UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JEFFERSON DA SILVA BARBOSA

CALICE - CATÁLOGO PARA A ANÁLISE DO IMPACTO DE CLONES EM CLASSES DE CUSTOMIZAÇÃO DE UMA LINHA DE PRODUTO DE SOFTWARE GLOBAL

FORTALEZA

2018

JEFFERSON DA SILVA BARBOSA

CALICE - CATÁLOGO PARA A ANÁLISE DO IMPACTO DE CLONES EM CLASSES DE CUSTOMIZAÇÃO DE UMA LINHA DE PRODUTO DE SOFTWARE GLOBAL

Dissertação apresentada ao Curso de do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade

Co-Orientador: Prof. Dr. João Bosco Ferreira Filho
Co-Orientador: Profa. Dra. Carla Ilane Moreira Bezerra

FORTALEZA

2018

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

B198c Barbosa, Jefferson da Silva.
Calice - catálogo para a análise do impacto de clones em classes de customização de uma linha de produto de software global / Jefferson da Silva Barbosa. – 2018.
109 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, , Fortaleza, 2018.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Profa. Dra. Carla Ilane Moreira Bezerra.

1. Linha de Produto de Software. 2. Clone. 3. Customização de Produtos. I. Título.

CDD

JEFFERSON DA SILVA BARBOSA

CALICE - CATÁLOGO PARA A ANÁLISE DO IMPACTO DE CLONES EM CLASSES DE CUSTOMIZAÇÃO DE UMA LINHA DE PRODUTO DE SOFTWARE GLOBAL

Dissertação apresentada ao Curso de do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Aprovada em: 29/11/2018

BANCA EXAMINADORA

Profa. Dra. Rossana Maria de Castro Andrade (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. João Bosco Ferreira Filho
Universidade Federal do Ceará (UFC)

Profa. Dra. Carla Ilane Moreira Bezerra
Universidade Federal do Ceará (UFC) - Campus Quixadá

Márcio de Medeiros Ribeiro
Universidade Federal de Alagoas (UFAL)

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará (UFC)

Rodrigo Dias Ferreira
LG Electronics

Aos meus pais pela generosidade e abstenção do agir mal, pelos ensinamentos de bom caráter e serenidade nas ações tomadas e pela humildade intrínseca que possibilitou o vislumbramento da realidade em diferentes contextos.

“Sonho que se sonha só é só um sonho que se sonha só, mas sonho que se sonha junto é realidade”

(Raul Seixas)

RESUMO

Desenvolvedores frequentemente adotam práticas de clonagem para acelerar a implementação, porém, a longo prazo, a clonagem pode afetar a evolução do software em relação aos custos e tempo de desenvolvimento, já que mudanças nos fragmentos clonados podem exigir modificações em várias partes do sistema. Esse problema pode aumentar caso a clonagem seja utilizada em classes usadas para adicionar, remover ou adaptar características do sistema em uma Linha de Produtos de Software (LPS), devido a essas classes estarem relacionadas a várias características e produtos da linha. Entretanto, é difícil saber até que ponto uma clonagem em classes de customização pode impactar um projeto de software. Assim, este trabalho conduz, em primeiro lugar, um estudo empírico, dentro de uma LPS usada por diversos clientes espalhados pelo mundo, denominada neste trabalho de LPS Global (LPSG), para analisar as práticas de clonagem e como as partes clonadas se relacionam com a capacidade de manutenção das classes de customização. Para construir o conjunto de dados deste trabalho, foram coletados e identificados durante 13 meses, clones entre as classes de customização, envolvendo 70 tipos de classes. Paralelamente, foram coletadas as respectivas *issues* do projeto, obtendo mais de 140 *issues* relacionadas as classes de customização. Posteriormente, foi realizada uma comparação entre o tempo gasto com *issues* que possuem fragmentos de código clonado e as que não possuem. Em seguida, foram feitas coletas de medidas relacionadas à complexidade, modificabilidade e tamanho das classes para realizar uma análise baseada na relação entre clone e a manutenibilidade das classes de customização. Por fim, este trabalho propõe um catálogo que lista os tipos de clones, os tipos de relações que um clone pode ter e os seus respectivos impactos, com intuito de auxiliar os desenvolvedores na fase de priorização da refatoração de código. Este catálogo auxilia a entender como a clonagem se associa à sustentabilidade no contexto da customização em massa, fornecendo insumos sobre a evolução da clonagem e seus impactos em um projeto real de uma LPSG.

Palavras-chave: Linha de Produto de Software. Clone. Customização de Produtos.

ABSTRACT

Developers often adopt cloning practices to speed up implementation. However, cloning-and-owning, in the long run, can severely affect software evolution, as changes in cloned fragments may require modifications in many parts of the system. This problem scales if cloning is applied in classes used to add, remove, or adapt features in a Software Product Line (SPL), because these classes are related to several features and products. Nevertheless, it is hard to know to which extent cloning in customization classes can impact a software project. Thus, this work first conducts an empirical study, within a SPL used worldwide, called here Global SPL (GSPL), to analyze cloning practices and how cloned parts relate to the maintainability of customization classes. To construct the dataset of this work, we collected and analyzed clones inside the GSPL customization classes during 13 months, involving 70 types of customization classes. In parallel, the respective issues were collected from the issue tracking tool of the GSPL project, obtaining over 140 issues related to customization classes. After that, we confronted the time spent to solve each issue with the fact whether it comes from cloning or not. Then, we collected measures related to complexity, modifiability and size of the classes to make an analysis based on the relationship between clone and maintainability. At the end, this work proposes a clone impact catalog to help developers prioritize refactoring of code fragments. This catalog helps to understand how cloning associates to maintainability in the context of mass customization, giving insights about cloning evolution and its impacts in a GSPL project.

Keywords: Software Product Line. Clone. Product Customization

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CBO	*Coupling between objects*
CFG	*Configuration File*
CSV	*Comma Separated Values*
CVL	*Common Variability Language*
DIT	*Depth Inheritance Tree*
DSL	*Domain-Specific Language*
ELPS	Engenharia de Linha de Produto de Software
JSON	*JavaScript Object Notation*
LCOM	*Lack of Cohesion of Method*s
LOC	*Lines of code*
LPS	Linha de Produto de Software
LPSG	Linha de Produto de Software Global
NOC	*Number of Children*
NOF	*Number of Fields*
NOM	*Number of Methods*
NOPF	*Number of Public Fields*
NOPM	*Number of Public Methods*
NOSF	*Number of Static Fields*
NOSI	*Number of Static Invocations*
NOSM	*Number of Static Methods*
PROP	*Property File*
RFC	*Response for a Class*
WMC	*Weight Method Class*
XLS	*eXtensible Stylesheet Language*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Essa dissertação tem como objetivo propor um catálogo que lista os tipos de clones, os tipos de relações que um clone pode ter e os seus respectivos impactos em um projeto de Linha de Produto de Software para ajudar a priorização da refatoração de clones. Na Seção 1.1 é apresentado o contexto em que o trabalho está inserido. A Seção 1.2 apresenta a motivação para o desenvolvimento do trabalho. A Seção 1.3 apresenta os objetivos da dissertação. A Seção 1.4 apresenta a metodologia utilizada para realização desse trabalho. Ao final desse capítulo, na Seção 1.5, é apresentado a estrutura desta dissertação.

1.1 Contextualização

Em um mercado cada vez mais competitivo, a qualidade e o tempo de desenvolvimento de um software pode ser um fator crucial para a escolha de um cliente. O desenvolvimento de um software individual sem a reutilização de artefatos previamente desenvolvidos nem sempre é uma alternativa viável. Na realidade, grandes empresas que trabalham em domínios específicos, produzem uma grande variedade de produtos de software semelhantes reusando software existente, proporcionando menores custos de produção e manutenção de software, entregas mais rápidas e maior produtividade (SOMMERVILLE *et al.*, 2011; ACHER *et al.*, 2013).

Um paradigma que pode ser adotado para intensificar as vantagens do reuso de software é a Linha de Produto de Software (LPS). O conceito de LPS é definido como a reutilização sistemática de um conjunto de aplicações com uma arquitetura comum e componentes compartilhados, em que cada aplicação é derivada das partes comuns para atingir diferentes requisitos (POHL *et al.*, 2005; SOMMERVILLE *et al.*, 2011).

Embora as LPSs possam trazer vários benefícios para uma organização, o contexto em que ela pode ser aplicada e a forma que ocorre o desenvolvimento de seus produtos, nem sempre é claro. Para isso, foi definido um paradigma de Engenharia em LPS para lidar com a complexidade de seu desenvolvimento. Esse paradigma dita como ocorre as diferentes fases do desenvolvimento de produtos de software usando plataformas e customização em massa (POHL *et al.*, 2005).

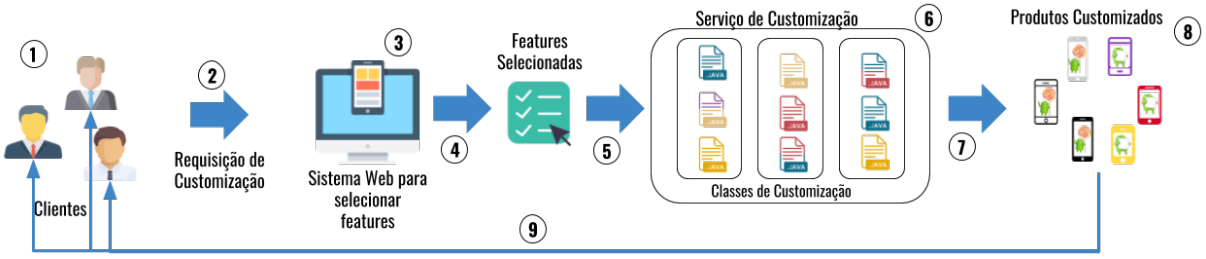
Para facilitar a implantação das fases de engenharia, Pohl *et al.* (2005) propuseram um *framework* que define um processo de engenharia para LPS com intuito de melhorar o gerenciamento do projeto. No entanto, com intuito de realizar entregas mais rápidas, as empresas

desenvolvem produtos para atender uma necessidade específica do cliente, definidos como produtos *ad-hoc*, sem se preocupar com a reuso sistemático dos artefatos gerados. Com isso, a manutenibilidade dos produtos gerados podem ser comprometidos ao longo do tempo devido a inserção de acoplamentos indesejados (RUBIN *et al.*, 2015).

Para implementar esses produtos *ad-hoc*, os desenvolvedores tendem a clonar fragmentos de código, uma ou mais linhas de código, dentro de um sistema para acelerar o processo de desenvolvimento. Essa prática também está presente nas LPSs, na implementação da família de produtos relacionados e nos próprios artefatos da LPS (e.g. modelos de *features*, operadores de derivação ou os diferentes artefatos gerados durante o desenvolvimento da LPS) (RUBIN *et al.*, 2015).

Neste trabalho de dissertação foi analisado um projeto real de uma LPS, aqui denominada de Linha de Produto de Software Global (LPSG), desenvolvido para uma empresa multinacional que tem parceria com o grupo de pesquisa GREat[1] ao qual esta dissertação está vinculada. O termo Global foi adotado porque essa LPS gera centenas de produtos distribuídos em vários países. A LPSG estudada se encontra no domínio de aplicações móveis, onde é fornecido aos clientes um ambiente para customizar parte das *features* dos *smartphones* para diferentes tipos de modelos de dispositivos móveis. Essa customização[2] é importante para as operadoras multinacionais distribuírem seus celulares com variações em seus sistemas, dependendo de cada país, sistema operacional ou requisitos de uma linha de produtos. A visão geral da LPSG está ilustrada na Figura 1.

Figura 1 – Visão geral do projeto da LPSG



Fonte: Elaborada pelo autor

A Figura 1 inicialmente ilustra um grupo de operadoras telefônicas espalhadas pelo

[1] O Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) é vinculado ao Departamento de Computação da Universidade Federal do Ceará (UFC) e é composto por professores, pesquisadores colaboradores e estudantes de pós-graduação e graduação da UFC e de outras Instituições de Ensino Superior.

[2] Nesta dissertação, customização é o processo de modificar ou estender uma feature de software que requer um código ou alguma forma de implementação.

mundo (1). Cada operadora possui um conjunto de requisitos que devem ser customizados e aplicados em diferentes modelos de celulares. Alguns exemplos são: papel de parede, aplicativos pré-instalados, idioma padrão e assim por diante. Para gerar esses recursos automaticamente, essas operadoras podem fazer uma solicitação para um sistema web (2). O sistema web (3) é a interface que o técnico responsável pela geração de produtos interage para selecionar as *features* que serão customizadas. Depois de selecionar as *features* (4), o sistema web se comunica com um serviço de customização que receberá as *features* que a operadora deseja customizar e gerar os respectivos arquivos para customização. Então, o serviço de customização recebe a solicitação do sistema web (5). O serviço de customização contém as classes responsáveis pela geração dos arquivos de customização (6). Estas são as classes que serão analisadas neste trabalho. Os arquivos gerados pelas classes de customização são empacotados e retornados para serem inseridos em dispositivos móveis (7). Após a inserção dos arquivos de customização em diferentes tipos de dispositivos, são derivados diferentes produtos customizados de acordo com os requisitos das operadoras (8). Com isso, esses produtos possuem uma customização, que pode ser replicada em uma linha de produtos, atendendo a diferentes tipos de requisitos de operadoras (9).

A customização dos dispositivos é feita pelas classes de customização mencionadas anteriormente. As classes Java são responsáveis por gerar automaticamente um conjunto de arquivos: *eXtensible Markup Language (XML), Vorbis Compressed Audio File OGG, Configuration File (CFG), Property File (PROP), JavaScript Object Notation (JSON)* e/ou *Image File JPG*. Esses arquivos são inseridos no aplicativo em tempo de *design* para adaptar parte dos componentes do software. Os arquivos de customização podem estar relacionados em várias partes do software para cobrir diferentes requisitos. Alguns exemplos de *features* customizáveis são configurações, sons, rede, interface, perfil, aplicativos, mensagens e lista telefônica.

1.2 Motivação

Uma clonagem ocorre quando as variantes existentes são copiadas e podem ser modificadas para satisfazer os requisitos de um novo cliente ou segmento de mercado (RUBIN *et al.*, 2015). Essa prática de clonagem é usada quando, por exemplo, há a necessidade de reduzir o tempo de desenvolvimento. No entanto, essa prática também pode afetar a manutenibilidade da LPS ao longo do tempo, tanto no âmbito perfectivo como no âmbito corretivo, aumentando as possibilidades de propagação de *bugs*, permitindo que os defeitos se espalhem por muitos

produtos derivados, aumentando o acoplamento e gerando uma maior dificuldade na modificação e um aumento dos gastos com manutenção (KIM *et al.*, 2005; DUBINSKY *et al.*, 2013).

A prática de clonagem pode ter um impacto mais significativo na fase de derivação de produtos de uma LPS. Nessa fase os geradores de código são usados para automatizar o processo de derivação que estão relacionados a várias *features* do sistemas. Como uma *feature* pode ser reutilizada em diferentes produtos, um problema na geração de código pode afetar vários produtos gerados a partir desses mecanismos de derivação.

Um exemplo seria uma *feature* para gerenciar os arquivo de um aparelho telefônico necessita de métodos de abertura, leitura e escrita de arquivos. Ao derivar um produto, podem existir diversas *features* que usam o gerenciador de arquivos como, por exemplo, uma *feature* para gerenciar perfil de usuário onde o usuário pode fazer *upload* de foto. Caso a função *upload* seja replicada em todas as *features* que desejam usar o gerenciador de arquivos, uma alteração, por exemplo, nos tipos de arquivos que são permitidos pelo gerenciador pode afetar várias *features* e consequentemente vários produtos.

Para este trabalho, conforme mencionado anteriormente, foi analisado um projeto desenvolvido em parceria com o GREat e uma empresa multinacional de tecnologia, que usa classes de customização escritas em Java para adicionar, remover e adaptar algumas *features* de acordo com a necessidade do cliente. Esse projeto foi selecionado devido a disponibilidade de dados relacionados ao gerenciamento e ao processo de desenvolvimento. Outro motivo da seleção do projeto está relacionado à aplicação de boas práticas do gerenciamento de mudanças, uma vez que todas as alterações realizadas dentro do projeto estão registradas em ferramentas de forma que possam ser rastreadas no código. O projeto considerado como objeto de estudo empírico possui cerca de 70 classes de customização relacionadas a 378 *features*. Considerando este nível de magnitude, uma modificação em um fragmento de código clonado pode afetar várias *features* e, consequentemente, centenas de produtos para centenas de clientes.

Dessa forma, este trabalho teve como motivação, identificar pontos que necessitam ser refatorados baseados no tempo gasto de desenvolvimento de tarefas que envolvem partes clonadas, na taxa de alteração dessas partes clonadas e no nível de complexidade, e modificabilidade que essas partes clonadas inserem na LPSG.

1.3 Objetivo

Considerando os impactos que as práticas de clonagem podem causar na manutenibilidade de um projeto de software, este trabalho tem como objetivo propor um catálogo com os tipos de clones e seus respectivos impactos para ajudar na identificação e priorização de pontos de refatoração em LPSGs. O intuito do catálogo é facilitar a fase de refatoração de código e, assim, proporcionar um aumento da produtividade e do retorno de investimento da organização envolvida no desenvolvimento da mesma.

Para a construção do catálogo foi utilizado um processo que define os passos de coleta e análise dos dados. A partir das análises, foi possível quantificar o impacto de clones em uma LPSG de acordo com o tempo gasto para resolver *issues* com clones de acordo com o tempo registrado pelos desenvolvedores em cada *issue*, verificar a taxa de mudança em partes clonadas, e analisar como os clones afetam a manutenibilidade das classes de customização.

Para atingir esse objetivo foram levantadas as seguintes questões de pesquisa com um conjunto de hipóteses associadas as mesmas:

QP1 - Quanto tempo de desenvolvimento e manutenção é gasto para resolver issues com clones em classes de customização em uma LPSG?

H_0: *Issues* do tipo desenvolvimento que possuem código clonado em classes de customização levam menos tempo para serem solucionadas.

H_1: *Issues* do tipo desenvolvimento que possuem código clonado em classes de customização levam mais tempo para serem solucionadas.

H_2: *Issues* do tipo manutenção que possuem código clonado em classes de customização levam menos tempo para serem solucionadas.

H_3: *Issues* do tipo manutenção que possuem código clonado em classes de customização levam mais tempo para serem solucionadas.

QP2 - Qual o nível de alterações sofrida por partes clonadas ao longo do tempo em projetos de LPSG?

H_4: Fragmentos de código clonado em classes de customização têm uma taxa de mudança maior em comparação com partes sem clones.

H_5: Fragmentos de código clonado em classes de customização têm uma taxa de mudança igual ou menor em comparação com partes sem clones.

QP3 - Como os clones afetam a manutenibilidade das classes de customização?

H_6: Medidas de manutenibilidade não possuem uma forte correlação com a taxa de clones em classes de customização.

H_7: Medidas de manutenibilidade possuem uma forte correlação com a taxa de clones em classes de customização.

Para responder a essas questões, foram obtidas e analisadas mais de 120 mil linhas de código durante 13 meses de um projeto feito em parceria com o GREat e uma empresa multinacional de tecnologia. Entre os dados coletados da LPSG pode-se listar: um conjunto de *issues* coletados a partir de sistema de gerenciamento de *issues* e os *commit logs* que representam as modificações realizadas no projeto durante seu tempo de vida. Além disso, também foram coletados os clones e as medidas relacionadas a complexidade, modificabilidade e tamanho das classes de customização. A partir desses dados, é possível acompanhar a evolução da LPSG e analisar os impactos dos clones ao longo do tempo.

É importante ressaltar que embora a empresa multinacional que possui essa LPSG exija a não divulgação de seus detalhes de código-fonte, ainda pode-se fornecer ao leitor um conjunto de dados anônimos [3].

1.4 Metodologia

Este trabalho utilizou a metodologia ilustrada na Figura 2 para a sua execução, a qual auxilia a responder as questões de pesquisa levantadas anteriormente e é composta de cinco fases. Essa dissertação usou como base o trabalho de Wohlin *et al.* (2012) que apresenta um modelo para realizar estudos empíricos, o qual foi utilizado no decorrer da metodologia.

Figura 2 – Metodologia do trabalho

Fonte: Elaborada pelo autor

A descrição geral de cada fase é apresentada a seguir:

- **Revisão da Literatura:** Nesta primeira fase foi executado uma *string* de busca para coletar trabalhos que pudessem dar um embasamento teórico do estado da arte relacionado

[3] https://github.com/GREatResearches/Software-Product-Line

a coleta e análise de clones além de proporcionar uma visão dos desafios em aberto da área;

- **Estudo do Projeto:** Nesta fase foi verificado como o projeto estava organizado, quais suas funções, qual sua arquitetura, quais os sistemas dependentes e quais os seus objetivos. Com isso, foi possível definir um processo de coleta e análise do projeto;
- **Coleta dos Dados:** Na terceira fase foram executadas diversas coletas de clones, medidas e *logs* relacionadas às classes de customização do projeto. As coletas foram feitas para um período de 13 meses do projeto com intuito de avaliar como ocorreu sua evolução;
- **Cruzamento dos Dados:** Na quarta fase foi feito o cruzamento de todos os dados para que se pudesse gerar conclusões de correlação entre os dados e assim propor uma análise baseada nessa correlação; e
- **Análise dos Dados e Construção do Catálogo:** Como última etapa, foi feita a análise de como os clones poderiam impactar no desenvolvimento do projeto baseado na correlação entre os dados coletados.

A partir dessa metodologia foi possível construir um *dataset*, uma ferramenta para fazer o cruzamento de dados entre as medidas de manutenibilidade e os clones, e um catálogo com os tipos de clones e os seus respectivos impactos no projeto. Espera-se então que este trabalho ajude na identificação de pontos de refatoração, além de proporcionar uma base de dados para a replicação do experimento e uso para novos estudos.

1.5 Organização da dissertação

O restante desta dissertação é composta por mais cinco capítulos.

O Capítulo 2 apresenta o embasamento teórico das diversas áreas abordadas por esse trabalho: Linha de Produto de Software, Clones e Medidas de Manutenibilidade.

No Capítulo 3 são apresentados os trabalhos que foram analisados e que estavam relacionados aos diferentes passos do processo para análise dos impactos dos clones executados para a obtenção dos resultados deste trabalho.

O Capítulo 4 descreve o processo para a análise do impacto de clones em uma LPSG, apresentando como foram executados os passos desse trabalho, indicando os artefatos gerados em cada etapa, como foram planejados os passos de coleta, cruzamento e análise dos dados. Também é apresentado o projeto usado como Estudo de Caso para a execução deste trabalho.

No Capítulo 5 são apresentados os resultados obtidos a partir da execução do pro-

cesso. Nesse capítulo também é feito uma análise estatística desses resultados.

Por fim, o Capítulo 6 apresenta o fechamento com os resultados alcançados neste trabalho, quais as ameaças a validade advindas do processo que podem afetar a validade dos resultados e quais os trabalhos futuros que podem advir como continuidade deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Na fundamentação teórica desta dissertação são abordados os conceitos utilizados em seu desenvolvimento. Na Seção 2.1, é especificado o que é uma Linha de Produto de Software, como funciona seu processo de engenharia, como a LPS pode ser representada, o que é derivação de produtos e quais os mecanismos que podem auxiliar essa derivação. Na Seção 2.3, são descritos os conceitos de clones, quais suas vantagens e desvantagens, como as práticas de clonagem podem afetar um projeto de software, quais são os seus tipos e quais as técnicas e ferramentas para detecção de clones. Na Seção 2.4 é apresentado o conceito de manutenibilidade, quais medidas podem ser utilizadas para analisar a manutenibilidade e qual a relação entre a manutenibilidade de um projeto e seus clones.

2.1 Linha de Produto de Software

O conceito de LPS é definido como a reutilização sistemática de artefatos que são compartilhados entre vários produtos, em que cada produto pode ser derivado desses artefatos e possuir um conjunto de novas funcionalidades para atender diferentes requisitos (POHL *et al.*, 2005).

As vantagens da aplicação de uma LPS são consequências de um conjunto de atividades que visam definir um reuso sistemático com a finalidade de obter benefícios organizacionais, explorando semelhanças entre um conjunto de produtos que tratam de um segmento específico do mercado e gerenciando a variabilidade desses produtos. As partes comuns são usadas para criar uma plataforma que deve ser utilizada como uma base na qual um conjunto de produtos podem ser desenvolvidos e produzidos de forma eficiente (ERIKSSON; HAGGLUNDS, 2003).

Para o gerenciamento das fases de desenvolvimento de uma LPS é definido na literatura a engenharia de LPS (POHL *et al.*, 2005).

2.1.1 Engenharia de Linha de Produto de Software

A Engenharia de Linha de Produto de Software (ELPS) é um paradigma para desenvolver software usando plataformas e customização em massa (POHL *et al.*, 2005). De acordo com Meyer e Lehnerd (1997), essas plataformas são um conjunto de subsistemas de software e interfaces que formam uma estrutura comum.

Para facilitar a implantação do reuso sistemático em projetos a partir da LPS, Pohl *et*

al. (2005) definiram um *framework* onde são descritas as atividades em cada fase de desenvolvimento do projeto. Em geral, o *framework* divide a ELPS em duas fases: Engenharia de Domínio e Engenharia de Aplicação. Na fase de Engenharia de Domínio, são definidas as partes comuns e variáveis de uma família de software. Já na fase de Engenharia de Aplicação, é construído um produto usando a reutilização de artefatos do domínio e verificação da variabilidade de um conjunto de produtos (POHL *et al.*, 2005).

De acordo com Pohl *et al.* (2005) a fase de Engenharia de Domínio é composta por cinco etapas e a Engenharia de Aplicação é composta por quatro etapas. A descrição e os artefatos de entrada e saída de cada etapa estão listados a seguir:

Engenharia do domínio:

- *Gestão de produtos* - Tem como finalidade delimitar o escopo da LPS e gerenciar o portfólio de produtos da empresa. Essa etapa tem como entrada os objetivos do projeto. A saída consiste de um conjunto de características comuns e variáveis dos produtos.
- *Requisitos da engenharia de domínio* - Tem como finalidade levantar e documentar os requisitos comuns e variáveis da LPS. Essa etapa tem como entrada as características comuns e variáveis dos produtos e tem como resultado um conjunto de requisitos reutilizáveis e um modelo de variabilidade.
- *Design do domínio* - Tem como finalidade definir uma arquitetura de referência que poderá ser usada por todos os produtos derivados da LPS. Essa etapa tem como entrada os requisitos e o modelo de variabilidade. A saída consiste de uma arquitetura de referência.
- *Desenvolvimento do domínio* - Nessa fase são desenvolvidos os componentes de software que poderão ser reusados pelos produtos da LPS. Essa etapa tem como entrada a arquitetura de referência. A saída consiste de um conjunto de componentes reutilizáveis.
- Teste *do domínio* - Tem como finalidade verificar e validar os componentes de software. Essa etapa tem como entrada os requisitos do domínio, a arquitetura de referência e os componentes implementados. Na saída são produzidos documentos com os resultados dos testes.

Engenharia da aplicação:

- *Requisitos da aplicação* - Tem como finalidade especificar requisitos para um produto específico e reusar os requisitos do domínio para as partes comuns da aplicação. Essa etapa tem como entrada os requisitos de domínio e a lista de características da LPS que serão usadas para o desenvolvimento da aplicação. Ao final dessa etapa são produzidos os

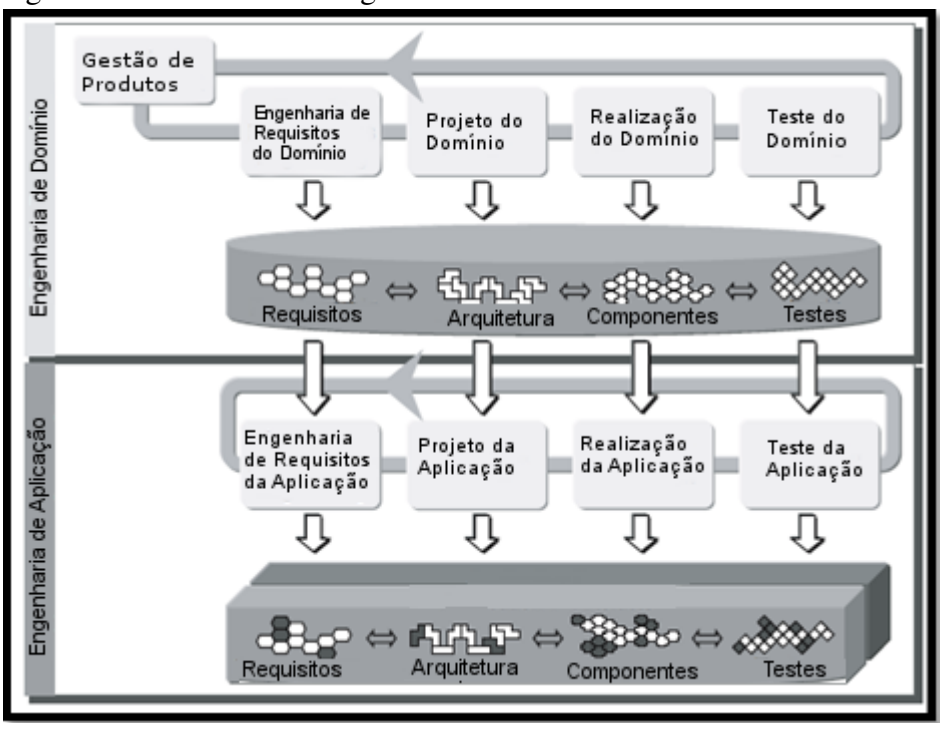
requisitos de uma aplicação específica.

- *Design da aplicação* - Tem como finalidade produzir a arquitetura do produto, selecionando, configurando e adaptando partes da arquitetura de referência. Essa etapa tem como entrada a arquitetura de referência e as especificações de requisitos da aplicação e como saída uma arquitetura de aplicação.
- *Desenvolvimento da aplicação* - Tem como finalidade produzir o produto a partir dos componentes de software e adicionar funções específicas da aplicação. Essa etapa possui como entrada os componentes reutilizáveis e como saída uma aplicação derivada da LPS.
- *Teste da aplicação* - Tem como finalidade validar e verificar uma aplicação em relação à sua especificação. Tem como entrada os testes do domínio e como saída os resultados dos testes da aplicação.

.

Para cada etapa do processo é gerado um artefato específico. Os artefatos gerados na fase de engenharia de domínio compõem a base da linha de produto de software e são armazenados em um repositório comum, já os artefatos de aplicação incluem todos os artefatos de desenvolvimento de um produto específico, incluindo a própria aplicação configurada e testada (POHL *et al.*, 2005). A visão geral do *framework* está ilustrada na Figura 3.

Figura 3 – Framework de engenharia de uma LPS



Fonte: Pohl *et al.* (2005)

A vantagem da divisão em engenharia de domínio e aplicação é que há uma separação de propósitos para construir uma base robusta e para construir aplicações específicas. Para serem eficazes, os dois processos devem interagir de uma maneira que ambos sejam beneficiados (SILVA *et al.*, 2011).

A execução deste trabalho terá como foco a fase de Engenharia da Aplicação, mais especificamente na parte de derivação de produtos. Será explorado o mecanismo de derivação de um projeto real verificando como os clones podem afetar este projeto.

Durante a fase de Engenharia da Aplicação um conjunto de produtos pode ser derivado a partir de um subconjunto dos artefatos gerados nas etapas do processo de Engenharia do Domínio (DEELSTRA *et al.*, 2005). A derivação tem como finalidade definir regras que especificam como as partes comuns de uma LPS devem ser compostas para se obter um produto específico dessa LPS (SÁNCHEZ *et al.*, 2008).

Para auxiliar a derivação de produtos, foram propostas técnicas dirigidas a modelos de variabilidade que ajudam a obter produtos mais confiáveis, diminuem o tempo de desenvolvimento do produto e minimizam os custos (PERROUIN *et al.*, 2008).

2.1.2 Modelos de variabilidade

A variabilidade é descrita como uma composição de pontos de variação e variantes, em que, um ponto de variação localiza as vinculações entre as diversas variantes, já uma variante é uma alternativa que representa uma característica do sistema (TRIGAUX; HEYMANS, 2003).

Os modelos de variabilidade são formas de representar a variabilidade da LPS (BERGER *et al.*, 2013). A partir dos modelos de variabilidade é possível obter um maior controle sobre a variabilidade da LPS, das especificações de requisitos e dos produtos derivados (BERGER *et al.*, 2013). Para expressar essa variabilidade de um domínio específico, foi proposto a *Domain-Specific Language (DSL)*, que faz a captura da variabilidade entre os modelos em um determinado domínio. Contudo, a DSL causa uma dependência da linguagem de modelagem ao modelo base da LPS (HAUGEN *et al.*, 2008).

