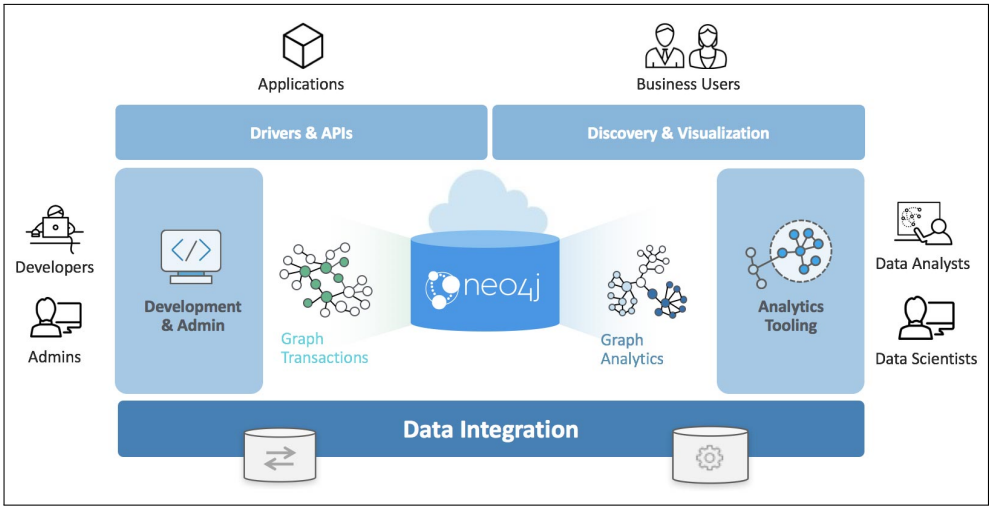
O Neo4j é composto de aplicações que se baseiam em banco de dados nativos. Os principais produtos ofertados para apoio em análises gráficas.

- Banco de Dados Gráfico Nativo *Neo4j*: é uma conexão escalável. À medida que os dados são armazenados o *Neo4j* conecta a aplicação. Com isso, a consulta ocorrerá de forma mais assertiva;

- Análise e Visualização Gráfica: tem por objetivo auxiliar na comunicação dos dados e no processo de representatividade das possíveis perspectivas dos dados;

[3] https://neo4j.com/

Figura 4 – Serviços Neo4j



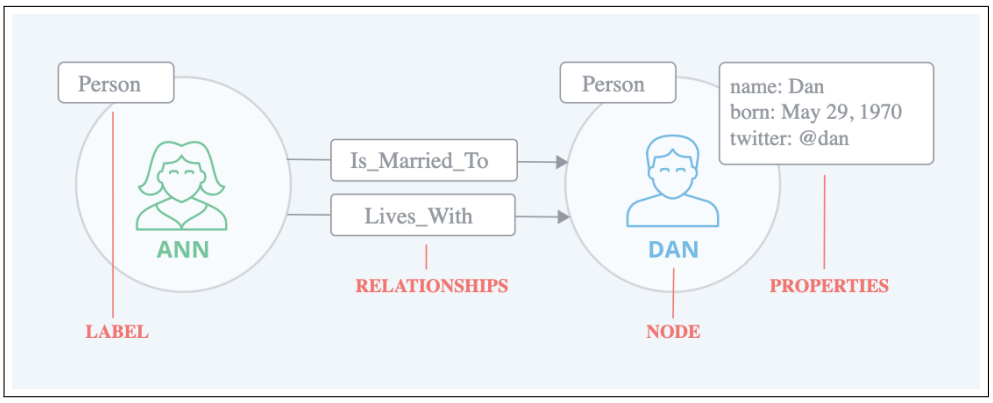
Fonte: NEO4F (2020)

- *Neo4j* ETL e Integração de Dados: apresenta as conexões dos dados armazenados em bancos de dados gerenciados. Logo, atua na proposta de acelerar a conversão dos dados e o processo de *big data* em formato de gráficos;
- Linguagem de Consulta de Gráfico *Cypher*: linguagem padrão para o contexto de gráficos. A qual fornece análise de consulta em *big data*;
- Arquitetura Corporativa: tem por objetivo fornecer estrutura para suportar os dados gráficos.

As aplicações do *Neo4j* tem por objetivo abordar a importância de conexões persistentes nas transições, dando a ideia de uma representação gráfica em um modelo lógico. Baseado nisso, a implementação em um modelo físico, utilizando-se de uma linguagem para consulta e à persistência em um banco de dados, conforme a Figura 5.

Nos fundamentos do *Neo4j* são atribuídos quatro rótulos básicos, que são: Nós, Relacionamentos, Propriedades e Etiquetas, conforme é apresentado na Figura 5. Os Nós são os elementos principais de dados, o Relacionamento é o conector dos nós. Já as Propriedades são valores e textos, podendo ser indexados, restringidos e compostos. Por fim, as Etiquetas são os rótulos usados para agrupar o nós em conjuntos, a qual tem por objetivo principal auxiliar na localização e indexação dos dados principais (Nós) no gráfico.

Figura 5 – Propriedades do Neo4j



Fonte: NEO4F (2020)

3.2 INSTRUMENTOS E ANÁLISE DE DADOS

Para ser possível a análise e classificação dos dados, bem como, a avaliação da evolução de mudanças dos projetos, foi utilizado como instrumento o Modelo de Entidade Relacionamento (MER) para descrever os dados relacionados da base sumarizada. Assim como, detalhes dos objetos, características e relacionamentos entre as entidades. No caso, proporcionou um conjunto com 11 arquivos *csv* apoiados nos relacionamentos das entidades do MER, conforme a Figura 6.

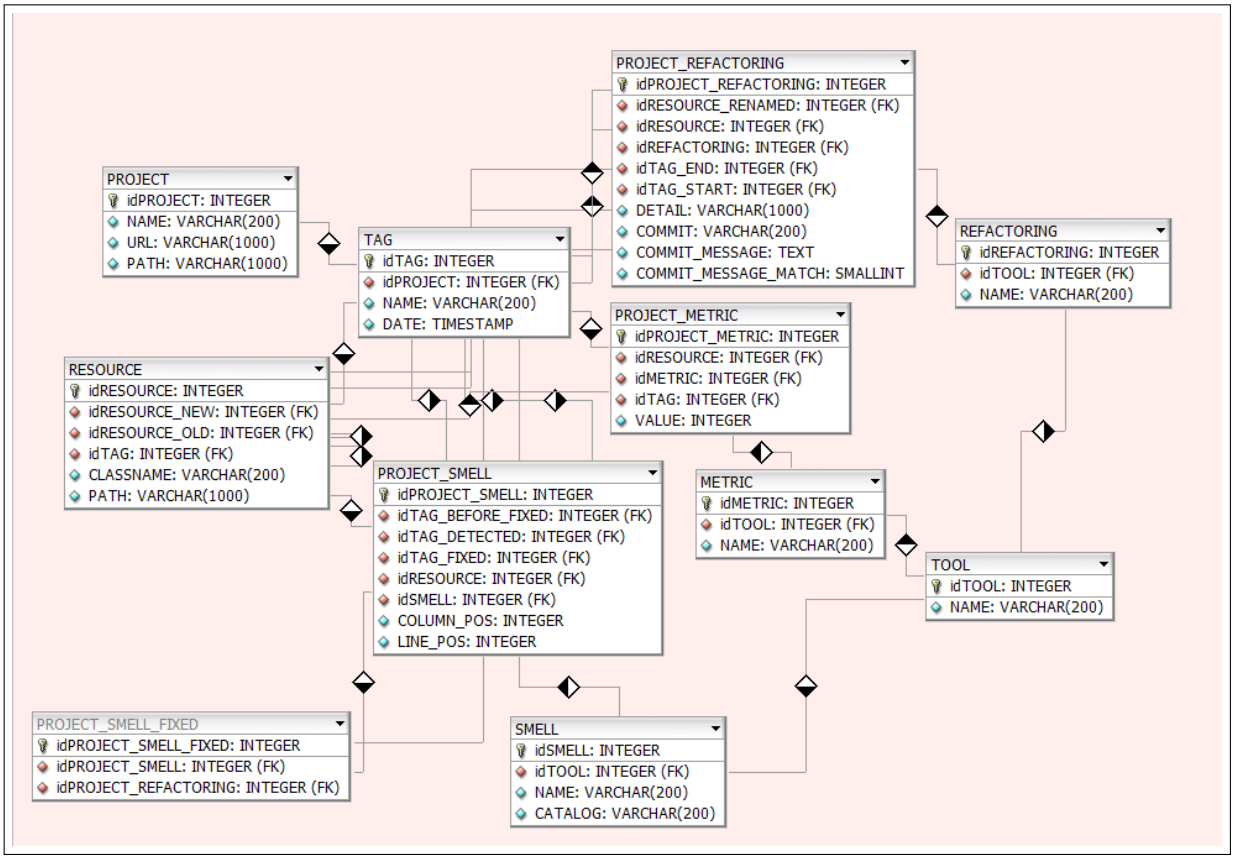
O MER ilustrado na Figura 6, é composto por um conjunto de onze entidades, com seus respectivos relacionamentos e atributos, que representam projetos com características de refatoramento. Conforme já elucidado, tais projetos foram absorvidos de um repositório *online*. O trabalho de OLIVEIRA (2020), apresenta um dicionário de dados relacionado a esse modelo.

Nesse domínio, as principais dependências se dão por meio da entidade *TAG*, *PROJECT*, *REFACTORING* E *METRIC*. Essas entidades têm o papel de interligar e relacionar os projetos envolvidos e refatoramentos em seu histórico, assim como as medidas associadas.

Com base nos estudos realizados, as medidas adotadas tiveram como base o trabalho de Carneiro (2003). Utilizando-se das medidas de *Lack of Cohesion of Methods* (LCOM), *Weighted Methods Per Class* (WMC), *Number of Public Methods* (NPM), *Lines of Code* (LOC), *Depth of Inheritance Tree* (DIT)e *Number of Children* (NOC)/ *NumbersQTY*. Identificando as devidas medidas baseado nos atributos de suas características internas do código: Acoplamento, Coesão, Complexidade e Tamanho.

A estruturação da base de dados teve por objetivo permitir estabelecer, para cada *PROJECT*, a relação entre *METRIC*, *CODE SMELLS* e *REFACTORING*.

Figura 6 – MER da base de dados da pesquisa (OLIVEIRA, 2020)



Fonte: OLIVEIRA (2020)

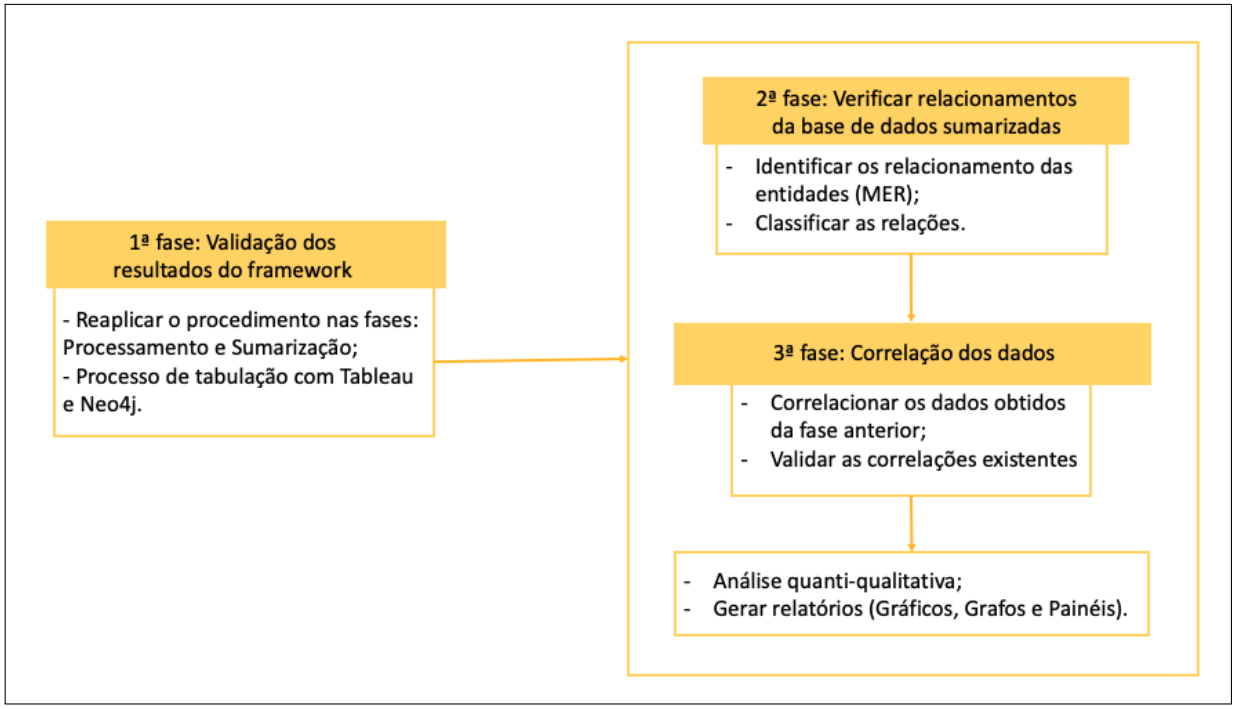
Para a obtenção das análises, foi realizado o manuseio dos *softwares Tableau* e *Neo4j*. Tais *softwares* foram adotados com o objetivo de realizar a leitura dos dados e apresentação dos relacionamentos obtidos pela base de dado, resultando em grafos, gráficos e painéis.

Para obtenção do resultado, a análise compreendeu algumas etapas. A primeira etapa foi a tabulação dos dados. O objetivo dessa etapa foi configurar e relacionar as tabelas. Para realizar essa configuração é necessário atribuir as cardinalidades dos relacionamentos, a fim de agrupar as tabelas ou obter os relacionamentos de forma automática, utilizando ferramentas de análise de BI. Logo, com posse da estrutura lógica do banco de dados, tornou-se viável a visualizações dos principais relacionamentos, assim como, possíveis análises exploratórias.

Para ficar mais claro, separamos o procedimento metodológico em três etapas, ilustradas na Figura 7.

Podemos identificar, conforme a Figura 7, três etapas para concepção do procedimento: a primeira etapa (1) se deu pela replicação do trabalho de OLIVEIRA (2020), no que tange nas fases de mineração e processamento dos repositórios. Diante disso, surgiu a necessidade de identificar a relação entre os dados gerados a partir dos arquivos *csv*. OLIVEIRA (2020) dispo-

Figura 7 – Procedimento metodológico



Fonte: A autora (2021)

nibilizou acesso público à base de dados sumarizada em csv[4]. Após realizar a tabulação dos dados por meio das ferramentas de BI, *Tableau* e *Neo4j*, demos início à segunda etapa. A segunda etapa (2) identifica os relacionamentos existentes no MER incorporado nas ferramentas de apoio, permitindo compreender quais e quantos projetos foram refatorados.

Diante disso, surgiu a necessidade de identificar e classificar as relações encontradas na etapa anterior. O processo de incorporação dos dados se deu pela inserção do arquivo em *csv* nas ferramentas *Tableau* e *Neo4j*. Com isso, possibilitando a interpretação dos dados obtidos mediante a relação e análise de cada entidade. Por fim, a terceira etapa (3), se deu pela relação dos dados obtidos na base de dados advindas do *Tableau* e *Neo4j*. Isso porque, existia a necessidade em reclassificar as relações obtidas na etapa 2. Logo, com o uso da ferramenta *Neo4j*, foi atribuído os identificadores de nós raízes e filhos, para conhecer e compreender até onde as informações estavam presentes e se os dados realmente se correlacionavam entre eles.

