
**MALLÚ EDUARDA BATISTA**

# DETECÇÃO DE CÓDIGO CLONADO EM LINHA DE PRODUTOS DE SOFTWARE

**LAVRAS – MG**

**2018**

**MALLÚ EDUARDA BATISTA**

# DETECÇÃO DE CÓDIGO CLONADO EM LINHA DE PRODUTOS DE SOFTWARE

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Softare, para a obtenção do título de Mestre(a).

Prof. Heitor Augustus Xavier Costa
Orientador

**LAVRAS – MG**
**2018**

**Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos**

**da Biblioteca Universitária da UFLA**

Batista, Mallú Eduarda

Detecção de código clonado em Linha de Produtos de Software / Mallú Eduarda Batista. – Lavras : UFLA, 2018.

67 p. : il.

Dissertação–Universidade Federal de Lavras, 2019.

Orientador: Prof. Heitor Augustus Xavier Costa.

Bibliografia.

1. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

# RESUMO

O resumo deve conter palavras representativas do conteúdo do trabalho, localizadas abaixo do resumo, separadas por dois espaços, antecedidas da expressão palavras-chave. Essas palavras representativas são grafadas com a letra inicial em maiúscula, separadas entre si por ponto.

**Palavras-chave:** Resumo. Palavras. Representativas.

## ABSTRACT

The abstract should contain representative words of the work content, located below the abstract, separated by two spaces, preceded by the keyword expression. These representative words are spelled with the first letter capitalized, separated by point.



**Keywords:** Summary. Words. Representative.

**SUMÁRIO**

# 1 INTRODUÇÃO

Uma das maiores inovações tecnológicas da era industrial é conhecida como linha de produção ou linha de montagem. Concebida no ano de 1913 por Henry Ford (CORRÊA; CORRÊA, 2000), o processo de produção em série possibilitou redução de custos e a produção em massa.

Aliados a evolução e a automatização dos processos na produção industrial, destaca-se, na área de Engenharia de Software, a evolução de softwares legados ou criação de novos sistemas utilizando o conceito de Linha de Produtos de Software (LPS) (do inglês *Software Product Line* ou SPL) (LAGUNA; CRESPO, 2013a). Tal conceito atende à sistemas de softwares que compartilham um conjunto comum de funcionalidades desenvolvidas a partir de uma base em comum, e que são voltados à atender necessidades de um segmento de mercado ou missão.

Clones de código são encontrados em diversos contextos de desenvolvimento de software a identificação desses clones possibilita a prevenção sa propagação de erros, identificação e correção de bugs o que também acarreta em maior facilidade de manutenção. Trechos de código-fonte idênticos ou similares são classificados como clones de código, uma vez que eles surgem de diversas maneiras e são comumente oriundos de práticas de "*copy and paste*"por programadores intencionalmente ou não. Algumas dessas práticas são reutilização de código *ad-hoc*, adição de funcionalidades semelhantes e outros.

Por conseguinte, pesquisas recentes mostram a existência de uma vasta gama de ferramentas que realizam tal detecção. Elas identificam tipos variados de clones e são implementadas por técnicas baseadas, em sua maioria, em texto (*text-based*), tokens (*token-based*), árvores (*tree-based*), grafos (*graph-based*), métricas (*metric-based*) e híbridas (*hybrid-based*). Nesse cenário, a maioria dessas ferramentas de detecção são desenvolvidas para detectar clones de código em softwares implementados sob o Paradigma Orientado a Objetos (POO). Contudo, por se tratar de um paradigma emergente, o Paradigma Orientado a características vem se

destacando na utilização para implementar LPS's, isto é, Linhas de Produtos de Software Orientada a Características.

## 1.1 Motivação

Como uma LPS gera vários produtos derivados de comunalidades e acrescidos de variabilidades, a detecção de clones diretamente na linha de produtos visa a propagação de alterações por todos os produtos dessa família de softwares. A utilização do POC na implementação de LPS é beneficiada por não possuir limitações especificadas por conceitos de herança, por exemplo.

Constatada a quase inexistência de pesquisas acerca da detecção de clones de código em Linhas de produtos de Software com uma pesquisa de mapeamento sistemático relatada no capítulo 4 deste trabalho de dissertação (Clonagem de código - Estado da arte) observamos uma lacuna de pesquisa a ser preenchida com os resultados esperados deste trabalho.

## 1.2 Objetivos

Neste trabalho propomos um abordagem híbrida para detecção de clones em código fonte, baseada na análise de chamadas de métodos e instruções, em Linha de Produtos de software Orientada a Características. Para da suporte ao desenvolvimento dessa abordagem, foram estabelecidos os seguintes objetivos específicos:

- **Estado da arte**: Para de compreender oa cenárioa de Clonagem de Código e Linha de produtos de Software, é necessário estudar ambos os conceitos abordados pela literatura.

- **Identificar abordagens existentes**: Para obter apoio no desenvolvimento da abordagem é necessário identificar a existência de ferramentas e abor-

dagens para detecção de clones de código e em qual paradigma elas são identificadas.

- **Proposta de abordagem**: Formalizar a proposta de abordagem de detecção de clones em LPS Orientada a Características, bem como a definição da linguagem e outros elementos julgadas importantes.