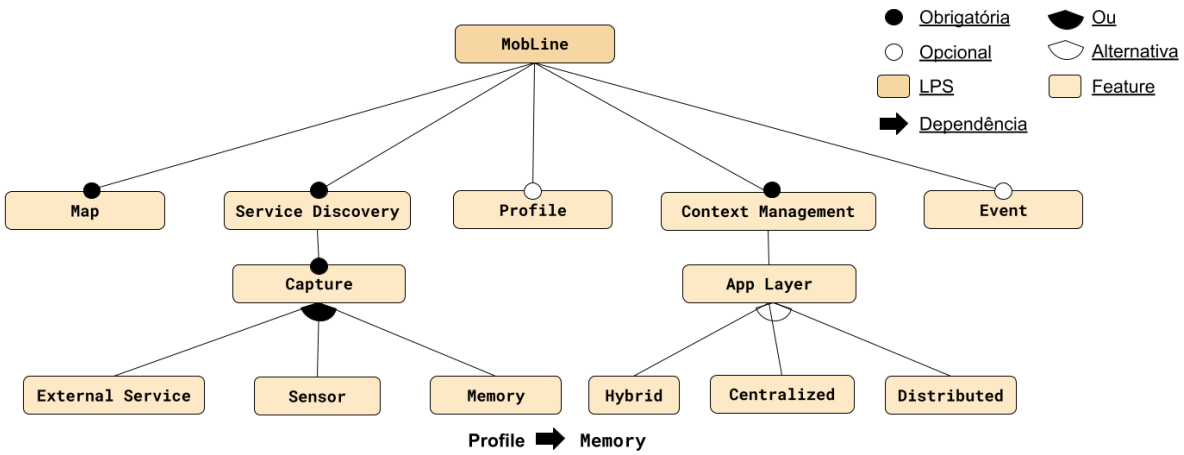
A linguagem *Common Variability Language (CVL)*, foi desenvolvida para contornar esse problema de dependência de linguagem. A CVL trata-se de uma linguagem genérica para modelagem de variabilidade em qualquer domínio que utiliza mecanismos de análise semântica para capturar os elementos do modelo. Um modelo CVL garante a independência de linguagem fazendo referências de sentido único para o modelo base, descrevendo como os elementos dos

modelos base podem variar. A execução de um modelo CVL ocorre sobre um domínio específico ou uma DSL, produzindo uma transformação do modelo base para um modelo de produto, onde a estrutura de alguns dos elementos do modelo base é modificada (SVENDSEN *et al.*, 2011).

No trabalho de (ALLIER *et al.*, 2015) é feito o uso da CVL e são descritos 4 operadores para derivar produtos a partir da linguagem. O operador Existência de Objetos adiciona ou remove qualquer elemento de código. O operador Link de Existência adiciona ou remove relações entre elementos, como por exemplo, entre classes. O operador Substituição de Objeto permite a substituição de elementos de código do mesmo tipo. O operador Substituição de Link permite que a relação entre duas classes A e B possa ser substituída de forma que a classe A referencie agora uma classe C.

Para a representação gráfica desses modelos são definidos modelos de *features* como um conjunto de características interligadas por um conjunto de relações que contêm dados que descrevem atributos dos recursos representados em formato de árvores. Um modelo de *features* pode ser exibido em diferentes níveis de detalhamento, assim como em outros diagramas, pode-se adotar a melhor visão para se trabalhar. Com isso, os modelos trazem como vantagem o melhor gerenciamento de variabilidade de uma LPS (GRISS *et al.*, 1998). A Figura 4 mostra um exemplo de modelo de *features* da LPS MobiLine desenvolvida no Laboratório de pesquisa GREat (MARINHO *et al.*, 2010).

Figura 4 – Modelo de *features* baseado na LPS MobiLine



Fonte: Elaborada pelo autor

A estrutura de um modelo de *features* é definida através das relações entre as *features*. A partir da LPS MobiLine, representado na Figura 4 é possível gerar diversos produtos, por

exemplo, um guia de museu, em que o produto gerado deve conter todas as *features* obrigatórias representadas pelo circulo preto. O produto pode ter a *feature Event* e *Profile* como opcionais, representada pelo circulo branco. A *feature Event* pode ser usada para mostrar exposições temporárias do museu e a *Profile* pode ser usada para gerenciar as informações dos visitantes do museu. No modelo está presente uma restrição de dependência entre *features*, representado pela seta, na qual indica que ao selecionar a *feature Profile* a *feature Memory* também deve ser selecionada. A *feature Memory* é um exemplo de *feature* do tipo "Ou"responsável por gerenciar as informações do contexto de um usuário coletadas da memoria do servidor. Um exemplo de *feature* alternativa seria a *Centralized* que indica a descoberta de serviços a partir de um *smartphone*. Benavides *et al.* (2010) definem alguns tipos dessas relações:

- *Obrigatória* – Inclusa em todos os produtos derivados da linha de produto.
- *Opcional* – Pode ou não estar inclusa em um produto.
- *Alternativa* – Apenas uma *feature* pode ser escolhida dentro de um grupo de *features*.
- *Ou* – Uma ou mais *features* podem ser escolhidas dentro de um grupo de *features*.
- *Inclusão* - Restrição que inclui uma *feature* B caso a *feature* A seja selecionada.
- *Exclusão* - Restrição que exclui uma *feature* B caso a *feature* A seja selecionada.

Existem algumas maneiras de representar uma estrutura de modelo de *features*, sendo elas através de elementos gráficos ou de fórmulas proposicionais. Segundo Benavides *et al.* (2010) “Uma fórmula proposicional consiste em um conjunto de símbolos primitivos de variáveis e um conjunto de conectivos lógicos que limitam os valores das variáveis.” De maneira formal, um conjunto finito *F* de *features* pode ser representado como um conjunto de variáveis booleanas e seus conjuntos de dependência entre as *features* Φ pode ser considerada como uma fórmula proposicional sobre *F* (BATORY, 2005). Dessa maneira foi definido um mapeamento dos modelos de *features* para uma fórmula proposicional, com o intuito de validar modelos ou mesmo para manipular esses modelos através de ferramentas utilizando essas fórmulas para automatizar o processo de derivação de produtos (CZARNECKI; WASOWSKI, 2007).

A derivação de produtos tem como finalidade gerar um ou mais produtos de software a partir de um conjunto de configurações. As configurações são feitas a partir da seleção de variáveis que serão incluídas no produto. Essa configuração dos produtos pode ocorrer em diferentes fases da derivação de produtos, podendo haver reconfigurações desses produtos durante a fase de derivação. Ao final do processo são gerados produtos que podem ser instalados e/ou implantados em algum sistema para o qual foi designado sua construção (CAPILLA, 2013).

Para esta dissertação foi adotado o conceito apresentado por Deelstra *et al.* (2005) no qual define que: "Um produto é derivado de uma família de produtos se for desenvolvido usando artefatos da família de produtos compartilhados. O termo derivação do produto refere-se, portanto, ao processo completo de construção de um produto a partir de ativos de software da família de produtos." Na literatura é possível encontrar ferramentas para auxiliar nessa fase de derivação de produtos (BEZERRA *et al.*, 2016; BENAVIDES *et al.*, 2007; ACHER *et al.*, 2013).

2.1.3 Ferramentas para auxiliar a derivação de produtos

Além das linguagens desenvolvidas para auxiliar a derivação de produtos, apresentadas na Seção 2.1.2, também foram criadas ferramentas que oferecem suporte ao gerenciamento de produtos derivados de uma LPS, além de automatizar parte do processo de derivação de produtos (THÜM *et al.*, 2014; COSTA *et al.*, 2015; KRUEGER, 2008; MENDONCA *et al.*, 2009).

A DyMMer é um exemplo de ferramenta que auxilia o controle da qualidade de produtos derivados a partir de modelos de *features* (BEZERRA *et al.*, 2016). A ferramenta é composta por 40 medidas para avaliar a manutenibilidade da LPS a partir do modelo de *features*. Além da avaliação da manutenibilidade a partir das medidas, a ferramenta ainda possibilita a importação, edição, visualização e exportação do modelo de *features*.

Outro exemplo de ferramenta que auxilia na derivação de produtos foi proposta por (COSTA *et al.*, 2015). A ferramenta *Fixture* trabalha com *features* que se adaptam de acordo com as informações recebidos do ambiente a partir de sensores e atuadores, definidas como *features* sensíveis ao contexto. A *Fixture* verifica e valida o modelo de *features* sensíveis ao contexto, avaliando automaticamente inconsistências entre os modelos e os produtos derivados. Para representar os modelos de *features* sensíveis ao contexto foi adicionado uma entidade de contexto que representa os contextos relevantes para um domínio e as *features* de informações de contexto que representam os dados que devem ser coletados para a adaptação de aplicativos de domínio (FERNANDES *et al.*, 2008).

Ainda existem outros mecanismos dirigidos a modelos que podem ser encontrados na literatura. A Tabela 1 apresenta exemplos de mecanismos que podem auxiliar o processo de derivação.

Devido a complexidade dos modelos de variabilidade em grande projetos, é necessário o uso de mecanismos que possam auxiliar na derivação dos produtos, proporcionando menor

Tabela 1 – Ferramentas de suporte à LPS

Nome	Descrição	Tipo	Referência
DyMMer	Ferramenta usada para avaliar a manutenibilidade da LPS a partir do modelo de *features*	Ferramenta Desktop	(BEZERRA *et al.*, 2016)
FAMA	*Framework* para a análise automatizada de modelos de *features* integrado com algumas das representações lógicas mais comuns	*Framework*	(BENAVIDES *et al.*, 2007)
FAMILIAR	Linguagem de configuração de modelos de *features* que suporta tipos complexos e primitivos	Ferramenta Desktop e WEB	(ACHER *et al.*, 2013)
FeatureIDE	Ferramenta que suporta todas as fases de desenvolvimento orientado a *features* de software para o desenvolvimento de linhas de produtos	Ferramenta para o Eclipse	(THÜM *et al.*, 2014)
Fixture	Ferramenta para verificação e validação de modelos de *features* sensíveis ao contexto	Ferramenta para o Eclipse	(COSTA *et al.*, 2015)
Gears SPL Lifecycle	*Framework* que suporta todo o ciclo de desenvolvimento de uma linha de produtos	*Framework*	(KRUEGER, 2008)
SPLOT	Fornece um ambiente para edição e configuração de modelos de *features*	Ferramenta WEB	(MENDONCA *et al.*, 2009)

Fonte: Elaborada pelo autor

esforço e custo (RABISER *et al.*, 2010). Porém, o uso desses mecanismos em projetos de LPS pode não ser viável devido a falta de gerenciamento bem definido da variabilidade (RUBIN *et al.*, 2015).

2.2 Linha de Produto de Software Global

O termo Linha de Produto de Software Global (LPSG) foi adotado neste trabalho para designar uma LPS cujos produtos gerados devem ser adaptados para diferentes clientes de diferentes nacionalidades, tendo um grande impacto no âmbito global.

A LPSG estudada nesta dissertação é composta por aplicações para dispositivos móveis, onde aplicações semelhantes devem ser derivadas para centenas de modelos de dispositivos móveis em centenas de operadoras telefônicas espalhadas pelo mundo.

Como a derivação de produtos de forma manual nesse contexto não é uma opção,

devido a grande quantidade de possibilidades de produtos que podem ser gerados, e não há um gerenciamento rígido da variabilidade, optou-se por desenvolver um conjunto de classes em Java[1] que tem como função, adicionar variantes de acordo com as configurações do dispositivo móvel, como por exemplo, a versão do sistema operacional. Esse conjunto de classes é definido no projeto LPSG como classes de customização.

De acordo com os desenvolvedores envolvidos no projeto da LPSG, foi relatado que havia um esforço extra na resolução de tarefas devido a replicação de código das classes de customização em várias partes do projeto. Segundo a documentação do JIRA[2] uma *issue*, pode ser considerada como uma requisição de tarefa composta por um tipo de requisição, prioridade, *status* e resolução. Por padrão, o JIRA define os tipos de requisição de tarefa como:

- *Bug* - Um problema que prejudica ou impede o funcionamento de um produto.
- *New Feature* - Um novo recurso ou funcionalidade do produto, que ainda não foi desenvolvido.
- *Task* - Tipo genérico para tarefa de desenvolvimento relacionadas ao projeto.
- *Improvement* - Melhoria de código de funcionalidades já existentes.
- *Custom Issue* - São *issues* que podem ser adicionadas de acordo com a necessidade do projeto para se adequar ao contexto de desenvolvimento.

.

Além dos tipos, uma *issue* pode conter uma prioridade dos tipos: *Blocker* - caracterizada pelo bloqueio do desenvolvimento e/ou teste. *Critical* - caracterizada por *crashes* ou perda de dados. *Major* - caracterizada pela execução incorreta de funcionalidades do sistema. *Minor* - caracterizada por pequena perda de funcionalidade e que seja fácil de solucionar. *Trivial* - problema cosmético, como palavras com erros ortográficos ou texto desalinhado.

Uma *issue* ainda pode ter um *status* que representa o seu estado atual. O *status* pode ser do tipo *Open*, que representa uma tarefa reportada, na qual será necessário uma pessoa responsável para solucionar essa tarefa. O *status In Progress* indica que a pessoa responsável já iniciou a execução da tarefa. O *status Resolved* indica que a tarefa foi solucionada. Quando a *issue* com *status Resolved* não foi solucionada de forma satisfatório é alterada para o *status Reopened*. Por fim, o *status Closed* indica a finalização da execução sobre uma *issue*.

Como última característica, uma *issue* pode ter uma *Resolution* que indica de que forma ocorreu sua finalização. Uma *Resolution* pode ter o tipo *Fixed* que indica que uma *issue* foi

[1] http://www.oracle.com/technetwork/pt/java/index.html
[2] https://br.atlassian.com/software/jira

testada e aprovada. *Won't Fix* é um problema que não será resolvido. *Duplicate* é um requisição que já foi reportada. *Incomplete* indica que falta informações sobre uma *issue*. *Cannot reproduce* indica que não foi possível reproduzir o cenário reportado no ambiente de desenvolvimento do responsável.

Além de todas essas características que uma *issue* pode ter, ainda pode-se registrar o tempo de execução que o responsável demorou para finalizar a tarefa. Cada *issue* possui um ID que identifica a tarefa com as alterações que foram feitas no sistema. Assim, é possível mapear todas as alterações no sistema com uma *issue* e o tempo que o desenvolvedor relatou para solucionar essa *issue*.

Com essas informações, neste trabalho, foi possível traçar uma relação entre as *issues* reportadas com os clones do sistema. A partir da relação entre *issues* e clones foi feita a análise do impacto dos clones na LPSG.

2.3 Clones

Não é muito frequente a implantação de uma LPS do zero. Na verdade, a implantação de reuso sistemático ocorre após o desenvolvimento de vários produtos similares. Esses produtos apresentam uma mesma base de código que é adaptada de acordo com os requisitos do cliente.

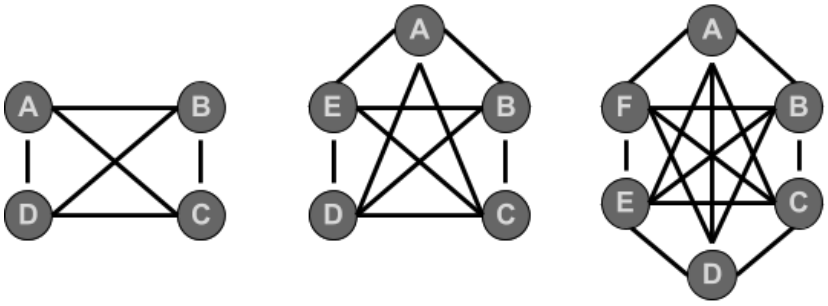
A cópia de fragmentos de código para a posterior reutilização, gerando ou não modificações no código original, reside em uma prática comum no desenvolvimento de software e recebe o nome de clonagem. O fragmento resultante de tal prática é chamado de clone (ROY; CORDY, 2007).

Nesse contexto de reuso entre produtos, a detecção de clones é algo importante para a reengenharia de um sistema com um conjunto de produtos *ad-hoc* em uma LPS (BASIT; JARZABEK, 2009).

Essa relação entre clones pode ser descrita em forma de grafo. A Figura 5 apresenta diferentes grafos que representam alguns exemplos de relações entre fragmentos de código, onde os vértices representam o fragmento de código e a aresta representa que esses fragmentos de código possuem as mesmas linhas de código. Neste trabalho foram considerados como clones apenas os fragmentos de código que não sofreram alteração e, portanto, possuem apenas relações de igualdade.

É possível induzir a partir da Figura 5 que a quantidade de relações entre clones pode ser representado da seguinte forma:

Figura 5 – Exemplos de representação das relações entre clones



Fonte: Elaborada pelo autor

Considerando que *n* é o número de vértices ou número de fragmentos de código, a quantidade de relações que pode existir entre todos os clones é igual a $\frac{n(n-1)}{2}$. Sendo assim, um clone pode ser definido como a quantidade de vértices de um grafo menos um (código primogênito) ou de maneira mais formal podemos definir que a quantidade de clones é igual a $\frac{\sqrt{8x+1}-1}{2}+1$, onde x é a quantidade de relações entre os fragmentos de código. Essa representação matemática é importante para que se possa fazer análises a partir da quantidade de relações entre fragmentos de código.

O surgimento de clones em sistemas de software ocorre devido a adoção de práticas de reuso que buscam uma maior redução de tempo e esforço. Alguns exemplos dessas práticas são: copiar e colar, *forking*, reuso de *design*, funcionalidades e lógica. Essas estratégias podem se tornar comuns em projetos devido à facilidade de execução e as suas vantagens de redução de tempo e esforço em relação a criar um código do zero (ROY; CORDY, 2007) (FALKE *et al.*, 2008).

Na literatura existem trabalhos que enumeram as causas que direcionaram a adoção de práticas de clonagem (ROY; CORDY, 2007) (FALKE *et al.*, 2008) (GUPTA; SURI, 2018). A Figura 6 mostra uma visão em formato de árvore da distribuição das causas que direcionam a prática de clonagem.

A partir da Figura 6, é possível perceber que os clones podem ser inseridos de maneira benéfica ao projeto. Uma empresa pode adotar estratégias de desenvolvimento baseada em reuso ou planejar o reuso de fragmentos de código de forma que possam ser replicados de maneira gerenciada com intuito de melhorar a manutenção de código.

Contudo, os clones podem ser inseridos de maneira indesejada no projeto, seja por limitações ou por acidente. Nesses casos, os clones podem apresentar um impacto negativo no projeto, pois há uma maior chance de criar inconsistências no código ou dependências indesejadas e assim, afetar a manutenibilidade do projeto.

Figura 6 – Visão geral das razões da clonagem

Fonte: Roy e Cordy (2007)

Embora a LPS controle a clonagem a partir do estabelecimento de componentes bem definidos e gerenciáveis, esse controle de clones não é evidente dentro das fases de Engenharia de uma LPS, pois a forma como uma LPS é implementada depende do contexto de seu desenvolvimento (POHL *et al.*, 2005).

Este trabalho aborda especificamente a fase de derivação de produtos, na qual ocorre a customização dos produtos gerados. Para a LPSG estudada, a fase de customização é feita a partir de classes na linguagem Java que geram arquivos XML que são inseridos nos produtos

em tempo de produção. Devido a grande quantidade de produtos derivados a partir das classes de customização, é importante verificar os clones nesse contexto, pois, caso um clone contenha algum problema ou haja a necessidade de alteração, essas modificações devem ser replicadas em todas as partes clonadas com intuito de evitar inconsistências e comportamentos não previstos. Nesse contexto, os clones podem ter um maior impacto, uma vez que uma classe de customização pode estar relacionada a diferentes *features*, na qual pode estar relacionada a centenas ou milhares de produtos gerados pela LPSG.

2.3.1 Tipos de clones

Para identificar os tipos de clones que podem surgir em um projeto, foi identificado na literatura um estudo que levantou dois tipos de clones: clone sintático e clone semântico (ROY; CORDY, 2007). Os clones sintáticos são os clones que possuem similaridade do texto do código. Os clones semânticos são os clones que possuem similaridade em suas funcionalidades sem serem textualmente semelhantes. Para esses dois tipos ainda há um subdivisão em 4 tipos, cuja lista é apresentada a seguir:

Clones sintáticos:

- *Tipo I* - Códigos iguais podendo variar em elementos como espaços em branco ou comentários.
- *Tipo II* - São os clones de Tipo I podendo haver variação em literais, tipos e identificadores.
- *Tipo III* - São clones de Tipo II podendo haver pequenas alterações de adição, remoção ou edição de declarações.

.

Clones semânticos:

- *Tipo IV* - Dois ou mais fragmentos de código que possuem o mesmo comportamento, mas estão implementados de formas distintas.

.

Esses clones podem variar o seu impacto de acordo com o contexto do projeto. Para este trabalho, foram analisados apenas os clones sintáticos do Tipo I e II, porque possuem um maior suporte de ferramentas e técnicas de detecção. Além disso, ainda não existe um consenso na literatura sobre a especificação do Tipo III, pois não é claro o quanto dois fragmentos que possuem relação de igualdade podem variar e ainda serem considerados clones entre eles (LILLACK *et al.*, 2014).

Para detectar os mais diferentes tipos de clones existem várias técnicas e ferramentas, nos quais são apresentados nas Seções 2.3.2 e 2.3.3. Cada tecnologia usada para detecção tem um contexto que ela pode ser usada, mas de maneira geral, os detectores de clones podem ser comparados em termos de retorno de clones detectados e precisão dos clones encontrados (FALKE *et al.*, 2008).

2.3.2 Detecção de clones

Um detector de clones é definido como uma abordagem, técnica ou ferramenta com a finalidade de encontrar fragmentos de código similares em sistema de software. A detecção de clones ainda é uma área de pesquisa em aberto e alguns dos problemas estão relacionados a quantidade de vezes que um código pode estar replicado no projeto, qual o tamanho mínimo de variação entre fragmentos de código similares podem ser inseridos e ainda serem considerados como clones entre eles (Tipo III) ou como melhorar o grau de assertividade e retorno relacionados aos clones do Tipo IV a partir de uma análise sintática do código. Esses problemas estão normalmente relacionados ao custo computacional e devido a isso são propostos vários tipos de algoritmos para detecção de clones em diferentes contextos (ROY; CORDY, 2007) (GHOSH; LEE, 2018).

Na literatura, existem várias abordagens para detectar os mais variados tipos de clones (ROY; CORDY, 2007) (GUPTA; SURI, 2018). Alguns exemplos são:

Baseadas em texto - Nesta abordagem há uma comparação de linha por linha entre fragmentos de código, buscando uma similaridade textual entre eles. Estas técnicas não dependem de tecnologia, filtragem ou normalização dos fragmentos de código, pois são aplicadas diretamente sobre o texto (GUPTA; SURI, 2018).

Baseadas em *tokens* - Esta abordagem transforma o código em uma sequência de *tokens* e tenta encontrar subsequências duplicadas de *tokens*. Essa abordagem é mais robusta em alterações de código, como formatação e espaçamento, porque faz comparações apenas entre *tokens* (ROY; CORDY, 2007).

Baseadas em árvore de análise - É uma abordagem que usa correspondência de padrões para encontrar sub-árvores similares. A partir dessa abordagem é possível encontrar clones do tipo III, podendo ser usado para isso a distância Euclidiana para determinar se um fragmento de código é similar ao outro ou não (GUPTA; SURI, 2018).

Baseadas em métricas - A abordagem baseada em métricas propõe uma comparação

entre métricas de diferentes fragmentos de código. Para isso, é definido a granularidade da detecção de código, essa granularidade pode ser uma classe, uma função, um método ou mesmo uma instrução. Em seguida, os valores de métricas são comparados para localizar os clones nessas unidades.

Para coletar os clones neste trabalho, foi usada uma abordagem híbrida baseada em *token* e árvore de análise, pois essa abordagem retorna uma alta taxa de clones e possui uma alta escalabilidade, embora sua precisão seja baixa, podendo retornar muitos falsos positivos (ROY; CORDY, 2007) (GUPTA; SURI, 2018). A escalabilidade foi tamnbém um fator importante para a seleção desta abordagem de detecção de clones, porque neste estudo foi analisado um projeto real com um grande número de linhas de código. Essa escalabilidade é fornecida pela detecção de clones baseada em *token* em conjunto com um algoritmo de *suffix tree* (ROY; CORDY, 2007).

Uma *suffix tree* é uma estrutura de dados amplamente usada para problemas envolvendo *Strings* com resoluções em tempo linear. O problema mais comum resolvido pelos algoritmos que usam essa estrutura está relacionado à localização de *substrings* de um texto em outro texto. Neste contexto, este tipo de algoritmo pode ser usado para detectar clones, dado um texto inicial **T** com tamanho **m** e é desejável verificar a ocorrência deste mesmo texto **T** em outro texto **S** com tamanho **n** (GUSFIELD, 1997).

2.3.3 Ferramentas

Em busca de auxílio para a execução da coleta de clones para que futuramente eles possam ser analisados a partir das medidas, foi realizada uma busca por ferramentas que executassem essa detecção em código Java e essas ferramentas estão identificadas na Tabela 2.

ConQat é uma ferramenta projetada para auxiliar as atividades de controle contínuo de qualidade em três âmbitos: monitoramento, análise e adaptação. Inicialmente, a ferramenta ConQat foi desenvolvida com intuito de agregar um conjunto de ferramentas para a construção eficiente de *Dashboards* de controle de qualidade. Ao passar do tempo, foram adicionadas ferramentas para auxiliar a detecção de clones e Análise de Conformidade da Arquitetura (CONQAT, 2013).

A ConQat auxilia na detecção de clones sintáticos que podem ser divididos em dois tipos, clones consistentes e inconsistentes. Clones consistentes são regiões de código cujo conteúdo é equivalente. Clones inconsistentes são regiões de código cujo conteúdo pode ser diferente, até certo número de operações de inserção ou deleção de declarações (CONQAT,

Tabela 2 – Ferramentas para auxiliar a detecção de clones

Nome	Descrição	Tipo de Detecção	Referência
ConQat	Uma *toolkit* que forcene detecção de clones a partir de um plugin do Eclipse para as linguagens Java, C#, C/C++, Visual Basic, Cobol and PL/I	*Token* com *Suffix tree*	(JUERGENS *et al.*, 2009)
CCFinder	Ferramenta Desktop com suporte as linguagens C, C++, Java, FORTRAN, LISP e COBOLL	*Token* com *Suffix tree*	(INOUE, 2002)
NiCad	Ferramenta como suporte a duas granularidades, funções e blocos, com possibilidade de detecção de clones com 0%, 10%, 20% e 30% de linhas diferentes entre fragmentos de código com suporte a cinco linguagens, C, C#, Java, Python e WSDL	Baseada em árvore	(CORDY; ROY, 2011)
Deckard	Ferramenta baseada em algoritmo de similaridade de árvore independente de linguagem.	Baseada em árvore	(JIANG *et al.*, 2007)
CloneDR	Uma *toolkit* paga que identifica clones exatos e aproximados e pode ser usada em diferentes linguagens	*Abstract Syntax Tree*	(BURD; BAILEY, 2002)

Fonte: Elaborada pelo autor

2013).

Para realizar a análise do código, a ConQat recebe como entrada os campos: *clone.minlength, input.dir, output.dir e output.reportname*. O campo *clone.minlength* representa o tamanho mínimo do clone, em linhas de código, que será detectado pela ferramenta. O campo *input.dir* deve conter o *path* até a pasta que contêm os códigos que são analisados. O campo *output.dir* deve conter o *path* para onde é enviado os resultados da inspeção. Por último o campo *output.reportname* representa o nome do relatório gerado como saída da análise (CONQAT, 2013).

Além das ferramentas para a detecção de clones, existem ferramentas que auxiliam a refatoração do código a partir dos clones detectados. A JDeodorant[3] é um exemplo de ferramenta para linguagem Java que ajuda na detecção de problemas de *design* a partir dos clones coletados,

[3] https://github.com/tsantalis/JDeodorant

utilizando para isso o módulo *Duplicated Code*.

O módulo *Duplicated Code* da JDeodorant utiliza uma das ferramentas CCFinder[4], ConQAT[5], NiCad[6], Deckard[7] ou CloneDR[8] para realizar a coleta dos clones e a partir dos dados coletados a JDeodorant propõe uma refatoração de forma que o sistema preserve seu comportamento (MAZINANIAN *et al.*, 2016).

Para visualizar os clones, a ferramenta faz um agrupamento dos clones coincidentes, utilizando um ID único para identificar cada grupo. Caso o usuário queira comparar ou visualizar um par de clones do mesmo grupo, basta selecionar os métodos e selecionar a função "*Show textual diff*." Após a ferramenta exibir as diferenças entre os clones, ela determina automaticamente a refatoração apropriada que pode ser aplicada, suportando até três cenários de refatoração:

- *Extract Method*: É aplicado quando os clones estão localizados no mesmo método ou em métodos diferentes da mesma classe.
- *Extract and Pull Up Method*: É aplicado quando os clones estão localizados em métodos pertencentes a diferentes subclasses dentro da mesma hierarquia de herança.
- *Extract Utility Method*: É aplicado quando os clones estão localizados em métodos de classes não relacionadas.

.

Apesar da JDeodorant possibilitar o gerenciamento de todos os clones de um projeto e os separe em grupos, só é possível visualizar a diferença textual apenas entre dois clones, limitando a refatoração do sistema (MAZINANIAN *et al.*, 2016).

Para essa dissertação, a ferramenta ConQat foi selecionada, pois é uma ferramenta com documentação extensa, suporta diferentes sistemas operacionais, possui medidas de clones já integradas na ferramenta e seu relatório de clones serve de entrada para a ferramenta JDeodorant, que é uma ferramenta para auxiliar a refatoração de clones. Embora a ferramenta JDeodorant não seja usada como parte desse trabalho, é importante mencioná-la, pois ela fornece suporte a fase de refatoração que pode ser aplicada após a detecção de clones. As outras ferramentas foram descartadas pelas seguintes razões: não possuíam uma versão compatível com o sistema operacional utilizado, permitem realizar coletas apenas entre duas classes, não oferecem uma documentação explicativa sobre a ferramenta e suas tecnologias, possuem uma interface não

[4] http://www.ccfinder.net/
[5] https://www.cqse.eu/en/products/conqat/overview/
[6] https://www.txl.ca/
[7] https://github.com/skyhover/Deckard
[8] http://www.semdesigns.com/products/clone/

intuitiva e sua última versão ultrapassa 5 anos.

2.4 Medidas de manutenibilidade

A ISO/IEC 25010 (Square) definiu um modelo de qualidade de software que categoriza os atributos de qualidade em características e subcaracterísticas (ISO/IEC, 2011). As características podem ser divididas em Adequação Funcional, Confiabilidade, Usabilidade, Eficiência de Desempenho, Manutenibilidade, Portabilidade, Segurança e Compatibilidade. Cada uma dessas características podem conter várias subcaracterísticas. Uma visão geral do modelo é apresentado na Tabela 3.

A Manutenibilidade é definida como o nível de eficácia e eficiência com que um produto ou sistema pode ser modificado (ISO/IEC, 2011). Para este trabalho, foi questionado os impactos que os clones podem causar na manutenibilidade de um projeto de customização de produtos. Mais especificamente, foi verificado como os clones estão relacionados a modificabilidade relacionadas a evolução as classes de customização.

Conforme mostrado na Tabela 3, a Manutenibilidade é composta pelas subcaracterísticas, Analisabilidade, Modificabilidade, Modularidade, Testabilidade e Reusabilidade. A **Analisabilidade** está relacionada com a facilidade de identificação do impacto de falhas ou mudanças no software. A **Modificabilidade** está relacionada a facilidade de modificação de uma parte do software sem a adição de erros e sem a degradação da qualidade. A **Modularidade** está relacionada ao nível em que um sistema é composto por componentes discretos, de forma que as mudanças tem pouco impacto sobre o sistema. A **Testabilidade** está relacionada a facilidade de testar o software. A **Reusabilidade** representa o grau que uma parte do software pode ser replicada em outros sistemas para ajudar na construção de novos produtos.

Para este trabalho foi utilizada a ferramenta CK tool[9], pois ela inclui medidas levantadas desde 1993 (LI; HENRY, 1993). Embora o início da pesquisa dessas medidas tenha iniciado no ano de 1993, a ferramenta ainda é muito utilizada e possui um repositório no GitHub atualizado, sendo possível contribuir com a evolução do projeto (CRUZ; ELER, 2017) (PADHY *et al.*, 2018) (BINANTO *et al.*, 2018). No total, a CK Tool possui quatorze medidas de manutenibilidade. A Tabela 4 apresenta a nomenclatura e a descrição de cada medida da ferramenta.

[9] https://github.com/mauricioaniche/ck

Tabela 3 – Características e subcaracterísticas de qualidade

Características de Qualidade	**Subcaracterísticas**
Adequação Funcional	Completude Funcional Corretude Funcional Funcionalidade Apropriada
Confiabilidade	Maturidade Tolerância a Falhas Recuperabilidade Disponibilidade
Usabilidade	Conhecimento Adequado Apreensibilidade Operabilidade Acessibilidade Proteção de Erro de Usuário Estética de Interface com o Usuário
Eficiência de Desempenho	Comportamento em Relação ao Tempo Comportamento em Relação aos Recursos Capacidade
Manutenibilidade	**Analisabilidade** **Modificabilidade** **Modularidade** **Testabilidade** **Reusabilidade**
Portabilidade	Adaptabilidade Instabilidade Substituibilidade
Segurança	Confidencialidade Integridade Não-Repúdio Responsabilização Autenticidade
Compatibilidade	Coexistência Interoperabilidade

Fonte: ISO/IEC (2011)

Nessa dissertação, após a coleta das medidas, foi possível realizar análises da evolução das classes relacionando as medidas de manutenibilidade e a taxa de clones. As análises foram baseadas em limites (*thresholds*) que delimitam um intervalo que uma medida pode variar, onde, medidas fora desse intervalo são consideradas discrepantes, podendo indicador problemas de manutenibilidade de acordo com o aspecto que a medida está relacionada. A partir dessas análises é possível identificar alguns dos fatores de impacto que os clones podem causar em classes de customização de uma LPS. Essa análise das medidas é descrita detalhadamente no capítulo 5.