Diante desse procedimento metodológico foi possível analisar e visualizar a base de conhecimento gerada com auxílio das ferramentas de *Tableau* e *Neo4j*, resultando em análises quantitativas e qualitativas, representados por Gráficos, Painéis e Grafos, as quais são detalhadas no próximo capítulo.

[4] https://cutt.ly/BvItU3T

4 RESULTADOS

Neste capítulo apresentaremos os resultados obtidos a partir da Análise e Visualização de Dados. A Seção 4.1 apresentará o processo de Análise e Visualização dos Dados, método desenvolvido para analisar e classificar as informações incorporadas com medidas de *software*. Também, exibimos os dados coletados, classificando os projetos analisados, refatorações e medidas de *software* por meio de análises descritivas.

Conforme definido anteriormente, o objetivo geral deste trabalho é realizar uma análise descritiva de visualização de dados, a fim de identificar refatoração e explorar o histórico observando as medidas e refatorações. Para que esse objetivo se concretizasse, este trabalho foi dividido em três etapas, conforme descrito no capítulo de metodologia.

4.1 ANÁLISE E VISUALIZAÇÃO DOS DADOS

4.1.1 Análise quantitativa

Nesta seção será apresentado o conjunto de informações exploradas da base de dados sumarizada, com a distinção de cada entidade e sua amostra no contexto desse domínio de projetos com refatoração, como também, os projetos que apresentaram algum tipo de medida de *software*. Na base de dados disponibilizada para este estudo foram obtidos 725 repositórios do *github*.

De acordo com o trabalho de OLIVEIRA (2020), foram aplicados alguns critérios para obtenção dos projetos diretamente na API do *Github*:

q=android+language:java+stars:>=725&sort=stars&order=desc

Logo, o uso de somente projetos *Android* era um dos critérios de seleção utilizados para o inclusão no conjunto de *dataset*, bem como, linguagem Java.

As tabelas estão estruturadas conforme o resumo acerca do dicionário de dados informado:

- *Project:* responsável por representar a entidade projeto. Cada repositório analisado possui um registro na representação do relacionamento e é composto por 725 projetos;
- *Tag:* representa as versões de cada projeto. No servidor do *Github* as versões são obtidas através do conceito de *tags*. Essa tabela é composta por 11.968 tags;

- *Resource:* um registro nessa tabela é representada por determinadas classes java. Essas classes podem estar associada a uma nova classe, caso esse tenha sido movido ou renomeado. A tabela é composta por 2.143.700 registros;
- *Tool:* representa as ferramentas que estão cadastradas no *Reflink*, conforme abordado no estudo de OLIVEIRA (2020), que são: *SmartSmell*, *mauricioaniche/ck* e *RefactoringMiner*;
- *Refactoring:* cada registro nessa entidade representa um tipo de refatoração suportada pela ferramenta (*tool*) associada. A tabela comporta 22 tipos de refatoração, todas associadas à ferramenta *RefactoringMiner*;
- *Smell:* idêntico à entidade *Refactoring*, cada registro representa um tipo de *code smell* suportado pela ferramenta. A tabela apresenta 19 *code smells*, todos pertencentes à ferramenta *SmartSmell*;
- *Metric:* a entidade *Metric*, apresenta as medidas associadas. Possui 37 medidas e todas associadas à ferramenta *mauricioaniche/ck*;
- *ProjectRefactoring:* representa o relacionamento com o *Resource*, *Tag* e *Refactoring*. Essa tabela armazena 197.525 registros;
- *ProjectSmell:* representa o relacionamento com *Resource*, *Tag*, *Smell*. A tabela armazena 4.738.240 registros;
- *ProjectMetric:* retrata o relacionamento com *Resource*, *Tag* e *Metric*. A tabela armazena 75.257.800 registros;
- *ProjectSmellFixed:* representa uma ocorrência sumarizada de um *Smell* que foi corrigido por uma ou mais refatorações (*Refactoring*).

Baseado na obtenção desse conjunto de dados, a composição das análises estão quantificadas da seguinte forma:

- Projetos com versões publicadas: 478 projetos com *Tags* registradas;
- Projetos com apenas uma versão publicada: 50 projetos registrados;
- Quantidade de versões: 11.593 *Tags*;

- Média de versões por projeto: ≈ 27.

Apresentar somente a abstração do quantitativo das entidades alocadas no domínio faz dessa uma apresentação indireta, sem representação dos respectivos relacionamentos do conjunto de dados. Logo, será disposta um visualização de introdução, bem como, as associações realizadas por meio de visualização de grafos. Essa análise tem por objetivo apresentar um histórico de registro dos projetos e de um único projeto, a fim de permitir a análise detalhada do banco de dados. Além disso, tem por finalidade responder e atender as perguntas de pesquisas levantadas neste trabalho:

1. Como visualizar, a partir da análise e classificação dos dados referentes à medidas de *software*, as características e projetos necessários para estabelecer relação entre refatorações e medidas de *software*?

2. É possível avaliar visualmente a evolução e mudanças nos projetos após sofrerem processo de refatoração?

4.1.2 Análise qualitativa

Para que a análise não fosse totalmente comprometida, foi selecionado um conjunto de 100 projetos para incorporar na base de dados no *Tableau* e *Neo4j*. O conjunto de dados foi incorporado com apoio do MER e incorporar todas as tabelas para que os relacionamentos fossems atribuídos e e o *dataset* modelado.

Logo, no trabalho de OLIVEIRA (2020), o conjunto de dados ele foi concebido com ferramentas que apoiavam em detecção de refatoração em código e captura de boas práticas de medidas de *software*, sendo elas: *SmartSmell*, *mauricioaniche/ck* e *RefactoringMiner*.

Cada ferramenta de identificação de refatoração suporta em sua estrutura analisar respectivos *code smells*. Diante disso, na Figura 8, a ferramenta *SmartSmell* (vértice vermelho) contempla e suporta dezenove tipos de *code smells* (vértice amarelo).

Já a ferramenta de *mauricioaniche/ck* tem por objetivo detectar e calcular medidas de *softwares* em classes, métodos, atributos e variáveis em projetos com código Java, conforme abordado na pesquisa de OLIVEIRA (2020). Persiste então um relacionamento entre as entidades de *Tool* e *Metric*, que o mesmo se dá pela composição de medidas suportadas pela ferramenta de medida de *software*. Desse modo, a ferramenta CK (vértice verde) abrange um escopo de

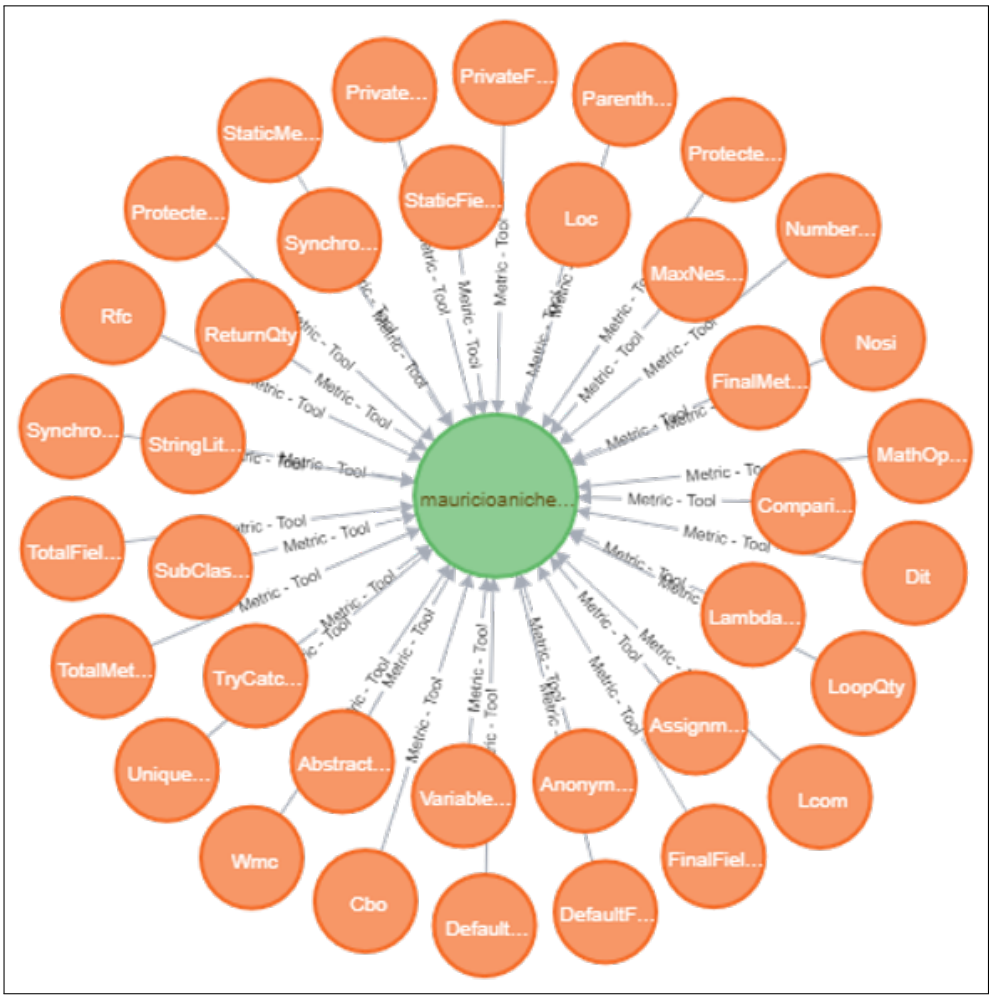
Figura 8 – Relação da Ferramenta SmartSmell e tipos de Smells

Fonte: A autora (2021)

37 medidas (vértices laranjas) suportadas, dentre elas a tabela armazena as medidas CBO, RFC, LCOM, DIT, WMC, LOC e *NumbersQTY*. As medidas são utilizadas como atributos para compor a base de conhecimento das ferramentas de análises, conforme é apresentado na Figura 9.

Conforme já mencionado, o estudo realizado por CARNEIRO (2003) apresenta um conjunto de medidas de *software*. A partir desse conjunto, esta pesquisa aborda as seguintes medidas: CBO que analisa a comunicação, atribuindo medidas de coesão e reuso; RFC que tem por objetivo identificar a complexidade e acoplamento, contabilizando o número de invocações de métodos exclusivas em uma classe; LCOM que verifica a coesão, complexidade e encapsulamento no uso de variáveis; DIT verifica herança, contabiliza o número de pais da classe, assim como, reuso, compreensão e teste das classes; WMC analisa a complexidade, tamanho, esforço para manutenção e o reuso; LOC e *NumbersQty* verificam o tamanho das classes e métodos.

Figura 9 – Relação da Ferramenta Mauricioaniche/CK e medidas suportadas



Fonte: A autora (2021)

A Figura 10 ilustra a ferramenta *RefactoringMiner* (vértice verde) e os tipos de refatoração (vértices azuis) suportados. Esta é a base para compreender os tipos de refatorações avaliados pela ferramenta em determinados códigos de projetos.

Nessa relação observamos: acoplamento, coesão, complexidade e tamanho. Em um primeiro momento, foram extraídos dos relacionamentos os projetos com características de acoplamento a fim de analisar o relacionamento entre os módulos do projeto, como também, identificar o quanto um depende do outro para funcionar. A Figura 11 apresenta a medida CBO (vértice laranja), a qual é suportada pela ferramenta *mauricioaniche/ck* (vértice vermelho), bem como, a referência da relação *ProjectMetric* (representados pelo id da entidade ProjectMetric - vértices verdes) diretamente associados à medida de acoplamento.

Além da medida CBO, o acoplamento pode ser identificado também pela medida RFC, que tem por responsabilidade identificar o acoplamento e complexidade por meio das chamadas das

Figura 10 – Relação da Ferramenta RefactoringMiner e os tipos de Refatoração

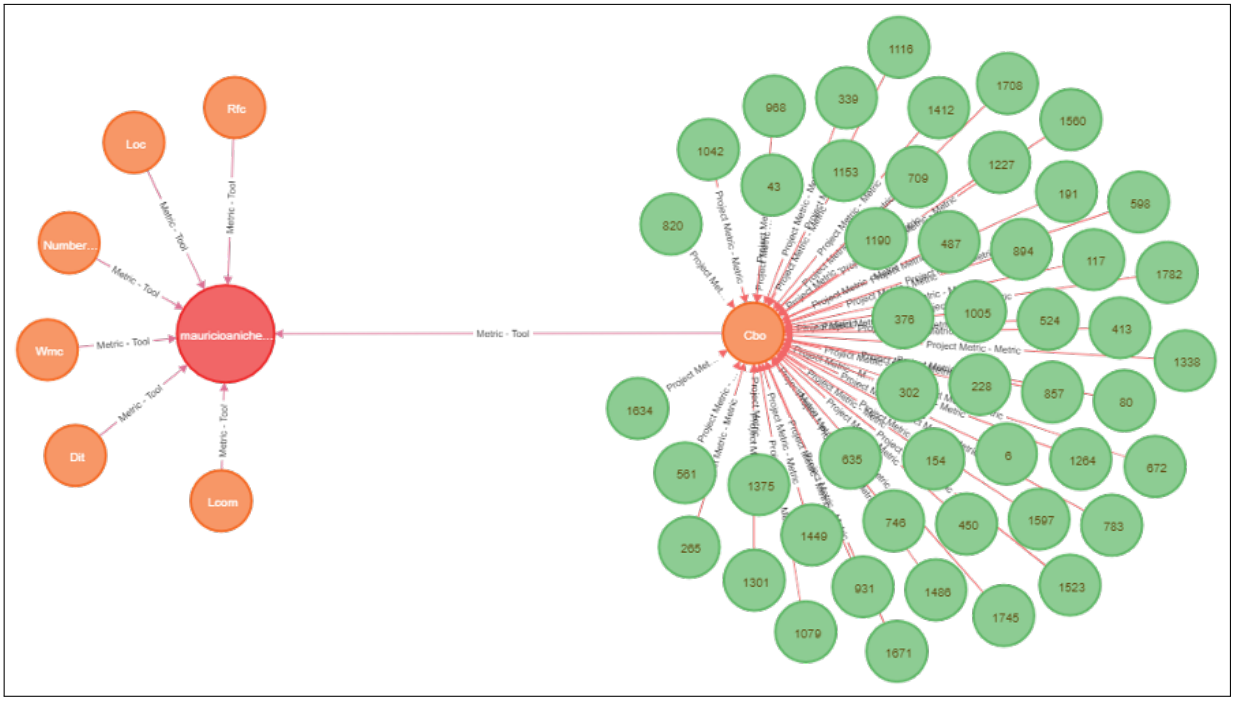
Fonte: A autora (2021)

classes. A Figura 12 apresenta também parte a associação existente entre Projetos e Medidas (*ProjectMetric* - vértices verdes) que contemplam a associação entre a medida RFC (vértice laranja) na base de conhecimento.

Outra medida explorada foi a de coesão, que pode ser identificada pela LCOM e CBO. Tais medidas têm o objetivo de medir a coesão do código, como também a complexidade, encapsulamento e uso de variáveis. A fim de representar os projetos que detêm medidas de coesão, a Figura 13 apresenta a atribuição da medidas LCOM (vértice laranja) para com a ferramenta (vértice vermelho) suportada e *ProjectMetrics* associados (vértices verdes).