Tabela 4 – Medidas manutenibilidade da ferramenta CK Metrics

Nome	**Descrição**
Coupling between objects (CBO)	Número de dependências de uma classe
Depth Inheritance Tree (DIT)	Profundidade de dependências das classes baseada no nível de hierarquia da classe.
Number of Children (NOC)	Número de filhos uma classe possui em sua hierarquia de dependência
Number of Public Fields (NOPF)	Número de campos públicos de uma classe
Number of Static Fields (NOSF)	Número de campos estáticos
Number of Fields (NOF)	Total de atributos que compõe uma classe
Number of Methods (NOM)	Total de métodos que compõe uma classe
Number of Public Methods (NOPM)	Número de métodos públicos
Number of Static Methods (NOSM)	Número de métodos estáticos
Number of Static Invocations (NOSI)	Número de vezes que métodos estáticos são chamados
Response for a Class (RFC)	Número de diferentes métodos invocados por uma classe
Weight Method Class (WMC)	Número de caminhos lógicos que pode existir em uma classe
Lines of code (LOC)	Número de linhas de código de uma classe, desconsiderando linhas em branco
Lack of Cohesion of Methods (LCOM)	Nível de divergência entre métodos de uma classe

Fonte: Li e Henry (1993)

3 TRABALHOS RELACIONADOS

Este capítulo é constituído de quatro seções. A Seção 3.1 descreve como foi realizado o processo de busca por trabalhos relacionados. A Seção 3.2 apresenta os principais trabalhos relacionados identificados na literatura com uma breve descrição de cada trabalho. Já a Seção 3.3 descreve os trabalhos relacionados à detecção de clones, a qual compõe parte do processo utilizado nesta dissertação. Por fim, a Seção 3.4 apresenta uma discussão sobre os trabalhos relacionados e faz uma comparação com o trabalho desenvolvido nesta dissertação.

3.1 Busca na literatura

Nesta dissertação foram identificados trabalhos relacionados aos impactos causados pela adoção de práticas de clonagem. Essa categoria está relacionada ao processo usado neste trabalho e ao contexto no qual este trabalho está inserido.

Foram utilizadas as fontes de busca da IEEE[1], ACM[2] e Springer[3]. Foram encontrados trabalhos que permitiram sedimentar o conhecimento sobre as abordagens já utilizadas para analisar e classificar clones em diferentes tipos de projetos. Para identificar os trabalhos nas fontes de pesquisa, foi utilizada a seguinte *string* de busca:

```
1 (("Product Line" OR "Product Family") AND ("Clone" OR "Code
    Derivation" OR "Code Similarity" OR "Product Copy") AND
    Maintain*)
```

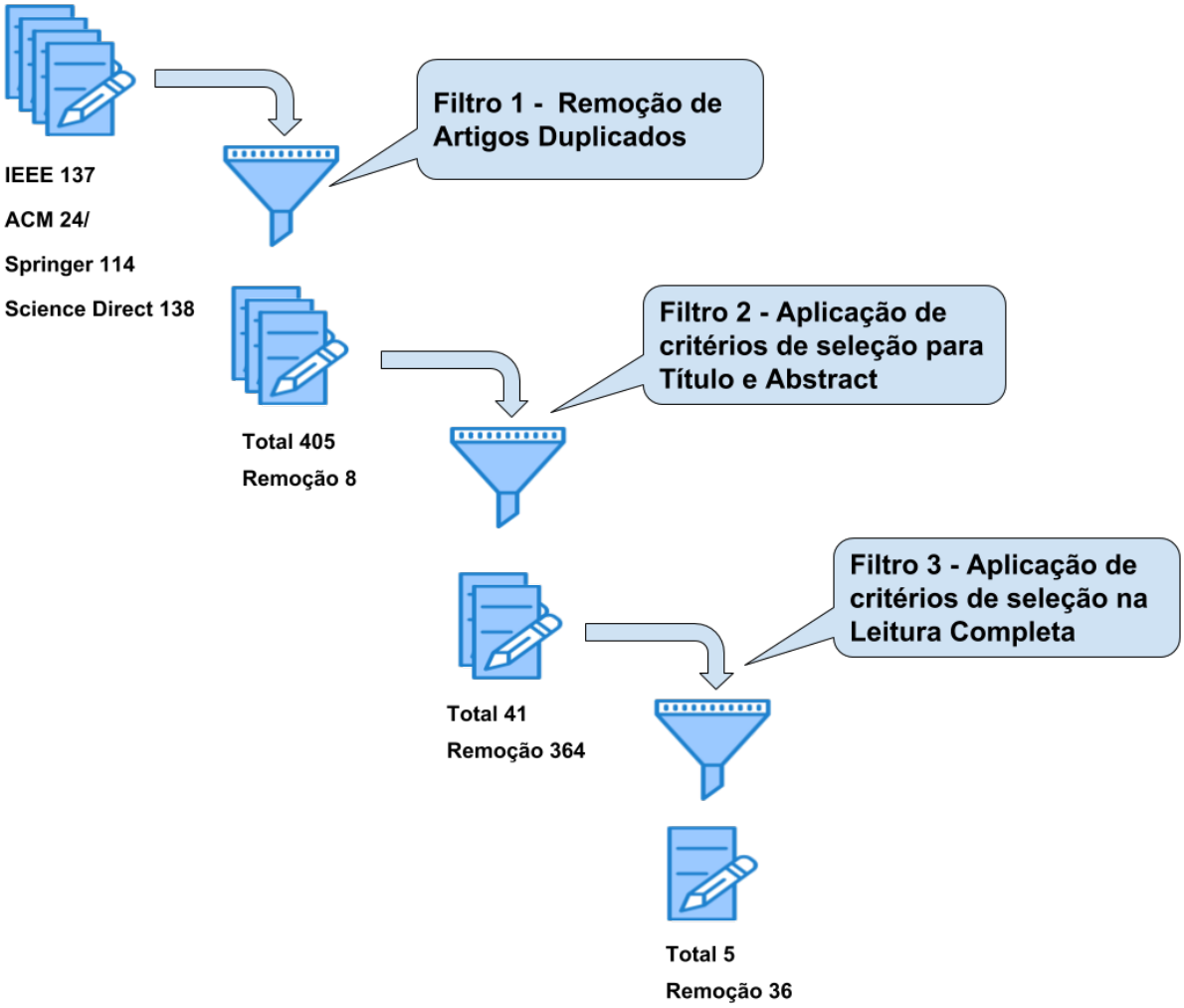
O primeiro fragmento da *string* é formado pelos termos *Product Line* e *Product Family*, pois havia o interesse de limitar os trabalhos para um contexto semelhante ao da LPSG estudada. O segundo fragmento foi composto por termos que pudessem remeter ao contexto de clonagem. Assim, os termos usados foram: *Clone*, *Code Derivation*, *Code Similarity*, *Product Copy*. Por fim, o terceiro fragmento *Maintain** da *string* remete a trabalhos de manutenibilidade que houvesse algum tipo de relação com a clonagem.

Com a *string* definida, foi realizada uma busca na literatura em 20 de março de 2017. Nessa primeira busca foram retornados 238 artigos, onde, 94 artigos eram da IEEE, 22 da ACM e 122 da Springer. Com o intuito de melhorar os resultados da busca, foi realizada uma atualização

[1] https://ieeexplore.ieee.org/Xplore/home.jsp
[2] https://dl.acm.org/
[3] https://link.springer.com/

da busca em 04 de outubro de 2017. Para essa atualização foi adicionada a base *Science Direct*.[4] A partir dessa nova busca foram retornados 137 trabalhos na base da IEEE, 24 da ACM, 114 da Springer e 138 na Science Direct, totalizando um conjunto de 413 artigos. A visão geral dos passos dessa busca na literatura é ilustrada na Figura 7.

Figura 7: Passos da revisão da literatura



Fonte: Elaborada pelo Autor

Os 413 artigos retornados foram submetidos ao primeiro filtro de verificação de artigos duplicados. Como resultado desse primeiro filtro foram removidos 8 artigos repetidos, restando 405 artigos.

O segundo filtro foi composto pela análise dos títulos e *abstracts* dos artigos. Como critério de inclusão, foram considerados apenas os artigos que estivessem relacionados a algum dos termos usados na *string* de busca ou a algum dos seguintes temas: *Variability, Feature, Bad*

[4] https://www.sciencedirect.com/

smells, Code Smells, Fork, Similar, Cross-Project, Aproximate Code, Automatic Production Systems. Além disso, os *abstracts* deveriam estar inseridos no contexto de *Clone Detection, Clone Analysis* ou *Clone Managment*. Todos os artigos que tivessem sua base de pesquisa relacionada a algum desses temas foram aprovados para o próximo filtro, enquanto o restante dos artigos foram descartados. Ao final dessa etapa foram coletados apenas 41 artigos, enquanto 364 artigos foram removidos.

Como última etapa, foi realizada a leitura dos 41 artigos onde foi buscado artigos relacionados à análise de impacto. A análise de impacto poderia está relacionada a experiência de desenvolvedores e pesquisadores com a prática de clonagem e como ela influência na produtividade dos desenvolvedores ou qualidade do software. A análise do impacto dos clones também poderia está voltada para avaliações estatísticas de como os clones podem influenciar em outras partes do projeto, usando para isso, um conjunto de medidas relacionadas ao tamanho dos clones, manutenibilidade do projeto ou mesmo a evolução da arquitetura do projeto. A partir da leitura completa dos artigos, foram verificados que apenas 5 artigos estavam diretamente relacionados à análise dos impactos dos clones, enquanto os outros 36 foram descartados. Uma visão geral dos cinco trabalhos selecionados é apresentado na Seção 3.2.

3.2 Análise dos clones

Esta dissertação teve como objetivo analisar os impactos dos clones e, assim, identificar pontos de melhoria a partir da identificação de correlação entre dados. Na literatura foram identificados cinco trabalhos relacionados que identificaram os impactos dos clones baseados em diferentes aspectos. Uma visão geral de cada trabalho é apresentada a seguir.

O trabalho de Chatterji *et al.* (2016) apresenta uma pesquisa exploratória sobre os impactos de clones a partir de questionários realizados por pesquisadores com experiência em reuso de software. Os questionários são direcionados para três principais objetivos. O primeiro objetivo está relacionado a concordância com o conceito de clones, o segundo está relacionado em como os desenvolvedores lidam com os clones e, por fim, o terceiro está relacionado aos impactos dos clones de acordo com a evolução do projeto.

Em relação à refatoração de clones, o trabalho mostra que existem algumas razões para desenvolvedores não refatorarem partes clonadas, entre elas, é afirmado que a refatoração pode ser um uso desnecessário de recursos quando os clones não são significativos. Por outro lado, a outra metade de entrevistados consideraram aceitável refatorar clones se eles pudessem

ser mesclados em uma função parametrizada. No fim de alguns questionários a pesquisa mostra alguns resultados inconclusivos e sugerem que mais estudos são necessários sobre questões relacionadas a qualidade do software ser afetada pela taxa de clones.

Ao final, o artigo apresenta um direcionamento dos impactos dos clones baseado nas respostas dos questionários. Esse direcionamento pode facilitar o entendimento em quais pontos os clones podem impactar.

O segundo trabalho de Dubinsky *et al.* (2013), analisa os impactos dos clones em uma LPS a partir de entrevistas e questionários com profissionais de seis LPSs industriais. O trabalho identifica quatro categorias relacionadas ao uso da clonagem no contexto das LPSs: Eficiência, *Overhead*, Pensamento de Curto Prazo e Falta de Governança.

Em relação a Eficiência, os desenvolvedores afirmam que a clonagem economiza tempo e reduz custos, pois é mais fácil iniciar o desenvolvimento com uma base inicial. O trabalho apresenta o *Overhead* como sendo uma categoria relacionada ao esforço de adaptação de artefatos clonados às novas necessidades. De acordo com os desenvolvedores, essas adaptações podem gerar *bugs* devido ao novo contexto que se encontra o fragmento de código clonado, criando novos cenário que ninguém tinha pensado antes. A categoria de Pensamento de Curto Prazo está relacionada aos profissionais terem a clonagem como o único mecanismo de reutilização disponível. Esse tipo de pensamento pode ser causado pela falta de planejamento da empresa, falta de recursos ou falta de conhecimento sobre reuso. Alguns desenvolvedores afirmam que não tinham ciência que estavam desenvolvendo uma LPS. A categoria Falta de Governança está relacionada ao direcionamento do processo de desenvolvimento para abordagem com reutilização sistemática. Nessa categoria quase todos os desenvolvedores afirmaram que não existe governança de desenvolvimento de linhas de produtos em suas organizações.

Ao final, o artigo apresenta uma lacuna informando que há uma falta de dados quantitativos sobre quanto os clones economizam ou custam e até onde pode continuar a clonar em um projeto. Essa lacuna foi utilizada como motivação para realizar este trabalho de dissertação, trazendo dados quantitativos sobre a LPS.

O terceiro trabalho de Zhang *et al.* (2012), estuda os clones na perspectiva técnica, pessoal e organizacional. A execução desse trabalho é constituída por três etapas. Na primeira etapa é realizada uma análise dos clones a partir de ferramentas. Na segunda etapa são coletadas informações de desenvolvedores a partir de questionários para entender as razões que leva um programador a adotar práticas de clonagem. Por último, é realizada uma entrevista com os

desenvolvedores para entender melhor as razões das práticas de clonagem. Ao final, o trabalho apresenta resultados relacionados à frequência da adoção de práticas de clonagem, o porque desenvolvedores usam ativamente essa prática de clonagem, e qual a relação dos clones com o *baseline* dos projetos.

O quarto trabalho de Wang e Godfrey (2011), apresenta um estudo sobre os *drivers* do Linux com várias arquiteturas. Cada *driver* tem um ciclo de vida diferente e esse trabalho tenta estudar a evolução dos clones dentro desse tipo de projeto a partir da coleta e análise de dados quantitativos. Como resultado, esse trabalho apresentou a evolução das arquiteturas em relação aos clones, apresentou o nível de alteração consistentes e inconsistente dos clones ao longo do tempo e a previsão de arquiteturas similares de acordo com a taxa de clones entre *drivers*.

O quinto e último trabalho de Rajapakse e Jarzabek (2007) estuda o impacto dos clones e o *trade-off* de aplicar *server page* como forma de unificar partes clonadas. Como resultado, o trabalho apresenta uma lista de *trade-off* que podem ser considerados para a aplicação da unificação de clones ou não. Como conclusão desse trabalho, foi verificado que na prática, nem sempre é possível unificar clones e que o entendimento dos *trade-offs* é importante para saber em quais circunstâncias a prática de clonagem pode ser vantajosa ou não.

Embora tenha sido detectado apenas 5 trabalhos relacionados, os resultados da busca na literatura retornaram vários artigos que descreviam diferentes processos para detectar clones. A Seção 3.3 apresenta os artigos relacionados à fase de detecção de clones que foram usados como base do processo proposto nesta dissertação.

3.3 Detecção de clones

Na literatura, existem trabalhos que apresentam métodos, ferramentas, algoritmos e aparatos para realizar a coleta de clones (FALKE *et al.*, 2008) (BASIT; JARZABEK, 2009) (ALALFI *et al.*, 2014) (LILLACK *et al.*, 2014) (BASIT *et al.*, 2015) (MAZINANIAN *et al.*, 2016).

O trabalho de Falke *et al.* (2008) apresenta tipos de clones, razões que influenciam a prática da clonagem e consequências da adoção dessa prática. Também é apresentado a execução dos algoritmos para detectar clones em profundidade, mostrando vantagens e desvantagens desses algoritmos.

Outro trabalho importante relacionado a detecção de clones foi proposto por Lillack

et al. (2014), no qual é descrito uma abordagem sobre como a detecção de clones pode ser aplicada a geradores de código. Esses geradores podem ser comparados as classes de customização, pois são usados para gerar diferentes variantes de código-fonte para diferentes clientes. Este trabalho também mostra uma ferramenta de detecção de clones e avalia a aplicação a vários geradores. No final, eles validam a ferramenta proposta e identificam que os clones são muito comuns em geradores de código. A principal diferença desse trabalho em relação a proposta desta dissertação é a granularidade e a natureza dos derivadores. Além disso, o trabalho desta dissertação vai além da detecção de clones e aborda o problema de determinar o impacto nas fases de desenvolvimento do projeto.

Em relação a ferramentas, algoritmos e técnicas, pode-se destacar o trabalho de Mazinanian *et al.* (2016), que apresenta uma ferramenta para auxiliar na refatoração de clones. A ferramenta recebe como entrada um arquivo de clones detectados. Esse trabalho apresenta uma visão inicial das ferramentas disponíveis para a prática da detecção de clones.

De uma maneira geral, os trabalhos nesse contexto apresentam novas tecnologias para auxiliar a detecção de clones mostrando benefícios relacionados ao *call-back* que está relacionado a quantidade de clones detectados, quantidade de falsos positivos retornados e escalabilidade para suportar grande volumes de dados. Cada técnica de detecção de clones possui suas vantagens e desvantagens e a partir dessa visão geral das tecnologias de detecções de clones, foi possível delimitar a abordagem de detecção de clones a partir de *tokens* para o contexto dessa dissertação.

Um grande diferencial deste trabalho de dissertação está relacionado a rastreabilidade dos clones e das alterações de todos os fragmentos de código em todo o projeto, devido a adoção de boas práticas de desenvolvimento. Com isso, é possível fazer uma análise quantitativa mais profunda dos dados e obter uma análise geral do projeto.

A partir dos trabalhos identificados na Seção 3.3, foi possível construir uma base teórica de como coletar clones, adaptando este conhecimento ao contexto de LPS na etapa de customização do produto.

3.4 Discussão

Com a revisão da literatura foi possível consolidar o conhecimento relacionado a clones e verificar as lacunas de pesquisa na área. A Tabela 5 mostra uma comparação entre os trabalhos relacionados apresentados na Seção 3.2.

Na Tabela 5, a coluna Coleta de Dados representa a forma como os trabalhos relacionados coletaram as informações sobre a prática de clonagem. Os trabalhos com questionário e entrevista apresentavam um conjunto de perguntas relacionadas às experiências de pesquisadores e desenvolvedores com clones e, a partir das respostas, eram analisados os impactos dos clones nos projetos de acordo com a experiência dos participantes. Os trabalhos que usaram ferramentas como coleta de dados apresentavam uma análise estatística com um grande volume de dados para fazer análises sobre o impacto de clones no código.

Tabela 5: Visão geral dos trabalhos relacionados

Trabalhos Relacionados	**Coleta de Dados**	**Tipo de Análise**	**Domínio**	**Análise da Evolução**
(CHATTERJI *et al.*, 2016)	Questionário	Qualitativo - *Grounded Theory*	Geral	Sim
(DUBINSKY *et al.*, 2013)	Questionário e Entrevistas	Qualitativo - *Grounded Theory*	LPS	Não
(ZHANG *et al.*, 2012)	Questionários, Entrevistas e Ferramentas	Qualitativa	Geral	Sim
(WANG; GODFREY, 2011)	Ferramentas	Quantitativo	Linux drivers	Sim
(RAJAPAKSE; JARZABEK, 2007)	Ferramenta	Quantitativo	Aplicação Web	Sim
Processo da Dissertação	Ferramentas	Quantitativo - Teste de Hipótese	LPS	Sim

Fonte: Elaborada pelo autor

A coluna Tipo de Análise representa como os artigos conduziram suas análises de acordo com os dados coletados. Os trabalhos que faziam interações diretas com participantes através de questionários e entrevistas conduziram suas pesquisas de forma qualitativa usando o *Grounded Theory* como metodologia de análise. Já os trabalhos que coletaram dados a partir de ferramentas, fizeram análises de forma Quantitativa usando métodos estatísticos.

A coluna de Domínio representa o contexto da tecnologia do trabalho relacionado. Os trabalhos com tipo Geral não apresentaram o contexto do sistema analisado, sendo de propósito geral para qualquer sistema que contenha clones.

Por fim, a coluna Análise da Evolução especifica se o trabalho fez uma análise ou não dos clones de acordo com a evolução do projeto com intuito de analisar como os clones podem afetar um projeto ao longo do tempo.

A partir da Tabela 5, é possível perceber que os trabalhos têm uma preocupação com a evolução dos clones, tendo como principal ideia que os clones podem ajudar na agilização das entregas. Porém, ao longo do tempo, eles podem gerar inconsistências e gerar um maior esforço na alteração de partes clonadas. Com a coleta e leitura dos artigos, também foi possível perceber a falta de dados quantitativos que pudesse ser usado pela comunidade, o que serviu de motivação para a aplicação de uma abordagem quantitativa nesta dissertação.

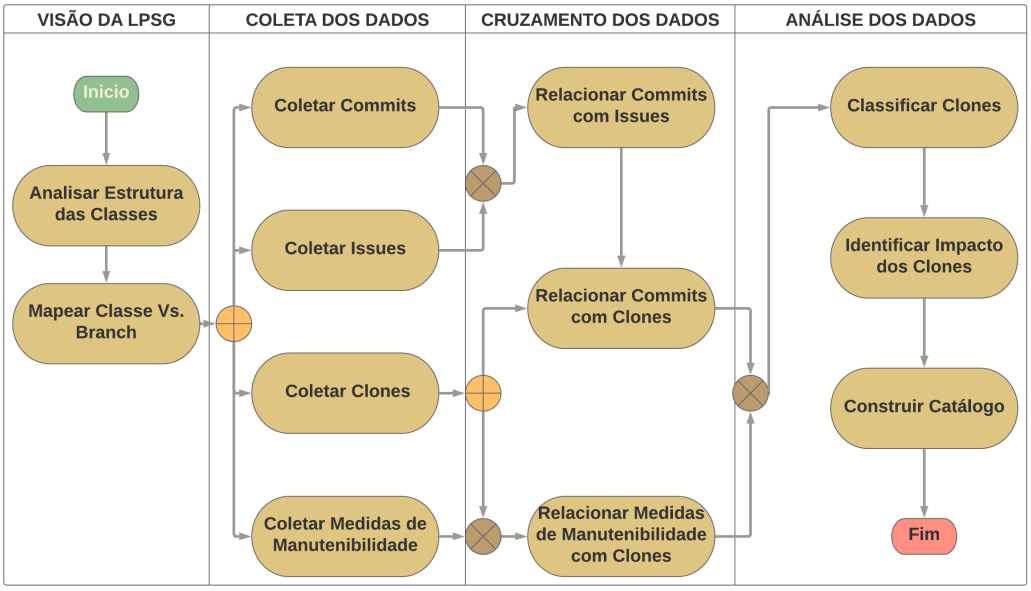
4 PROCESSO PARA A ANÁLISE DO IMPACTO DE CLONES EM UMA LPSG

Neste capítulo é apresentado o processo usado para analisar os impactos dos clones nas fases de desenvolvimento e manutenção de classes de customização. A Seção 4.1 apresenta os passos do processo, descrevendo cada passo e apresentando exemplos dos artefatos gerados em cada etapa, na Seção 4.2 é contextualizado o estudo de caso usado para esta dissertação e quais questões esse estudo ajuda a responder e, na Seção 4.3, é apresentado como o processo foi aplicado sobre o estudo de caso, indicando quais os dados coletados e como foram analisados.

4.1 Visão geral do processo

Para realizar a análise do impacto que os clones podem causar em classes de customização em uma LPSG foi definido um processo contendo quatro fases principais: Visão do Projeto, Coleta dos Dados, Cruzamento dos Dados e Análise dos Dados. Cada fase do processo é composto por um conjunto de etapas que é ilustrado na Figura 8.

Figura 8: Processo para análise do impacto de clones



Fonte: Elaborada pelo autor

A fase de Visão do Projeto é composta pelas atividades, analisar a estrutura das classes e mapear classes em cada produto, onde cada produto possui um fluxo de desenvolvimento (*branch*). A fase de coleta dos dados é composta pelas atividades de coletar *issues*, *commits*, clones e medidas de manutenibilidade. A fase de cruzamento dos dados é composta pelas atividades relacionar *commits* com *issues*, relacionar *commits* com clones e relacionar medidas

de manutenibilidade com clones. Por fim, a análise dos dados é composta pelas atividades de classificação dos clones, identificação dos impacto dos clones e construção do catálogo. A descrição geral de cada atividade é apresentada a seguir:

- **Analisar Estrutura das Classes:** A primeira etapa do processo tem como entrada o projeto que será analisado. Essa etapa consiste em verificar como o projeto organiza suas classes de customização e como funciona sua execução. Como resultado dessa etapa são selecionadas as classes e seus respectivos *branches* que serão analisadas.
- **Mapear Classes vs. *Branch*:** O artefato de entrada dessa etapa é formado pelas classes que foram selecionadas na etapa de Análise da Estrutura das Classes. Esse passo consiste na construção de uma matriz de rastreabilidade para rastrear quais classes estão presentes em cada *branch* e quais as classes foram adicionadas e removidas ao longo do tempo. Como resultado dessa etapa é gerada a matriz de rastreabilidade das classes de customização contidas em cada *branch*.
- **Coletar *Issues*:** Nessa etapa é feita a coleta de todas as *issues* abertas dentro do projeto usado como base de estudo desse trabalho. Como resultado dessa etapa é gerado um arquivo em formato *eXtensible Stylesheet Language* (XLS) contendo todas as informações das *issues* da LPSG.
- **Coletar *Commits*:** Essa etapa tem como entrada as classes que serão analisadas. Essa atividade consiste na coleta de *logs* de *commits* feitos no projetos para rastrear as alterações feitas nas classes de customização. Como resultado dessa etapa é gerado um arquivo no formato texto contendo as alterações da LPSG.
- **Coletar Clones:** Essa atividade tem como entrada as classes que serão analisadas. Nessa fase são coletados os clones entre as classes de customização e entre os *branches*. Essa coleta também é feita ao longo do tempo para verificar como as clones são alterados. Como resultado dessa etapa é gerado um arquivo no formato XML que indica a localização dos clones.
- **Coletar Medidas de Manutenibilidade:** Essa etapa tem como entrada as classes que serão analisadas. A coleta das medidas também é feita sobre as classes de customização para futuramente serem analisados a relação entre os clones e a manutenibilidade das classes. Como resultado dessa etapa é gerado um arquivo no formato CSV organizado por data e por *branch*.
- **Relacionar *Issues* com *Commits*:** A entrada dessa etapa é formada pelos *commits* e

issues extraídos da fase de coleta e consiste no mapeamento entre as *issues* registradas e as alterações do projeto baseado no ID da *issue*. Este mapeamento serve como base para análise dos impactos dos clones no projeto. Como resultado dessa etapa é gerado um *log* contendo todas as *issues* alteradas por um *commit*.

- **Relacionar *Commits* com Clones:** Essa atividade tem como entrada os *commits* e clones extraídos na fase de coleta. Nessa etapa é feito outro cruzamento entre os *commits* e clones para rastrear exatamente quais foram as alterações feitas nas classes de customização que impactaram em fragmentos de código clonados. Como resultado dessa etapa é gerado um arquivo XLS contendo os clones presentes em cada *commmit* e consequentemente em cada *issue*.
- **Relacionar Clones com Medidas de Manutenibilidade:** Essa atividade tem como entrada os clones e as medidas de manutenibilidade extraídos na fase de coleta. Com o cruzamento dos dados entre clones e medidas de manutenibilidade é possível verificar como os clones e suas alterações podem afetar a manutenibilidade das classes de customização a partir de uma análise estatística de correlação. Como resultado dessa etapa é gerado um arquivo XLS contendo os clones presentes em cada classe e um conjunto de medidas de manutenibilidade vinculadas a essa classe.
- **Classificar Clones:** Essa etapa tem como entrada o arquivo XML contendo os clones coletados. Nesse passo os clones são classificados de acordo com suas relações para que haja um entendimento de quais os tipos de clones são mais utilizados, quais que ocasionam mais problemas na LPSG, quais são os tipos de clones mais estáticos e mais voláteis e assim quantificar os tipos que apresentam um maior impacto no projeto de acordo com a subcaracterística de manutenibilidade que se deseja avaliar. Como resultado dessa etapa é gerado um lista com todos os tipos de relações entre os clones.
- **Identificar Impacto dos Clones:** Essa atividade tem como entrada os arquivos gerados no cruzamento dos dados entre clones, medidas de manutenibilidade, *commits* e *issues*. É realizada uma análise dos dados coletados de acordo com a classificação dos dados. Nessa etapa é feita uma avaliação estatística usando teste de hipótese para aceitar ou refutar as hipóteses levantadas. Como resultado dessa etapa é gerado os *thresholds* das medidas de manutenibilidade, um arquivo XSL contendo os impactos dos clones baseados no tempo e na taxa de alteração. O cálculo do tempo foi baseado no tempo registrado pelos desenvolvedores no JIRA que está relacionado ao esforço gasto para solucionar uma *issue*.

- **Construir Catálogo:** Essa atividade tem como entrada o arquivo gerado na fase de Identificação dos Impactos dos Clones. Ao final da análise é feito o cruzamento de todos os dados obtidos para construir o catálogo de clones. Como resultado dessa etapa é gerado uma tabela contendo todas as informações cruzadas dos clones e seus respectivos impactos.

Ao final dessas atividades se obtêm uma matriz de rastreabilidade das classes de customização, uma lista com as *issues* relacionadas aos clones, a quantidade em linhas de código alteradas nas classes de customização, uma classificação dos clones de acordo com os tipos de relações entre fragmentos de código, uma tabela com o nível de relação entre medidas de manutenibilidade e taxa de clones e os impactos que os clones trouxeram para o desenvolvimento do projeto listados a partir do catálogo construído. Um exemplo de todo o processo é apresentado a seguir.

Na primeira etapa do processo foram feitas análises da estrutura do projeto e o mapeamento das classes em seus respectivos *branches*. Na Figura 9 é apresentado um exemplo de classe de customização.

Figura 9: Exemplo de classe de customização

```
public class ClassA extends ScriptSlave<Master> {
    public static final String FILENAME = "filename.xml";
    public static final String ENTRY = "ENTRY";

    @Override
    public void run() throws Exception {
        \\Do something
    }

    private void createAttribute(Element parentElement, String attr, String item,
            String defaultValue) {
        if (item != null || ClassUtuils.isNotEmpty(defaultValue)) {
             String value = x.getValue(item, true);
             String result = ClassUtils.isNotEmpty(value) ? value : defaultValue;
             if (ClassUtils.isNotEmpty(result)) {
                parentElement.setAttribute(new Attribute(attr, result));
             }
        }
    }

    private void createEmptyAttributeIfNoValue(Element parentElement, String attr,
        String item, String defaultValue) {

        if (item != null || ClassUtils.isNotEmpty(defaultValue)) {
            createAttribute(parentElement, attr, item, defaultValue);
        } else {
            parentElement.setAttribute(new Attribute(attr, ""));
        }
    }
}
```

Fonte: Elaborada pelo autor

A partir da Figura 9 é possível verificar que não há diferença de uma classe de

customização para uma outra classe qualquer. Na verdade, a grande diferença dessas classes são os arquivos de configuração gerados que são inseridos nos dispositivos móveis que estão relacionados a uma grande quantidade de produtos derivados da LPSG. Uma outra característica do projeto é a estrutura de divisão das classes em diferentes *branches* como uma forma de separar as características que serão inseridas em cada produto. Contudo, essa separação pode ocasionar a replicação das classes em múltiplos *branches*. Devido a isso, foi feito um mapeamento das classes para saber em quais *branches* cada classe foi replicada. A Tabela no apêndice D mostra todas as classes do projeto e em quais *branches* elas estão presentes.

Após a fase de análise do projeto foi iniciado a fase de coleta. A coleta de *commits* se deu a partir de *scripts* em *batch* onde eram feitas buscas pelos IDs inseridos nos comentários do *commit*. Um exemplo de *commit* é apresentado na Figura 10.

Figura 10: Exemplo de *commit* com adição e remoção de linhas de código

```
Author: authorName
Date:   Fri Jun 24 14:24:36 2016 -0300
    Description:
    Implement upperCase for some fields.

    Jira:
    JIRALink_ID_496
@@ -182,19 +182,19 @@ public class ClassA extends ScriptSlave<Master> {
     }

     private void createAttribute(Element parentElement, String attr, String item,
-            String defaultValue) {
+            String defaultValue, boolean upperCase) {
         if (item != null || ClassUtuils.isNotEmpty(defaultValue)) {
             String value = x.getValue(item, true);
             String result = ClassUtils.isNotEmpty(value) ? value : defaultValue;
             if (ClassUtils.isNotEmpty(result)) {
-                parentElement.setAttribute(new Attribute(attr, result));
+                parentElement.setAttribute(new Attribute(attr, upperCase ?
+                   result.toUpperCase() : result));
             }
         }
     }

     private void createEmptyAttributeIfNoValue(Element parentElement, String attr,
        String item, String defaultValue) {
         if (item != null || ClassUtils.isNotEmpty(defaultValue)) {
-            createAttribute(parentElement, attr, item, defaultValue);
+            createAttribute(parentElement, attr, item, defaultValue, false);
         } else {
             parentElement.setAttribute(new Attribute(attr, ""));
         }
```

Fonte: Elaborada pelo autor

Na Figura 10 é feito um destaque em amarelo para o comentário contendo o ID da alteração, um destaque em vermelho para a linha removida e um destaque verde para um linha de código adicionada. A partir desses *commits* é possível relacionar as alterações feitas no projeto

com os seus respectivos responsáveis, a data de alteração e quais as linhas de código alteradas.

Outra parte da coleta consiste na coleta de *issues*, onde cada *issue* deve conter informações relacionadas a uma tarefa executada dentro do projeto. O projeto contém todas as suas tarefas registradas na ferramenta JIRA que pode exportar essas tarefas em formato de tabela. Um exemplo de tarefa relacionada ao *commit* com ID 496 é apresentado na Tabela 6.