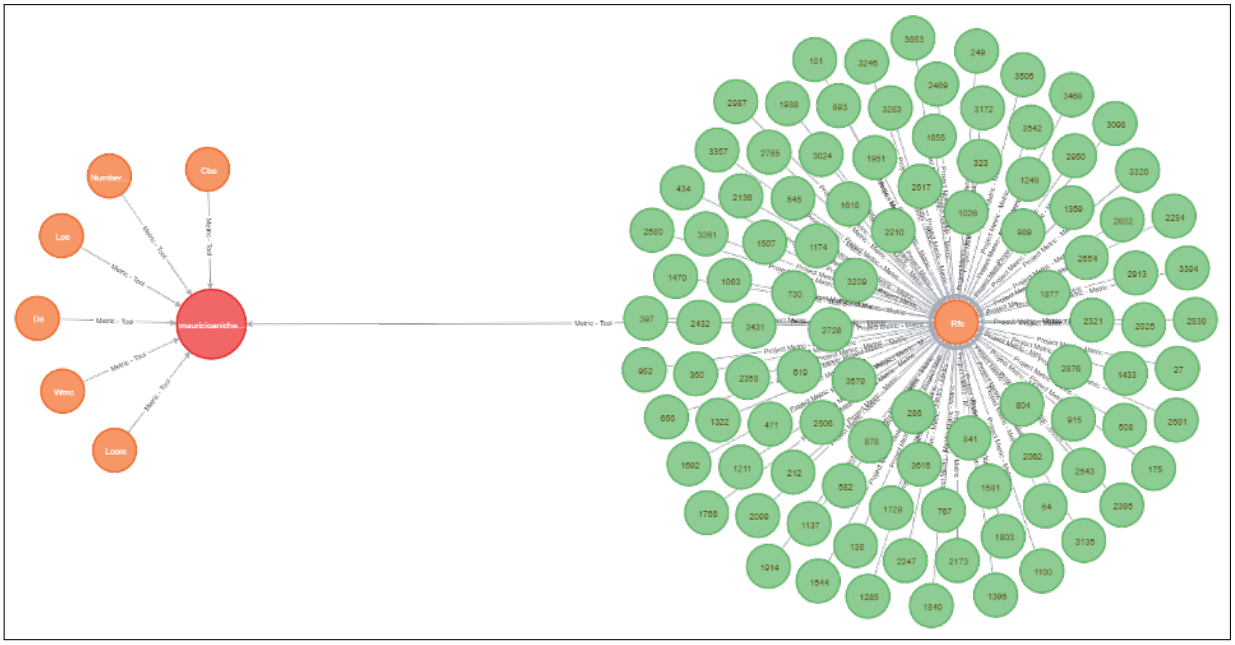
Ademais, é possível analisar medidas de complexidade, como DIT e WMC, que são responsáveis por medir o esforço, complexidade e tamanho por meio de dependências entre as classes. As Figuras 14 e 15 ilustram a relação de responsabilidade das entidades *Metric*, *Project* e *Tool* utilizadas no processo de refatoramento. Assim como as medidas citadas acima, as Figuras 14 e 15 indicam os *ProjectMetrics* (ids da entidade de relacionamento - vértices

Figura 11 – Relação dos projetos com a medida CBO



Fonte: A autora (2021)

Figura 12 – Relação dos projetos com a medida RFC

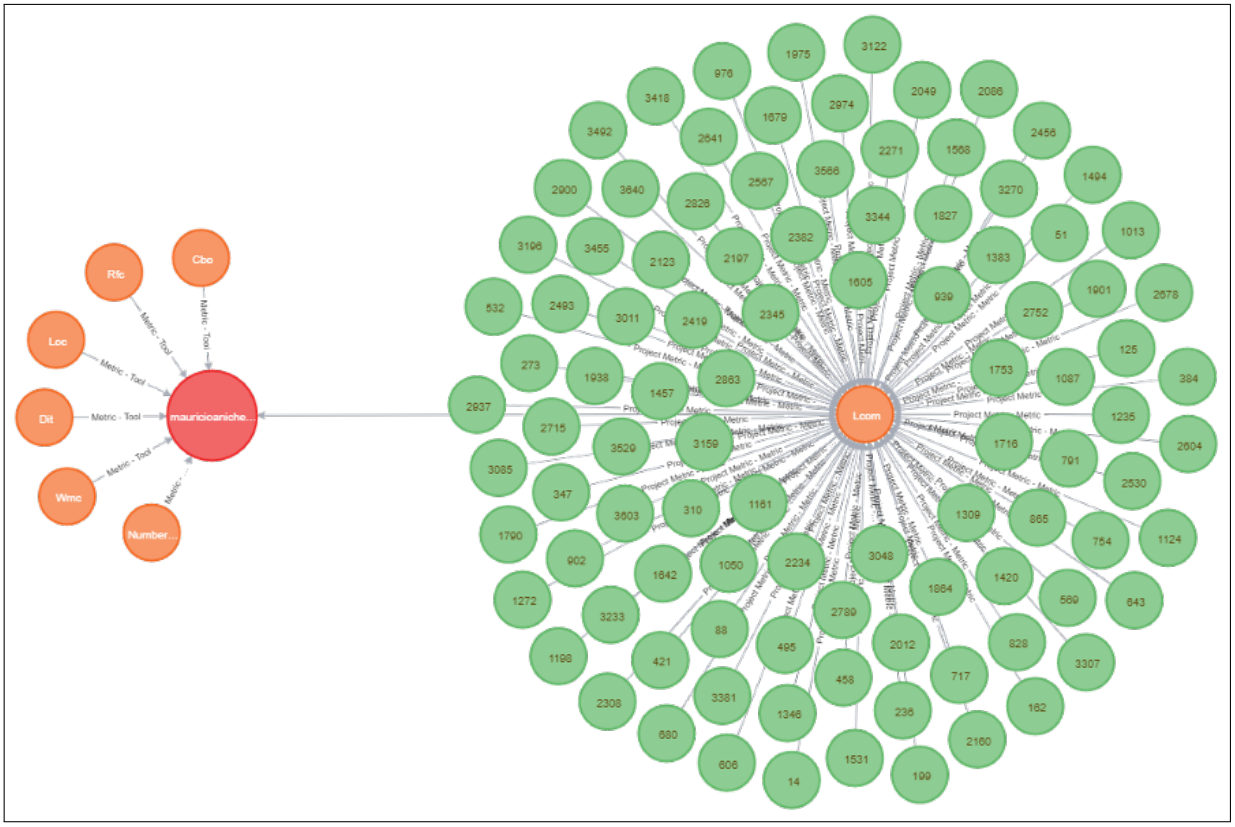


Fonte: A autora (2021)

verdes) foram detectados medidas de DIT e WMC (vértices laranjas), como também, por meio de qual ferramenta (vértice vermelho) foi detectada.

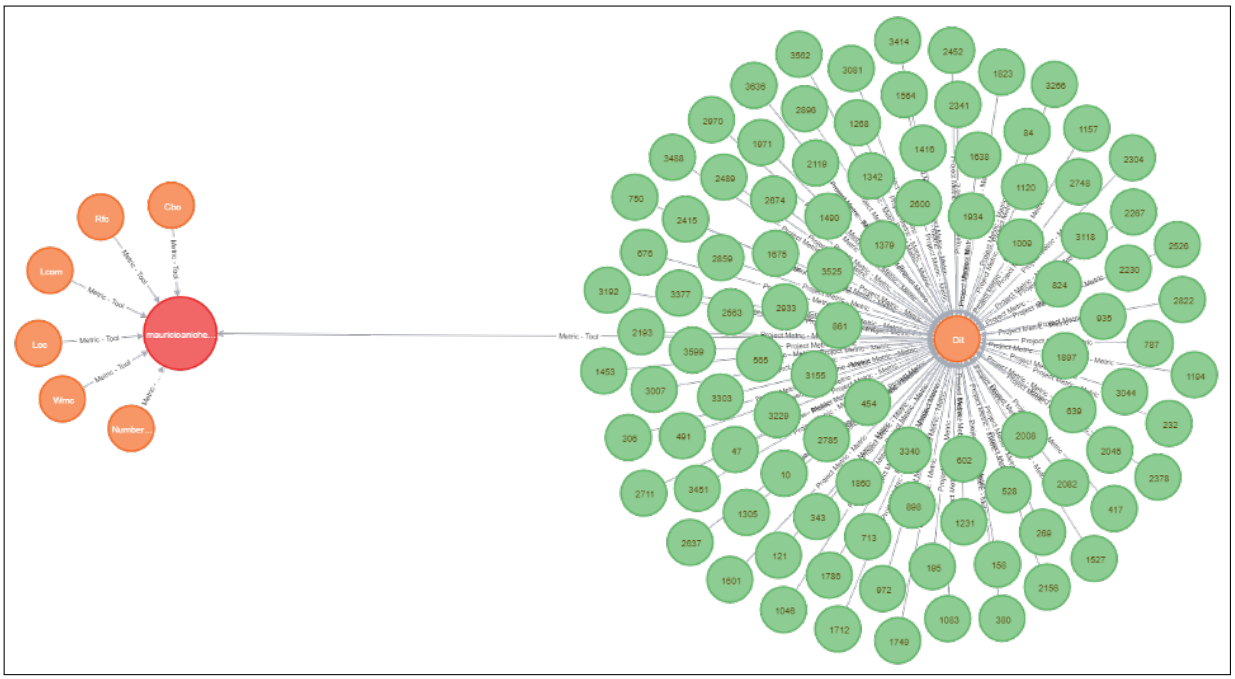
A partir da agregação dos *ProjectMetrics* (vértices verdes), *Tool* (vértice vermelho) e *Metric* (vértices laranjas) foi possível capturar características de medidas de tamanho. As

Figura 13 – Relação dos projetos com a medida LCOM



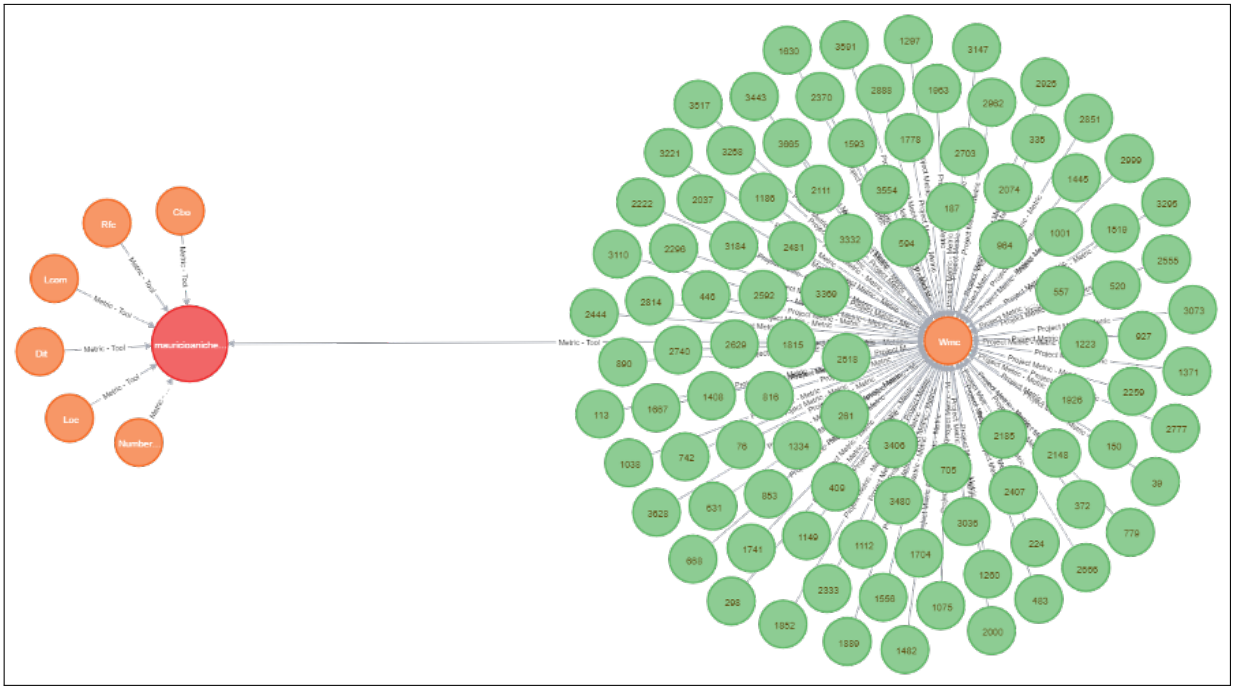
Fonte: A autora (2021)

Figura 14 – Relação dos projetos com a medida DIT



Fonte: A autora (2021)

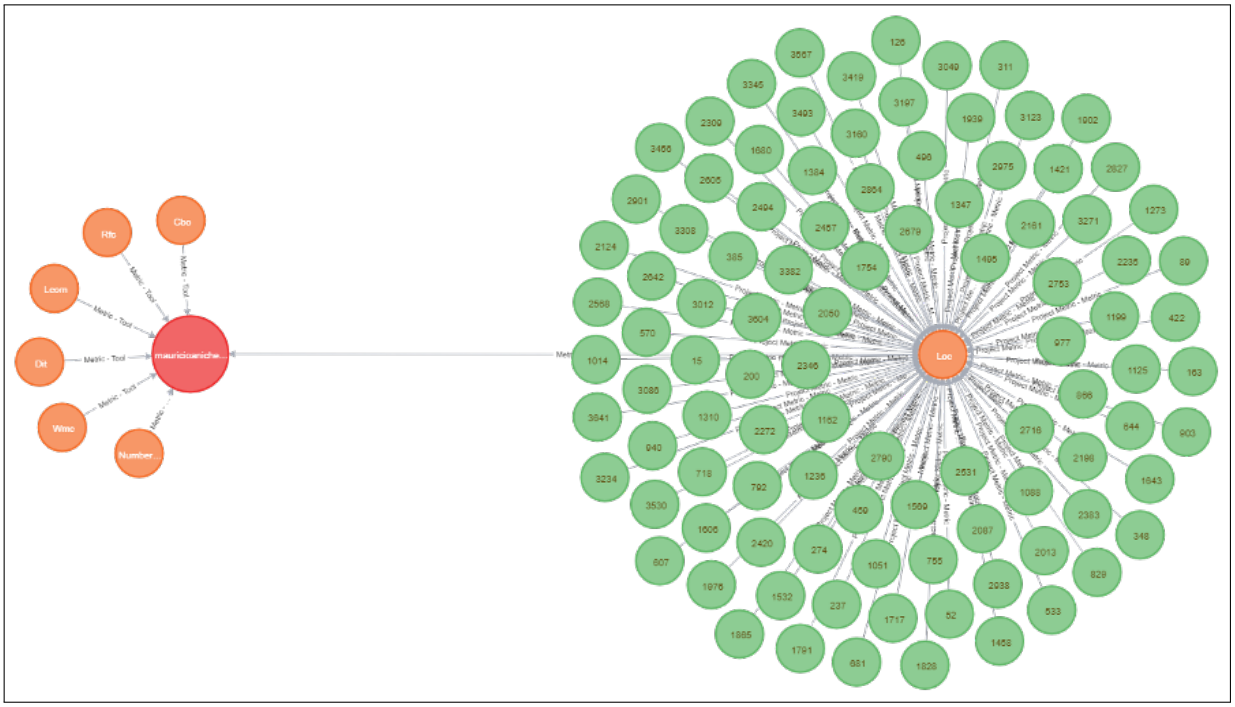
Figura 15 – Relação dos projetos com a medida WMC



Fonte: A autora (2021)