Tabela 6: Exemplo da *issue* ID-496

Tipos de Issue	Issue ID	Responsável	Relator	Prioridade	Status	Resolução	Criação	Tempo
Bug	ID-496	dev1	dev2	Major	Resolved	Fixed	23/06/16 16:38	2 horas

Fonte: Elaborada pelo autor

A partir da *issue* mostrada na Tabela 6 é possível identificar o tipo da *issue*, o seu ID e o tempo de resolução. Essas informações foram usadas como base dessa dissertação.

A terceira tarefa de coleta consiste na coleta de clones a partir da ferramenta ConQat. A saída dessa etapa é formada por: O conjunto de classes que possuem clones entre elas representado na primeira coluna, o tamanho do clone na segunda coluna, a quantidade de fragmentos de código clonados representado na terceira coluna, a quantidade de linhas de código representado na quarta coluna, a linha de código inicial e final de cada fragmento de código clonado em cada classe na quinta e sexta coluna, respectivamente. Um exemplo de retorno é mostrado na Figura 11.

Figura 11: Exemplo de resultado da coleta de clones

Clone Class [801]	13	2	26		
Branch3/ClassA				181	201
Branch2/ClassA				179	199
Clone Class [1144]	13	3	39		
Branch3/ClassB				38	55
Branch3/ClassB				31	48
Branch3/ClassB				40	57
Clone Class [1775]	13	3	39		
Branch1/ClassC				134	155
Branch2/ClassD				141	163
Branch4/ClassE				144	166
Clone Class [11987]	12	5	60		
Branch5/ClassF				74	86
Branch5/ClassB				61	73
Branch5/ClassC				74	86
Branch5/ClassG				64	76
Branch5/ClassH				78	90

Fonte: Elaborada pelo autor

As linhas em destaque amarelo representam o código clonado na classe usada como exemplo apresentado na Figura 9.

Como última etapa da coleta foi usada a ferramenta CK Tool para coletar medidas de manutenibilidade das classes de customização. A Tabela 7 apresenta um conjunto de medidas de manutenibilidade coletados das classes de customização para o *branch* dois.

Tabela 7: Exemplo de medidas de manutenibilidade

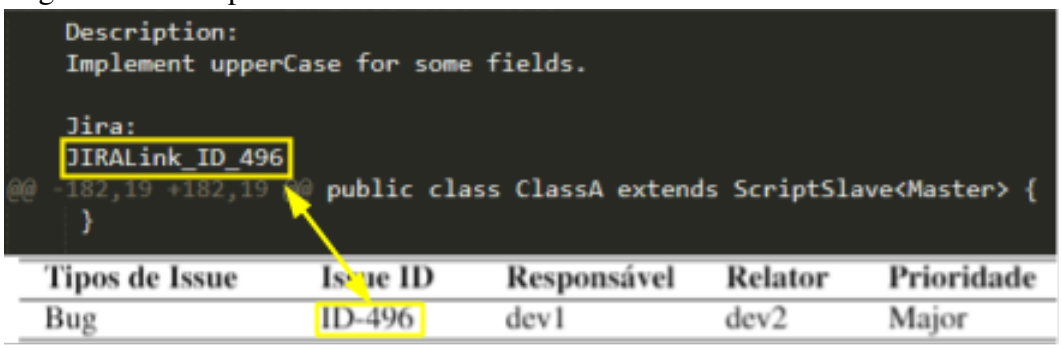
	cbo	wmc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc
ClassA.java	**8**	**19**	**21**	**3**	**3**	**1**	**0**	**2**	**2**	**2**	**0**	**188**
ClassB.java	9	34	22	1	2	1	0	8	4	7	6	140
ClassC.java	10	26	44	0	4	1	0	5	0	0	0	140
ClassD.java	11	60	35	47	11	2	0	12	11	0	0	240
ClassE.java	10	7	20	0	1	1	0	3	0	0	1	65
ClassF.java	9	17	27	10	5	1	0	0	0	0	0	92
ClassG.java	14	44	44	1	9	5	0	24	19	7	1	207
ClassH.java	14	34	43	7	6	2	0	4	2	1	0	163
ClassI.java	12	75	55	14	8	2	0	4	0	0	5	281
ClassJ.java	16	252	67	180	19	4	0	3	0	0	10	621
ClassK.java	11	87	78	463	31	1	0	5	0	0	19	462
ClassL.java	9	67	56	85	15	0	0	2	0	0	0	289
ClassM.java	17	98	79	188	21	2	0	22	0	20	7	487
ClassN.java	19	179	106	225	20	2	0	2	0	0	7	717
ClassO.java	10	63	64	147	18	1	0	8	2	0	0	283
ClassP.java	13	82	72	44	13	2	0	6	0	0	3	389
ClassQ.java	9	43	34	26	8	0	0	4	0	0	13	201
ClassR.java	10	14	31	3	3	3	0	2	2	0	0	92
ClassS.java	7	22	36	28	8	0	0	4	0	0	1	107
ClassT.java	9	19	34	0	5	1	0	6	0	4	0	130
ClassU.java	5	19	29	20	8	1	0	2	0	0	0	97

Fonte: Elaborada pelo autor

A primeira linha na Tabela 7 representa as medidas de manutenibilidade da classe usada como exemplo em um dos *branches* em que ela está presente.

Após a fase de coleta de dados é iniciada a fase de cruzamento dos dados. Na primeira fase do cruzamento é feito o mapeamento entre *issues* e *commits* a partir dos IDs. A Figura 12 ilustra o cruzamento entre o *commit*, representado na parte superior, e uma *issue*, representada na parte inferior.

Figura 12: Exemplo de cruzamento entre *issues* e *commits*

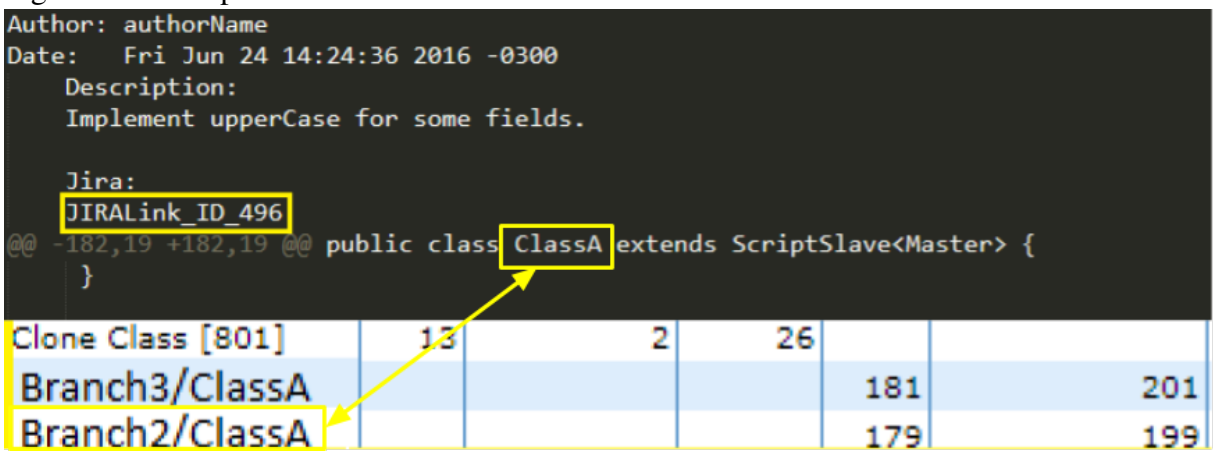


Fonte: Elaborada pelo autor

Na segunda etapa do cruzamento dos dados é feito o mapeamento entre os clones retornados pela ConQat e os *commits*. Nessa fase é feito o mapeamento a partir do nome da

classe alterada no *commit* em determinado *branch*. A Figura 13 ilustra o cruzamento entre um *commit* na *ClassA* no segundo *branch* representado na parte superior da figura e os clones detectados pela ConQat para essa classe representado na parte inferior da figura.

Figura 13: Exemplo de cruzamento entre clone e *commits*



Fonte: Elaborada pelo autor

A partir desse segundo cruzamento entre os dados é possível obter uma rastreamento entre clones, *commits* e *issues*, assim, obtendo um mapeamento triangular entre esses dados.

Como última etapa do cruzamento dos dados é feito uma separação de medidas de acordo com as classes e os seus respectivos *branches*. Na Tabela 7 já foram mostradas as classes e suas medidas de manutenibilidade separadas para o *branch* dois. Com essa separação é possível quantificar e rastrear as medidas de cada classe de customização em seus respectivos *branches*, além de saber quais os clones contidos em cada classe, quais as partes alteradas de cada classe e quais as *issues* relacionadas a cada alteração.

A última etapa do processo é formada pela análise dos dados. Como primeira etapa dessa fase é realizada uma classificação dos clones para diferenciar onde cada clone se enquadra. Para isso, os clones foram separados em quatro tipos: DI, ID, DD e II. DI representa diferentes *branches* com classes iguais, ID representa *branches* iguais com classes diferentes, DD representa diferentes *branches* e diferentes classes e II representa *branches* iguais e classes iguais. Essa classificação permite identificar a natureza das relações dos clones, e assim, entender os diferentes níveis de acoplamento que um fragmento de código pode estar envolvido. Além disso, um clone ainda pode ser classificado de acordo com nível de igualdade, onde o clone do Tipo I representa clones que são exatamente iguais e clone do Tipo II representa clones que podem haver variações na nomenclatura de suas variáveis, por exemplo. A Tabela 8 apresenta

exemplos de classificações que um clone pode ter.

Tabela 8: Catálogo contendo a mediana dos dados coletados

Tipo de Relação	**Branch 1**	**Branch 2**	**Branch 3**	**Branch 4**	**Branch 5**	**Tipo Clone I**	**Tipo Clone II**	**Tam. Clone**	**Nº Fragmentos Cod.**
DS	**Não**	**Sim**	**Sim**	**Não**	**Não**	**1**	**—**	**13**	**2**
SD, DS, DD	Sim	Sim	Não	Não	Não	1	2	13	3
DS	Sim	Sim	Não	Sim	Não	3	—	13	3
SD, DS, DD	Sim	Sim	Sim	Não	Não	4	6	12	5
SS, DS	Sim	Sim	Não	Não	Não	2	4	12	4
SD	Não	Não	Sim	Não	Não	—	3	12	3
SS, DS	Sim	Sim	Não	Não	Não	2	4	12	4
SD, DS, DD	Sim	Sim	Não	Não	Não	2	4	12	4
SS, DS	Sim	Sim	Não	Sim	Não	4	6	12	5
DS	Sim	Sim	Sim	Não	Não	1	2	12	3
SS, DS	Sim	Sim	Não	Não	Não	1	2	12	3
DD	Sim	Não	Sim	Não	Não	1	—	12	2
SS, DS	Sim	Sim	Não	Sim	Não	6	9	12	6
DS	Sim	Sim	Não	Não	Não	—	1	12	2

Fonte: Elaborada pelo autor

A primeira linha da Tabela 8 representa o exemplo da ClassA usada até o momento. A partir dessa linha é possível perceber que esse clone é do tipo DS, identificando que o clone está presente em diferentes *branches*, mas está dentro da mesma classe replicada nos *branches*. O clone está presente nos *branches* dois e três e possui apenas uma relação de Tipo I, que significa que o clone é exatamente igual entre os *branches* que possuem esse clone. Além disso, esse clone tem o tamanho de 13 linhas de código estando presente em dois fragmentos de código, um para cada classe em cada *branch*.

Na etapa de identificação do impacto dos clones foram considerados dois fatores: tempo de resolução de tarefas que envolvem clones e taxa de alteração que esse clones sofrem a partir dessas tarefas. O tempo é medido através dos registros dos desenvolvedores no JIRA como já foi mostrado um exemplo na Tabela 6. Os registros de alterações foram coletados a partir dos *commits*, um exemplo das taxas de alterações coletadas é apresentado na Tabela 9.

É possível verificar que as classes de customização não sofrem uma grande quantidade de alterações, porém essas alterações devem ser consideradas em cada *branch* que a classe se está presente.

Ao final de todos os passos foi feito a construção do catálogo. O catálogo consiste em uma tabela contendo os dados mapeados e classificados de cada clone detectado no projeto. A partir de *thresholds* é possível identificar clones que possam estar impactando na manutenibilidade do projeto e assim facilitar a refatoração do sistema. A Tabela 10 mostra um exemplo de mapeamento dos dados para um clone presente na *issue* com ID 496.

Tabela 9: Exemplo da quantidade de alterações

	Total LoC	**Qtd. LoC Alteradas**	**Taxa de Alteração**
ClassA	**131**	**3**	**0,02290076336**
ClassB	331	3	0,009063444109
ClassC	62	12	0,1935483871
ClassD	261	17	0,06513409962
ClassE	141	12	0,08510638298
ClassF	854	11	0,01288056206
ClassG	148	0	0
ClassH	151	18	0,119205298
ClassI	117	1	0,008547008547
ClassJ	365	1	0,002739726027
ClassK	112	0	0
ClassL	360	0	0
ClassM	40	0	0
ClassN	441	2	0,004535147392
ClassO	179	0	0
ClassP	137	17	0,1240875912
ClassQ	150	7	0,04666666667
ClassR	111	0	0
ClassS	890	31	0,03483146067
ClassT	148	0	0
ClassU	261	1	0,003831417625

Fonte: Elaborada pelo autor

Tabela 10: Exemplo do catálogo para o clone contido na ID-496

Class.	**Tipo 1**	**Tipo 2**	**Tam.**	**Qtd. Clone**	**wmc**	**rfc**	**nopm**	**loc**	**LoC Adic.**	**LoC Rem.**	**Issue**	**Tempo**
DS	1	X	13	2	19,5	21,5	1	189	1	2	ID-496	2 horas

Fonte: Elaborada pelo autor

A primeira coluna da Tabela 10 identifica que os clones foram replicados em *branches* distintos a partir de uma mesma classe. A segunda e terceira coluna indicam que os fragmentos de código são totalmente iguais, pois são do Tipo I e não possuem relações do Tipo II. O Tamanho do clone é igual a 13 e possui apenas dois fragmentos nessa relação. Em seguida é apresentado as medidas de manutenibilidade da classe na qual o clone está presente, a quantidade de modificações que esse fragmento sofreu, qual a *issue* essas modificações estão atreladas e qual o tempo para resolver toda a tarefa que ocasionou a alteração do fragmento de código clonado. O restante dos dados referente ao catálogo foi anexado no apêndice C.

O principal objetivo do processo é definir um conjunto de passos com entradas e saídas que podem ser replicados em diferentes projetos. Os passos do processo foram definidos de forma empírica, onde os dados coletados e as análises desses dados foram guiados pelas questões de pesquisa e hipóteses levantadas, apresentadas no Capítulo 1. Vale ressaltar que o objetivo do processo é guiar a análise do código dentro das classes de customização, e não dentro das variantes do produto.

4.2 Estudo de caso das classes de customização de uma LPSG

O estudo de caso selecionado para essa dissertação tem como principal característica o uso de classes de customização para gerar arquivos que adaptam *features* de *smartphones*. Os arquivos são inseridos nos dispositivos móveis em tempo de *design* para adequar as funcionalidades de acordo com as necessidades do cliente. O sistema conta com uma interface web que é responsável por gerenciar as solicitações de customização.

A LPSG estudada usa o padrão *Master-Slave* onde uma classe *Master* é responsável por gerenciar as requisições e enviar para as subclasses *Slaves* ou classes de customização. Dessa forma, a classe *Master* escuta as requisições que chegam através do sistema web e envia a solicitação para um conjunto de classes que irão gerar os arquivos de customização de acordo com as *features* selecionadas.

Em relação a arquitetura da LPSG, os desenvolvedores e arquitetos definiram que o mecanismo de customização seria separado por *branches*. Em Junho de 2017 a LPSG já continha 11 *branches*, onde cada uma continha um conjunto de classes de customização, podendo haver compartilhamento dessas classes entre *branches*. Quando o sistema recebe uma requisição de customização, é verificado qual *branch* atende aquela tipo de customização, essa checagem pode ser baseada, por exemplo, na versão do sistema operacional do modelo do *smartphone*.

Tendo em vista essa organização da LPSG, antes de iniciar a coleta dos dados desse projeto, foram formuladas as seguintes questões de pesquisa:

QP1 - Quanto tempo de desenvolvimento e manutenção é gasto para resolver *issues* com clones em classes de customização em uma LPSG? Respondendo a esta pergunta será possível quantificar o tempo gasto para resolver problemas no contexto de derivação de produtos, bem como fornecer uma visão dos impactos causados pelos clones nas classes de customização em diferentes fases do desenvolvimento da LPSG.

Para a QP1 foram levantadas duas hipóteses. A primeira hipótese relaciona o tempo gasto com *issues* de desenvolvimento com os clones. Uma *issue* é classificada como do tipo **Desenvolvimento** nos casos em que ela foi registrada no JIRA[1] em um dos seguintes tipos: implementação, *new feature* e tarefa. Uma *issue* de implementação está relacionada ao ato geral de incluir código para um requisito planejado; *new feature* é uma *issue* que está voltada para adição de novas funcionalidades; e a *issue* do tipo tarefa está ligada a um processo de negócios.

[1] https://br.atlassian.com/software/jira

H_0: *Issues* do tipo desenvolvimento que possuem código clonado em classes de customização levam menos tempo para serem solucionadas.

H_1: *Issues* do tipo desenvolvimento que possuem código clonado em classes de customização levam mais tempo para serem solucionadas.

Com essa hipótese, é possível verificar se *issues* que usam a prática de clonagem como desenvolvimento da solução são mais complicadas ou propensas a levarem mais ou menos tempo para serem solucionadas.

A segunda hipótese está relacionada à fase de manutenção. As *issues* que compõem a fase de manutenção são *bug* e melhoria. Sendo melhoria uma *issue* originada por uma necessidade de refatoração; e *bug* sendo problemas relacionados a defeitos reportados por testadores, usuários ou desenvolvedores.

H_2: *Issues* do tipo manutenção que possuem código clonado em classes de customização levam menos tempo para serem solucionadas.

H_3: *Issues* do tipo manutenção que possuem código clonado em classes de customização levam mais tempo para serem solucionadas.

A avaliação dessas hipóteses permitirá verificar os impactos da clonagem em um contexto de um mecanismo complexo de derivação usado para gerar milhares de produtos.

Para este trabalho também foi realizado um estudo sobre a evolução do projeto para entender como os clones neste tipo de projeto evoluem. Para isso, foi levantada a seguinte pergunta:

QP2 - Qual o nível de alterações sofrida por partes clonadas ao longo do tempo em projetos de LPSG? A resposta a essa pergunta permite quantificar a frequência das mudanças que ocorrem em fragmentos de códigos clonados e verificar se há um aumento ou redução de clones ao longo do tempo. Para isso, foi levantada a seguinte hipótese.

H_4: Fragmentos de código clonado em classes de customização têm uma taxa de mudança maior em comparação com partes sem clones.

H_5: Fragmentos de código clonado em classes de customização têm uma taxa de mudança igual ou menor em comparação com partes sem clones.

Com intuito de identificar outros impactos relacionados aos clones em classes de customização, foi elaborada a terceira questão de pesquisa sobre a correlação entre a manutenibilidade do projeto e a taxa de clones.

QP3 - Como os clones afetam a manutenibilidade das classes de customização?

Com a resposta a essa pergunta é possível quantificar quais as medidas de manutenibilidade estão mais relacionadas aos clones. Para a QP3 foi levantada a seguinte hipótese.

H_6: Medidas de manutenibilidade não possuem uma forte correlação com a taxa de clones em classes de customização.

H_7: Medidas de manutenibilidade possuem uma forte correlação com a taxa de clones em classes de customização.

Com a resposta da terceira pergunta é possível identificar as medidas que possuem relação com os clones e assim identificar como os clones podem impactar na manutenibilidade.

Com as questões levantadas é possível fazer uma relação entre elas para identificar os impactos dos clones em classes de customização baseadas no tempo gasto, na taxa de mudanças e na manutenibilidade, e assim, propor um catálogo com fatores de impacto como uma forma de ajudar desenvolvedores a priorizar a refatoração de clones. Além disso, é possível quantificar e prever como a inserção de um clone pode impactar o desenvolvimento no futuro.

4.3 Aplicação do processo

Após a especificação das questões de pesquisa, foi definido o intervalo de tempo de 13 meses para coletar todos os dados definidos no processo, entre o mês de Junho de 2016 até Junho de 2017 e, desse intervalo, foram selecionadas apenas classes de customização que estão presentes no projeto durante todo esse período. Neste trabalho, foram executadas dois tipos de coletas de clones: clones entre classes de customização e clones entre *branches*.

4.3.1 Sobre o projeto da LPSG

O primeiro passo ilustrado na Figura 8 consiste na análise da estrutura da LPSG para selecionar quais os artefatos que serão analisados.

No total, foram usadas 70 classes, distribuídas e replicadas em 11 diferentes *branches*. Neste projeto, os *branches* são independentes, e cada *branch* tem um objetivo específico e é responsável por gerar um conjunto de arquivos XML, OGG, CFG, PROP, JSON e JPG que são usados para customizar os dispositivos móveis.

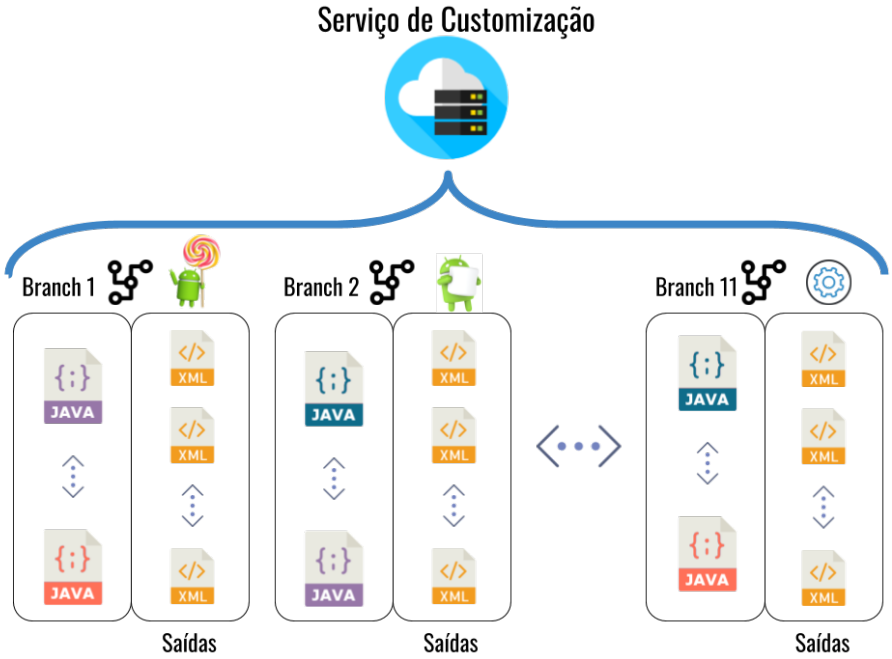
Esses 11 *branches* podem ser agrupados conforme os arquivos de customização gerados e suas funções. Para analisar os clones entre *branches* foram selecionados um *branch* de cada tipo como uma forma de reduzir a quantidade de clones analisados e obter uma representati-

vidade de cada grupo. A seguir é apresentado uma descrição de cada grupo que um *branch* pode pertencer na LPSG:

- *Grupo I* - *Branches* responsáveis por gerar arquivos de customização com base na versão do sistema operacional de dispositivos móveis;
- *Grupo II* - Customização no comportamento do dispositivo não vinculado a um aplicativo específico (por exemplo, internet, configurações de rede, nome da rede, idioma padrão, etc.);
- *Grupo III* - Customização de aplicativos (por exemplo, mensagens, navegador, chamada telefônica, etc.);
- *Grupo IV* - Customização para clientes específicos devido a suas regras de negócio; e
- *Grupo V* - Relacionado com a geração de APKs customizados.

O segundo passo do processo ilustrado na Figura 8 consiste em mapear cada classe em cada *branch* para entender a distribuição das classes de customização entre os *branches*, como elas mudaram ao longo do tempo e para saber de quais *branches* cada classe seria coletada. A Figura 14 mostra uma organização geral das *branches* no projeto

Figura 14: Organização do projeto



Fonte: Elaborada pelo autor

Conforme mostrado na Figura 14, cada *branch* é direcionado para um tipo de configuração específica. Por exemplo, o *branch* 1 é específico para modelos móveis com a versão

5.0 do Android, e o *branch* 11 gera um conjunto de arquivos de configuração independentes da versão do sistema operacional onde são empacotados e enviados para o cliente de forma automática. As classes de customização representadas pelos arquivos Java podem ser replicadas em diferentes *branches* do projeto. Essas classes são responsáveis por gerar os arquivos de customização que serão inseridos nos dispositivos móveis em tempo de projeto. A partir dessas classes de customização, foram coletados os dados que compõem este trabalho para a elaboração do *dataset*[2].

Além dos dados relacionados as classes, o *dataset* também é composto de *issues* e *logs* dos *commits* que foram usados para rastrear alterações no código. Para essa coleta foram verificados todos os *logs* e as *issues* que estavam relacionados as alterações das classes de customização. Ao final dessa coleta foram obtidas 293 *issues*. Na Tabela 11 são apresentados os principais dados do experimento.

Tabela 11: Dados gerais do estudo de caso

Dados	**N° Classes**	**Total LoC**	**N° *Issues***	**N° *Branches***	**Média/Mês Clones**
Valor	70	123472	293	5	275

Fonte: Elaborada pelo autor

4.3.2 Coleta dos dados

Nessa etapa foi iniciado o processo de coleta de dados do estudo de caso ao longo dos 13 meses. Para a execução da coleta de clones, foi utilizada a ferramenta ConQat[3], pois ela fornece uma extensa documentação, possui uma versão recente da ferramenta, usa algoritmos baseado em *suffix-tree* e é compatível com a ferramenta JDeodorant, na qual pode ser usada no futuro para refatoração de clones (MAZINANIAN *et al.*, 2016). Além das coletas relacionadas aos clones, foram coletados também *issues* e os *logs* de *commits* relacionados a cada *issue*. Como não há dependência entre os dados coletados, todas as atividades dessa fase do processo podem ser executadas em paralelo.

Coleta de Clones - Antes de realizar a primeira coleta de clones entre classes de customização e entre *branches*, foi definido um tamanho mínimo dos clones em linhas de código. Para obter um primeiro panorama da distribuição de clones no projeto, foi definido 20 linhas de código na ferramenta ConQat. Após a primeira coleta, foram realizados ajustes

[2] https://github.com/GREatResearches/Software-Product-Line/tree/master/Dataset
[3] https://www.cqse.eu/en/products/conqat/overview/

no tamanho mínimo para verificar a variação dos clones. No total, ocorreram quatro ajustes e, consequentemente, cinco coletas para o conjunto de 70 classes de customização, em que os tamanhos mínimos coletados foram: 10, 11, 12, 15 e 20.

Com intuito de amenizar os impactos dos falsos positivos retornados pela ferramenta, foi realizada uma análise manual de cada clone retornado pela ferramenta referente ao mês de Junho de 2016, pois foi o mês que retornou a maior quantidade de clones. Foi feita uma verificação dos falsos positivos para cada ajuste do tamanho mínimo, resultando nos seguintes valores: 2 falsos positivos com tamanho igual a 10, 1 com tamanho 12, 1 com tamanho 15, 1 com tamanho 17 e 1 com tamanho 19. Também foi verificado para o tamanho mínimo igual a 9, contudo a maioria dos clones coletados para esse tamanho apresentavam apenas um conjunto de declarações de variáveis ou um grande conjunto de *if* e *else*, não representando uma função do projeto. Com estes resultados, foi definido o tamanho mínimo igual a 10, pois esse tamanho retorna uma maior quantidade de clones e apresenta apenas 6 falsos positivos referente a soma dos falsos positivos do tamanho 10 a 20. Assim, seis falsos positivos não foram considerados como uma quantidade relevante para uma média de 275 clones coletados mensalmente durante 13 meses. O espaçamento entre as coletas dos clones foi definido como um mês devido à natureza do projeto, já que a equipe de desenvolvedores faz um fechamento de uma versão do produto mensalmente, entregando parte dos requisitos de forma iterativa.

Com o tamanho mínimo dos clones definido, foi iniciada a coleta de clones entre as classes de customização ao longo do tempo. O objetivo dessa coleta é verificar como os clones evoluem ao longo do tempo de vida do projeto usado como caso de uso. Para isso, foram selecionadas apenas as classes que tinham pelo menos um ano e um mês de criação, obtendo como resultado, 70 classes a serem analisadas.

Ainda relacionado a coleta de clones, foi selecionado 1 *branch* de cada grupo para coletar clones entre esses *branches*. Para essa coleta, foram utilizadas as mesmas condições de tamanho mínimo do clone e tempo da coleta realizada entre classes de customização.

Coleta de *Issues* - Para coletar as *issues*, foi usado a função de exportar da ferramenta JIRA, onde é possível obter um arquivo no formato *Comma Separated Values (CSV)* que facilita a manipulação dos campos através de tabelas e ferramentas de análise de dados como, por exemplo, a ferramenta R[4].

Como resultado dessa primeira coleta, foram obtidas 293 *issues*. Dessas 293 *issues*,

[4] https://www.r-project.org/

foram excluídas 130 porque não foram encontrados os *commits* respectivos no *log* do Git. Por fim, dessas 163 *issues*, foram excluídas mais 22 *issues*, pois o tempo de resolução gasto na solução não foi registrado no JIRA, resultando no final 141 *issues* que foram consideradas para esse estudo de caso.

Coleta de *Commits* - O projeto usado como estudo de caso, tem como parte do seu processo interno a política de adicionar o ID da respectiva *issues* que está sendo tratada na descrição do *commit*. Essa prática permite fazer o mapeamento entre *issues* e *logs* de *commits* relacionados ao projeto. A partir desse mapeamento foram rastreados os *commits* que possuíam os IDs das *issues* e salvos em arquivos de texto para serem analisados. O comando a seguir foi usado para retornar os *commits* do projeto com o ID correspondente de uma *issue*:

```
1 for /l \%\%id in (x, y, z) do git --no-pager
2 log --all --grep= \%\%id" -p > git-log/log\%\%id.txt
```

Esse comando cria um *loop* que começa com o valor da variável *x* (valor do ID da primeira *issue*), um iterador incrementado pelo valor de *y* (nesse caso, um por um, pois os IDs das *issues* são incrementadas dessa maneira) até que *x* atinja o valor de *z* (o ID da última *issue*) que é a condição de parada. Para cada etapa do *loop*, o comando busca pelos *commits* com o ID numérico. Em seguida, todos os *commits* relacionados a uma *issue* com o mesmo ID são adicionados a um arquivo de texto para que possam ser analisados automaticamente na fase de cruzamento de dados.

Coleta das Medidas de Manutenibilidade - Para a terceira e última parte da coleta de dados foi usado a ferramenta CK tool, apresentada no Capítulo 2. No total foram usadas as quatorze medidas da CK tool sendo elas: *Coupling between objects*, *Depth Inheritance Tree*, *Number of Children*, *Number of public fields*, *Number of static fields*, *Number of fields*, *Number of methods*, *Number of static methods*, *Coupling between objects*, *Number of static invocations*, *Response for a Class*, *Weight Method Class*, *Lines of code*, *Lack of Cohesion of Methods*. Para fazer a coleta das medidas, foi criado um *script* para executar a ferramenta de maneira automática para todas as classes de customização ao longo dos 13 meses analisados. A chamada da ferramenta através do *script* é apresentado a seguir:

```
1 git checkout nome_branch
2 java -jar ck.jar input_path_classes
3 output_path_meidas_classes
```

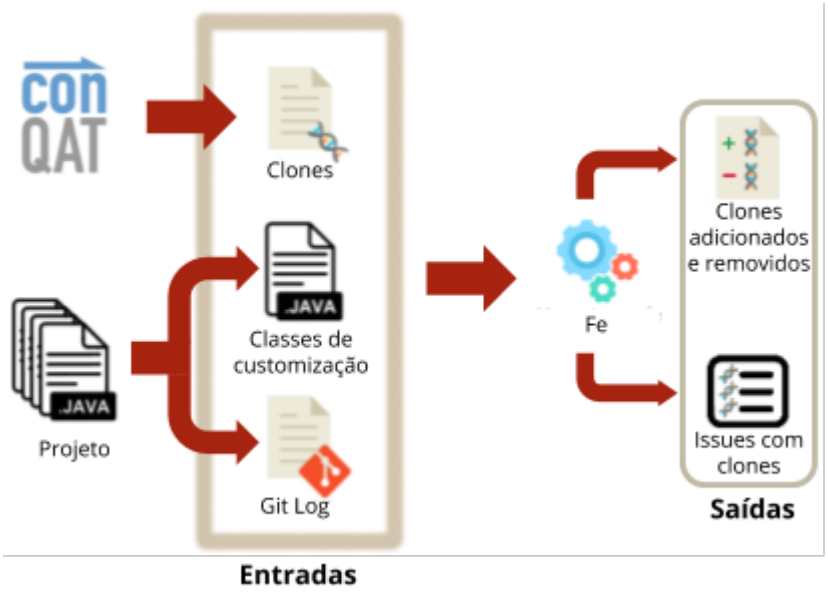
Como resultado dessa etapa, foi obtido um conjunto de arquivos CSVs[5] para cada mês e cada *branch*, contendo o conjunto de 14 medidas da CK tool.

4.3.3 Cruzamento dos dados

Após a coleta dos dados, foi iniciado o processo de cruzamento dos dados coletados para posteriormente serem analisados. Primeiramente, foi feito o relacionamento entre *issues* e clones. Com essa etapa, foram geradas informações importantes sobre os dados coletados, sendo agora possível diferenciar *issues* que contém fragmentos clonados e as que não possuem clones.

Relação entre *issues* e clones - Com as demais 141 *issues* retornadas da fase de coleta, foi iniciada a análise para extrair apenas as *issues* relacionadas aos fragmentos de código clonado. Uma *issue* tem relação com um clone quando qualquer um dos seus *commits* adiciona ou remove algum fragmento de código clonado. Para realizar esta análise, foi desenvolvido um mecanismo para identificar se as partes alteradas de um *commit* são partes de um clone. O desenvolvimento deste mecanismo foi baseado na linguagem Java. O algoritmo para este mecanismo tem como entrada três arquivos: classes de customização de um projeto Java, o histórico de *commits* do projeto e a lista de clones detectados. A visão geral do mecanismo é ilustrada na Figura 15.

Figura 15: Mecanismo para o cruzamento de clones com *issues*



Fonte: Elaborada pelo autor

[5] https://github.com/GREatResearches/Software-Product-Line/blob/master/Dataset/Ck-Metrics-Manutenibilidade.xlsx

O mecanismo lê os três arquivos e os armazena em objetos distintos para facilitar o gerenciamento de seus atributos. Os *commits* são armazenados como um conjunto de fragmentos alterados, onde um fragmento é marcado como *added* se ele tiver um símbolo "+" ou *removed* se o fragmento de código tiver o símbolo "−". Já os clones são armazenados como um objeto com uma linha inicial e linha final que indicam o começo e o término de um fragmento de código clonado. O mecanismo armazena as classes de customização como texto, onde as partes alteradas por *commits* serão pesquisadas. Para filtrar os *commits* com clones, primeiro, o mecanismo verifica se um fragmento de código alterado por um *commit* está entre o intervalo de linha inicial e final do objeto clone. No caso afirmativo, ele é classificado como um fragmento do tipo clone *added* ou *removed*, de acordo com a alteração feita. No final deste processo, é possível obter fragmentos de clones adicionados e removidos ao longo do tempo e as *issues* com clones vinculados. Esse mecanismo ajuda a automatizar grande parte do esforço de classificação, pois cada *issue* pode conter muitos *commits*. Foram detectadas *issues* com mais de 15 *commits* vinculados, portanto, verificar as 141 *issues* manualmente seria uma tarefa difícil e propensa a erros.

O mecanismo também permite mapear as linhas alteradas com as partes clonadas. Para a coleta das linhas alteradas de uma classe de customização, foi construído um *script* usando comandos do Git para verificar as mudanças das classes ao longo do tempo. Foi executado o comando abaixo para as 70 classes:

```
1 git --no-pager log --after=date-start --before=date-end
2 --pretty=tformat:  --numstat | grep classname.java | awk
3  {inserted+=$1; deleted+=$2;} END {printf " \%s \n \%s \n",
4 inserted, deleted}  -> git-changes/classname.txt
```

Esse comando faz uma busca em todos os *commits* entre duas datas para uma determinada classe de customização. As linhas de código adicionadas e removidas através desses *commits* são inseridas em variáveis. No final, os valores das variáveis também são exportados para um arquivo de texto. A partir dessa coleta será possível analisar o impacto das mudanças em fragmentos de código clonado em classe de customização.

Relacionar *Issues* com *Commits* - Para fazer esse cruzamento foi usado apenas os arquivos gerados no passo de coleta de *commits* na Seção 4.3.2. Esses arquivos foram criados a partir do mapeamento da chave de uma *issue* com a chave na descrição de um *commit*.

Relacionar Clones com Medidas de Manutenibilidade - Nessa etapa foi usada a ferramenta R para fazer a correlação entre as quatorze medidas de manutenibilidade da CK tool e a taxa de clones. Para isso, foi utilizado a correlação de *Spearman*, não paramétrica, pois essa correlação não requer que a relação entre as variáveis seja linear e não necessita que os dados apresentem uma distribuição específica (WOHLIN *et al.*, 2012). O cálculo de correlação é definido pela seguinte equação:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{4.1}$$

d = Diferença entre as variáveis usadas para verificar a correlação.

n = Quantidade de amostras usadas para fazer a correlação.

Como resultado dessa equação é definido o grau de relação entre duas medidas baseado na lista abaixo (WOHLIN *et al.*, 2012):

- 0.8 até 1.0 ou -0.8 até -1.0: correlação muito forte;
- 0.6 até 0.8 ou -0.6 até -0.8: correlação forte;
- 0.4 até 0.6 ou -0.4 até -0.6: correlação moderada;
- 0.2 até 0.4 ou -0.2 até -0.4: correlação fraca;
- 0.0 até 0.2 ou 0.0 até -0.2: correlação muito fraca ou não existe relação

Durante essa etapa foram removidas as medidas *Number of Children* e *Depth Inheritance Tree*, pois não apresentavam nenhuma variação entre as classes de customização. O resultado do cruzamento pode ser visto na Tabela 12.

Tabela 12: Correlação entre medidas de manutenibilidade e taxa de clones

	clones%	**cbo**	**wmc**	**rfc**	**lcom**	**nom**	**nopm**	**nosm**	**nof**	**nopf**	**nosf**	**nosi**	**loc**
clones%	X	-0,53	**-0,60**	**-0,68**	-0,57	-0,38	**0,68**	-0,46	-0,46	-0,04	-0,5	-0,47	**-0,65**
cbo	X	X	0,78	0,80	0,75	0,51	-0,85	0,48	0,48	0,00	0,69	0,52	0,64
wmc	X	X	X	0,86	0,85	0,57	-0,94	0,47	0,47	0,04	0,65	0,50	0,91
rfc	X	X	X	X	0,91	0,74	-0,86	0,47	0,47	-0,20	0,90	0,81	0,82
lcom	X	X	X	X	X	0,84	-0,84	0,47	0,47	-0,22	0,88	0,77	0,83
nom	X	X	X	X	X	X	-0,53	0,08	0,08	-0,66	0,86	0,90	0,62
nopm	X	X	X	X	X	X	X	-0,50	-0,50	-0,06	-0,65	-0,51	-0,88
nosm	X	X	X	X	X	X	X	X	1,00	0,52	0,42	0,09	0,46
nof	X	X	X	X	X	X	X	X	X	0,52	0,42	0,09	0,46
nopf	X	X	X	X	X	X	X	X	X	X	-0,41	-0,68	0,02
nosf	X	X	X	X	X	X	X	X	X	X	X	0,90	0,69
nosi	X	X	X	X	X	X	X	X	X	X	X	X	0,54
loc	X	X	X	X	X	X	X	X	X	X	X	X	X

Fonte: Elaborada pelo autor

A partir desse cruzamento foi possível identificar que 4 medidas de manutenibilidade que tinham uma maior relação com a taxa de clones: WMC, RFC, NOPM e LOC. Essas medidas

foram utilizadas no catálogo apresentado no Capítulo 5, desenvolvido para identificar os fatores de impacto na manutenibilidade causada por clones.

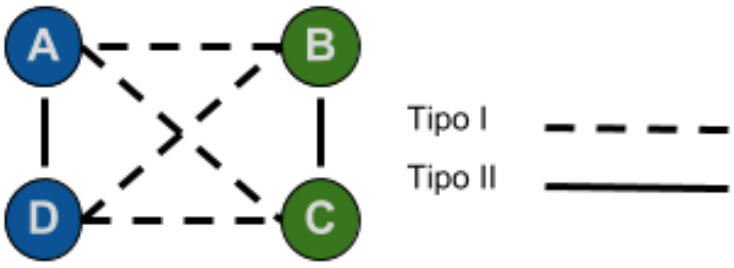
4.3.4 Análise dos dados

Conforme apresentado na Figura 4.1, a primeira etapa da análise de dados consiste em classificar os clones que foram coletados. A fase de classificação dos clones aconteceu através de uma análise manual de todas as relações existentes entre todos os fragmentos de código clonado. No total foram analisados 1398 relações entre 873 fragmentos de código. Para esta dissertação foi considerado que cada relação entre clones deve ter um indicador referente a localização de cada fragmento e outro indicador para o tipo de clone. Uma relação pode ter um dos quatro indicadores de localização, sendo eles:

- Relação de igualdade entre fragmentos de código de diferentes *branches* dentro de uma mesma classe replicada entre os *branches*;
- Relação de igualdade entre fragmentos de código de diferentes *branches* de diferentes classes;
- Relação de igualdade entre fragmentos de código de um mesmo *branch* dentro de uma mesma classe; e
- Relação de igualdade entre fragmentos de código de um mesmo *branch* de diferentes classes.

A segunda classificação representa o tipo de relação de clonagem entre dois fragmentos de código, que, para este trabalho, podem ser do Tipo I ou Tipo II, onde a descrição desses tipos pode ser vista na Seção 2.3.1. A Figura 16 apresenta um exemplo de relação entre 4 fragmentos de código clonados representados pelas letras A, B, C e D e 2 *branches* distintos, representados pela cor verde e azul.

Figura 16: Tipos de relações entre fragmentos de código



Fonte: Elaborada pelo autor

Com o exemplo pode-se visualizar a complexidade da análise de relações entre fragmentos de código, pois, quanto maior a quantidade de fragmentos de código vinculados a um clone, maior a quantidade de tipos de relações podem existir entre os clones. Ao final, essa classificação foi usada no catálogo para construir uma amostragem dos tipos de clones e os seus respectivos impactos.

A partir dos dados coletados com a execução do processo foi possível construir um *dataset* no qual foi usado como base para análise das correlações dos dados e, assim, identificar os impactos dos clones baseado na correlação desses dados. A execução do processo também possibilitou a classificação dos dados que foram usados para vincular os impactos dos clones a cada tipo de clone. A partir do cruzamento de todos os dados coletados foi possível construir um catálogo com os impactos dos clones.

As análises dos resultados foram baseadas em teste de hipóteses, onde é levantada uma hipótese nula em que os testes realizados tem a intenção de rejeitar essa hipótese para que a hipótese alternativa possa ser levada em consideração. A partir do teste de hipótese é possível definir a significância dos resultados, que é representado pelo *p-value* (WOHLIN *et al.*, 2012). Um *p-value* menor significa que há evidências mais fortes a favor da hipótese alternativa.

Para realizar os testes foi utilizada a ferramenta RStudio[6], na qual possui a função *t.test* que possui como entrada os dados referentes a hipótese que se deseja verificar e o tipo de comparação da hipótese alternativa com a hipótese nula. Os tipos aceitos pela função *t.test* são: *less*, *greater* ou *different*. Para todos os testes nessa dissertação usou-se uma significância de 5%, que corresponde a resultados com *p-value* menor que 0.05 um forte indício da hipótese alternativa ser verdadeira.

A partir da verificação dos testes é possível identificar o impacto dos clones e a correlação com a taxa de clones em classes de customização, possibilitando construir o catálogo que mapear pontos críticos de manutenibilidade no sistema. Os resultados e discussões sobre os impactos dos clones e o catálogo são descritos no Capítulo 5.

[6] https://rstudio.com/

5 IDENTIFICAÇÃO DO IMPACTO E CONSTRUÇÃO DO CATÁLOGO DE CLONES

Neste Capitulo é apresentada a análise estatística dos impactos dos clones usando teste de hipótese e a especificação do catálogo. Na Seção 5.1 é apresentada a comparação entre o tempo gasto com *issues* com clones e sem clones. A Seção 5.2 descreve a análise comparativa entre a taxa de alteração de fragmentos de código com clones e sem clones. Na Seção 5.3 é especificado o catálogo produzido a partir da análise da correlação entre medidas de manutenibilidade e a taxa de clones e o cruzamento de dados do tempo gasto com clones e a taxa de alteração. Por fim, na Seção 5.4 é realizada uma discussão baseada nos resultados obtidos dessa dissertação e no contexto em que o trabalho está inserido.

5.1 Análise do impacto dos clones

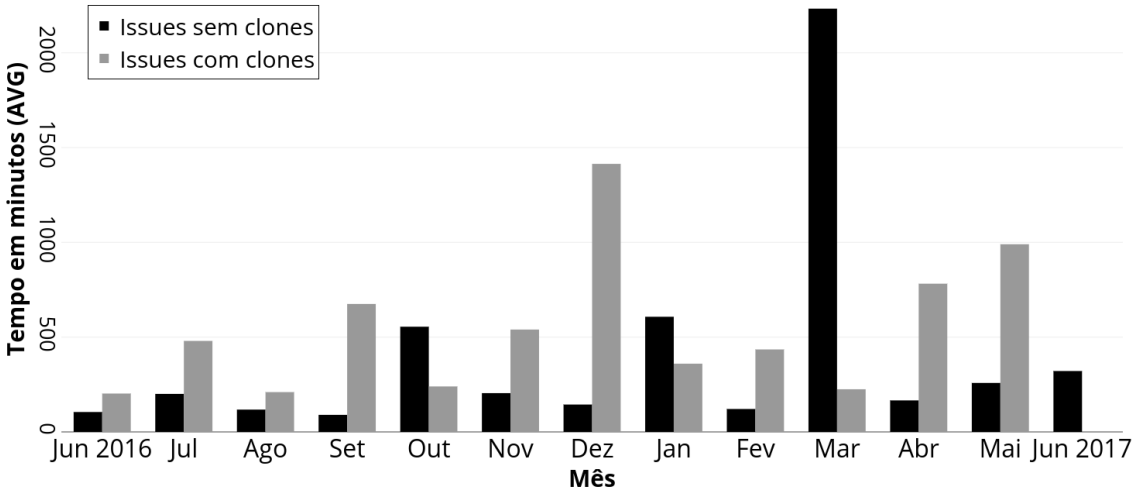
Com a coleta dos dados, foi criado um *dataset* para analisar os impactos dos clones nas classes de customização. A partir dessas análises foi possível construir um catálogo com os tipos de clones, os tipos de relações e o impacto vinculado a cada tipo.

A análise dos impactos dos clones foi realizada em dois âmbitos: impacto dos clones entre classes de customização e entre *branches*.

Para responder a questão de pesquisa **QP1**, citada no Capítulo 1, ocorreu uma separação das *issues* em dois grupos: *issues* com clones e *issues* sem clones. Após essa separação, foi contabilizado a média de tempo gasto com essas *issues* para cada grupo e para cada mês. A Figura 17 ilustra o tempo médio gasto em todas as *issues* de cada grupo ao longo do tempo.

É possível notar que, em geral, o tempo de resolução de *issues* com clones é maior que sem clones. As *issues* com clones levam em média 501.66 minutos e as *issues* sem clones 329.40 minutos. Também é possível perceber que no mês de Junho de 2017 não foi detectado nenhuma alteração em fragmentos de código clonado. Além disso, foi verificado a existência de somente uma *issue*, não relacionada a clones, com tempo discrepante no mês de Março. Essa *issue* causou considerável redução da diferença de tempo médio entre os grupos e tem o maior tempo com um total de 13620 minutos, enquanto a segunda tem apenas 3660 minutos. Se ignorarmos essa *issue* considerando-a como um *outlier*, o tempo médio de *issues* sem clones diminuiria de 329 para 212 minutos, resultando em uma diferença de 136% a mais no tempo de resolução de *issues* com clones. Mesmo com essa discrepância, as *issues* com clones ainda

Figura 17: Tempo para solucionar *issues* com clones e sem clones



Fonte: Elaborada pelo autor

obtiveram maior custo de tempo para serem solucionadas.

Para entender como esses clones podem afetar a fase de desenvolvimento das classes de customização, definimos a seguinte hipótese:

H_0: *Issues* do tipo desenvolvimento que possuem código clonado em classes de customização levam menos tempo para serem solucionadas.

H_1: *Issues* do tipo desenvolvimento que possuem código clonado em classes de customização levam mais tempo para serem solucionadas.

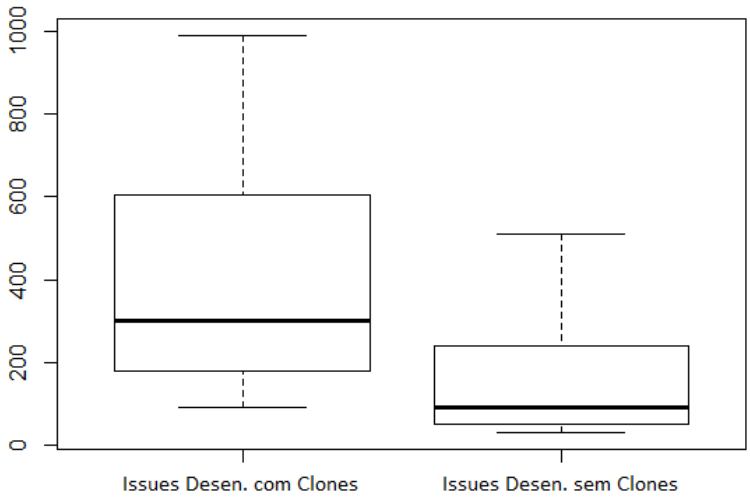
Como primeiro passo, houve uma separação dos dados entre *issues* com clone e *issues* sem clones que estavam relacionadas a fase **Desenvolvimento**, sendo elas: *desenvolvimento*, *nova feature* e *tarefa*. A Figura 18 apresenta um gráfico *boxplot* com intuito de ilustrar a diferença do tempo de resolução de *issues* com clones vs. *issues* sem clones.

Com o resultado, foi obtida uma mediana de 300 minutos ou 5 horas para *issues* com clones e 90 ou 1 hora e 30 minutos para *issues* sem clones. A partir desses resultados, foi aplicado o teste de hipótese para verificar se os dados amostrais trazem evidências que apoiem a hipótese levantada. O teste de hipótese teve um intervalo de confiança de 95% e obteve como resultado um *p-value* igual a 1.812e-05, **tornando possível considerar a hipótese alternativa H_1 como verdadeira para o contexto de classes de customização**.

A mesma análise foi feita para as *issues* do tipo de manutenção. As *issues* contidas no tipo **Manutenção** podem ser dos tipos: *bug* ou *melhoria*. Similar à fase de desenvolvimento, nessa etapa foi levantada a seguinte hipótese.

H_2: *Issues* do tipo manutenção que possuem código clonado em classes de customi-

Figura 18: Tempo gasto com *issues* de desenvolvimento

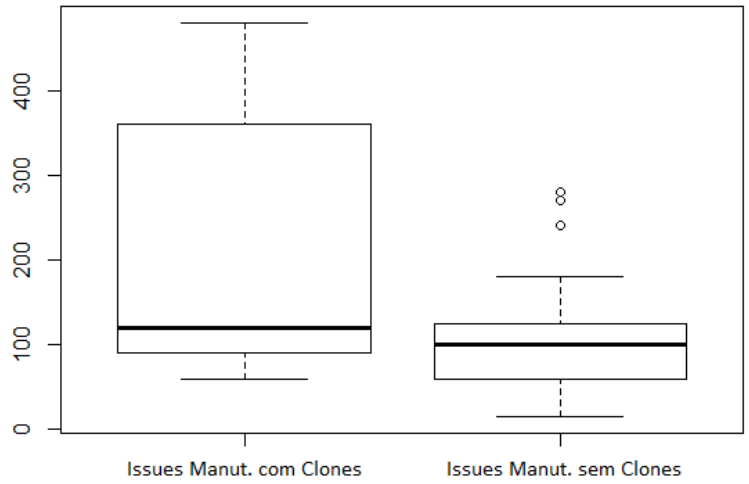


Fonte: Elaborada pelo autor

zação levam menos tempo para serem solucionadas.

H_3: *Issues* do tipo manutenção que possuem código clonado em classes de customização levam mais tempo para serem solucionadas.

Figura 19: Tempo gasto com *issues* de manutenibilidade



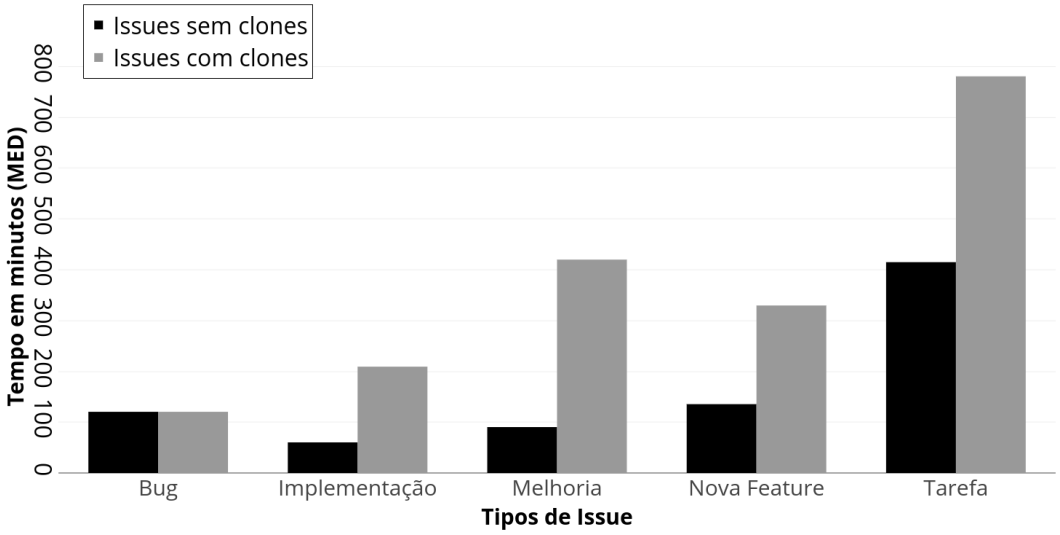
Fonte: Elaborada pelo autor

Como resultado foi obtido uma mediana de 120 minutos ou 2 horas para *issues* com clones vinculados e 100 ou 1 hora e 40 minutos para *issues* sem clones. Aplicando o teste de hipótese a um intervalo de confiança de 95%, foi obtido um p próximo de um, não sendo possível considerar a hipótese alternativa H_3 para esse contexto.

A partir da Figura 20, é possível obter uma visão da distribuição dos tempos de soluções das *issues* de acordo com os seus tipos.

De acordo com a Figura 20, é possível perceber que as *issues* relacionadas à melhoria

Figura 20: Visão geral do tempo gasto com *issues* agrupadas pelos seus tipos



Fonte: Elaborada pelo autor

possuem a maior diferença de tempo entre todos os grupos. Também é possível notar que todos os grupos com clones tiveram um tempo médio maior em sua resolução, reforçando os resultados retornados pelos testes de hipótese.

5.2 Análise do impacto dos clones em relação a taxa de alteração

Para a questão de pesquisa **QP2**, foi buscado entender como ocorre a evolução dos clones do projeto com o passar do tempo. Primeiramente, foi verificado como a taxa de clones é alterada com o passar do tempo. É possível observar a partir da Figura 21 que os clones podem evoluir de diferentes maneiras.

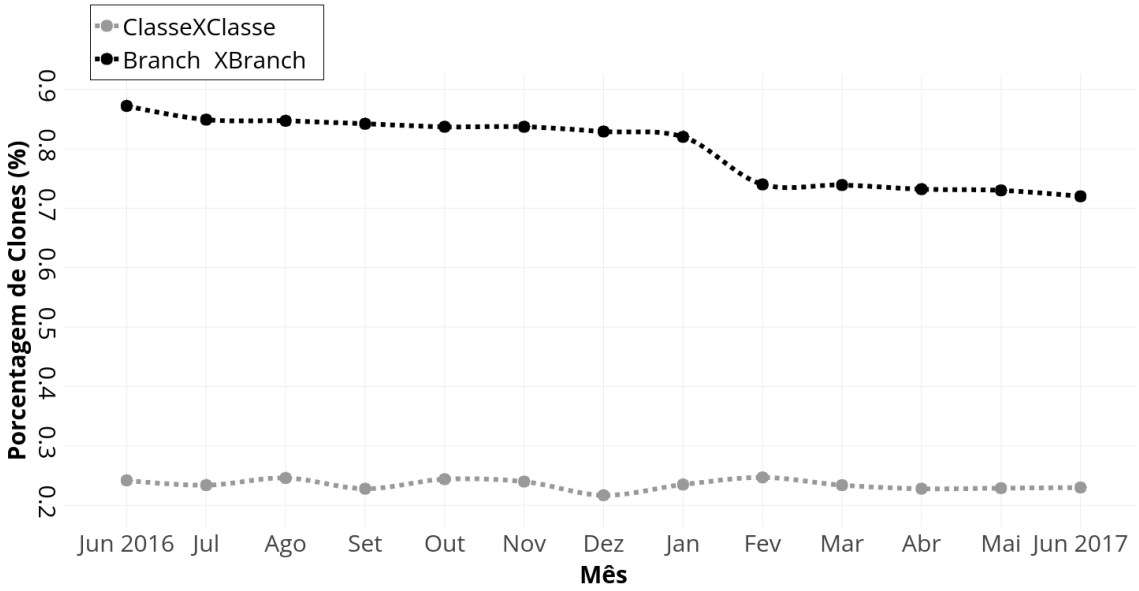
Observando a linha preta, é notável que a porcentagem de clones entre *branches* sempre diminui com a passagem do tempo. Por outro lado, clones em classes de customização têm comportamento semelhante aos clones de projetos de software individuais, pois à medida que novos clones são adicionados, outros são removidos, mantendo equilíbrio no número de clones ao longo do tempo, como foi relatado no trabalho de (KIM *et al.*, 2005).

Para entender o nível de magnitude da taxa de mudança dos clones, foi realizada uma coleta das taxas de alteração das classes de customização e, para essa etapa, levantada a seguinte hipótese.

H_4: Fragmentos de código clonado em classes de customização têm uma taxa de mudança maior em comparação com partes sem clones.

H_5: Fragmentos de código clonado em classes de customização têm uma taxa de

Figura 21: Evolução dos clones em % entre *branches* e entre classes de customização



Fonte: Elaborada pelo autor

mudança igual ou menor em comparação com partes sem clones.

O resultado mostra uma taxa média de mudanças de 0,13 % para LoC com clones e 0,96 % para LoC sem clones. Aplicando o teste de hipótese a um intervalo de confiança de 95%, obteve-se um valor de p = 0,07117 aproximando-se do limiar de significância, mas não sendo possível refutar a hipótese nula.

A partir da Figura 22, observa-se que as classes de customização não possuem uma taxa de mudança significativa e, de acordo com a hipótese, não há uma diferença substancial entre a taxa de mudança entre fragmentos de código clonados e não-clonados.

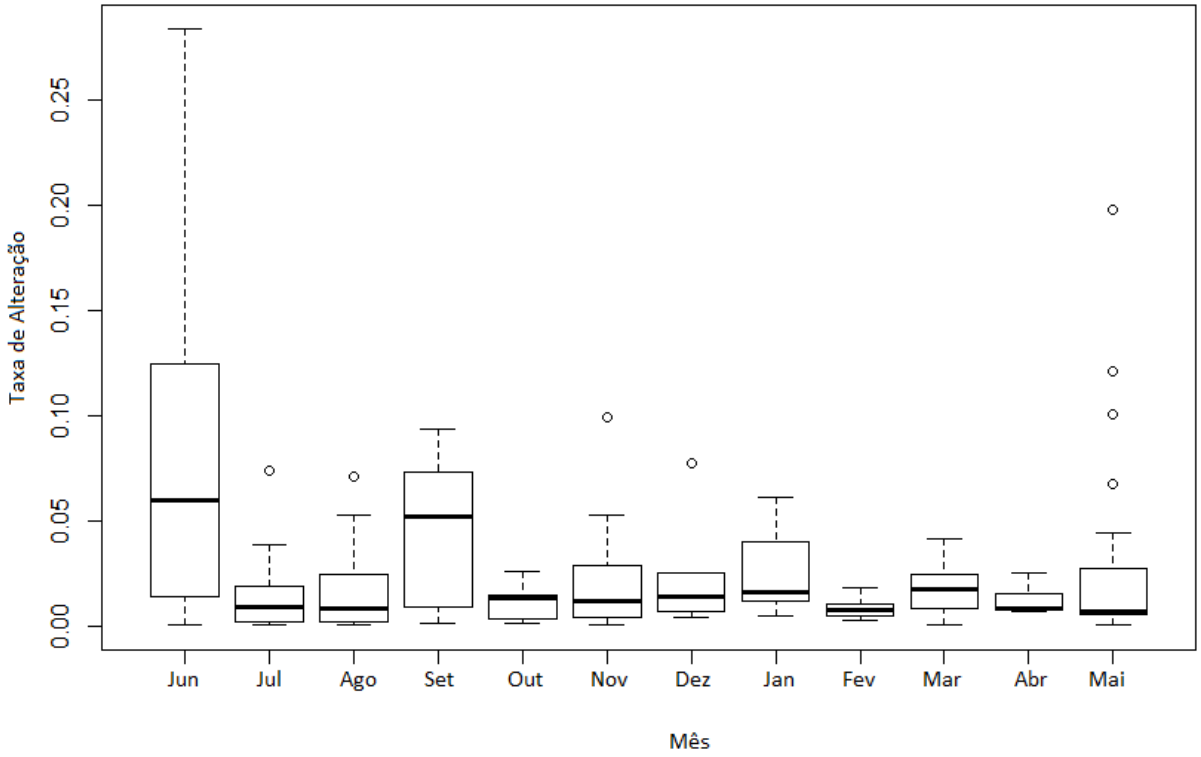
Nesse caso, é possível verificar que, embora as classes tenham funções específicas de customização e que as classes de customização tenham obtido estabilidade ao longo dos anos, mudanças no projeto ainda podem impactá-las. Esses impactos podem ser causados pela adição e remoção de classes em diferentes *branches*.

Para a questão de pesquisa **QP3** foram consideradas quatorze medidas de manutenibilidade coletadas a partir da ferramenta CK-tool. Para cada medida foi realizada a correlação com a taxa de clones e depois foi efetuado o teste hipótese para verificar o nível de correlação. A hipótese nula e alternativa são descritas respectivamente a seguir:

H_6: Medidas de manutenibilidade não possuem uma forte correlação com a taxa de clones em classes de customização.

H_7: Medidas de manutenibilidade possuem uma forte correlação com a taxa de clones em classes de customização.

Figura 22: Taxa de alteração em partes clonadas em classes de customização



Fonte: Elaborada pelo autor

Para o teste de hipótese da questão QP3 foi usado o teste de correlação de *Spearman* [1], pois os dados não apresentaram um padrão em sua distribuição. Assim, foram considerados dados não paramétricos. A partir dos resultados dos testes foram obtidos os seguintes p-values do nível de correlação das medidas de manutenibilidade retiradas da Tabela 12 na Seção 4.3.3 com a taxa de clones. Para a medida WMC foi obtido um p-value de 0.02884, a medida RFC obteve um p-value de 0.0112, a medida NOPM teve um p-value igual a 0.01056 e a medida LOC teve um p-value igual a 0.01555.

A partir desses resultados é possível verificar que as medidas WMC, RFC, NOPM e LOC usadas no catálogo possuem uma forte relação com a taxa de clones das classes de customização. Assim, é possível considerar a hipótese alternativa como verdadeira. Uma descrição dos dados coletados dessas medidas é apresentado na Seção 5.3.

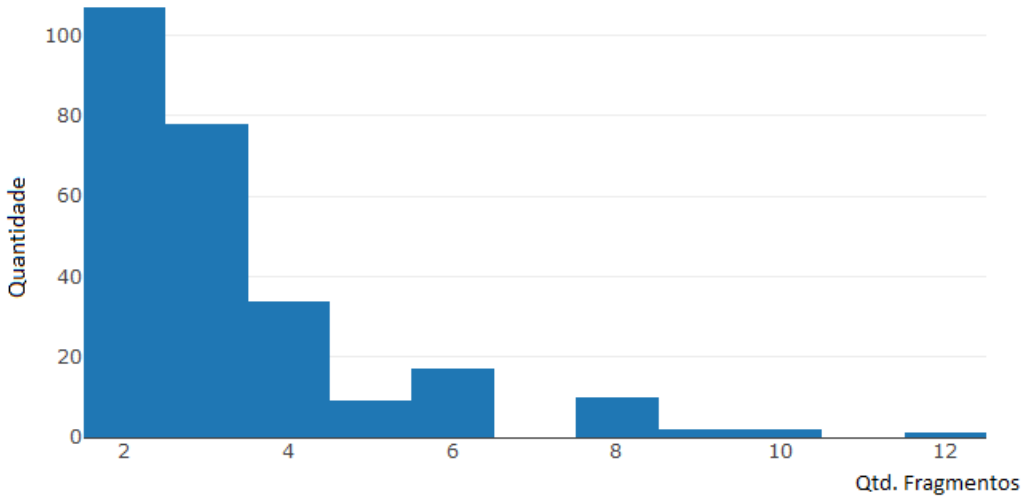
[1] É usado para analisar o grau de relação entre duas variáveis baseado no p-value retornado do teste. Quanto menor o p-value maior é o nível de correlação entre as variáveis. Esse teste é usado para variáveis discretas e não há necessidade de haver uma relação linear entre as variáveis, as variáveis não precisam estar em escalas de intervalo ou taxa e não requer o conhecimento da distribuição de valores das variáveis (BOLBOACA; JÄNTSCHI, 2006)

5.3 Catálogo de clones

Para a construção do catálogo foram analisados todos os tipos de clones dentro do caso de estudo dentro do mês de Junho de 2016, pois foi o mês com a maior taxa de clones retornados entre todos os analisados. Foi feita uma análise manual de todos os clones retornados e removidos todos os falsos positivos. Um clone foi classificado como falso positivo nos casos em que o fragmento identificado não represente uma funcionalidade do sistema. Isso ocorre nos casos de um conjunto de instanciações de variáveis ou, por exemplo, na manipulação dos valores de variáveis a partir de *getters* e *setters*. Ao final, foram removidos 15 clones em um total de 278, onde cada clone é representado por uma linha do catálogo. O catálogo com todos os dados é apresentado no apêndice C.