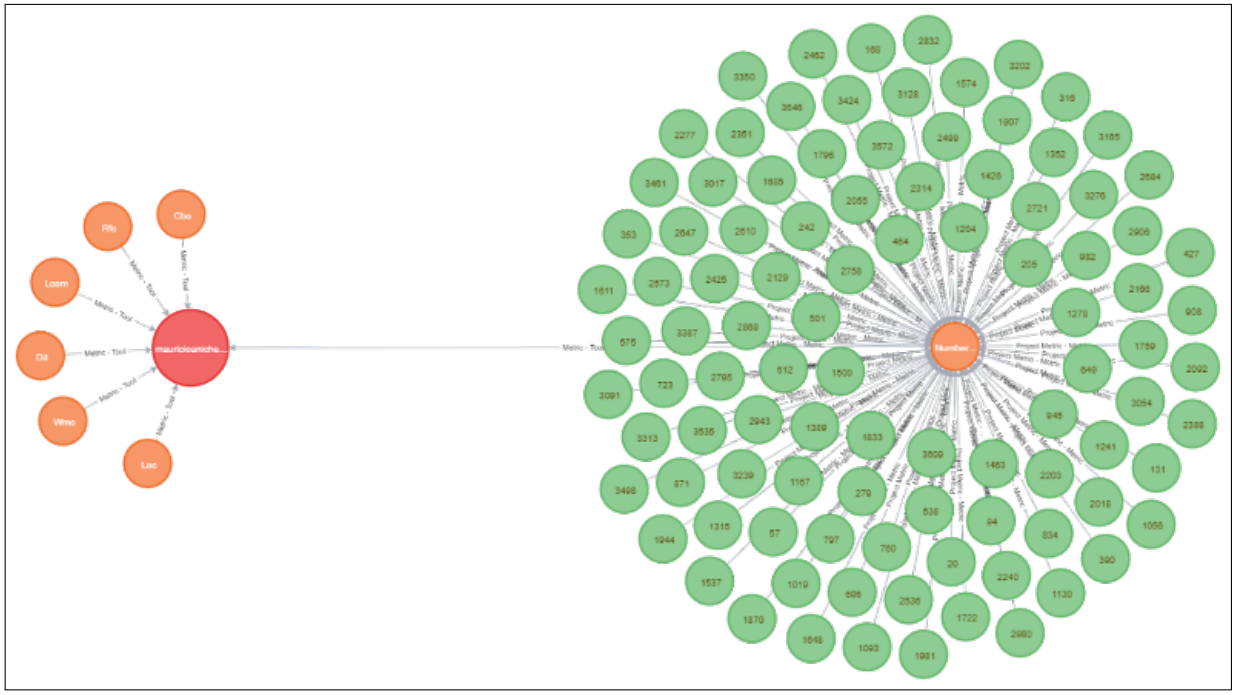
medidas responsáveis por essa característica são LOC e *NumbersQTY* (vértices laranjas). Tais resultados são ilustrados nas Figuras (16 e 17).

Figura 16 – Relação dos projetos com a medida LOC



Fonte: A autora (2021)

Figura 17 – Relação dos projetos com a medida NumbersQTY



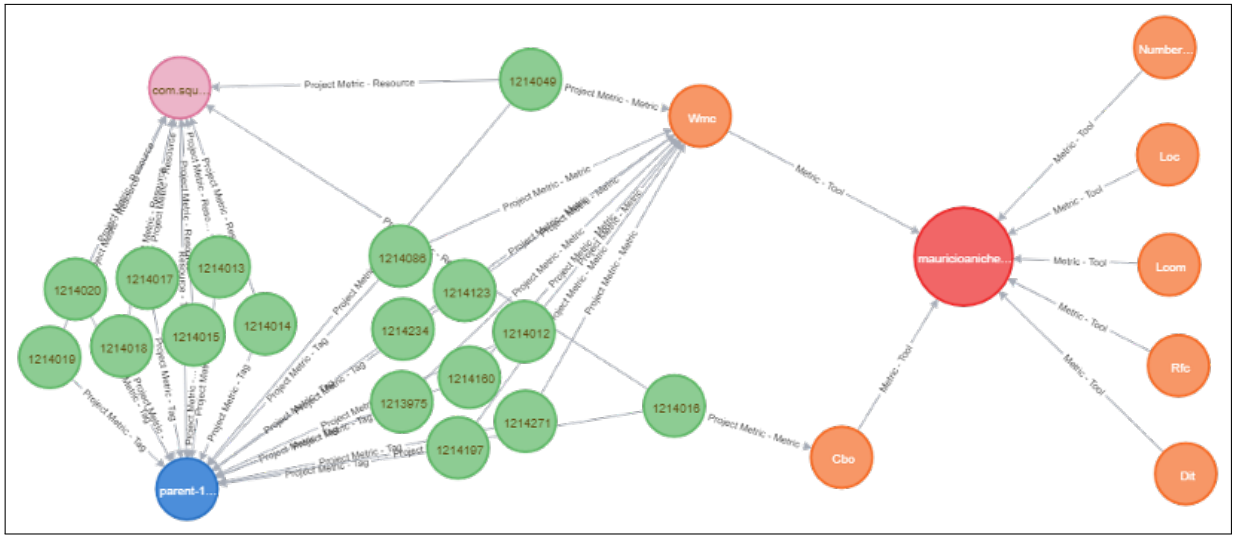
Fonte: A autora (2021)

A fim de apresentar a relação das responsabilidades por meio dos recursos e medidas, foi selecionado um projeto aleatório, com uma lista reduzida de versões, listadas das entidades *Metric* e *Resource* que foram aplicadas no projeto. Além disso, foram constatados alguns casos em que são identificados pelo menos dois tipos de medidas de *software* para um único projeto, bem como, apresentar uma dependência entre os recursos e o projeto analisado.

Na Figura 18, é possível observar que com a ferramenta *mauricioaniche/ck* (vértice vermelho) pode-se extrair duas medidas associadas, WMC e CBO (vértices laranjas). Para as medidas suportadas por essa ferramenta observa-se os *ProjectMetrics* (vértices verdes) correlacionados, assim como, a atribuição para com o registro (vértice rosa - entidade *Resourse*) que foi identificado, ou seja, movido ou renomeado. Baseado nisso, pode-se considerar que as versões (*Tags* - vértice azul) associadas ao projeto representado obtém-se de tratamento de medidas de complexidade e coesão de código. Com isso, é possível identificar o tamanho, esforço e reuso do código com refatoração, uma vez que foi possível identificar pela ferramenta Ck que avalia o código para analisar o histórico do projeto.

Na Figura 19, observamos que a ferramenta *mauricioaniche/ck* extrai duas medidas associadas, WMC e DIT. Portanto, para as medidas (vértices laranjas) suportadas por essa ferramenta (vértice vermelho), as versões (vértice azul) do projeto que estão correlacionadas, indicando a atribuição para com o registro (vértice rosa) que foi alterado. Fundamentado

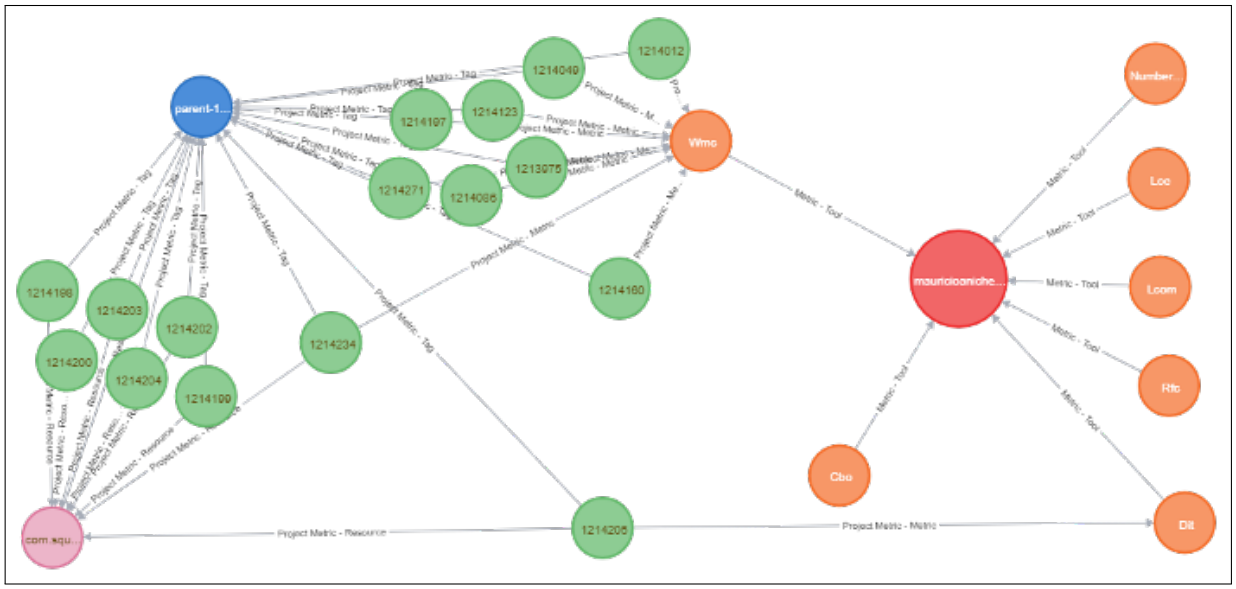
Figura 18 – Relação das medidas WMC e CBO



Fonte: A autora (2021)

nisso, pode-se considerar que as versões (vértices azuis) marcadas no projeto compreendem de características de complexidade (vértices laranjas). Desse modo, com a competência de tratar e medir o esforço realizado no código.

Figura 19 – Relação das medidas WMC e DIT

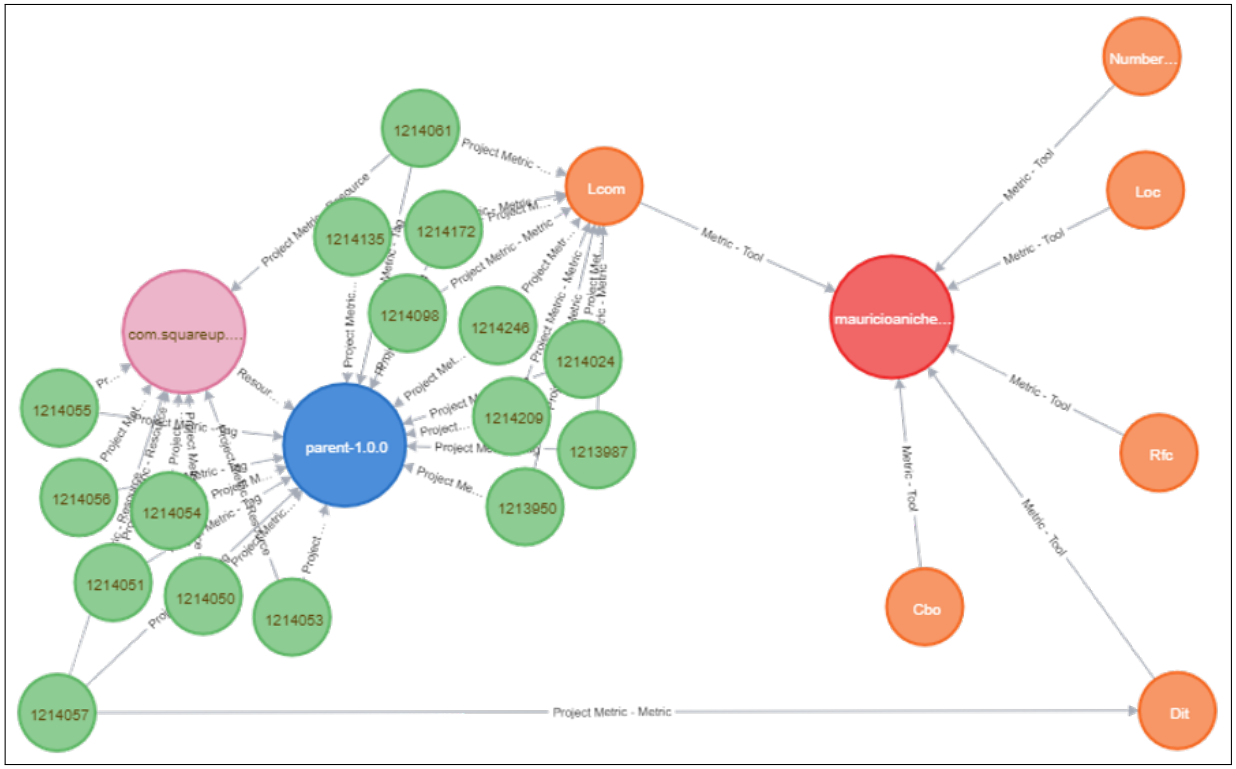


Fonte: A autora (2021)

A Figura 20, as medidas de LCOM e DIT estão com responsabilidades de identificar e realizar abordagens de coesão, complexidade e encapsulamento do código. Logo, é possível visualizar a relação dos *ProjectMetrics* (vértices verdes) com o registro (vértice rosa) e suas versões (vértice azul), que foram abordados com as medidas de coesão e complexidade (vértices

laranjas).

Figura 20 – Relação das medidas LCOM e DIT



Fonte: A autora (2021)

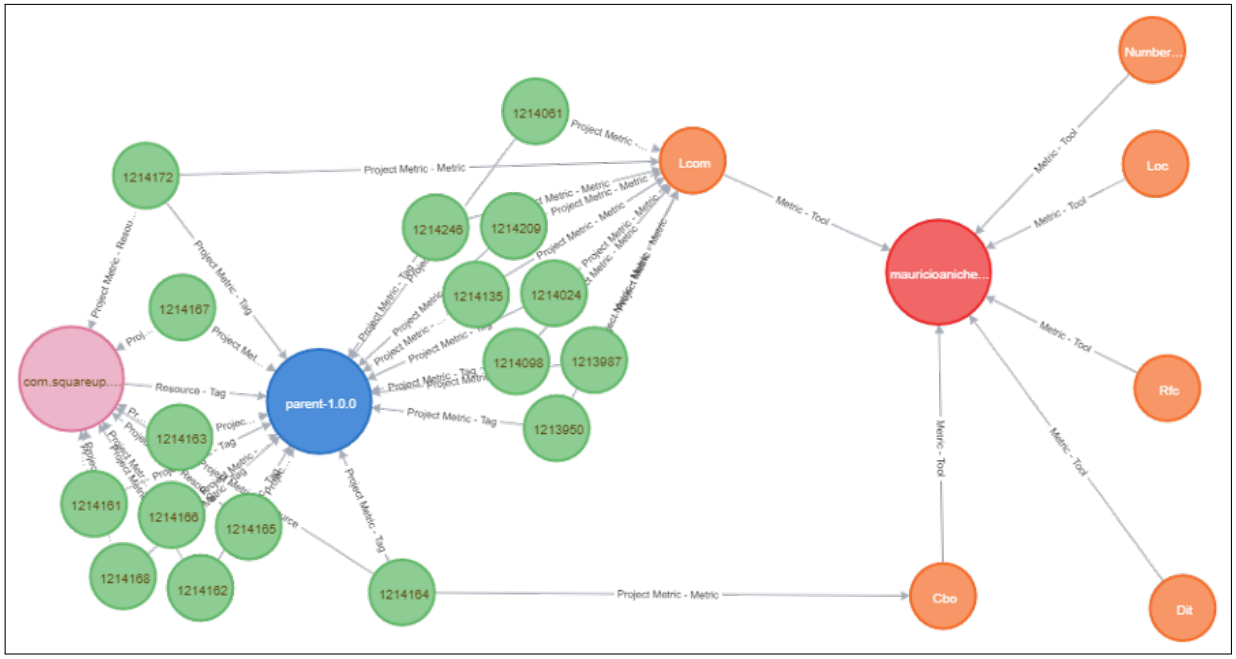
Em suma, na Figura 21 retrata a relação de medidas com coesão e acoplamento do código. Bem como, atributos de complexidade, encapsulamento e reuso de código.