Os 278 clones coletados eram compostos por 873 fragmentos de código e esses 873 fragmentos de código tinham 1398 relações de igualdade entre eles. Na Figura 23 é possível visualizar a distribuição da quantidade de fragmentos de código em cada clone.

Figura 23: Distribuição dos fragmentos de código em cada clone

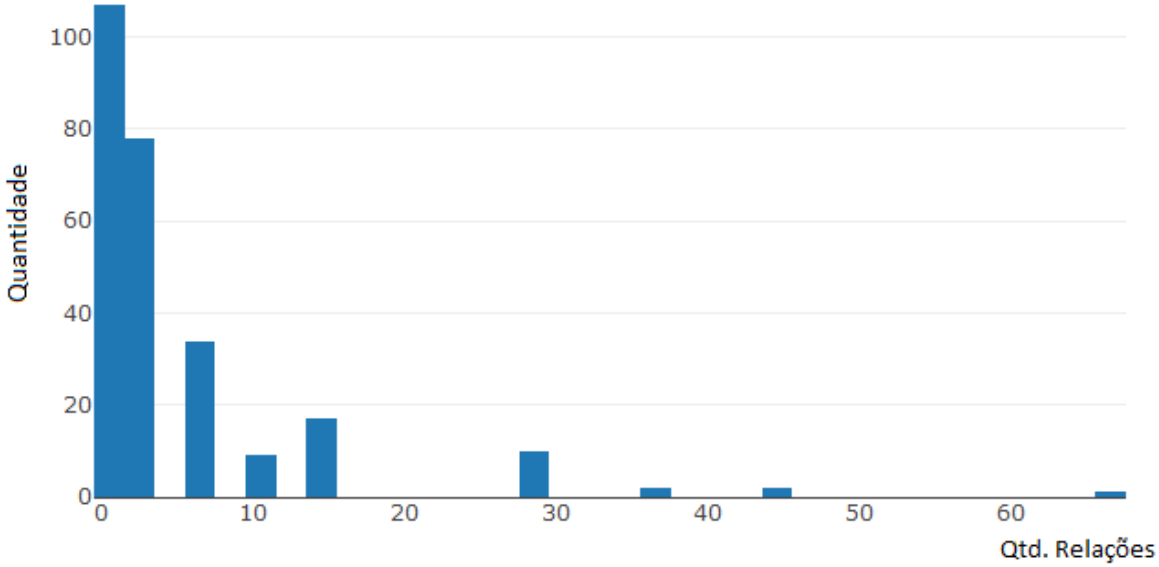


Fonte: Elaborada pelo autor

A partir da Figura 23 é possível perceber que a maioria dos clones tem apenas dois fragmentos de código e que apenas um clone tem doze fragmentos de códigos. Alguns dos clones com maior quantidade de fragmentos envolvidos, como por exemplo, 8, 9, 10 ou 12, podem ser considerados como pontos candidatos de refatoração, já que envolvem diferentes partes do projeto. Esses fragmentos podem gerar um alto valor de dependência devido a grande quantidade

de relações entre eles. A Figura 24 apresenta uma perspectiva da quantidade de relações entre os fragmentos.

Figura 24: Quantidade de relações entre fragmentos de código



Fonte: Elaborada pelo autor

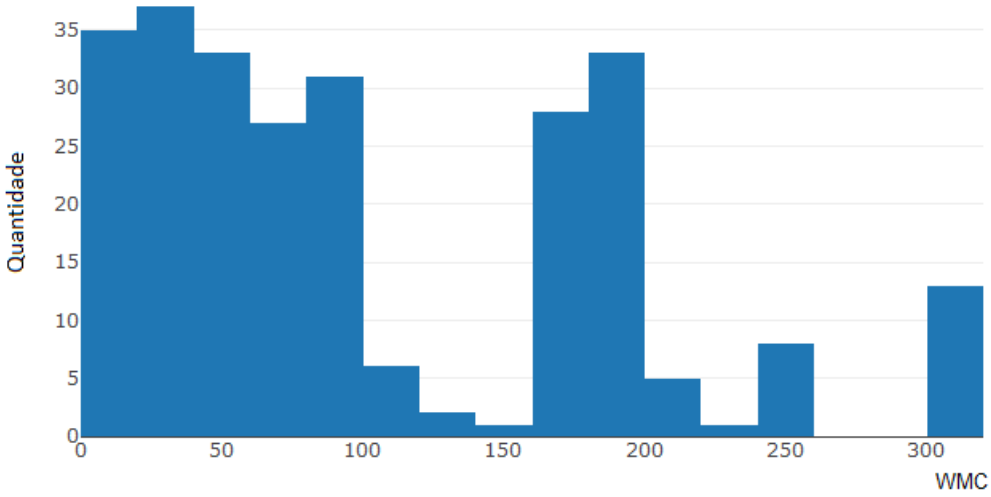
Tanto a Figura 23 quanto a Figura 24 apresentam distribuições semelhantes, pois são resultados de medidas correlacionadas. A Figura 24 mostra que a quantidade de 12 fragmentos gera mais de 60 relações entre os fragmentos, assim como o de 10 possui mais de 40 relações. O clone com 9 fragmentos possui mais de 30 e o de 8 fragmentos possui quase 30 relações.

O trabalho de Benlarbi *et al.* (2000) define vários *thresholds* para as medidas abordadas nesta dissertação. Os autores definem *thresholds* igual a 100 para WMC, 100 para RFC e 500 para LoC. Contudo, esses *thresholds* podem não representar o contexto do estudo de caso abordado neste trabalho. Para isso, foi verificado para cada medida o valor $\mu + \sigma$, que é a média somada ao desvio padrão para determinar o limite superior com intuito de identificar possíveis pontos de melhoria.

A primeira medida apresentada que teve uma alta correlação com a taxa de clones foi o WMC, que representa a complexidade de uma classe de acordo com a quantidade de caminhos lógicos que ela pode executar. A Figura 25 mostra a distribuição da complexidade para as classes de customização com clones.

Em geral a média da medida WMC teve um valor aproximado igual a 105 e seu

Figura 25: Valores da medida WMC das classes com clones

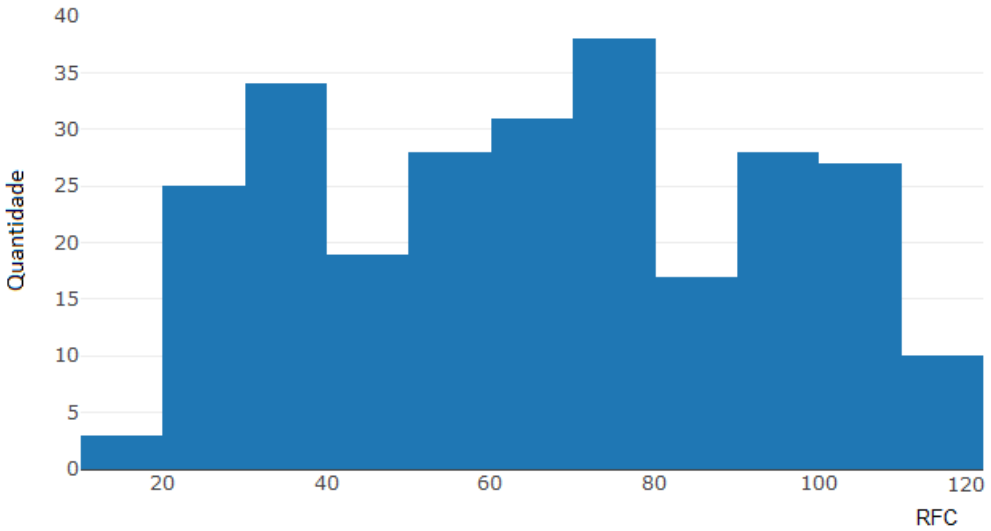


Fonte: Elaborada pelo autor

desvio padrão igual a 83. Com isso, um valor que pode ser tomado como base do limite superior é igual a 188. Logo, as classes com valor superior a esse limite, poderiam ser consideradas como pontos de refatoração em potencial para diminuição da complexidade.

Uma segunda medida coletada foi a RFC, que representa a quantidade de métodos invocados por uma classe. Essa medida está relacionada com o acoplamento, podendo ajudar na identificação do nível de modificabilidade de uma classe. A Figura 26 apresenta os níveis de RFC das classes contendo clones.

Figura 26: RFC das classes com clones

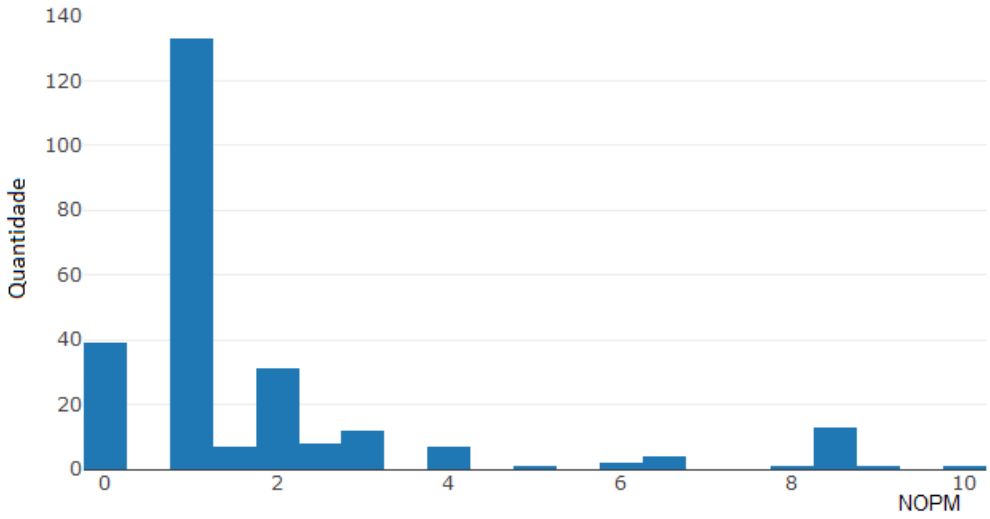


Fonte: Elaborada pelo autor

Para o RFC foi obtido uma média aproximada de 67 e um desvio padrão de 28, portanto o limite superior é igual a 95. A partir da Figura 26 é possível identificar uma quantidade considerável de classes que estão acima do limite. Portanto, podem ser consideradas como alvos de refatoração para melhoria da modificabilidade.

A terceira medida coletada foi a NOPM, que representa a quantidade de métodos públicos. Assim como a medida LoC, a NOPM está relacionada a medidas de tamanho. Esse tipo de medida pode ser importante para questões de legibilidade do código. A Figura 27 ilustra a medida NOPM das classes de customização com clones.

Figura 27: Valores da medida NOPM das classes com clones



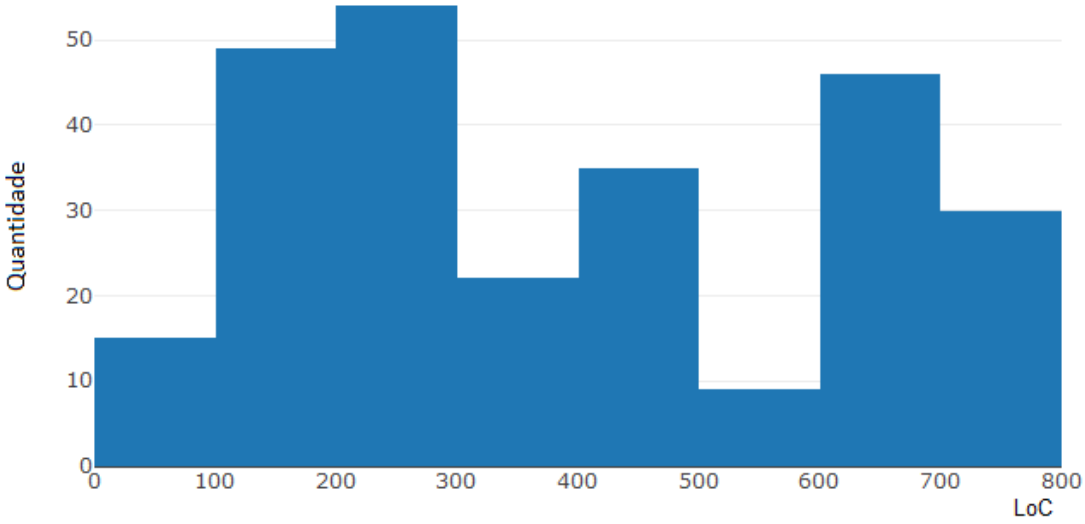
Fonte: Elaborada pelo autor

A partir da Figura 27 é possível visualizar que a maioria das classes possuem apenas um método público, enquanto o restante contém de 0 até 10 métodos públicos. Para essa medida NOPM foi obtido uma média de 2 métodos públicos para cada classe e o desvio padrão igual a 2, resultando em um limite superior de 4 métodos públicos.

A quarta e última medida de manutenibilidade relacionada a alta taxa de correlação com a taxa de clones é a LoC. É natural que a medida LoC tenha alta relação com os clones uma vez que a remoção e adição de clones estão diretamente relacionados com LoC. No entanto, não necessariamente a remoção de um clone indicará a diminuição do LoC, pois existe a possibilidade de adição de dependências ou novas funções durante as melhorias. A Figura 28 apresenta os diferentes tamanhos das classes de customização com clones.

Em média as classes de customização possuem um tamanho aproximado igual a 400

Figura 28: Valores da medida LoC das classes com clones



Fonte: Elaborada pelo autor

LoC e um desvio padrão de 231. Dessa forma, classes com LoC maiores que 631 podem ser consideradas como grandes.

A partir dos diversos gráficos é possível verificar que os dados não seguem uma distribuição normal, o que reforçou o uso da correlação de *Spearman* por ser um coeficiente de correlação não paramétrico.

Para fazer o relacionamento com os clones, foi realizada uma classificação da medida que a análise manual era executada. Os clones poderiam conter até quatro tipos de relações: DI, ID, DD e II. A Tabela 13 apresenta o resumo do catálogo com as medianas dos dados coletados.

Tabela 13: Resumo do catálogo contendo a mediana dos dados coletados

Tipo de Relação	**Tipo Clone I**	**Tipo Clone II**	**Tam. Clone**	**Nº Fragmentos Cod.**	**Relações entre Clones**	**WMC**	**RFC**	**NOPM**	**LoC**	**LoC Adicionadas**	**LoC Removidas**	**Tempo Issue**
DI	1	2	37	2	1	64	64	1	296	0	2	420
ID, DI, DD	4	7	14	5	10	62	52	1	278	0	5	420
II, DI	6	9	12	6	15	187	89	2	694	0	5	0
DI, DD	2	2	28	3	3	179	97	0	688	0	1	480
ID	1	1	14	2	1	48	54	1	208	2	2	120
ID, DD	0	6	10	4	6	87	71	1	356	0	0	0
ID, DI	1	2	33	3	3	22	43	1	134	0	0	0
II	0	1	11	2	1	41	85	0	347	0	0	0
II, ID, DI, DD	6	16	11	7	22	125	88	1	477	0	0	0
II, ID, DI	2	4	24	4	6	253	69	4	626	0	0	0
II, DI, DD	12	16	13	8	28	180	97	0	693	0	0	0
DD	1	1	172	2	1	176	95	0	674	8	13	480

Fonte: Elaborada pelo autor

A partir desse resumos dos dados na Tabela 13, é possível ter uma visão geral da distribuição dos clones. Com essa tabela é possível construir um *Ranking* de acordo com o ponto que se deseja refatorar. Por exemplo, um projeto deseja diminuir a complexidade do código, para isso, pode-se verificar que os clones com as relações DI e II na linha 3 da Tabela, apresentam o maior valor da medida WMC. Além de estarem próximas do limite superior, que representa uma maior complexidade do código. Outro fator interessante é a quantidade de relações entre clones do tipo II, DI e DD na penúltima linha da Tabela. Essas relações podem indicar um alto grau de acoplamento entre classes, o que pode indicar um ponto de refatoração em casos que a equipe quer diminuir o acoplamento do código para melhorar a modificabilidade.

Como usar - O apêndice C apresenta o catálogo de clones completo, baseado nos dados coletados do projeto em estudo. Cada linha é um clone que pode estar vinculado com uma ou mais classes de customização e consequentemente, pode estar distribuído entre os diferentes *branches* da LPSG, na qual essa distribuição é representada pela primeira coluna da tabela do apêndice C. O desenvolvedor ao visualizar esse catálogo irá perceber um conjunto de medidas vinculadas a cada clone do projeto e assim, poderá rastrear onde cada clone se encontra. Durante a fase de refatoração ou melhoria da qualidade do código, o desenvolvedor pode verificar o catalogo e identificar as medidas que estão fora dos limites dos *thresholds* e rastrear diretamente qual a parte do código está afetando a manutenibilidade do projeto. Outra forma de utilizar o catálogo é como forma de controle da evolução do projeto, onde é possível analisar, por exemplo, a cada 6 meses a quantidade de clones inseridos e removidos no projeto e como o isso impactou na manutenibilidade do projeto.

O catálogo tem o intuito de proporcionar aos desenvolvedores um conjunto de indícios para auxiliar na refatoração de código. A partir das colunas Tipo I, Tipo II e Número de Fragmentos de Código é possível visualizar a quantidade de relações de igualdade entre clones. Essa medida pode ajudar a rastrear a quantidade de mudanças que deverá ser realizada ao alterar uma parte do código, além de poder ser usada como um fator de esforço para refatorar determinados tipos de clones. Já a coluna Tamanho do Clone proporciona uma visão da granularidade dos clones. Essa medida é importante para identificar os clones menores que são mais difíceis de rastrear e tendem a possuir uma maior quantidade de relações. Esses pequenos clones podem ocorrer devido ao reuso de partes de funções ou lógicas que seguem um mesmo fluxo. O catálogo ainda conta com um conjunto de medidas de qualidade que podem ajudar a rastrear pontos de melhoria da manutenibilidade do projeto. Por fim, o catálogo faz uma relação com os tipos de

clones e o esforço relacionado ao tempo gasto com as *issues* e a quantidade de alteração sofrida pelos fragmentos de código clonado. O diferencial desse catálogo é a capacidade de rastrear os códigos clonados e vincular com medidas que possam ajudar os desenvolvedores a identificar pontos de melhoria do código.

Com o catálogo, é possível identificar que clones em classes de customização podem afetar o projeto em diferentes aspectos. Um ponto importante que vale ressaltar dos resultados é que os clones apresentaram um maior tempo para a solução das *issues* em quase todos os tipos analisados. Esse maior custo de tempo pode ter sido influenciado pela alta taxa de replicação do código em diferentes *branches*. Portanto, é importante que clones em classes de customização sejam gerenciados, pois a partir do conhecimento da natureza dos clones, é possível identificar pontos de refatoração que podem impactar diretamente na manutenibilidade do projeto e assim influenciar no custo de tempo e consequentemente no Retorno de Investimento.

5.4 Discussão dos resultados

O principal objetivo desta dissertação foi analisar os impactos dos clones em um projeto de LPSG, mais especificamente na fase de derivação do produto e a partir dessas análises propor um catálogo com os tipos de clones e suas relações com os respectivos impactos vinculados. Os impactos analisados estão relacionados ao tempo de resolução de *issues* de desenvolvimento e manutenção, a taxa de mudanças em fragmentos clonados e a manutenibilidade do projeto. Como esse estudo é baseado em métodos estatísticos é necessário um grande volume de dados para que possam ser analisados e assim gerar resultados mais próximos da generalização. Embora esse trabalho possa ser aplicado em projetos de larga escala, o contexto da LPSG é importante para que possa trazer esse grande volume de código replicado, uma vez que para cada novo produto derivado da LPSG é gerado uma grande quantidade de código clonado.

Com os resultados obtidos, é possível verificar que o tempo médio para *issues* com clones é maior que *issues* sem clones. Esse tempo médio é maior em 9 dos 13 meses avaliados. Também é notável que os tempos das *issues* sem clones têm uma variação maior que as *issues* com clones. As *issues* sem clones variam de 30 a 3660 minutos e as *issues* com clones podem variar de 60 a 2250 minutos.

Outro fator importante é que as *issues* do tipo *Desenvolvimento* que tinham clones vinculados apresentaram um tempo médio de 720 minutos, enquanto que as *issues* sem clones obtiveram um tempo médio de 238,8 minutos. Esse resultado pode indicar que a arquitetura do

projeto propicie um maior esforço no desenvolvimento do sistema devido a replicação em várias partes do sistema para suprir um requisito. Essa análise é reforçada pela *issue* do tipo **Melhoria**, uma vez que uma melhoria deve ser aplicada em diversas partes do sistema.

Um segundo indício encontrado é que as classes de customização não possuem uma grande taxa de alteração. Isso pode ser causado pela natureza das classes de customização, uma vez que cada classe tem um papel específico de customizar determinadas *features*. Essas funções específicas não costumam mudar ao longo do tempo, pois estão fortemente conectadas à *baseline* do projeto, garantindo uma maior estabilidade das mudanças. Outro fator que pode influenciar essa taxa de mudança é a maturidade do projeto que possui registro nos *logs* desde 2013.

Embora as classes de customização possuam essa estabilidade, vale ressaltar que a *issue* do tipo **Bug** teve a maior frequência entre todos os tipos, com um total de 49 das 141 analisadas. Contudo, não há uma grande alteração das partes clonadas de *issues* do tipo **Bug**. Uma hipótese que pode ser levantada é que as partes clonadas por serem replicadas em várias partes, passaram por uma maior carga de testes e assim adquiriram maior estabilidade ao longo do tempo.

Também é possível observar a partir dos resultados, que os *branches* apresentaram um decréscimo da taxa de clones ao longo do ano, houve uma variação de 87% até 72%, resultando em uma variação de 15%. É possível considerar a partir dos resultados que cada *branch* tende a se tornar mais específico para atender um determinado requisito, o que leva a remoção de clones ao longo do tempo. Além disso, é plausível acreditar que o desenvolvimento de uma nova *branch* é baseado em *fork*, onde são geradas cópias de um *branch* mais estável, e ao longo do tempo, são adicionadas e removidas funções, ou classes de customização específicas para cada *branch*. Alguns dos problemas dessa abordagem está relacionada a questões de rastreabilidade dos clones o que pode gerar um alto custo de manutenção ao longo do tempo.

Uma LPS Ad-hoc. Como em muitos projetos de desenvolvimento reais e de grande escala, é difícil conceber nos estágios iniciais se um projeto se tornará, em algum momento, uma Linha de Produtos de Software. O gerenciamento de variabilidades e pontos comuns é deixado de lado para dar espaço a uma perspectiva de sistema único mais básica/ágil. A LPSG abordada nesta dissertação começou de maneira *ad-hoc*, com adaptações de sistemas operacionais feitas manualmente para cada novo modelo de celular que foi lançado no início dos anos 2000. Desde então, o projeto evoluiu para incorporar seleção de recursos (ou seja, configuração) em uma interface baseada na Web e customização automática com um conjunto de classes Java. Dado

esse contexto, é natural iniciar esses mecanismos de automação com linguagens de propósito geral conhecidas, como Java e C. O uso de uma estrutura de LPS estava longe de ser uma alternativa, principalmente por causa da maturidade dos *frameworks* disponíveis na época.

A expressividade tem um preço. O uso de classes Java para implementar um mecanismo de customização *ad-hoc* é, como mencionado, uma alternativa direta para permitir a derivação automática do produto. O projeto de customização do estudo de caso pode ser claramente comparado a mecanismos de derivação que aparecem como alternativas mais estruturadas. Essas alternativas, como linhas de produtos *delta-oriented* (SCHAEFER *et al.*, 2010; HABER *et al.*, 2013) e linhas de produtos baseadas em modelo (HAUGEN *et al.*, 2012; FILHO *et al.*, 2015), fornecem operações que podem ser reutilizadas, compostas e verificadas muito mais facilmente do que operadores *ad-hoc* implementados diretamente em classes Java. Um contraponto de usar mecanismos específicos de LPS em vez de linguagens de propósito geral em um mecanismo de derivação é que esses mecanismos restringem a expressividade do que pode ser feito. Por outro lado, facilitam a verificabilidade e a computabilidade em geral. O Java também é uma linguagem amplamente adotada, o que facilita na curva de aprendizado e, portanto, a agilidade do projeto.

Refatoração de classes de customização. Como mencionado anteriormente, ao definir ou usar um mecanismo de derivação/customização mais estruturado, é possível minimizar não apenas as práticas de clonagem, mas também vários *bad smells* dentro dessas classes Java. Uma refatoração mais radical poderia identificar e extrair blocos de código Java e transformá-los em comandos de derivação mais abstratos, como os da camada de realização da linguagem CVL (HAUGEN *et al.*, 2012). Essa transformação também exigiria que os artefatos de destino fossem mapeados em modelos com seus respectivos meta-modelos. Uma maneira menos radical seria rastrear os clones e aplicar pequenas refatorações ao longo do tempo eliminando clones com maior impacto, tendo como fator direcional as subcaracterísticas de qualidade que se deseja melhorar e o catálogo de clones gerado por esta dissertação, como por exemplo, a modificabilidade poderia ser afetada pela redução de clones com maior número de relações já que um fragmento pode estar acoplado com muitos outros.

6 CONCLUSÃO

Este capítulo apresenta as considerações finais relativas a esta dissertação. A Seção 6.1 apresenta uma visão geral dos resultados alcançados a partir da execução dessa dissertação. A Seção 6.2 identifica possíveis ameaças aos resultados e conclusões do trabalho e como essas ameaças foram mitigadas. Finalmente, na Seção 6.3 são apresentadas as possíveis direções futuras de evolução que esta pesquisa pode seguir.

6.1 Resultados alcançados

Este trabalho teve como objetivo analisar os impactos que a prática de clonagem pode causar em classes de customização a partir de um estudo de caso em um projeto de uma LPSG real, definir um processo contendo todos os passos e artefatos gerados em cada passo para a identificação dos impactos dos clones e propor um catálogo com os tipos de clones e seus respectivos impactos para auxiliar na identificação e priorização de pontos de refatoração.

Ao quantificar e identificar os clones e o tempo gasto com eles, os desenvolvedores e gerentes de projeto podem identificar as melhores abordagens de refatoração de código para eliminar esses clones. Com base nos impactos quantificados, é possível priorizar clones a serem refatorados.

Como resultado direto deste trabalho de dissertação, foi identificado que *issues* relacionadas à clonagem levam em média 136% mais tempo a serem resolvidas e a taxa de mudança de clone é de 0,13% em média para os clones entre as classes. Então, para este trabalho, pode-se concluir que os fragmentos clonados entre as classes de customização não mudam com frequência, mas o esforço gasto com a resolução de *issues* com clones leva mais que o dobro do tempo em relação a *issues* sem clones. Isso pode estar relacionado a natureza do projeto que reusa uma grande quantidade de classes de customização em diferentes *branches*. Essas classes estão presentes no projeto por um longo período de tempo, o que pode garantir uma maior estabilidade do código em relação as mudanças, embora cause uma grande quantidade de replicação de código.

A partir desse trabalho de dissertação também foi possível realizar o desenvolvimento de um mecanismo para cruzamento de clones com issues, *scripts* e um agrupamento de informações relacionadas ao projeto, que possibilitou a construção e análises de um extenso

dataset[1] relacionado a diferentes aspectos da LPSG, utilizado no estudo de caso, no qual pode ser usado pela comunidade para replicar ou estudar novas informações sobre clonagem dentro do contexto de LPS.

Em resumo, entre os principais resultados alcançados, é possível listar:

- **Ferramenta para mapeamento de clones com *commits*** - O primeiro resultado gerado por esta dissertação foi uma ferramenta desenvolvida em JAVA para relacionar clones com *commits*, assim, é possível identificar quais alterações estavam relacionadas a clones e consequentemente, quais *issues* continham partes clonadas que foram alteradas.
- **Processo para análise dos impactos de clones** - Um outro resultado que pode ser usado pela comunidade é o processo usado neste trabalho como forma de análise dos impactos de clones. O processo foi descrito de forma que possa ser replicado em outros contexto e assim, ajudar na identificação de pontos de refatoração.
- **Catálogo com os impactos dos clones** Ao final, foi gerado um catálogo que realizou o mapeamento e relacionamento de todas as informações coletadas desse estudo, como forma de auxiliar desenvolvedores a selecionar partes do código que precisam ser refatoradas.

Também é possível listar um conjunto de resultados secundários obtidos através da execução do processo apresentado no capítulo 4.

- **Mapeamento das classes de customização em cada *branch*** - Este trabalho construiu uma matriz de rastreabilidade que identificou quais classes de customização estavam presentes em cada *branch* e como essas classes foram adicionadas e removidas ao longo do tempo de cada *branch*. Esse mapeamento ajudou a entender como o projeto estava organizado e como evoluiu ao longo do tempo.
- **Medidas relacionadas aos clones** - Um segundo resultado deste trabalho, foi a coleta das taxa de clones entre *branches* e entre classes de customização. Com essa medida, foi possível identificar uma relação com a matriz de rastreabilidade apresentada no item anterior e a porcentagem de clones entre dois *branches*.
- ***Script* para mapeamento e coleta de dados** - Ao total, foi gerado um *script* para a coleta de *commits* baseado nos IDs de cada tarefa registrada no JIRA, um *script* para a coleta de medidas de manutenibilidade em cada *branch* ao longo de 13 meses usando a ferramenta CK Metrics e um *script* para coletar a quantidade de linhas de código alteradas, baseado nos *commits* realizados ao longo do tempo.

[1] https://github.com/GREatResearches/Software-Product-Line/tree/master/Dataset

- **Cruzamento de dados entre medidas de manutenibilidade e clones** - O trabalho também foi composto por um passo que coletou e identificou o nível de relação entre clones e medidas de manutenibilidade. Como resultado, foi gerada uma matriz com o grau de relação de 14 medidas de manutenibilidade com a taxa de clones ao longo de 13 meses.
- **Classificação de clones baseado nos tipos de relações entre fragmentos de código** - Como resultado deste trabalho, foi gerado um tabela contendo os tipos de relações que um fragmento de código clonado pode ter e a quantidade de relações de cadas tipo. Os dados gerados por essa classificação, podem ser usados em outros estudos como fonte de análise do acoplamento entre fragmentos de código.

Todos os resultados estão disponíveis no repositório do GREat[2] e podem ser usados como forma de replicação da metodologia usada neste trabalho.

6.2 Ameaças à validade

As ameaças à validade foram identificadas nesta dissertação baseadas no trabalho de Wohlin *et al.* (2012). A descrição de cada ameaça e sua mitigação são apresentados a seguir:

Ameaças à validade externa - A primeira ameaça à validade está relacionada ao fato de ser analisado apenas um projeto de uma LPSG real. Devido a isso, este trabalho não procura fornecer generalização para cada projeto de LPS. Em vez disso, é fornecido um caso extenso, com mais de 120 mil linhas de código, e real que mostra evidências sobre práticas de clonagem dentro de um mecanismo de customização usado globalmente. Outra ameaça está relacionada ao uso de apenas uma ferramenta para a detecção de clones que usa algoritmo baseado em *tokens*. Para mitigar essa ameaça, o algoritmo escolhido para a detecção de clones foi classificado com o algoritmo com maior retorno de clones detectados, embora possa trazer vários falsos positivos como resultado dessa coleta.

Ameaças à validade interna - Outra ameaça está relacionada ao intervalo de tempo para as coleta dos dados. Em cada mês os desenvolvedores podem ter níveis de produtividade distintas que podem causar variação dos dados relacionados aos *commits* e, consequentemente, impactar a taxa de variação de clones. Para evitar dados discrepantes entre os meses coletados, foram feitas análises a partir de gráficos *box-plot* que possibilitam visualizar *outliers* e desconsiderar esses dados discrepantes das análises.

Ameaças à validade da construção - O desenvolvimento dessa dissertação foi

[2] https://github.com/GREatResearches/Software-Product-Line/tree/master/Dataset

motivado após desenvolvedores informarem que os clones estavam prejudicando a produtividade do desenvolvimento e manutenção do código. A partir dessa demanda real em um projeto de software, a pesquisa foi voltada para identificar apenas os aspectos negativos que a prática de clonagem pode causar. Contudo, é possível que os clones possam trazer benefícios para o desenvolvimento com a entrega mais rápida de demandas e diminuir *bugs* em casos que o fragmento de código clonado já foi massivamente testado. Sabendo disso, o estudo definiu *thresholds* para tentar identificar apenas clones que possam estar causando maior impacto na manutenibilidade do código.

Ameaças à validade da conclusão - Quanto à validade da conclusão, devido o processo usado para detecção de clones baseada em *tokens*, existe uma grande quantidade de clones retornados, porém existe a possibilidade de retornar vários falsos positivos. Para minimizar essa ameaça, os clones antes de serem submetidos para a análise da ferramenta, foram analisados manualmente referente ao mês de Junho de 2016, pois era o mês que retornou a maior quantidade de clones. Ao final, o conjunto de dados analisados era composto por 877 fragmentos de código, resultando em quase 1.400 relações entre fragmentos de código clonados. Também foram realizadas várias adaptações e análises do tamanho mínimo do clone retornado pela ferramenta ConQat, onde foi verificado se o clone detectado é realmente um clone. Com essa verificação, foi identificado que alguns clones eram apenas um grande conjunto de instanciações de atributos ou um agrupamento de funções para gerenciar esses atributos (e.g. *set* e *get*). Esses tipos de clones foram classificados como falsos positivos, porque eles não faziam parte da funcionalidade do sistema, eles eram apenas formas de manipular os atributos das classes.

6.3 Trabalhos futuros

Uma direção para o trabalho futuro é o uso dos fatores de impacto dos clones em ferramentas de refatoração e manutenção de clones. Com uma visão futura, os passos do processo utilizados neste trabalho podem ser acoplados em uma única ferramenta para automatizar a análise dos impactos. Com a automatização da análise, seria possível reduzir o esforço dos desenvolvedores na transformação de uma LPS *ad-hoc* para um LPS gerenciada, a partir da identificação de clones e transformação dessas partes clonadas em variantes que podem ser reutilizadas de forma gerenciada.

Uma outra direção deste trabalho seria a adição de uma análise qualitativa dos dados junto com os desenvolvedores, usando questionários e entrevistas. Com isso, poderiam ser

identificados pontos que os desenvolvedores consideram mais importantes e assim verificar como a análise quantitativa pode estar relacionada aos dados qualitativos. Para essa análise qualitativa, também seria possível classificar as perspectivas dos diferentes desenvolvedores de acordo com o nível de experiência e maturidade dentro do projeto.

REFERÊNCIAS

ACHER, M.; COLLET, P.; LAHIRE, P.; FRANCE, R. B. Familiar: A domain-specific language for large scale management of feature models. **Science of Computer Programming**, [S.l.], Elsevier, v. 78, n. 6, p. 657–681, 2013.

ALALFI, M. H.; CORDY, J. R.; DEAN, T. R. Analysis and clustering of model clones: An automotive industrial experience. In: IEEE. **Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on**. [S.l.], 2014. p. 375–378.

ALLIER, S.; BARAIS, O.; ACHER, M.; BAUDRY, B. *et al.* Assessing product line derivation operators applied to java source code: an empirical study. In: ACM. **Proceedings of the 19th International Conference on Software Product Line**. [S.l.], 2015. p. 36–45.

BASIT, H. A.; JARZABEK, S. A data mining approach for detecting higher-level clones in software. **IEEE Transactions on Software engineering**, [S.l.], IEEE, v. 35, n. 4, p. 497–514, 2009.

BASIT, H. A.; KHAN, H. S.; HAMID, F.; SUHAIL, I. Tool support for managing method clones. In: IEEE. **Software Clones (IWSC), 2015 IEEE 9th International Workshop on**. [S.l.], 2015. p. 40–46.

BATORY, D. Feature models, grammars, and propositional formulas. In: SPRINGER. **International Conference on Software Product Lines**. [S.l.], 2005. p. 7–20.

BENAVIDES, D.; SEGURA, S.; RUIZ-CORTÉS, A. Automated analysis of feature models 20 years later: A literature review. **Information Systems**, [S.l.], Elsevier, v. 35, n. 6, p. 615–636, 2010.

BENAVIDES, D.; SEGURA, S.; TRINIDAD, P.; CORTÉS, A. R. Fama: Tooling a framework for the automated analysis of feature models. **VaMoS**. [S.l.], v. 2007, p. 01, 2007.

BENLARBI, S.; EMAM, K. E.; GOEL, N.; RAI, S. Thresholds for object-oriented measures. In: IEEE. **Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on**. [S.l.], 2000. p. 24–38.

BERGER, T.; RUBLACK, R.; NAIR, D.; ATLEE, J. M.; BECKER, M.; CZARNECKI, K.; WĄSOWSKI, A. A survey of variability modeling in industrial practice. In: ACM. **Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems**. [S.l.], 2013. p. 7.

BEZERRA, C. I.; BARBOSA, J.; FREIRES, J. H.; ANDRADE, R.; MONTEIRO, J. M. Dymmer: a measurement-based tool to support quality evaluation of dspl feature models. In: ACM. **Proceedings of the 20th International Systems and Software Product Line Conference**. [S.l.], 2016. p. 314–317.

BINANTO, I.; WARNARS, H. L. H. S.; GAOL, F. L.; ABDURACHMAN, E.; SOEWITO, B. Measuring the quality of various version an object-oriented software utilizing ck metrics. In: **Information and Communications Technology (ICOIACT), 2018 International Conference on**. [S.l.: s.n.], 2018. p. 41–44.

BOLBOACA, S.-D.; JÄNTSCHI, L. Pearson versus spearman, kendall's tau correlation analysis on structure-activity relationships of biologic active compounds. **Leonardo Journal of Sciences**. [S.l.], v. 5, n. 9, p. 179–200, 2006.

BURD, E.; BAILEY, J. Evaluating clone detection tools for use during preventative maintenance. In: IEEE. **Source Code Analysis and Manipulation, 2002. Proceedings. Second IEEE International Workshop on**. [S.l.], 2002. p. 36–43.

CAPILLA, R. Variability realization techniques and product derivation. In: **Systems and Software Variability Management**. [S.l.]: Springer, 2013. p. 87–99.

CHATTERJI, D.; CARVER, J. C.; KRAFT, N. A. Code clones and developer behavior: results of two surveys of the clone research community. **Empirical Software Engineering**, [S.l.], Springer, v. 21, n. 4, p. 1476–1508, 2016.

CONQAT, T. Conqat user guide 2013.10. 2013. Dísponivel em: <<https://www.cqse.eu/download/conqat/conqat-book-2013.10.pdf>.> Acesso em: 07 de Maio de 2017.

CORDY, J. R.; ROY, C. K. The nicad clone detector. In: IEEE. **Program Comprehension (ICPC), 2011 IEEE 19th International Conference on**. [S.l.], 2011. p. 219–220.

COSTA, P. A. da S.; MARINHO, F. G.; ANDRADE, R. M. de C.; OLIVEIRA, T. Fixture-a tool for automatic inconsistencies detection in context-aware spl. In: **ICEIS (2)**. [S.l.: s.n.], 2015. p. 114–125.

CRUZ, R. C. da; ELER, M. M. An empirical analysis of the correlation between ck metrics, test coverage and mutation score. In: **ICEIS (2)**. [S.l.: s.n.], 2017. p. 341–350.

CZARNECKI, K.; WASOWSKI, A. Feature diagrams and logics: There and back again. In: IEEE. **Software Product Line Conference, 2007. SPLC 2007. 11th International**. [S.l.], 2007. p. 23–34.

DEELSTRA, S.; SINNEMA, M.; BOSCH, J. Product derivation in software product families: a case study. **Journal of Systems and Software**, [S.l.], Elsevier, v. 74, n. 2, p. 173–194, 2005.

DUBINSKY, Y.; RUBIN, J.; BERGER, T.; DUSZYNSKI, S.; BECKER, M.; CZARNECKI, K. An exploratory study of cloning in industrial software product lines. In: IEEE. **Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on**. [S.l.], 2013. p. 25–34.

ERIKSSON, M.; HAGGLUNDS, A. An introduction to software product line development. In: **Proceedings of Umeå's Seventh Student Conference in Computing Science, UMINF**. [S.l.: s.n.], 2003. v. 3, p. 26–37.

FALKE, R.; FRENZEL, P.; KOSCHKE, R. Empirical evaluation of clone detection using syntax suffix trees. **Empirical Software Engineering**, [S.l.], Springer, v. 13, n. 6, p. 601–643, 2008.

FERNANDES, P.; WERNER, C.; MURTA, L. G. P. Feature modeling for context-aware software product lines. In: **SEKE**. [S.l.: s.n.], 2008. p. 758–763.

FILHO, J. B. F.; ALLIER, S.; BARAIS, O.; ACHER, M.; BAUDRY, B. *et al.* Assessing product line derivation operators applied to java source code: an empirical study. In: ACM. **Proceedings of the 19th International Conference on Software Product Line**. [S.l.], 2015. p. 36–45.

GHOSH, A.; LEE, Y. An empirical study of a hybrid code clone detection approach on java byte code. **GSTF Journal on Computing (JoC)**. [S.l.], v. 5, n. 2, 2018.

GRISS, M. L.; FAVARO, J.; D'ALESSANDRO, M. Integrating feature modeling with the rseb. In: IEEE. **Software Reuse, 1998. Proceedings. Fifth International Conference on**. [S.l.], 1998. p. 76–85.

GUPTA, A.; SURI, B. A survey on code clone, its behavior and applications. In: **Networking Communication and Data Knowledge Engineering**. [S.l.]: Springer, 2018. p. 27–39.

GUSFIELD, D. **Algorithms on strings, trees and sequences: computer science and computational biology**. [S.l.]: Cambridge university press, 1997.

HABER, A.; HÖLLDOBLER, K.; KOLASSA, C.; LOOK, M.; RUMPE, B.; MÜLLER, K.; SCHAEFER, I. Engineering delta modeling languages. In: ACM. **Proceedings of the 17th International Software Product Line Conference**. [S.l.], 2013. p. 22–31.

HAUGEN, Ø.; MØLLER-PEDERSEN, B.; OLDEVIK, J.; OLSEN, G. K.; SVENDSEN, A. Adding standardized variability to domain specific languages. In: IEEE. **Software Product Line Conference, 2008. SPLC'08. 12th International**. [S.l.], 2008. p. 139–148.

HAUGEN, O. *et al.* Common variability language (cvl). **OMG Revised Submission**. [S.l.], 2012.

INOUE, K. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. **Annual report of Osaka University: academic achievement**, [S.l.], Osaka University, v. 2001, p. 22–25, 2002.

ISO/IEC. **ISO-IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models**. [S.l.]: ISO, 2011.

JIANG, L.; MISHERGHI, G.; SU, Z.; GLONDU, S. Deckard: Scalable and accurate tree-based detection of code clones. In: IEEE COMPUTER SOCIETY. **Proceedings of the 29th international conference on Software Engineering**. [S.l.], 2007. p. 96–105.

JUERGENS, E.; DEISSENBOECK, F.; HUMMEL, B. Clonedetective-a workbench for clone detection research. In: IEEE COMPUTER SOCIETY. **Proceedings of the 31st International Conference on Software Engineering**. [S.l.], 2009. p. 603–606.

KIM, M.; SAZAWAL, V.; NOTKIN, D.; MURPHY, G. An empirical study of code clone genealogies. In: ACM. **ACM SIGSOFT Software Engineering Notes**. [S.l.], 2005. v. 30, n. 5, p. 187–196.

KRUEGER, C. W. The biglever software gears unified software product line engineering framework. In: IEEE. **Software Product Line Conference, 2008. SPLC'08. 12th International**. [S.l.], 2008. p. 353–353.

LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. **Journal of systems and software**, [S.l.], Elsevier, v. 23, n. 2, p. 111–122, 1993.

LILLACK, M.; BUCHOLDT, C.; SCHILLING, D. Detection of code clones in software generators. In: ACM. **Proceedings of the 6th International Workshop on Feature-Oriented Software Development**. [S.l.], 2014. p. 37–44.

MARINHO, F. G.; COSTA, A. L.; LIMA, F. F.; NETO, J. B.; FILHO, B. J.; ROCHA, L.; DANTAS, V. L.; ANDRADE, R. M.; TEIXEIRA, E.; WERNER, C. An architecture proposal for nested software product lines in the domain of mobile and context-aware applications. In: IEEE. **Software Components, Architectures and Reuse (SBCARS), 2010 Fourth Brazilian Symposium on**. [S.l.], 2010. p. 51–60.

MAZINANIAN, D.; TSANTALIS, N.; STEIN, R.; VALENTA, Z. Jdeodorant: clone refactoring. In: ACM. **Proceedings of the 38th International Conference on Software Engineering Companion**. [S.l.], 2016. p. 613–616.

MENDONCA, M.; BRANCO, M.; COWAN, D. Splot: software product lines online tools. In: ACM. **Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications**. [S.l.], 2009. p. 761–762.

MEYER, M. H.; LEHNERD, A. P. **The power of product platforms**. [S.l.]: Simon and Schuster, 1997.

PADHY, N.; SINGH, R.; SATAPATHY, S. C. Software reusability metrics estimation: algorithms, models and optimization techniques. **Computers & Electrical Engineering**, [S.l.], Elsevier, v. 69, p. 653–668, 2018.

PERROUIN, G.; KLEIN, J.; GUELFI, N.; JÉZÉQUEL, J.-M. Reconciling automation and flexibility in product derivation. In: IEEE. **Software Product Line Conference, 2008. SPLC'08. 12th International**. [S.l.], 2008. p. 339–348.

POHL, K.; BÖCKLE, G.; LINDEN, F. J. van D. **Software product line engineering: foundations, principles and techniques**. [S.l.]: Springer Science & Business Media, 2005.

RABISER, R.; GRÜNBACHER, P.; DHUNGANA, D. Requirements for product derivation support: Results from a systematic literature review and an expert survey. **Information and Software Technology**, [S.l.], Elsevier, v. 52, n. 3, p. 324–346, 2010.

RAJAPAKSE, D. C.; JARZABEK, S. Using server pages to unify clones in web applications: A trade-off analysis. In: IEEE. **Software Engineering, 2007. ICSE 2007. 29th International Conference on**. [S.l.], 2007. p. 116–126.

ROY, C. K.; CORDY, J. R. A survey on software clone detection research. **Queen's School of Computing TR**. [S.l.], v. 541, n. 115, p. 64–68, 2007.

RUBIN, J.; CZARNECKI, K.; CHECHIK, M. Cloned product variants: from ad-hoc to managed software product lines. **International Journal on Software Tools for Technology Transfer**, [S.l.], Springer, v. 17, n. 5, p. 627–646, 2015.

SÁNCHEZ, P.; LOUGHRAN, N.; FUENTES, L.; GARCIA, A. Engineering languages for specifying product-derivation processes in software product lines. In: SPRINGER. **International Conference on Software Language Engineering**. [S.l.], 2008. p. 188–207.

SCHAEFER, I.; BETTINI, L.; BONO, V.; DAMIANI, F.; TANZARELLA, N. Delta-oriented programming of software product lines. In: SPRINGER. **International Conference on Software Product Lines**. [S.l.], 2010. p. 77–91.

SILVA, F. A. P. da; NETO, P. A. d. M. S.; GARCIA, V. C.; MUNIZ, P. F. Linhas de produtos de software: Uma tendência da indústria. **V Encontro Regional de Informática Ceará-Piauí (ERCEMAPI 2011)**. [S.l.], volume 1, 2011.

SOMMERVILLE, I.; MELNIKOFF, S. S. S.; ARAKAKI, R.; BARBOSA, E. de A. **Engenharia de software**. [S.l.]: São Paulo: Pearson Education do Brasil, 2011. v. 9.

SVENDSEN, A.; HAUGEN, Ø.; MØLLER-PEDERSEN, B. Analyzing variability: capturing semantic ripple effects. In: SPRINGER. **European Conference on Modelling Foundations and Applications**. [S.l.], 2011. p. 253–269.

THÜM, T.; KÄSTNER, C.; BENDUHN, F.; MEINICKE, J.; SAAKE, G.; LEICH, T. Featureide: An extensible framework for feature-oriented software development. **Science of Computer Programming**, [S.l.], Elsevier, v. 79, p. 70–85, 2014.

TRIGAUX, J.-C.; HEYMANS, P. Software product lines: State of the art. **Product Line ENgineering of food TraceabilitY software, FUNDP-Equipe LIEL**. [S.l.], p. 9–39, 2003.

WANG, W.; GODFREY, M. W. A study of cloning in the linux scsi drivers. In: IEEE. **Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on**. [S.l.], 2011. p. 95–104.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

ZHANG, G.; PENG, X.; XING, Z.; ZHAO, W. Cloning practices: Why developers clone and what can be changed. In: IEEE. **Software Maintenance (ICSM), 2012 28th IEEE International Conference on**. [S.l.], 2012. p. 285–294.

APÊNDICE A – AJUSTES NO TAMANHO MÍNIMO DO CLONE COLETADO

Tabela 14: Tamanho mínimo igual a 20

Taxa Clones	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
P1	-	35,4%	35,1%	46,6%	35%	36,3%	46,9%	46,1%	35,1%	35,4%	44,8%
P2	-	-	36,5%	33,5%	95,1%	37,5%	33,2%	33,8%	66,3%	36,5%	33%
P3	-	-	-	46,6%	36,9%	90,4%	48,3%	68,1%	36,6%	86%	50,8%
P4	-	-	-	-	32,8%	48,1%	88,9%	72,7%	32,4%	48%	79,7%
P5	-	-	-	-	-	38%	33,6%	34,2%	66,8%	37%	33,5%
P6	-	-	-	-	-	-	50,5%	72%	37,4%	92,5%	53,4%
P7	-	-	-	-	-	-	-	77,9%	33,8%	50,7%	86,9%
P8	-	-	-	-	-	-	-	-	33,7%	69,7%	69%
P9	-	-	-	-	-	-	-	-	-	38,1%	36,3%
P10	-	-	-	-	-	-	-	-	-	-	54,2%
P11	-	-	-	-	-	-	-	-	-	-	-

Tabela 15: Tamanho mínimo igual a 15

Taxa Clones	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
P1	-	39%	38,2%	52,7%	38,8%	39,7%	52,4%	51,1%	38,7%	38,7%	49,1%
P2	-	-	39,4%	38,9%	95,8%	41%	37,7%	37,4%	69,1%	39,7%	37,7%
P3	-	-	-	52,2%	39,9%	91,7%	52%	71%	38,7%	87,9%	54,4%
P4	-	-	-	-	38,5%	54,1%	91,9%	76,2%	39%	53,4%	83,4%
P5	-	-	-	-	-	41,5%	38%	37,8%	69,7%	40,2%	38,2%
P6	-	-	-	-	-	-	54,6%	74,7%	40,1%	93,7%	57,2%
P7	-	-	-	-	-	-	-	80,3%	39%	54%	88,8%
P8	-	-	-	-	-	-	-	-	37,7%	72,8%	72,1%
P9	-	-	-	-	-	-	-	-	-	40,6%	41,4%
P10	-	-	-	-	-	-	-	-	-	-	57,9%
P11	-	-	-	-	-	-	-	-	-	-	-

Tabela 16: Tamanho mínimo igual a 12

Taxa Clones	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
P1	-	42,8%	43,4%	56,5%	42,4%	44,4%	56,4%	54,3%	42,5%	43,5%	53,1%
P2	-	-	42%	41,5%	96%	43,4%	40,4%	39,8%	71,5%	42,4%	40,6%
P3	-	-	-	56%	42,4%	92,5%	56,4%	73,4%	41,5%	88,9%	59%
P4	-	-	-	-	41,1%	57,6%	92%	77,1%	41,4%	57,2%	84,6%
P5	-	-	-	-	-	43,7%	40,6%	40,1%	72%	42,8%	40,8%
P6	-	-	-	-	-	-	58,6%	76,7%	42,6%	93,9%	61,4%
P7	-	-	-	-	-	-	-	81,5%	41,2%	58,2%	90,1%
P8	-	-	-	-	-	-	-	-	40,3%	75%	74,1%
P9	-	-	-	-	-	-	-	-	-	43,5%	43,5%
P10	-	-	-	-	-	-	-	-	-	-	62,3%
P11	-	-	-	-	-	-	-	-	-	-	-

Tabela 17: Tamanho mínimo igual a 11

Taxa Clones	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**P11**
P1	-	44,7%	45,1%	57,3%	44,3%	46,2%	57,2%	55%	44,2%	45,4%	54,3%
P2	-	-	43,7%	43,6%	96,4%	45,1%	42,4%	41,8%	72,5%	44,3%	42,1%
P3	-	-	-	58,4%	44%	93,4%	58,9%	75,3%	42,8%	89,9%	61,1%
P4	-	-	-	-	43,1%	6%,	92,5%	77,8%	43,2%	59,8%	85,7%
P5	-	-	-	-	-	45,4%	42,8%	42,1%	73,1%	44,6%	42,5%
P6	-	-	-	-	-	-	61,1%	78,3%	44%	94,5%	63,5%
P7	-	-	-	-	-	-	-	82,3%	43,1%	60,9%	91,3%
P8	-	-	-	-	-	-	-	-	42,2%	76,4%	75,4%
P9	-	-	-	-	-	-	-	-	-	45,1%	44,8%
P10	-	-	-	-	-	-	-	-	-	-	64,4%
P11	-	-	-	-	-	-	-	-	-	-	-

Tabela 18: Tamanho mínimo igual a 10

Taxa Clones	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**P11**
P1	-	46,2%	46%	58,1%	45,8%	47,2%	58,1%	56,1%	45,7%	46,4%	55,3%
P2	-	-	44,7%	44,6%	97,2%	46,4%	43,4%	42,9%	74,4%	45,6%	43,2%
P3	-	-	-	59%	45,1%	93,6%	59,3%	75,6%	44%	90,3%	61,5%
P4	-	-	-	-	44,1%	60,8%	92,6%	78,5%	44,1%	60,8%	86,1%
P5	-	-	-	-	-	46,8%	43,8%	43,2%	75%	45,9%	43,6%
P6	-	-	-	-	-	-	61,8%	78,7%	45,5%	94,9%	64,2%
P7	-	-	-	-	-	-	-	82,8%	44,1%	61,8%	91,6%
P8	-	-	-	-	-	-	-	-	43,5%	76,9%	76,4%
P9	-	-	-	-	-	-	-	-	-	46,4%	45,9%
P10	-	-	-	-	-	-	-	-	-	-	65,2%
P11	-	-	-	-	-	-	-	-	-	-	-

APÊNDICE B – MEDIANA DAS MEDIDAS DE MANUTENIBILIDADE COLETADASDURANTE 13 MESES

	Clones%	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc
13/06/2016	0,872	6,5	26,8	2	0	22,8	12,6	7,1	2	0	2,8	0	0,6	1,1	130,4
13/07/2016	0,849	6,9	30,1	2	0	26,5	13,4	6,3	1,9	0,2	3,1	0,4	0,6	1	135,7
13/08/2016	0,847	6,9	30,1	2	0	26,7	13,4	6,4	1,9	0,2	3,1	0,4	0,7	1	136,3
13/09/2016	0,842	6,9	30,1	2	0	26,5	13,4	6,4	1,9	0,2	3,1	0,4	0,7	1	137,1
13/10/2016	0,837	7,4	30,7	2	0	27,4	14	6,8	1,8	0,2	3,1	0,4	0,7	1	136,4
13/11/2016	0,837	7	28,3	2	0	27,5	13,8	7,3	1,9	0,2	3,1	0,2	1	1,4	133,6
13/12/2016	0,829	7	30,4	2	0	28,3	15,9	7,6	1,9	0,2	3,1	0,2	1	1,4	136,7
13/01/2017	0,82	7	30,6	2	0	28,3	17,3	7,7	1,8	0,2	3,1	0,2	1	1,4	137,8
13/02/2017	0,74	7	31,1	2	0	28,3	17,3	7,7	1,8	0,2	3,1	0,2	1	1,4	137,8
13/03/2017	0,739	7	31,3	2	0	28,4	17,3	7,6	1,8	0,2	3,1	0,2	1	1,4	137,7
13/04/2017	0,732	7,1	31,8	2	0	28,8	16,3	7,5	1,5	0,2	3,1	0,2	1	1,4	138,3
13/05/2017	0,73	7,1	31,8	2	0	29,4	17,3	7,5	1,5	0,2	3,1	0,4	1	1,4	141,3
13/06/2017	0,72	6,9	29,9	2	0	26,5	12,8	6,3	1,9	0,2	3,1	0,4	0,6	1	136,1

APÊNDICE C – CATÁLOGO

Neste apêndice é apresentado todos os dados coletados para construir o catálogo. A tabela é composta pelo **Tipo de Relação** que indica a natureza do acoplamento entre dois fragmentos de código clonados, sendo eles: DI representa diferentes *branches* com classes iguais, ID representa *branches* iguais com classes diferentes, DD representa diferentes *branches* e diferentes classes e II representa *branches* iguais e classes iguais. As colunas **Tipo 1** e **Tipo 2** representam a quantidade de relações de cada tipo dos clones, sendo o Tipo 1 fragmentos exatamente iguais e Tipo 2 fragmentos com variação apenas nos seus *statements*. Em seguida, a colunas **Tam. Clone** representa o tamanho do clone detectado, junto com a **Qtd. Fragmentos** e **Qtd. Relações** que representam respectivamente, as quantidade de fragmentos envolvidos no clone e a quantidade de relações entre esses fragmentos. Logo após são representadas as 14 medidas de manutenibilidade coletada a partir da CK-tool. Por fim, estão presentes as quantidades de linhas adicionadas e removidas de de cada clone, junto com o tempo gasto na *issue* que envolvevoram as alterações dessas linhas clonadas.

As tabelas a seguir estão ordenadas de acordo com o tamanho do clone detectado.

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	—	1	354	2	1	19,00	179,00	3,00	0,00	106,00	247,50	21,00	2,00	0,00	2,00	0,00	0,00	7,00	720,00	—	—	—
DS	1	—	351	2	1	14,00	309,00	3,00	0,00	90,00	473,00	33,00	9,00	0,00	18,50	16,00	0,00	5,00	705,00	0,00	19,00	NA
DS	1	—	319	2	1	16,00	197,00	2,00	0,00	108,00	267,00	27,00	1,00	0,00	21,50	4,50	4,50	19,00	647,00	—	—	—
DS	—	1	291	2	1	19,00	195,50	3,00	0,00	113,00	292,50	23,00	2,00	0,00	3,50	0,00	0,50	7,00	779,50	—	—	—
DS	—	1	257	2	1	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	—	—	—
DS	1	—	245	2	1	14,00	306,00	3,00	0,00	87,50	451,50	32,00	8,00	0,00	18,00	16,00	0,00	4,00	684,00	0,00	5,00	NA
DS	1	—	236	2	1	12,00	203,50	2,00	0,00	119,50	1.146,50	49,50	1,00	0,00	8,00	1,00	2,00	5,00	755,00	—	—	—
DS	—	1	215	2	1	12,00	203,50	2,00	0,00	119,50	1.146,50	49,50	1,00	0,00	8,00	1,00	2,00	5,00	755,00	—	—	—
DS	—	3	211	3	3	19,00	185,00	3,00	0,00	108,67	270,67	22,00	2,00	0,00	2,67	0,00	0,33	7,00	741,33	—	—	—
DD	—	1	202	2	1	12,00	181,50	3,00	0,00	97,50	724,00	38,50	0,00	0,00	1,50	0,00	1,50	13,50	693,50	6,00	9,00	480,00
DS, DD	—	3	199	3	3	11,67	179,33	3,00	0,00	95,67	692,00	37,67	0,00	0,00	1,67	0,00	1,67	13,00	685,00	—	—	—
DS	—	1	198	2	1	19,00	195,50	3,00	0,00	113,00	292,50	23,00	2,00	0,00	3,50	0,00	0,50	7,00	779,50	—	—	—
DS	1	—	179	2	1	11,00	91,00	2,00	0,00	80,00	463,00	31,00	1,00	0,00	5,50	0,50	0,50	19,50	469,00	0,00	5,00	420,00
DS	1	—	173	2	1	18,00	55,00	2,00	0,00	77,00	20,00	8,00	3,00	0,00	5,00	4,00	4,00	2,00	268,00	0,00	2,00	NA
DD	—	1	172	2	1	11,50	176,00	3,00	0,00	94,50	628,00	36,00	0,00	0,00	2,00	0,00	2,00	12,00	673,50	10,00	16,00	480,00
DS	3	—	166	3	3	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	0,00	5,00	NA
DS	1	—	152	2	1	12,00	75,50	2,00	0,00	55,50	14,00	8,00	2,00	0,00	4,00	0,00	0,00	5,00	282,50	—	—	—
DS	1	—	148	2	1	11,00	60,00	3,00	0,00	35,00	47,00	11,00	2,00	0,00	12,00	11,00	0,00	0,00	240,00	—	—	—
DS	1	—	147	2	1	9,50	62,50	2,00	0,00	64,00	147,00	18,00	1,00	0,00	8,00	2,00	0,00	0,00	282,00	—	—	—
DS	3	3	145	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	3	—	142	3	3	15,67	163,33	2,00	0,00	104,67	219,67	24,00	1,00	0,00	20,00	3,00	3,00	18,67	566,67	—	—	—
DS	—	1	138	2	1	19,00	186,50	3,00	0,00	109,00	246,50	21,00	2,00	0,00	2,50	0,00	0,00	7,00	746,00	—	—	—
DS	1	—	134	2	1	14,00	55,00	2,00	0,00	44,00	83,00	18,00	10,00	0,00	10,00	0,00	3,00	4,00	293,00	—	—	—
DS	1	2	131	3	3	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	0,00	6,00	NA
DS	1	—	128	2	1	14,00	44,00	4,00	0,00	44,00	1,00	9,00	5,00	0,00	24,00	19,00	7,00	1,00	207,00	—	—	—
DS	3	—	128	3	3	8,33	66,67	3,00	0,00	57,00	75,67	14,33	0,00	0,00	3,33	1,33	1,33	0,00	296,33	—	—	—
DS	3	3	127	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	1	—	124	2	1	9,00	23,50	2,00	0,00	32,50	6,00	3,00	1,00	0,00	2,00	2,00	2,00	3,00	396,00	5,00	1,00	180,00
DS	1	—	124	2	1	19,00	188,00	3,00	0,00	110,00	293,50	23,00	2,00	0,00	3,00	0,00	0,50	7,00	753,50	—	—	—
DS	1	—	120	2	1	17,00	91,50	2,00	0,00	77,00	195,50	21,50	2,50	0,00	19,00	0,00	17,00	7,00	474,50	0,00	1,00	NA

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	1	—	116	2	1	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	—	—	—
DS	1	2	115	3	3	19,00	190,00	3,00	0,00	110,67	270,00	22,00	2,00	0,00	3,00	0,00	0,33	7,00	758,67	—	—	—
DS	3	—	111	3	3	16,33	63,33	2,00	0,00	61,33	79,00	11,00	1,00	7,00	7,00	1,00	5,00	11,00	316,00	1,00	1,00	NA
DS, DD	3	3	106	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	0,00	1,00	NA
DS	1	—	101	2	1	13,00	57,00	3,00	0,00	40,00	79,00	15,00	6,00	0,00	4,00	2,00	0,00	5,00	192,00	—	—	—
DS	3	3	97	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	3	—	92	3	3	17,00	95,33	2,00	0,00	78,67	198,00	21,67	2,67	0,00	20,00	0,00	18,00	7,00	475,67	0,00	1,00	NA
DS	1	—	91	2	1	10,00	26,00	2,00	0,00	44,00	0,00	4,00	1,00	0,00	5,00	0,00	0,00	0,00	140,00	—	—	—
DS	3	—	90	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	0,00	5,00	420,00
DS	1	—	90	2	1	12,50	81,00	2,00	0,00	71,00	38,00	12,50	2,00	0,00	6,00	0,00	0,00	3,00	387,00	—	—	—
DS	3	—	88	3	3	12,33	80,33	2,00	0,00	70,33	33,67	12,00	1,67	0,00	6,33	0,33	0,33	3,00	384,33	—	—	—
DS	3	—	87	3	3	17,67	53,00	2,00	0,00	75,67	20,00	8,00	3,00	0,00	5,00	4,00	4,00	2,00	261,33	0,00	2,00	NA
DS	1	—	84	2	1	14,00	34,00	3,00	0,00	43,00	7,00	6,00	2,00	0,00	4,00	2,00	1,00	0,00	163,00	—	—	—
DS	—	1	83	2	1	16,00	197,00	2,00	0,00	108,00	267,00	27,00	1,00	0,00	21,50	4,50	4,50	19,00	647,00	—	—	—
DS	1	—	80	2	1	8,50	42,00	3,00	0,00	33,00	22,50	7,50	0,00	0,00	4,00	0,00	0,00	13,00	203,00	—	—	—
DS	1	—	76	2	1	17,00	91,50	2,00	0,00	77,00	195,50	21,50	2,50	0,00	19,00	0,00	17,00	7,00	474,50	0,00	1,00	NA
DS	1	—	75	2	1	19,00	179,00	3,00	0,00	106,00	247,50	21,00	2,00	0,00	2,00	0,00	0,00	7,00	720,00	—	—	—
DS	1	—	75	2	1	8,00	26,00	2,00	0,00	34,00	19,00	7,00	1,00	0,00	3,00	3,00	3,00	0,00	132,00	—	—	—
DS	1	—	74	2	1	8,00	19,50	2,00	0,00	21,50	3,00	3,00	1,00	0,00	2,00	2,00	2,00	0,00	189,00	14,00	37,00	120,00
DS	1	—	73	2	1	12,50	81,00	2,00	0,00	71,00	38,00	12,50	2,00	0,00	6,00	0,00	0,00	3,00	387,00	—	—	—
DS	—	1	71	2	1	15,50	147,50	2,00	0,00	103,50	196,00	22,50	1,00	0,00	19,50	2,50	2,50	18,50	534,00	—	—	—
DS	1	—	70	2	1	17,00	100,50	2,00	0,00	80,50	195,50	21,50	2,50	0,00	22,00	0,00	20,00	7,00	482,50	0,00	1,00	NA
DS	1	2	70	3	3	15,67	163,33	2,00	0,00	104,67	219,67	24,00	1,00	0,00	20,00	3,00	3,00	18,67	566,67	—	—	—
DS	6	—	65	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	1	—	65	2	1	10,00	49,50	2,00	0,00	58,00	130,50	16,50	1,00	0,00	5,50	0,50	0,50	12,00	266,50	—	—	—
DS	1	2	63	3	3	9,67	62,67	2,00	0,00	64,67	147,00	18,00	1,00	0,00	8,33	2,33	1,00	0,00	285,00	—	—	—
DS	1	—	62	2	1	12,00	79,50	2,00	0,00	69,50	28,50	11,50	1,50	0,00	6,50	0,50	0,50	3,00	382,00	—	—	—
DS	—	1	61	2	1	19,00	195,50	3,00	0,00	113,00	292,50	23,00	2,00	0,00	3,50	0,00	0,50	7,00	779,50	—	—	—
DS	3	—	60	3	3	12,33	80,33	2,00	0,00	70,33	33,67	12,00	1,67	0,00	6,33	0,33	0,33	3,00	384,33	—	—	—
DS, DD	1	2	59	3	3	11,67	179,33	3,00	0,00	95,67	692,00	37,67	0,00	0,00	1,67	0,00	1,67	13,00	685,00	1,00	0,00	NA