Com essas correlações foi possível compreender o papel das medidas e alterações realizadas em determinado projeto (Figura 28), com o intuito de conceder responsabilidades e proporcionar uma abordagem assertiva, para analisar as características compostas nos projetos, por meio das correlações trabalhadas entre as entidades *Metric*, *Resource* e *Tag*.

No conjunto de projetos consolidados para essa análise foi possível detectar a classificação das *tags* associadas aos projetos. A Figura 26 ilustra a classificação atribuída, tais *tags* são caracterizadas como processadas ou não.

Da população coletada de 725 projetos, foram identificadas 11.968 versões interligadas compostas no servidor do *Github*. Dessas versões, conforme apresenta a Figura 26, 97,28% estão como processadas, ou seja, o *status* de *processed* é distinguido como *true*. Enquanto isso, as versões não publicadas contam com uma amostra de 2,72% (325 versões não processadas). Baseado na representatividade dos *status* das versões dos projetos, foi possível associar as versões conforme a classificação de seus status no projeto, como evidencia a Figura 26, bem

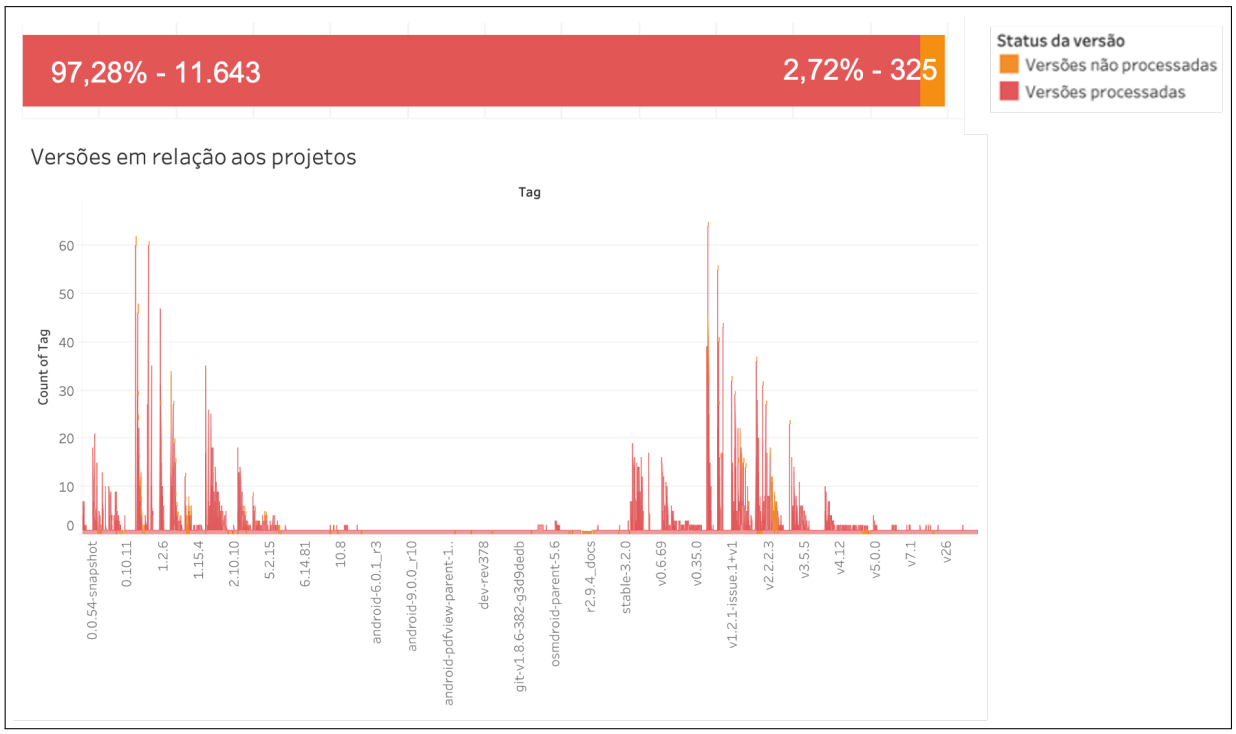
Figura 21 – Relação das medidas LCOM e CBO



Fonte: A autora (2021)

como distinguir as versões processadas das versões não processadas.

Figura 22 – Relação dos projetos e versões



Fonte: A autora (2021)

Concordante com as informações expostas, as versões processadas correspondem a um quantitativo mais representativo no que tange à distribuição dos projetos em relação às versões,

totalizando cerca de 478 projetos com *tags* registradas. Porém, as versões não processadas têm por média de uma *tag* por versão não publicadas, contabilizando uma amostra de 50 projetos com pelo menos uma versão registrada.

Para se fazer possível identificar e classificar as análises realizadas, os próximos resultados apresentados terão como objetivos evidenciar e categorizar os projetos da base de conhecimento com fatos que discriminam:

- existe algum tipo de refatoração;
- foi utilizado algum tipo de medida ou atributo para identificar refatoração;
- existe algum problema no código;
- é possível comparar a evolução por meio de versão(ões) do(s) código(s) refatorado(s);

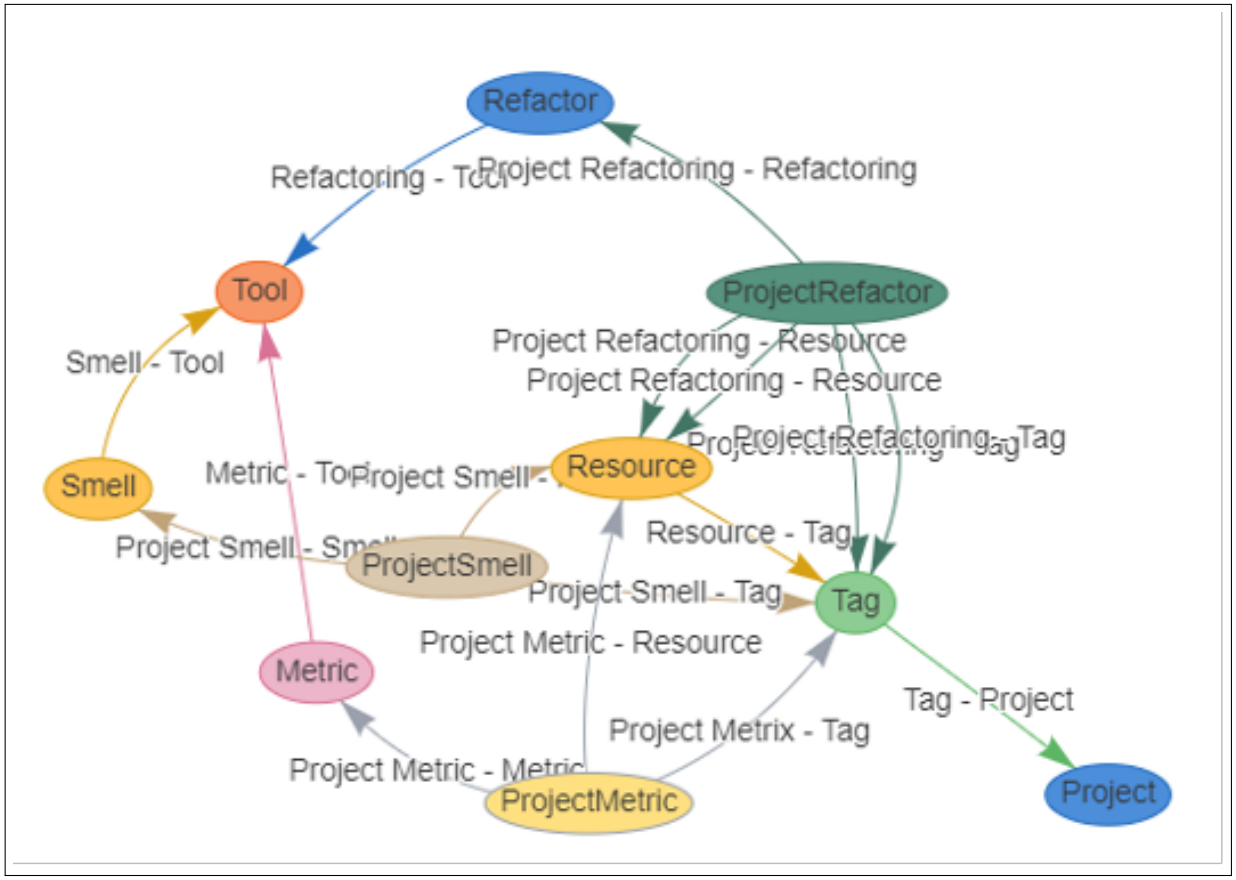
A fim de atender os aspectos que foram discriminados, se fez necessário analisar os relacionamentos entre as entidades do conjunto de dados, conforme a apresentação na Tabela 5 e Figura 23.

Tabela 5 – Relacionamento entre as tabelas da base de dados

Variável inicial(Nó inicial)	Variável final(Nó final)	Relacionamento
project_refactoring	resource	Project Refactoring - Resource
project_smell	tag	Project Smell - Tag
project_metric	resource	Project Metric - Resource
tag	project	Tag - Project
metric	tool	Metric - Tool
smell	tool	Smell - Tool
project_metric	tag	Project Metric - Tag
project_refactoring	tag	Project Refactoring - Tag
project_smell	smell	Project Smell - Smell
refactoring	tool	Refactoring - Tool
project_metric	metric	Project Metric - Metric
resource	tag	Resource - Tag
project_refactoring	refactoring	Project Refactoring - Refactoring
project_refactoring	tag	Project Refactoring - Tag

A refatoração pode ser identificada pela presença de refatoramentos que estão detalhados na tabela de *ProjectRefactoring*. De acordo com CARNEIRO (2003), as refatorações podem ser classificadas por meio de três níveis de abstração, sendo elas a procedural, orientada à objeto e

Figura 23 – Ilustração das relações entre os nós



Fonte: A autora (2021)

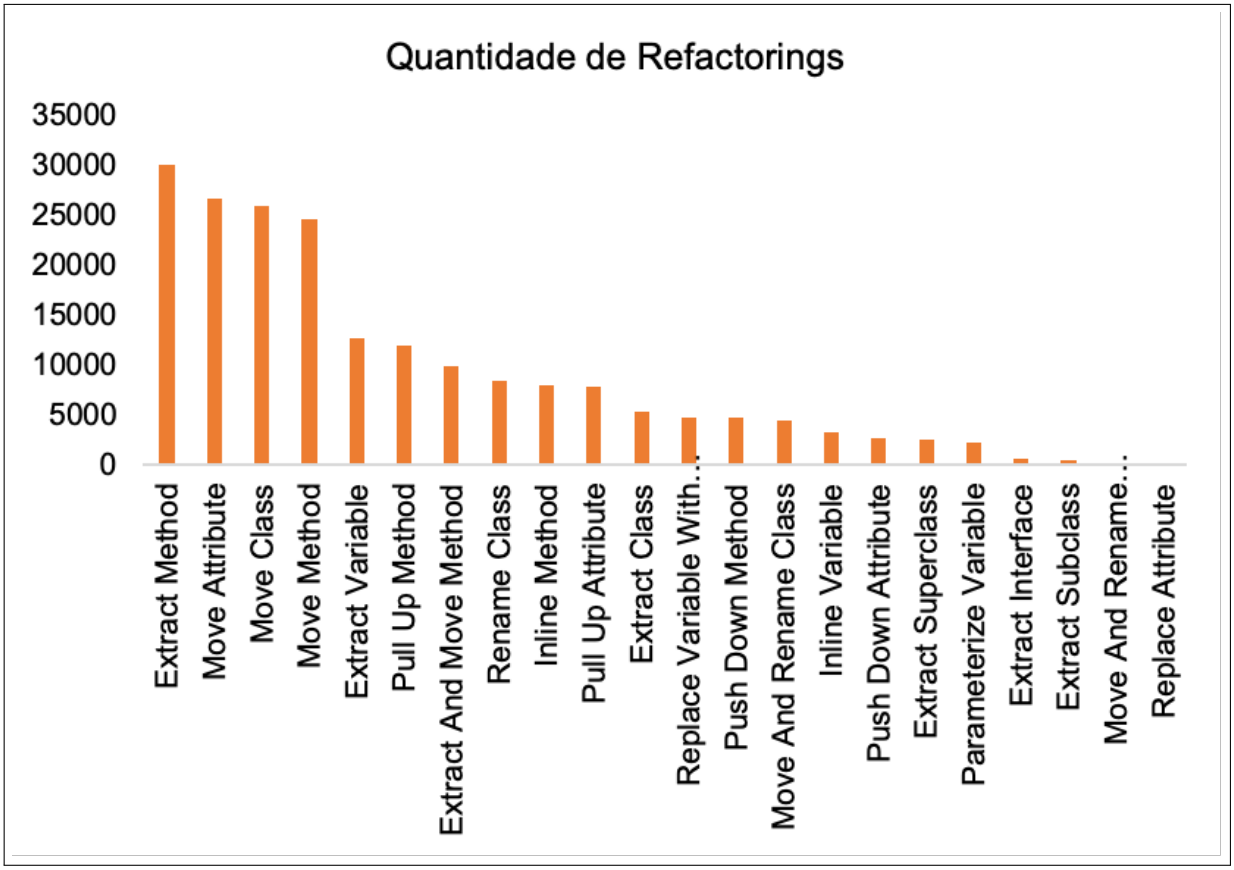
conceitual. A refatoração procedural representa o código, como por ex: *Composing Methods*. A orientada à objeto atua de forma direta no projeto do sistema e afeta a sua estrutura, como por ex: *Moving Features between Object*s. Logo, a refatoração conceitual atua na modificação de conceitos do sistema, como *Organizing Data*.

No cenário levantado nesta pesquisa foram analisados 197.525 refatorações existentes, conforme os agrupamentos da entidade *Refactoring*. Com isso, foi possível a representação das refatorações, ilustradas por meio da Figura 24 e no Anexo **??**.

Durante esse processo de análise, observamos medidas relativas à coesão, acoplamento, complexidade e tamanho como indicadores relativos a refatorações. Ademais, o uso de suas medidas, que concerne em CBO, RFC, LCOM, DIT, WMC, LOC e *NumbersQTY*. Esses atributos e medidas contribuíram para a composição da análise para identificar possibilidades de refatorações nos projetos da base de dados explorada.

Dentre os 725 projetos, 474 possuem *code smells*, totalizando 4.738.238 *code smells* nos recursos dos projetos, conforme ilustrado na Figura 25 e no Anexo **??**.