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	1	—	56	2	1	8,00	16,00	2,00	0,00	28,00	8,00	5,00	1,00	0,00	3,00	0,00	0,00	1,00	95,00	—	—	—
DS	1	—	56	2	1	9,00	19,00	2,00	0,00	34,00	0,00	5,00	1,00	0,00	6,00	0,00	4,00	0,00	130,00	—	—	—
DS	3	—	55	3	3	17,00	95,33	2,00	0,00	78,67	198,00	21,67	2,67	0,00	20,00	0,00	18,00	7,00	475,67	0,00	1,00	NA
DS	1	—	54	2	1	10,00	14,00	2,00	0,00	31,00	3,00	3,00	3,00	0,00	2,00	2,00	0,00	0,00	93,00	—	—	—
DS	1	—	52	2	1	10,00	49,50	2,00	0,00	58,00	130,50	16,50	1,00	0,00	5,50	0,50	0,50	12,00	266,50	—	—	—
DS	1	—	52	2	1	16,00	63,50	2,00	0,00	61,50	79,00	11,00	1,00	7,00	7,00	1,00	5,00	11,00	316,00	—	—	—
DS	1	2	51	3	3	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
DS	3	—	51	3	3	16,33	63,33	2,00	0,00	61,33	79,00	11,00	1,00	7,00	7,00	1,00	5,00	11,00	316,00	—	—	—
DS	1	—	49	2	1	9,50	20,50	2,00	0,00	28,50	12,50	5,50	1,00	0,00	0,00	0,00	0,00	3,00	112,50	—	—	—
DS, DD	1	2	48	3	3	11,67	179,33	3,00	0,00	95,67	692,00	37,67	0,00	0,00	1,67	0,00	1,67	13,00	685,00	3,00	3,00	480,00
DS, DD	1	2	48	3	3	11,67	179,33	3,00	0,00	95,67	692,00	37,67	0,00	0,00	1,67	0,00	1,67	13,00	685,00	—	—	—
DS	3	—	47	3	3	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	0,00	1,00	NA
DS	1	—	47	2	1	6,00	5,00	2,00	0,00	14,00	0,00	1,00	1,00	0,00	2,00	0,00	0,00	0,00	94,00	—	—	—
DS	1	—	46	2	1	10,50	88,00	2,00	0,00	76,00	478,50	31,50	1,00	0,00	5,00	0,00	0,00	20,00	460,00	—	—	—
DS	—	1	45	2	1	7,50	16,50	2,00	0,00	30,50	10,00	5,00	1,00	0,00	1,50	1,50	1,50	0,00	103,00	—	—	—
DS, DD	3	3	45	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
DS	1	—	44	2	1	12,00	75,50	2,00	0,00	55,50	14,00	8,00	2,00	0,00	4,00	0,00	0,00	5,00	282,50	—	—	—
DS	1	—	43	2	1	17,00	94,00	2,00	0,00	78,50	203,00	22,00	3,00	0,00	19,00	0,00	17,00	7,00	470,00	—	—	—
DS	1	—	42	2	1	10,50	183,00	3,00	0,00	100,00	780,50	40,00	0,00	0,00	3,00	1,50	2,00	15,50	712,00	—	—	—
DS	1	—	42	2	1	12,00	203,50	2,00	0,00	119,50	1.146,50	49,50	1,00	0,00	8,00	1,00	2,00	5,00	755,00	—	—	—
DS	—	1	42	2	1	7,50	14,50	2,00	0,00	28,50	10,00	5,00	1,00	0,00	1,50	1,50	1,50	1,00	95,00	—	—	—
DS, DD	2	4	41	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
DS	1	—	41	2	1	9,50	15,50	2,00	0,00	41,50	4,00	4,00	1,00	0,00	5,00	0,00	0,00	1,00	123,00	—	—	—
DS	3	—	41	3	3	9,67	62,67	2,00	0,00	64,67	147,00	18,00	1,00	0,00	8,33	2,33	1,00	0,00	285,00	—	—	—
DS	1	—	40	2	1	11,00	91,00	2,00	0,00	80,00	463,00	31,00	1,00	0,00	5,50	0,50	0,50	19,50	469,00	—	—	—
DS	1	2	39	3	3	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
DS	1	—	38	2	1	8,00	16,50	2,00	0,00	20,50	1,00	2,00	1,00	0,00	2,00	2,00	2,00	0,00	98,00	—	—	—
DS	1	—	38	2	1	10,50	92,00	2,00	0,00	78,00	478,50	31,50	1,00	0,00	5,50	0,50	0,50	20,50	467,00	0,00	4,00	420,00
DS	3	—	37	3	3	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
SD, DS	1	2	36	3	3	9,67	22,33	2,00	0,00	43,00	1,33	4,00	1,00	0,00	5,00	0,00	0,00	0,33	134,00	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	—	1	35	2	1	7,00	22,00	3,00	0,00	36,00	28,00	8,00	0,00	0,00	4,00	0,00	0,00	1,00	106,00	—	—	—
DS	3	—	34	3	3	17,67	53,00	2,00	0,00	75,67	20,00	8,00	3,00	0,00	5,00	4,00	4,00	2,00	261,33	0,00	2,00	NA
DS	1	2	34	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
DS	3	—	33	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	0,00	4,00	420,00
DS	1	—	33	2	1	9,50	62,50	2,00	0,00	64,00	147,00	18,00	1,00	0,00	8,00	2,00	0,00	0,00	282,00	—	—	—
DS	3	—	33	3	3	13,00	59,67	3,00	0,00	45,33	88,67	15,67	6,67	0,00	4,33	2,00	0,00	5,67	201,33	—	—	—
DS	1	—	33	2	1	11,00	16,50	2,00	0,00	32,50	4,00	4,00	1,00	0,00	3,00	0,00	0,00	3,00	98,50	—	—	—
SD, DS	1	2	33	3	3	7,67	22,00	2,00	0,00	32,00	16,00	6,33	1,00	0,00	2,67	2,67	2,67	0,33	119,33	—	—	—
DS	1	—	32	2	1	9,50	62,50	2,00	0,00	65,00	147,00	18,00	1,00	0,00	8,50	2,50	1,50	0,00	286,00	—	—	—
DS	1	—	32	2	1	4,50	18,50	3,00	0,00	30,00	16,50	7,50	1,00	0,00	2,00	0,00	0,00	0,00	97,50	—	—	—
SD, DS, DD	6	—	32	4	6	7,75	20,25	2,00	0,00	31,25	14,50	6,00	1,00	0,00	2,25	2,25	2,25	0,50	113,50	—	—	—
DS	1	—	32	2	1	11,50	31,00	2,00	0,00	37,00	10,00	5,00	1,00	0,00	3,00	0,00	2,00	2,00	118,00	0,00	1,00	NA
DS	1	—	31	2	1	10,50	183,00	3,00	0,00	100,00	780,50	40,00	0,00	0,00	3,00	1,50	2,00	15,50	712,00	—	—	—
DS	3	—	31	3	3	9,67	62,67	2,00	0,00	64,67	147,00	18,00	1,00	0,00	8,33	2,33	1,00	0,00	285,00	—	—	—
DS	3	—	31	3	3	9,33	16,00	2,00	0,00	42,00	3,00	3,67	1,00	0,00	5,33	0,33	0,33	1,00	123,33	—	—	—
DS, DD	2	4	30	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
DS	3	3	29	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
SD, DS, DD	2	4	29	4	6	8,00	18,00	2,00	0,00	21,00	2,00	2,50	1,00	0,00	2,00	2,00	2,00	0,00	143,50	—	—	—
DS	1	—	28	2	1	10,00	7,00	2,00	0,00	20,00	0,00	1,00	1,00	0,00	3,00	0,00	0,00	1,00	65,00	—	—	—
DS	1	—	28	2	1	9,00	19,50	2,00	0,00	27,50	6,00	3,00	1,00	0,00	2,00	2,00	2,00	2,00	117,00	—	—	—
DS	1	—	28	2	1	9,50	15,50	2,00	0,00	41,50	4,00	4,00	1,00	0,00	5,00	0,00	0,00	1,00	123,00	—	—	—
DS	1	—	28	2	1	9,50	50,00	2,00	0,00	58,00	138,50	17,00	1,00	0,00	5,50	0,50	0,50	12,50	270,00	—	—	—
DS, DD	3	3	28	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	0,00	2,00	480,00
DS	—	1	28	2	1	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	0,00	1,00	NA
DS	1	—	28	2	1	9,00	23,50	2,00	0,00	32,50	6,00	3,00	1,00	0,00	2,00	2,00	2,00	3,00	396,00	—	—	—
DS	1	—	28	2	1	8,00	66,50	3,00	0,00	57,50	71,00	14,00	0,00	0,00	4,00	2,00	2,00	0,00	300,00	—	—	—
DS	1	—	27	2	1	8,50	66,00	3,00	0,00	55,50	78,00	14,50	0,00	0,00	2,00	0,00	0,00	0,00	285,00	—	—	—
DS	1	—	26	2	1	9,50	50,00	2,00	0,00	58,00	138,50	17,00	1,00	0,00	5,50	0,50	0,50	12,50	270,00	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	3	—	26	3	3	15,00	188,00	3,00	0,00	63,33	133,67	15,67	3,00	0,00	3,00	0,00	0,00	8,67	492,33	—	—	—
SD, DS, DD	10	—	26	5	10	9,60	20,00	2,00	0,00	42,80	1,80	3,80	1,00	0,00	5,20	0,20	0,20	0,60	130,00	—	—	—
DS	1	2	26	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
DS	3	—	26	3	3	8,33	66,67	3,00	0,00	57,00	75,67	14,33	0,00	0,00	3,33	1,33	1,33	0,00	296,33	—	—	—
DS	3	—	26	3	3	12,33	80,33	2,00	0,00	70,33	33,67	12,00	1,67	0,00	6,33	0,33	0,33	3,00	384,33	—	—	—
DS	1	—	25	2	1	10,00	21,00	2,00	0,00	37,00	8,00	5,00	1,00	0,00	2,00	0,00	1,00	1,00	95,00	—	—	—
DS	1	—	25	2	1	11,50	31,00	2,00	0,00	37,00	10,00	5,00	1,00	0,00	3,00	0,00	2,00	2,00	118,00	—	—	—
DS	1	—	25	2	1	9,00	19,50	2,00	0,00	27,50	6,00	3,00	1,00	0,00	2,00	2,00	2,00	2,00	117,00	—	—	—
SS, SD, DS	2	4	25	4	6	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	—	—	—
DS	3	—	24	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
DS	3	—	24	3	3	17,00	95,33	2,00	0,00	78,67	198,00	21,67	2,67	0,00	20,00	0,00	18,00	7,00	475,67	—	—	—
DS	3	—	24	3	3	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
DS	—	1	24	2	1	7,50	14,50	2,00	0,00	28,50	10,00	5,00	1,00	0,00	2,50	2,50	2,50	1,00	95,00	—	—	—
DS	1	—	23	2	1	8,00	9,00	2,00	0,00	15,00	1,00	2,00	2,00	0,00	1,00	0,00	0,00	0,00	60,00	—	—	—
DS	1	—	23	2	1	11,00	32,50	2,00	0,00	38,00	12,50	5,50	1,00	0,00	3,50	0,00	2,50	2,50	124,50	0,00	1,00	NA
DS	1	—	22	2	1	5,00	11,00	3,00	0,00	18,00	19,00	7,00	0,00	0,00	1,00	1,00	1,00	0,00	62,00	—	—	—
SD, DS, DD	6	9	22	6	15	11,00	64,00	2,00	0,00	63,17	82,00	14,33	1,33	0,00	5,83	0,33	0,33	7,50	323,17	—	—	—
SS, SD, DS	2	4	22	4	6	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	—	—	—
DS	3	—	22	3	3	11,33	31,67	2,00	0,00	37,33	11,67	5,33	1,00	0,00	3,33	0,00	2,33	2,33	124,67	—	—	—
SD	—	1	22	2	1	10,00	39,50	2,00	0,00	53,00	75,50	11,00	1,00	0,00	6,50	1,00	0,00	0,50	203,50	—	—	—
SD	—	1	22	2	1	9,50	40,00	2,00	0,00	54,50	74,00	10,50	1,00	0,00	7,50	2,00	2,00	0,50	207,50	—	—	—
SD, DS	2	4	22	4	6	9,75	32,50	2,00	0,00	48,75	37,75	7,50	1,00	0,00	6,00	0,75	0,75	0,25	173,25	—	—	—
DS, DD	1	2	21	3	3	9,67	22,67	2,00	0,00	29,67	11,67	5,33	1,00	0,00	0,67	0,67	0,67	2,67	117,00	—	—	—
SD, DS, DD	2	4	21	4	6	9,00	21,50	2,00	0,00	30,00	6,00	3,00	1,00	0,00	2,00	2,00	2,00	2,50	256,50	—	—	—
SD, DS, DD	13	15	20	8	28	9,63	36,00	2,00	0,00	51,00	56,25	9,13	1,00	0,00	6,38	1,00	0,50	0,38	188,13	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	1	2	20	3	3	9,67	62,67	2,00	0,00	64,67	147,00	18,00	1,00	0,00	8,33	2,33	1,00	0,00	285,00	—	—	—
DS	1	2	20	3	3	11,33	31,67	2,00	0,00	37,33	11,67	5,33	1,00	0,00	3,33	0,00	2,33	2,33	124,67	0,00	1,00	NA
SD, DS, DD	36	—	20	9	36	12,78	63,78	2,00	0,00	62,56	81,00	13,22	1,22	2,33	6,22	0,56	1,89	8,67	230,78	—	—	—
DS	3	3	20	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	1	2	20	3	3	15,00	188,00	3,00	0,00	63,33	133,67	15,67	3,00	0,00	3,00	0,00	0,00	8,67	492,33	0,00	1,00	NA
DS	3	—	20	3	3	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
DS	3	—	20	3	3	17,67	53,00	2,00	0,00	75,67	20,00	8,00	3,00	0,00	5,00	4,00	4,00	2,00	261,33	—	—	—
SD, DS, DD	4	6	19	5	10	13,40	119,40	2,60	0,00	51,00	81,80	11,00	2,20	0,00	3,00	0,00	0,00	6,40	334,80	—	—	—
DS	3	—	19	3	3	15,67	163,33	2,00	0,00	104,67	219,67	24,00	1,00	0,00	20,00	3,00	3,00	18,67	566,67	—	—	—
DS	1	2	19	3	3	9,67	23,00	2,00	0,00	43,67	0,33	3,67	1,00	0,00	5,33	0,33	0,33	0,33	134,67	—	—	—
SD, DS, DD	6	9	19	6	15	9,67	19,83	2,00	0,00	30,83	5,33	3,33	1,00	0,00	2,33	1,33	1,33	2,67	203,83	—	—	—
SD, DS, DD	4	11	18	6	15	12,83	108,83	2,50	0,00	53,17	92,67	12,00	2,00	0,00	3,50	0,17	0,17	7,50	326,83	—	—	—
DS	1	—	18	2	1	9,00	16,00	2,00	0,00	42,00	2,50	3,50	1,00	0,00	5,50	0,50	0,50	1,00	123,00	—	—	—
DS	—	1	18	2	1	11,50	180,50	3,00	0,00	95,00	724,00	38,50	0,00	0,00	1,50	0,00	1,50	13,50	688,00	—	—	—
DS, DD	1	2	17	3	3	11,67	179,33	3,00	0,00	95,67	692,00	37,67	0,00	0,00	1,67	0,00	1,67	13,00	685,00	—	—	—
DS, DD	6	—	17	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
SD	—	3	17	3	3	10,33	48,67	2,00	0,00	57,33	193,67	17,00	1,00	0,00	5,00	0,00	0,00	10,33	277,33	—	—	—
SD, DS, DD	6	39	17	10	45	11,30	73,90	2,30	0,00	43,90	58,00	8,40	1,60	0,00	2,90	0,90	0,90	5,50	298,70	—	—	—
DS	1	2	17	3	3	17,00	95,33	2,00	0,00	78,67	198,00	21,67	2,67	0,00	20,00	0,00	18,00	7,00	475,67	—	—	—
DS	—	1	17	2	1	8,50	42,00	3,00	0,00	33,00	22,50	7,50	0,00	0,00	4,00	0,00	0,00	13,00	203,00	—	—	—
DS	1	—	17	2	1	11,00	16,50	2,00	0,00	32,50	4,00	4,00	1,00	0,00	3,00	0,00	0,00	3,00	98,50	—	—	—
DS	1	2	16	3	3	15,00	188,00	3,00	0,00	63,33	133,67	15,67	3,00	0,00	3,00	0,00	0,00	8,67	492,33	—	—	—
DS	3	—	16	3	3	9,33	16,00	2,00	0,00	42,00	3,00	3,67	1,00	0,00	5,33	0,33	0,33	1,00	123,33	—	—	—
SS, DS	12	16	16	8	28	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
SD	—	1	16	2	1	9,50	36,50	2,00	0,00	53,50	74,00	10,00	1,00	0,00	6,00	1,00	1,00	7,00	205,50	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	3	—	16	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
DS	6	—	16	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
SD, DS, DD	3	12	15	6	15	8,33	18,50	2,00	0,00	23,17	3,33	2,67	1,00	0,00	2,00	2,00	2,00	0,67	134,67	—	—	—
DS	—	1	15	2	1	10,50	88,00	2,00	0,00	76,00	478,50	31,50	1,00	0,00	5,00	0,00	0,00	20,00	460,00	—	—	—
SD, DS, DD	2	4	15	4	6	7,75	21,25	2,00	0,00	32,25	14,50	6,00	1,00	0,00	2,25	2,25	2,25	0,00	117,50	—	—	—
DS	—	1	15	2	1	9,00	30,50	2,00	0,00	22,00	1,00	2,00	1,00	0,00	7,50	3,50	6,50	5,50	130,50	—	—	—
DS, DD	6	—	15	4	6	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
SD	—	1	15	2	1	13,50	186,50	3,00	0,00	72,00	290,00	25,00	8,50	0,00	11,50	9,00	0,00	5,50	458,00	—	—	—
SD, DS, DD	6	—	15	4	6	7,75	20,25	2,00	0,00	31,25	14,50	6,00	1,00	0,00	2,75	2,75	2,75	0,50	113,50	—	—	—
SD, DS, DD	3	3	15	4	6	13,75	246,75	3,00	0,00	80,75	371,25	28,50	8,25	0,00	15,00	12,50	0,00	5,25	575,50	—	—	—
SD	—	3	15	3	3	9,33	49,33	2,00	0,00	55,67	209,33	17,67	1,00	0,00	5,00	0,00	0,00	11,33	277,67	—	—	—
DS	3	—	15	3	3	7,67	14,67	2,00	0,00	28,67	10,00	5,00	1,00	0,00	2,00	2,00	2,00	1,00	95,33	—	—	—
SD, DS, DD	4	24	14	8	28	8,50	19,75	2,00	0,00	25,50	4,00	2,75	1,00	0,00	2,00	2,00	2,00	1,25	200,00	—	—	—
SS, DS	6	9	14	6	15	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	0,00	5,00	NA
SD, DS, DD	10	—	14	5	10	7,80	19,60	2,00	0,00	31,20	13,60	5,80	1,00	0,00	2,40	2,40	2,40	0,40	111,60	—	—	—
DS	3	—	14	3	3	17,00	95,33	2,00	0,00	78,67	198,00	21,67	2,67	0,00	20,00	0,00	18,00	7,00	475,67	0,00	2,00	NA
DS	1	—	14	2	1	9,00	30,50	2,00	0,00	22,00	1,00	2,00	1,00	0,00	7,50	3,50	6,50	5,50	130,50	—	—	—
SS, DS	12	16	14	8	28	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	—	1	14	2	1	11,50	180,50	3,00	0,00	95,00	724,00	38,50	0,00	0,00	1,50	0,00	1,50	13,50	688,00	—	—	—
SS, DS	6	9	14	6	15	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
SD	—	1	14	2	1	8,50	18,50	2,00	0,00	24,50	3,50	2,50	1,00	0,00	2,00	2,00	2,00	1,00	108,50	—	—	—
SS, DS, DD	6	4	14	5	10	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
DS	3	—	13	3	3	17,67	53,00	2,00	0,00	75,67	20,00	8,00	3,00	0,00	5,00	4,00	4,00	2,00	261,33	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
SS, DS	6	9	13	6	15	15,67	163,33	2,00	0,00	104,67	219,67	24,00	1,00	0,00	20,00	3,00	3,00	18,67	566,67	—	—	—
SS, DS	7	8	13	6	15	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
SS, DS	6	9	13	6	15	13,00	59,67	3,00	0,00	45,33	88,67	15,67	6,67	0,00	4,33	2,00	0,00	5,67	201,33	—	—	—
SD	—	1	13	2	1	8,50	17,50	2,00	0,00	23,50	3,50	2,50	1,00	0,00	2,00	2,00	2,00	1,00	106,50	—	—	—
SS, DS, DD	12	16	13	8	28	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
DS, DD	1	2	13	3	3	10,67	177,33	3,00	0,00	97,00	665,67	37,00	0,00	0,00	3,00	1,00	2,33	13,33	687,67	—	—	—
DS	3	—	13	3	3	11,33	31,67	2,00	0,00	37,33	11,67	5,33	1,00	0,00	3,33	0,00	2,33	2,33	124,67	0,00	1,00	NA
DS, DD	1	2	13	3	3	11,67	179,33	3,00	0,00	95,67	692,00	37,67	0,00	0,00	1,67	0,00	1,67	13,00	685,00	—	—	—
DS	—	1	13	2	1	9,00	30,50	2,00	0,00	22,00	1,00	2,00	1,00	0,00	7,50	3,50	6,50	5,50	130,50	—	—	—
SD	1	—	13	2	1	8,00	18,50	2,00	0,00	21,50	2,00	2,50	1,00	0,00	2,00	2,00	2,00	0,00	144,50	2,00	2,00	120,00
DS	1	2	13	3	3	8,67	41,67	3,00	0,00	31,33	23,00	7,67	0,00	0,00	8,33	4,00	0,33	13,33	210,33	—	—	—
SD, DS, DD	6	4	13	5	10	16,40	161,60	3,00	0,00	96,20	233,00	20,40	2,20	0,00	4,80	2,20	2,40	6,40	651,60	—	—	—
DS	1	—	13	2	1	8,00	19,50	2,00	0,00	21,50	3,00	3,00	1,00	0,00	2,00	2,00	2,00	0,00	189,00	1,00	2,00	120,00
SD, DS, DD	1	2	13	3	3	7,00	21,67	3,00	0,00	34,00	27,33	8,00	0,00	0,00	4,00	0,00	0,00	1,00	101,67	—	—	—
DS	3	—	13	3	3	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
SD, DS, DD	4	6	12	5	10	9,60	43,80	2,00	0,00	55,40	89,80	12,40	1,00	0,00	7,00	1,40	0,60	0,40	220,20	—	—	—
SS, DS	2	4	12	4	6	12,00	203,50	2,00	0,00	119,50	1.146,50	49,50	1,00	0,00	8,00	1,00	2,00	5,00	755,00	—	—	—
SD	—	3	12	3	3	10,33	50,67	2,00	0,00	58,67	57,67	10,33	1,00	0,00	6,33	1,00	1,00	5,67	263,33	—	—	—
SS, DS	2	4	12	4	6	12,00	203,50	2,00	0,00	119,50	1.146,50	49,50	1,00	0,00	8,00	1,00	2,00	5,00	755,00	—	—	—
SD, DS, DD	2	4	12	4	6	10,50	111,25	2,00	0,00	76,75	573,25	27,25	1,00	0,00	7,00	0,50	3,00	2,50	442,50	—	—	—
SS, DS	4	6	12	5	10	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
DS	1	2	12	3	3	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
SS, DS	1	2	12	3	3	10,50	88,00	2,00	0,00	76,00	478,50	31,50	1,00	0,00	5,00	0,00	0,00	20,00	460,00	—	—	—
DD	1	—	12	2	1	10,00	25,00	2,00	0,00	31,00	12,50	5,50	1,00	0,00	1,00	1,00	1,00	2,50	124,00	—	—	—
SS, DS	6	9	12	6	15	15,67	163,33	2,00	0,00	104,67	219,67	24,00	1,00	0,00	20,00	3,00	3,00	18,67	566,67	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
DS	—	1	12	2	1	10,50	25,50	2,00	0,00	35,50	11,50	5,50	1,00	0,00	3,00	0,00	2,00	2,00	112,50	—	—	—
SD, DS, DD	1	5	12	4	6	9,75	60,75	2,00	0,00	62,25	272,75	21,00	1,00	0,00	5,25	0,25	0,25	13,50	327,25	—	—	—
DS	3	3	12	4	6	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
SS, DS, DD	12	16	12	8	28	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
SD	1	—	12	2	1	8,00	17,50	2,00	0,00	20,50	2,00	2,50	1,00	0,00	2,00	2,00	2,00	0,00	142,50	—	—	—
SS	—	1	12	2	1	15,00	6,00	3,00	0,00	85,00	143,00	18,00	0,00	0,00	10,00	0,00	0,00	17,00	347,00	—	—	—
DS	1	2	12	3	3	15,00	188,00	3,00	0,00	63,33	133,67	15,67	3,00	0,00	3,00	0,00	0,00	8,67	492,33	—	—	—
DS	3	—	12	3	3	9,67	62,67	2,00	0,00	64,67	147,00	18,00	1,00	0,00	8,33	2,33	1,00	0,00	285,00	—	—	—
SS, DS	2	4	11	4	6	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	—	—	—
DS, DD	1	2	11	3	3	10,67	177,33	3,00	0,00	97,00	665,67	37,00	0,00	0,00	3,00	1,00	2,33	13,33	687,67	0,00	1,00	480,00
SS, DS	6	9	11	6	15	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
SD, DS, DD	1	9	11	5	10	10,20	64,60	2,00	0,00	63,80	224,60	19,20	1,20	0,00	5,40	0,20	0,20	11,40	338,80	—	—	—
DS, DD	3	—	11	3	3	9,67	22,67	2,00	0,00	29,67	11,67	5,33	1,00	0,00	0,67	0,67	0,67	2,67	117,00	—	—	—
SS, DS	6	9	11	6	15	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
SS, SD, DS, DD	6	22	11	8	28	15,50	138,50	2,25	0,00	99,75	200,50	22,50	0,75	0,00	17,50	2,25	2,25	18,25	511,75	—	—	—
DS	1	2	11	3	3	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
SD, DS, DD	18	18	11	9	36	10,56	68,78	2,00	0,00	66,11	233,33	20,56	1,11	0,00	5,56	0,33	0,33	12,67	355,89	0,00	5,00	420,00
SD, DS, DD	3	3	11	4	6	10,25	24,75	2,00	0,00	36,25	9,75	5,25	1,00	0,00	2,50	0,00	1,50	1,50	107,75	—	—	—
SD, DS, DD	2	8	11	5	10	7,60	29,80	3,00	0,00	33,60	25,40	7,80	0,00	0,00	4,00	0,00	0,00	5,80	142,20	—	—	—
SD, DS, DD	7	8	11	6	15	15,50	229,67	3,00	0,00	91,50	345,00	26,67	6,17	0,00	11,17	8,33	0,17	5,83	643,50	—	—	—
SS, DS	6	9	11	6	15	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
SS	—	1	11	2	1	9,00	180,00	3,00	0,00	102,00	741,00	39,00	0,00	0,00	5,00	3,00	3,00	16,00	716,00	—	—	—

Tipo Relação	Tipo 1	Tipo 2	Tam. Clone	Qtd. Fragmentos	Qtd. Relações	cbo	wmc	dit	noc	rfc	lcom	nom	nopm	nosm	nof	nopf	nosf	nosi	loc	LoC add	LoC rem.	Tempo gasto
SS, DS, DD	12	16	11	8	28	11,00	179,50	3,00	0,00	97,25	704,25	38,00	0,00	0,00	2,50	0,75	2,00	13,75	692,75	—	—	—
DS	3	—	11	3	3	13,00	59,67	3,00	0,00	45,33	88,67	15,67	6,67	0,00	4,33	2,00	0,00	5,67	201,33	—	—	—
SD, DS, DD	28	—	11	8	28	13,50	102,63	2,50	0,00	65,25	136,00	12,25	1,50	0,00	2,38	1,00	1,13	3,50	446,63	—	—	—
DS	—	1	11	2	1	10,50	92,00	2,00	0,00	78,00	478,50	31,50	1,00	0,00	5,50	0,50	0,50	20,50	467,00	—	—	—
SS, DS	6	9	11	6	15	14,00	307,33	3,00	0,00	89,00	459,00	32,33	8,33	0,00	18,33	16,00	0,00	4,67	694,00	—	—	—
SS, SD, DS, DD	6	9	11	6	15	10,50	111,25	2,00	0,00	76,75	573,25	27,25	1,00	0,00	7,00	0,50	3,00	2,50	442,50	—	—	—
SS, DS	13	32	11	10	45	19,00	187,25	3,00	0,00	109,50	270,00	22,00	2,00	0,00	2,75	0,00	0,25	7,00	749,75	—	—	—
DS	1	2	10	3	3	15,00	188,00	3,00	0,00	63,33	133,67	15,67	3,00	0,00	3,00	0,00	0,00	8,67	492,33	—	—	—
SS, DS	3	3	10	4	6	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
SD	—	1	10	2	1	10,50	48,00	2,00	0,00	56,00	13,00	7,00	1,00	0,00	6,50	1,00	1,00	2,00	251,50	—	—	—
SD	1	—	10	2	1	9,00	22,00	2,00	0,00	30,50	6,00	3,00	1,00	0,00	2,00	2,00	2,00	2,50	255,50	—	—	—
SD, DS, DD	39	27	10	12	66	12,00	67,42	2,00	0,00	64,92	187,25	18,17	1,08	1,75	5,92	0,50	1,50	12,25	345,92	—	—	—
SD, DS, DD	6	9	10	6	15	10,17	76,50	2,00	0,00	71,33	310,17	24,67	1,00	0,00	6,83	1,33	0,67	10,00	375,17	0,00	5,00	420,00
DS	1	2	10	3	3	9,67	47,67	2,00	0,00	56,00	130,33	16,67	1,00	0,00	5,33	0,33	0,33	12,00	262,00	—	—	—
SS, DS	3	3	10	4	6	10,67	90,33	2,00	0,00	78,00	473,33	31,33	1,00	0,00	5,33	0,33	0,33	20,00	465,33	—	—	—
DS	—	1	10	2	1	8,50	14,00	2,00	0,00	23,50	8,00	4,50	1,00	0,00	0,00	0,00	0,00	0,00	76,00	—	—	—
SS	—	1	10	2	1	9,00	41,00	3,00	0,00	28,00	24,00	8,00	0,00	0,00	17,00	12,00	1,00	14,00	225,00	—	—	—
SS, DS	8	20	10	8	28	16,00	253,00	3,00	0,00	69,00	180,00	19,00	4,00	0,00	3,00	0,00	0,00	10,00	625,50	—	—	—
SD, DS, DD	3	3	10	4	6	15,50	138,50	2,25	0,00	99,75	200,50	22,50	0,75	0,00	17,50	2,25	2,25	18,25	511,75	—	—	—
DS	3	—	10	3	3	13,00	59,67	3,00	0,00	45,33	88,67	15,67	6,67	0,00	4,33	2,00	0,00	5,67	201,33	—	—	—
SD, DD	—	6	10	4	6	11,75	86,75	2,00	0,00	70,75	110,00	14,50	1,00	0,00	10,00	1,75	1,75	9,00	355,50	—	—	—
SD, DS, DD	1	2	10	3	3	10,33	82,33	2,00	0,00	74,00	368,00	27,00	1,00	0,00	6,67	1,33	1,33	13,67	408,33	—	—	—

APÊNDICE D – MAPEAMENTO DAS CLASSES DE CUSTOMIZAÇÃO ENTRE OS BRANCHES

Taxa Clones	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
C1				■			■	■			
C2	■										
C3	■			■			■	■			■
C4	■										
C5	■			■			■	■			■
C6		■			■				■		
C7			■			■		■			
C8			■			■		■		■	
C9	■										
C10					■				■		
C11		■			■				■		
C12		■			■				■		
C13			■			■		■		■	
C14		■			■				■		
C15	■										
C16			■			■		■		■	
C17		■			■				■		
C18	■	■			■				■		
C19		■			■				■		
C20			■			■		■		■	
C21		■			■				■		
C22		■			■				■		
C23	■		■			■		■		■	
C24	■		■			■		■		■	
C25		■			■				■		
C26			■			■		■		■	
C27		■			■				■		
C28		■			■				■		
C29	■	■			■				■		
C30	■	■			■				■		
C31	■										
C32				■							
C33	■										
C34		■			■				■		
C35			■			■		■		■	

Taxa Clones	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
C36											
C37											
C38											
C39											
C40											
C41											
C42											
C43											
C44											
C45											
C46											
C47											
C48											
C49											
C50											
C51											
C52											
C53											
C54											
C55											
C56											
C57											
C58											
C59											
C60											
C61											
C62											
C63											
C64											
C65											
C66											
C67											
C68											
C69											
C70											