Figura 24 – Quantidade de Refactorings Identificados



Fonte: A autora (2021)

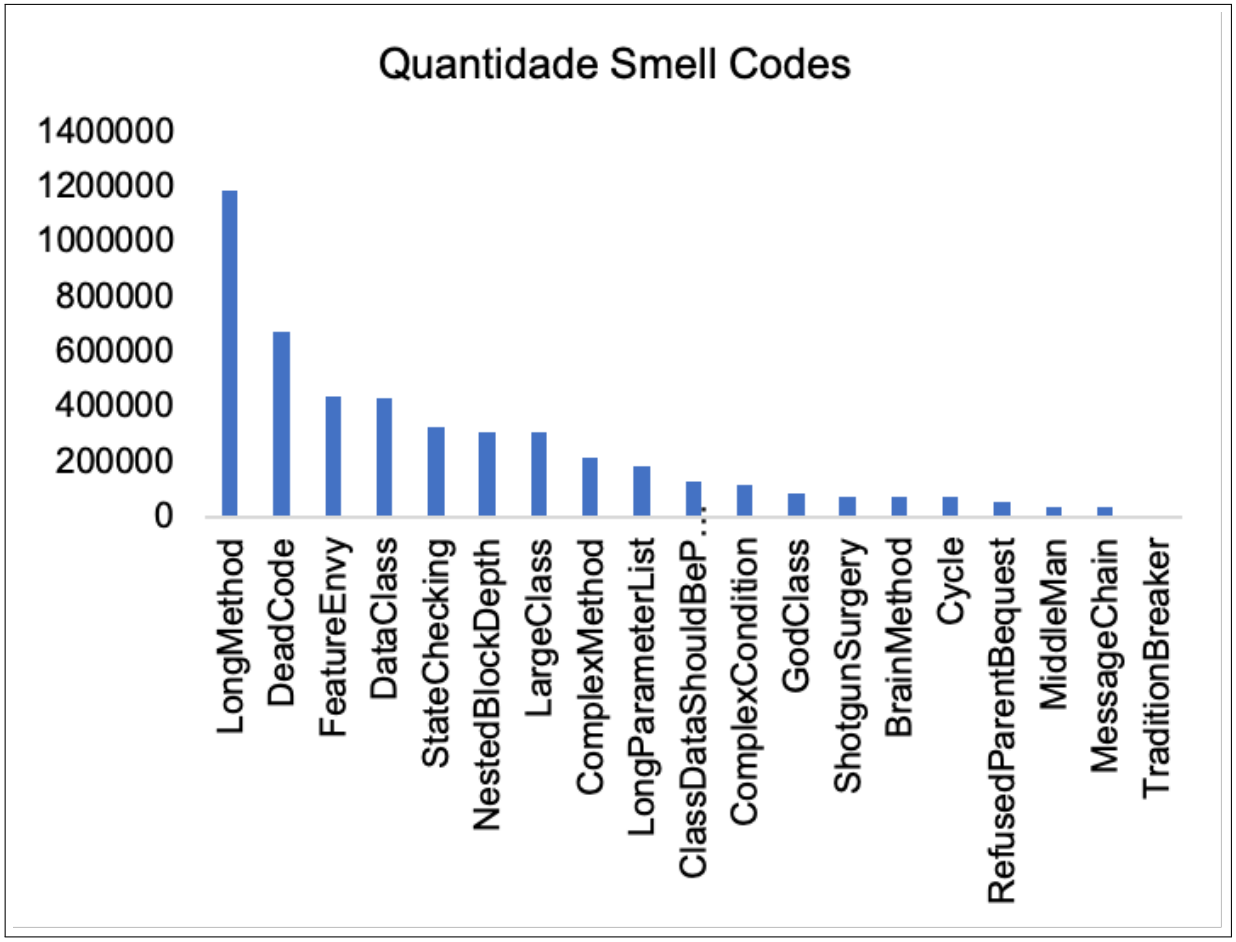
Assim como ilustrado na Figura **??**, dentre os projetos analisados, o *Lawnchair* é o que possui maior número de *code smells*, com um total de 810.270. O *ranking* com os 10 projetos com mais *code smells* também pode ser verificado na Figura **??**.

Em relação à base de dados explorada, os projetos com menor número de *code smells* foram: *seismic*, *VerticalViewPager*, *Cockroach* e *Android-StepsView*, com o quantitativo de 1 *smell* cada. Na Figura 27 ilustra o *ranking* com os 10 projetos com menos *code smells*. Que foi possível detectar após análise realizada pela ferramenta Neo4j, a fim de auxiliar na tomada de decisão em relação ao acompanhamento e desenvolvimento dos projetos.

A fim de representar o histórico do projeto, se fez necessário atribuir uma relação entre as entidades *Project*, *Tag* e *Smells*. Então, constatamos que o projeto *Lawnchair* possui 500 *tags* identificadas. Desse modo, a *tag* android-9.0.0_r11 surge com a maior quantidade de *smells*, com 374 identificados como *LongMethod*.

Diante das análises de classificações apresentadas, foi possível identificar que as características necessárias para estabelecer a relação entre refatorações e medidas são os *Refactoring*(s), *Tag*(s), *Resource*(s), *Tool*, *Metric*(s) e o(s) *Projeto*(s) associados. Além disso, nas Figuras 28

Figura 25 – Quantidade de Code Smells Identificados nos Projetos



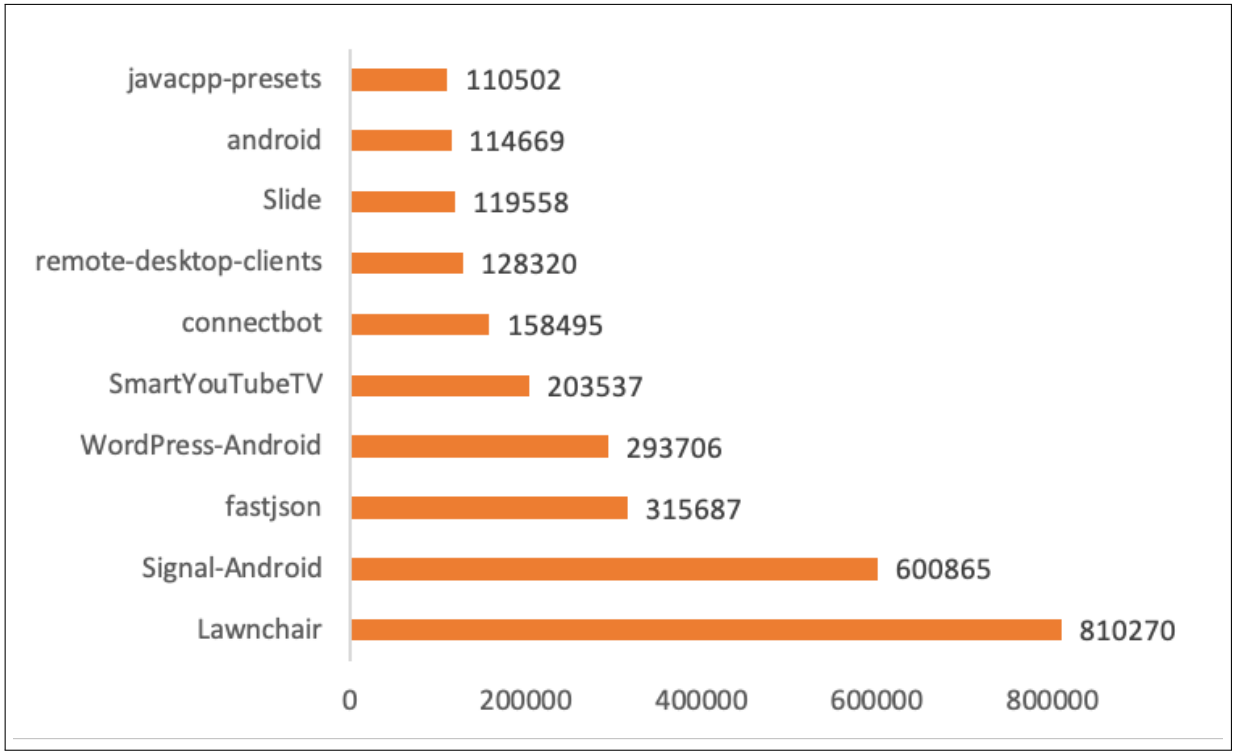
Fonte: A autora (2021)

e 29, exploramos um pouco mais o contexto dos projetos, no que tange a cadeia de relacionamento das entidades.

De acordo com a demonstração da Figura 29, se fez provável representar a entidade *Project*, identificando cada versão (*Tag*) associada. Além disso, percebemos a existência de um novo recurso(*Resource*) alterado, caracterizando alguma mudança no código fonte. Com uso das ferramentas, foi possível identificar a presença de *Smell*(s), medida(s) de *software* e *Refactoring*(s), por meio da navegação nos grafos. Tal como, apresentar o relacionamento com os recursos de *Tag*, *Refactoring*, *Smell* e *Metric* no projeto de cada repositório escolhido.

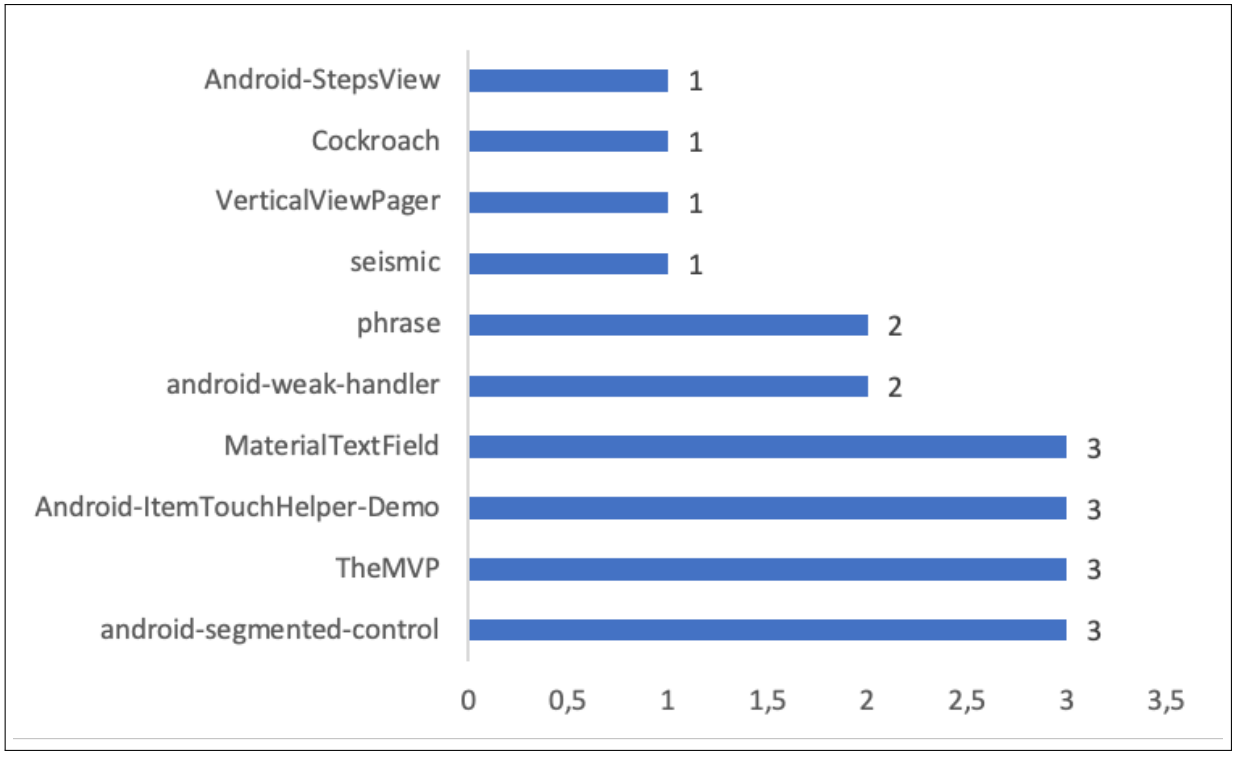
A fim de compreender a obtenção dos resultados, iremos descrever o caminho percorrido para se estabelecer a relação das tabelas da base de dados já sumarizada. Baseado nisso, dentre os projetos analisados podemos destacar o *Lawnchair*, que possui a maior quantidade de *smells* e como foi concedido o relacionamento com as demais entidades da base de dados analisada.

Figura 26 – Ranking de Projetos com mais Code Smells Identificados



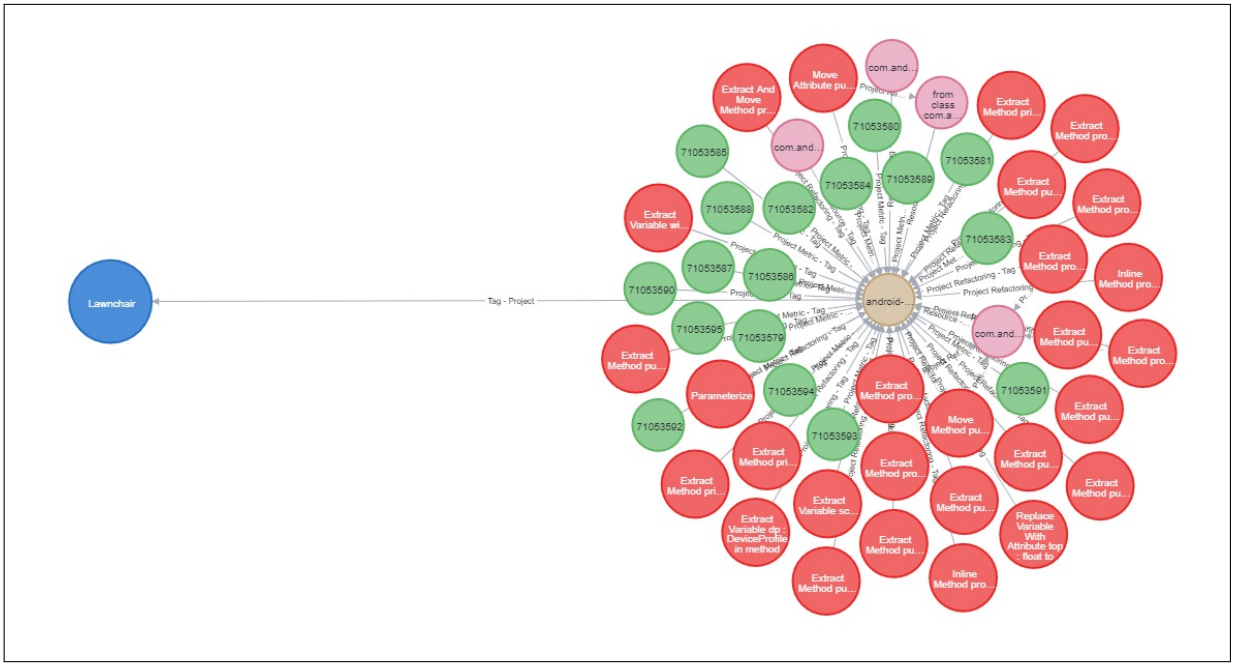
Fonte: A autora (2021)

Figura 27 – Ranking de Projetos com menos Smells Identificados



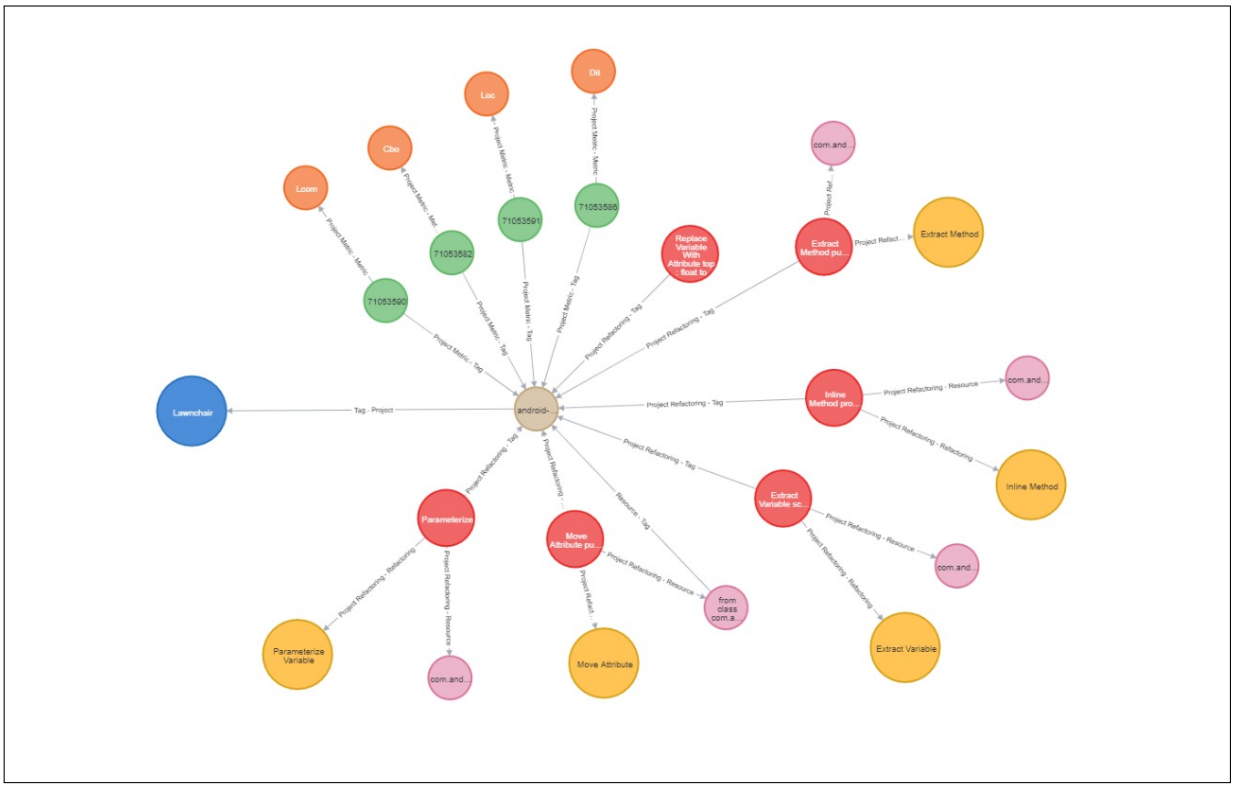
Fonte: A autora (2021)

Figura 28 – Projeto com a cadeia de relação dos *refactorings*



Fonte: A autora (2021)

Figura 29 – Relação de Project, Resource, Refactoring, Project Refactoring, Tag, Project Metric e Metric



Fonte: A autora (2021)

Para selecionar o projeto *Lawnchair*, utilizamos a seguinte instrução:

MATCH (n:Project{name:'Lawnchair'}) **RETURN** n (Figura 30)

Figura 30 – Seleção do Project Lawnchair

Fonte: A autora (2021)

Dentre as *Tag*'s do projeto *Lawnchair* a que possui mais ocorrências é a *android-9.0.0_r11*, possuindo a relação entre *Tag – Project* com o *Project*. Para selecionar a *Tag* e o *Project* associado, utilizamos da seguinte *query*:

MATCH p=(n:Tag{name:'android-9.0.0_r11'})-[:'Tag - Project']->() **RETURN** p

Onde será exibido a *Tag android-9.0.0_r11*. Como também, a relação *Tag – Project* e o *Project Lawnchair*, conforme a Figura 31.

Figura 31 – Relação de Project e Tag

Fonte: A autora (2021)

A *Tag* android-9.0.0_r11 possui relação com os nós *project_refactoring*, com a relação *Project Refactoring - Tag*, *Resource* com *Resource - Tag* e *project_metric* com *Project Metric - Tag*, identificando uma possível refatoração no projeto, conforme ilustrado na Figura 32:

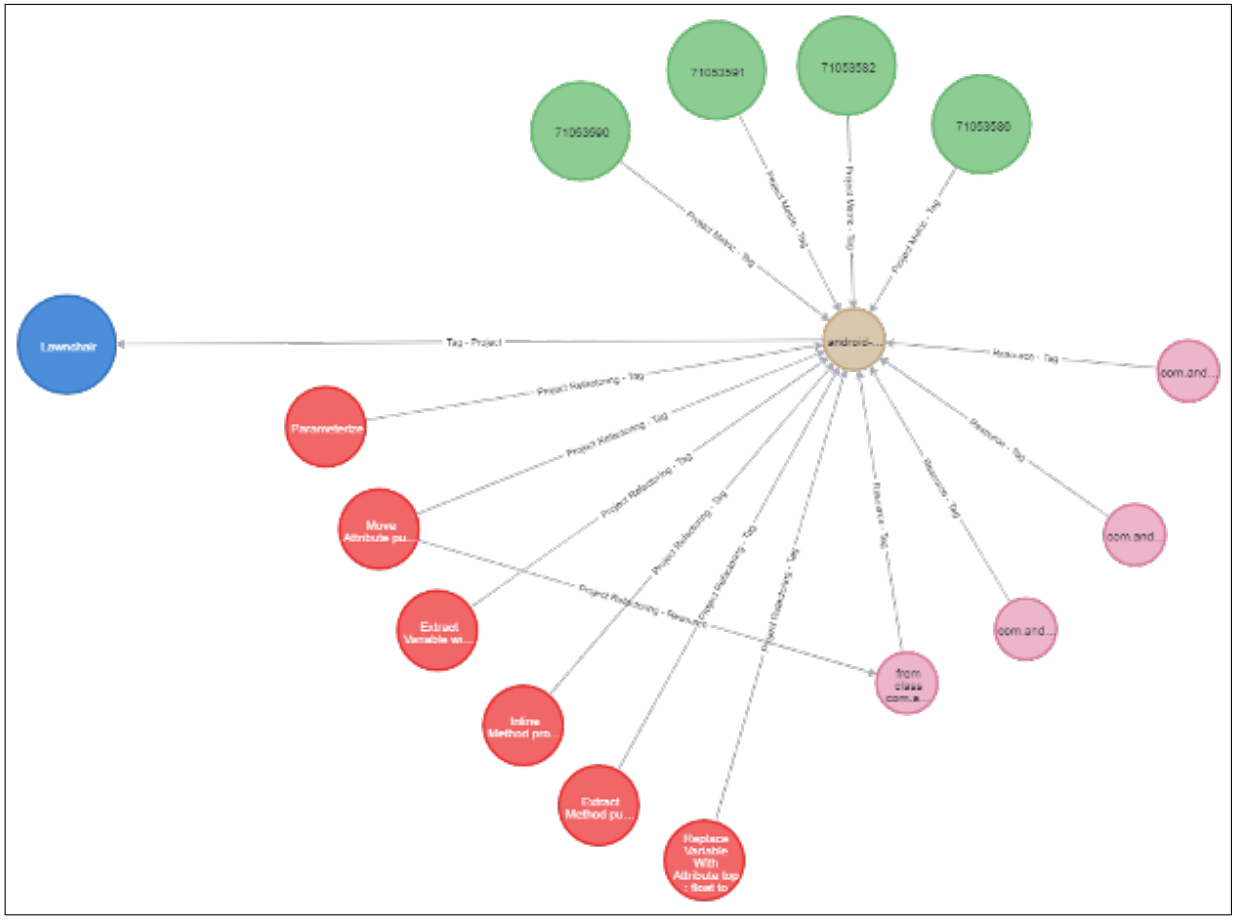
Cada *Refactoring* possui uma relação *Project Refactoring – Refactoring* com a tabela *Project_Refactoring* e em alguns casos com a tabela *Resource* com a relação *Project Refactoring - Resource*, como ilustrado na Figura 33.

Dentre os projetos que estão relacionados com a *Tag* android-9.0.0_r11, eles também se relacionam com a tabela *Metric* por meio da relação *Project Metric - Metric*, conforme ilustrado na Figura 34.

Dessa forma, foi possível identificar e classificar o projeto correspondente a determinadas medidas, bem como, as refatorações existentes, versões e recursos afetados.

Diante disso, verificar o impacto dessa refatoração, no que tange a(s) versão(ões), recurso(s). Diante dos resultados obtidos, as relações das tabelas resultou na exploração e explicação da base de conhecimento incorporada nas ferramentas *Tableau* e *Neo4j*, utilizadas

Figura 32 – Relação dos nós Project, Refactoring, Resource, Tag e Metric



Fonte: A autora (2021)

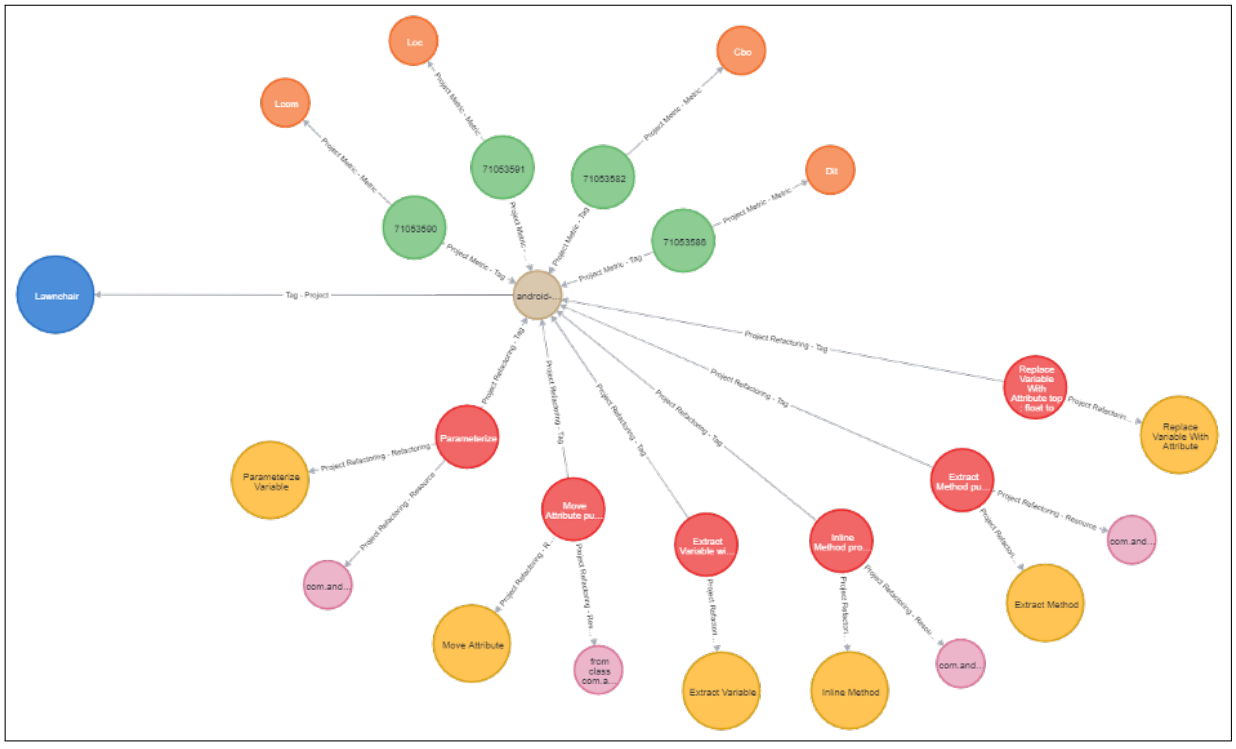
no processo de análise e visualização dos dados. A fim de auxiliar na gestão do conhecimento no contexto de projetos com refatoração e medidas de *softwares*.

Utilizando visualização de dados, foi realizada uma análise descritiva de visualização de dados. A partir disso, conseguimos identificar refatorações, assim como exploramos o histórico relacionado, adotando medidas de *software* viáveis para esse contexto.

Figura 33 – Relação dos nós Project, Refactoring, Resource, Tag e Metric

Fonte: A autora (2021)

Figura 34 – Relação dos nós Project, Refactoring, Resource, Tag e Metric



Fonte: A autora (2021)

5 CONCLUSÃO

Neste capítulo apresentamos nossas considerações finais, as principais contribuições, ameaças à validade, limitações e trabalhos futuros oriundos desta dissertação. O objetivo central deste trabalho consistiu em realizar uma análise descritiva de visualização de dados, a fim de identificar refatoração observando medidas de *software* ao longo da história do projeto de *software*.

Com o propósito de examinar o levantamento realizado no trabalho de OLIVEIRA (2020), com base na relação entre as medidas de *software*, foi possível proporcionar outras perspectivas para avaliar e comparar a evolução de projetos de *software*, no intuito de investigar se houve alguma melhoria com base aplicação de medidas de qualidade de *software* no código refatorado no projeto do *dataset* extraído e avaliado no trabalho de OLIVEIRA (2020).

Analisamos as relações dos dados extraídos, que permitiu identificar medidas de *software* para auxiliar na classificação e tomada de decisão dos projetos, bem como, apresentar análises descritivas e exploratórias por meio de ferramentas de visualização de dados. A possibilidade de uso dessas ferramentas permitiu a visualização e análise dos dados referente a projetos de *software*, com o objetivo de também explorar correlações existentes.

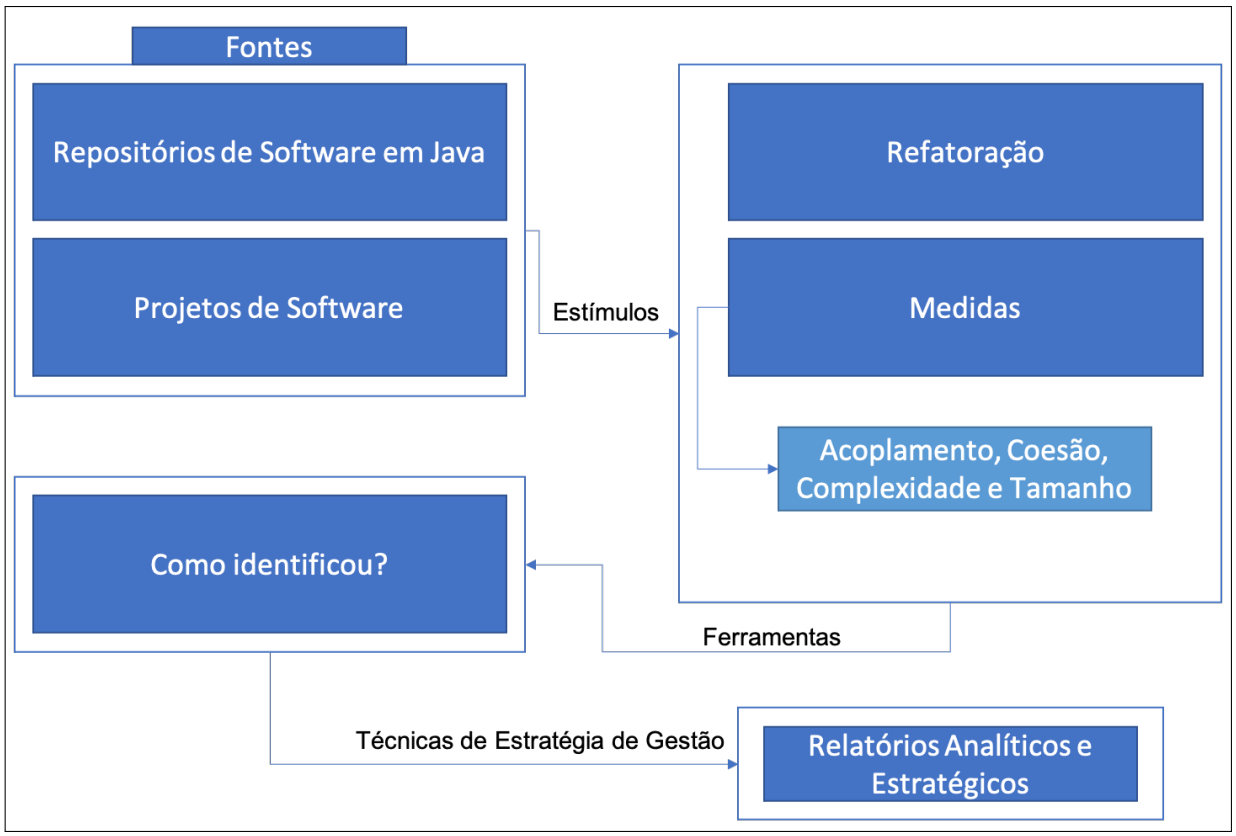
Também combinamos um conjunto de ferramentas de análise de dados (*Tableau* e *Neo4j*). Elaborar análises e gerar relatórios elaborados desta forma pode viabilizar a tomada de decisões, a fim de se possibilitar a tomada de decisão. No que se faz característico a Gestão do Conhecimento, facultando avaliar informações sob diferentes perspectivas, no que tange a associação das entidades da base de dados para auxiliar na evolução de projetos de *softwares*. Bem como, identificar padrões, modelos e relacionamentos em um determinado conjunto de dados.

As classificações e análises apresentados no decorrer deste trabalho permitiram descobrir relações existes no conjunto de dados, validar a utilidade de medidas *software* no contexto de refatoração, assim como definir atributos para classificar os dados no contexto de refatoração. O trabalho de OLIVEIRA (2020) foi a base para obter o fluxo ilustrado na Figura **??** para se alcançar a proposta estabelecida para este trabalho.

As fontes extraídas e analisadas (i) se deu com base na realização do trabalho de OLIVEIRA (2020), disponibilizando de uma base processada e sumarizada. Com relação ao contexto de refatoração e medidas de *softwares* (ii), classificando nas principais medidas associadas às aná-

lises realizadas, que foram acoplamento, coesão, complexidade e tamanho. A identificação da existência de refatoração e medidas de *software* se deu com uso de ferramentas para detecção de refatoramento e medidas (iii), processo que foi realizada no trabalho de OLIVEIRA (2020). Com base nas associações disponibilizadas foi realizado análises dos dados para proporcionar visualizações quantitativas e qualitativas dos dados analisados e classificados (iv) por meio de tabelas, gráficos, grafos e painéis ilustrativos. O resultado gerado se deu com o objetivo de auxiliar em tomada de decisões para evoluir projetos de *softwares*.

Figura 35 – Fluxo realizado para alcançar a proposta desta pesquisa



Fonte: A autora (2021)

O fluxo resultante desta proposta se deu para auxiliar na evolução de projetos de *softwares* no que tange a conhecer o projeto em seus atributos característicos de construção do código fonte, para estabelecer estratégias de resoluções durante o processo de desenvolvimento do código fonte dos projetos, por meio das análises obtidas neste trabalho. Após a seleção da base de dados incorporada e classificada por OLIVEIRA (2020). Avançamos no processo de apresentação das refatorações identificadas, com as devidas medidas aplicadas e com os atributos que as classificaram por meio das ferramentas *Tableau* e *Neo4j*.

Esta dissertação apresenta como principais contribuições: (i) método sistemático para aná-

lise de projetos com refatoração e medidas de *software*. (ii) processo para identificação e classificação das medidas de refatoramento e *software* nos projetos. (iii) mensuração para as medidas de *softwares* nos projetos. (iv) método de análise e visualização de dados de projetos característicos de refatoramento, com proposição de modelo lógico e relatórios com painéis, grafos e gráficos dos resultados obtidos, com apoio de ferramentas de visualização de dados.

5.1 AMEAÇAS À VALIDADE

Esta seção apresenta as principais ameaças à validade relacionadas à esta pesquisa, assim como as justificadas adotadas para o desenvolvimento deste trabalho.

5.1.1 Critérios de seleção

No que se trata aos critérios de seleção da base de dados, foi definido utilizar o repositório de *software* já tratado e extraído do trabalho de OLIVEIRA (2020), a fim de realizar e proporcionar a análise e visualização dos dados coletados, ou seja, houve replicação da pesquisa no que se trata das etapas de Processamento e Sumarização. Uma vez que, o estudo proposto por OLIVEIRA (2020) auxiliou de forma objetiva no desenvolvimento desta dissertação. De forma, explorar com profundidade as possibilidades de análises e visualização de dados.

Também com as ferramentas utilizadas neste trabalho, houve replicação da pesquisa de OLIVEIRA (2020) da base já sumarizada, com tratamento dos dados com medidas de *software* e refatoração. Para que a base fosse espelhada na sua íntegra, se fez necessário reaplicar a pesquisa desde a etapa de processamento dos dados até a de sumarização.

5.1.2 Representatividade da amostra

Utilizamos uma base com 725 repositórios de projetos voltados para *Android*, na linguagem Java no ambiente do *github*. A amostra selecionada foi obtida a partir dos dados já processados na pesquisa de OLIVEIRA (2020). Apesar da amostrar ser de apenas 725 repositórios, tais projetos possuem características diversificadas. Além disso, os resultados obtidos demonstram que, a partir da aplicação do procedimento metodológico, foi possível validar tal amostra. O que por sua vez, viabilizou a identificação e classificação das características investigadas no conjunto de dados.

5.2 LIMITAÇÕES

No desenvolvimento deste trabalho, se utilizou como apoio o trabalho de OLIVEIRA (2020), no que tange à seleção e ao tratamentos dos dados. Utilizamos também critérios de seleção para realizar análise descritiva do conjunto de dados. Contudo, essa pesquisa proposta possui algumas limitações em sua execução. Primeiramente, o método proposto foi um apoio na concepção, tratamento e análise dos dados coletados. Porém, para obtermos um abordagem mais exploratória se faz necessário atingir outros aspectos para evolução e tratamento das determinadas limitações:

- Explorar a incorporação de demais medidas de *softwares* apoiadas na resolução de detectar e classificar refatorações. Este trabalho utilizou como base a pesquisa de OLIVEIRA (2020), no que fere à seleção das medidas para apoiar no processo de análise e classificação do conjunto de dados;
- Explorar necessidade de levantamento de incorporação de demais ferramentas para detecção e tratamentos de refatorações, no que tange ao processo de IA, como por exemplo comparação com o processo e automatização de processos de *Machine Learning*. O método proposto deste trabalho, utilizou como base a pesquisa de OLIVEIRA (2020), o processo de critérios de seleção das ferramentas. Se fazendo necessário, investigar propostas de aplicações de outros trabalhos;
- Aprimoramento do conjunto de dados, abordando outros contextos e práticas de desenvolvimento de *software*;
- Possibilitar a atualização da base de dados. Além disso, proporcionar análise em tempo real, com investigação no código;
- Realizar um experimento para caracterizar a facilidade do uso de visualização para tomada de decisões, quando comparado ao uso estrito da base de dados por meio de consultas.

Logo, frente às limitações expostas, se faz necessário explorá-las, com a finalidade de refletir e por consequência evoluir o espoco desta dissertação.

5.3 TRABALHOS FUTUROS

O escopo desta dissertação ainda pode ser incrementado, assim, como trabalhos futuros, sugere-se:

- Ampliar o contexto do conjunto de dados para diversas linguagens, tal qual como os repositórios;
- Incrementar análise e visualização de dados para outras ferramentas de *Business Intelligence*;
- Proporcionar análise dos dados em tempo real, com apoio de uma consulta direta nos repositórios que serão incorporados. A partir disso, realizar uma consulta programada dos dados;
- Explorar a inclusão de mais práticas e medidas de *software* para auxiliar na classificação das refatorações;
- Agir diretamente nas limitações abordadas a respeito do contexto desta pesquisa.

REFERÊNCIAS

BANSIYA, J.; ETZKORN, L. H.; DAVIS, C. G.; LI, W. A class cohesion metric for object-oriented designs. *JOOP*, v. 11, p. 47–52, 1999.

BARROS, V. P. A. *Um Método para a Refatoração de Software Baseado em Frameworks de Domínio*. 99 p. — Universidade Tecnológica Federal do Paraná, Ponta Grossa - PR, 2015.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. *Pattern-oriented Software Architecture: A System of Patterns*. 1. ed. [S.l.]: Wiley, 1996.

CARNEIRO, G. D. F. *USANDO MEDIÇÃO DE CÓDIGO FONTE PARA REFACTORING*. 141 p. — Universidade Salvador, Salvador, 2003. Disponível em: <https://pdfs.semanticscholar.org/d9f1/e0ca94c9db285e249d42e5b302be1cb017e3.pdf>.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, IEEE, v. 20, n. 6, p. 476–493, 1994.

ELGENDY, N.; ELRAGAL, A. Big data analytics: A literature review paper. In: PERNER, P. (Ed.). *Advances in Data Mining. Applications and Theoretical Aspects*. Cham: Springer International Publishing, 2014. p. 214–227. ISBN 978-3-319-08976-8.

FORNO, G. M. B. D.; MULLER, F. M. Fatores críticos em projetos de desenvolvimento de software. *Revista Pretexto*, v. 18, n. 2, p. 100–115, 2017.

FOWLER, M. e. o. *Refactoring: improving the design of existing code*. [S.l.]: Pearson Education India, 1999.

GALDINO, N. Big data: Ferramentas e aplicabilidade. *XIII SEGeT - Simpósio de Excelência em Gestão e Tecnologia*, 2016. Disponível em: <https://www.aedb.br/seget/arquivos/artigos16/472427.pdf>.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J.; SALGADO, F. B. P. L. A. M. *Padrões de Projetos: Soluções Reutilizáveis*. [S.l.]: Bookman, 2000.

Ge, X.; DuBose, Q. L.; Murphy-Hill, E. Reconciling manual and automatic refactoring. In: *2012 34th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2012. p. 211–221.

GOLDSCHMIDT, R.; PASSOS, E.; BEZERRA, E. *Data Mining*. [S.l.]: Elsevier Brasil, 2015.

LOH, S. Bi na era do big data para cientistas de dados. *Porto Alegre: Amazon*, 2014.

MACHADO, J. P. L. *Uma abordagem para apoio à decisão de refatoração em sistemas de software*. 73 p. — Universidade Federal de Uberlândia, Uberlândia, 2017.

MENS, T.; TOURWE, T. A survey of software refactoring. In: *IEEE Transactions on Software Engineering (Volume: 30 , Issue: 2 , Feb 2004)*. IEEE, 2004. p. 126–139. Disponível em: <https://ieeexplore.ieee.org/document/1265817>.

NAIRNEI, R. D. F. M. *PROPOSTA DE UM MODELO USANDO AGENTES PARA REFATORAÇÃO DE SOFTWARE*. 62 p. — Universidade Tecnológica Federal do Paraná, Ponta Grossa - PR, 2018.

NEO4F, I. *Neo4f*. 2020. Disponível em: <https://neo4j.com/product/>.

OLIVEIRA, E. *Relação entre refatorações e code smells na evolução de projetos de software e seu reflexo em medidas de software*. 96 p. — Universidade Federal de Pernambuco, Recife, 2020.

OPDYKE, W. F. *REFACTORING OBJECT-ORIENTED FRAMEWORKS*. Tese (Doutorado), 1992. Disponível em: <http://www.laputan.org/pub/papers/opdyke-thesis.pdf>.

PAIXãO, M.; UCHôA, A.; BIBIANO, A.; OLIVEIRA, D.; GARCIA, A.; KRINKE, J.; ARVONIO, E. Behind the intents: An in-depth empirical study on software refactoring in modern code review. In: . [S.l.: s.n.], 2020.

PRESSMAN, R. S. *Software engineering : a practitioner's approach - 7th ed*. [S.l.]: McGraw-Hill, 2010.

RIKHARDSSON, P.; YIGITBASIOGLU, O. Business intelligence analytics in management accounting research: Status and future focus. *International Journal of Accounting Information Systems*, v. 29, p. 37 – 58, 2018. ISSN 1467-0895. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1467089516300616>.

SANTOSO, L. W. et al. Data warehouse with big data technology for higher education. *Procedia Computer Science*, Elsevier, v. 124, p. 93–99, 2017.

SHEN, B.; ZHANG, W.; ZHAO, H.; LIANG, G.; JIN, Z.; WANG, Q. Intellimerge: A refactoring-aware software merging technique. *Proc. ACM Program. Lang.*, Association for Computing Machinery, New York, NY, USA, v. 3, n. OOPSLA, out. 2019. Disponível em: <https://doi.org/10.1145/3360596>.

SOFTWARE, L. T. *Tableau Desktop*. 2003–2020. Disponível em: <https://www.tableau.com/pt-br/products/desktop>.

SOMMERVILLE, I. *Engenharia de Software*. [S.l.]: Pearson, 2007.

TOURWE, T.; MENS, T. Identifying refactoring opportunities using logic meta programming. In: *Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.* Benevento, Italy, Italy: IEEE, 2003. Disponível em: <https://ieeexplore.ieee.org/document/1265817>.

YI, T.; WU, F.; GAN, C. A comparison of metrics for uml class diagrams. *SIGSOFT Softw. Eng. Notes*, Association for Computing Machinery, New York, NY, USA, v. 29, n. 5, p. 1–6, set. 2004. ISSN 0163-5948. Disponível em: <https://doi.org/10.1145/1022494.1022523>.

APÊNDICE A – REFATORAÇÕES REALIZADAS PELO REFACTORINGMINER

RefactoringMiner 1.0 & 2.0	RefactoringMiner 2.0
Extract Method	Move and Rename Class
Inline Method	Extract Class
Rename Method	Extract Subclass
Move Method	Extract Variable
Move Attribute	Inline Variable
Pull Up Method	Parameterize Variable
Pull Up Attribute	Rename Variable
Push Down Method	Rename Parameter
Push Down Attribute	Rename Attribute
Extract Superclass	Move and Rename Attribute
Extract Interface	Replace Variable with Attribute
Move Class	Replace Attribute (with Attribute)
Rename Class	Merge Variable
Extract and Move Method	Merge Parameter
Change Package (Move, Rename, Split, Merge)	Merge Attribute
	Split Variable
	Split Variable
	Split Attribute
	Change Variable Type
	Change Parameter Type
	Change Return Type
	Change Attribute Type
	Extract Attribute
	Move and Rename Method
	Move and Inline Method
	Add Method Annotation
	Remove Method Annotation
	Modify Method Annotation
	Add Attribute Annotation
	Remove Attribute Annotation
	Modify Attribute Annotation
	Add Class Annotation
	Remove Class Annotation
	Modify Class Annotation
	Add Parameter
	Remove Parameter
	Reorder Parameter

ANEXO A – REFATORAÇÕES IDENTIFICADAS NA BASE DE DADOS

Tabela 6 – Refatorações identificadas na base de dados

Refactoring	Quantidade de Refactorings
Extract Method	30070
Move Attribute	26665
Move Class	26011
Move Method	24570
Extract Variable	12645
Pull Up Method	11981
Extract And Move Method	9973
Rename Class	8396
Inline Method	8026
Pull Up Attribute	7850
Extract Class	5295
Replace Variable With Attribute	4744
Push Down Method	4723
Move And Rename Class	4430
Inline Variable	3294
Push Down Attribute	2634
Extract Superclass	2514
Parameterize Variable	2281
Extract Interface	576
Extract Subclass	548
Move And Rename Attribute	156
Replace Attribute	143
Total	197.525

ANEXO B – CODE SMELLS IDENTIFICADAS NOS PROJETOS

Tabela 7 – Code Smells Identificadas nos Projetos

Smell	Quantidade de Code Smells
LongMethod	1185006
DeadCode	672265
FeatureEnvy	438699
DataClass	432503
StateChecking	324284
NestedBlockDepth	310170
LargeClass	305100
ComplexMethod	214053
LongParameterList	181062
ClassDataShouldBePrivate	125534
ComplexCondition	116642
GodClass	83790
ShotgunSurgery	75203
BrainMethod	73240
Cycle	71974
RefusedParentBequest	54185
MiddleMan	36381
MessageChain	35204
TraditionBreaker	2943
TOTAL	4.738.238