- **Ferramenta de apoio computacional**: Para automatizar o processo de detecção, será desenvolvida uma ferramenta que implementa a abordagem proposta.

## 1.3 Estrutura do Trabalho

Afim de estabelecer e identificar o conteúdo abordado por este trabalho, este documento está organizado da seguinte forma. Na Seção 2, a metodologia do trabalho é descrita em termos da classificação da pesquisa e do método de pesquisa utilizado. Na Seção 3, é descrito [CONTINUAR]

# 2 METODOLOGIA DE PESQUISA

Metodologia determina formas a serem utilizadas para reunir dados necessário para o êxito do trabalho, através de técnicas de coleta e análise de dados (MORESI et al., 2003).

O restante desse capítulo está organizado da seguinte forma. A Seção 2.1 classifica a pesquisa de acordo com MORESI et al., 2003. Na Seção 2.2, são apresentadas as etapas de desenvolvimento do projeto bem como suas especificações no processo.

## 2.1 Classificação da Pesquisa

Segundo MORESI et al. 2003, a pesquisa pode ser classificada quanto a:

- **Natureza**: do ponto de vista de sua natureza, a pesquisa classifica-se em (a) básica, que tem por objetivo gerar conhecimentos úteis para obter avanços na ciência, sem previsão de aplicação prática e envolve verdades e interesses universais, ou (b) aplicada, que tem por objetivo gerar conhecimento, aplicáveis na prática, buscando solução para problemas específicos e envolve verdades e interesses locais. Este trabalho pode ser classificado como **pesquisa aplicada**, pois visa o desenvolvimento de uma abordagem de detecção de clones de código em Linhas de Produtos de Software.

- **Abordagem:** do ponto de vista da abordagem do problema, a pesquisa classifica-se em (a) quantitativa, que tem por objetivo traduzir informações em números para classifica-las e analisa-las através do uso de técnicas de estatística, ou (b) qualitativa, que utiliza de interpretação de fenômenos e atribuição de significado, sendo considerada descritiva. Este trabalho pode ser classificado como **pesquisa quantitativa** pois utiliza de recursos e estatística para análise dos resultados.

- **Finalidade**: do ponto de vista da finalidade, a pesquisa pode ser (a) exploratória, que a investigação acontece onde existe pouco conhecimento acumulado e sistematizado, e/ou (b) descritiva, onde a pesquisa é direcionada a mostrar características de determinados fenômenos ou populações, e/ou (c) explicativa, que tenta tornar algo compreensível, através da justificativa dos motivos, e/ou (d) metodológica, que é referente a elaboração de instrumentos de captação ou de manipulação da realidade, e/ou (e) intervencionista, que objetiva principalmente interferir na realidade estuda visando modifica-la. Este trabalho pode ser classificado como **pesquisa exploratória e metodológica** visto que não foram encontrados estudos sobre detecção de clones de código em linha de produtos de software orientada a características em um mapeamento sistemático da literatura realizado, e que o método de detecção proposto disporá de uma ferramenta que semi-automatiza o processo.

- **Meios de investigação:** do ponto de vista das maneiras de busca pela informação, a pesquisa pode ser (a) de campo, que consiste na investigação empírica no local onde ocorre/ocorreu um fenômeno ou que disponibiliza elementos que o explica, ou (b) de laboratório, que a experiência é realizada em local restringido, ou (c) telematizada, que utiliza computador e telecomunicações, ou (d) documental, que faz investigação de documentos, ou (e) bibliográfica, que é o estudo sistematizado desenvolvido baseando-se em material disponível ao publico em geral, ou (f) experimental, que é investigação empírica utilizando variáveis independentes de forma a manipula-las e controla-las para observar variações que as mesmas surtem em variáveis dependentes, ou (g) *ex post facto*, que refere-se a um fato já ocorrido, ou (h) participante, que introduz a fronteira pesquisador/pesquisado ao contexto do problema investigado, ou (i) pesquisa-ação, que supõe intervencionar participativamente na realidade social, (j) estudo de caso, que é delimitado a uma ou poucas unidades e tem caráter de detalhamento. Este trabalho pode ser

classificado como **pesquisa bibliográfica** pois realiza o estudo sistemático de materiais contidos em bibliotecas digitais, livros e outros.

## 2.2 Método de Pesquisa

O presente trabalho iniciou-se em março de 2018 e tem como previsão o término para fevereiro de 2020. Neste trabalho, o objetivo é desenvolver uma abordagem híbrida de detecção de clones de código em Linhas de Produtos de Software Orientada a Características. Para realizar a análise dessa abordagem, serão abrangidos os contextos de análise de desempenho e cobertura. Logo, para atigir os objetivos propostos, a metodologia a ser utilizada nesse trabalho segue as seguintes etapas:

1. **Estado da Arte - Clonagem de Código:** nessa etapa foi realizada a revisão da literatura, abordando como a detecção de clonagem de código ocorre no paradigma de orientação a objetos e verificar como essa detecção é feita no paradigma de orientação a características. Para tal, foi feito um estudo exploratório utilizando a técnica de mapeamento sistemático da literatura. Esse processo utilizou artigos científicos obtidos em pesquisas realizadas em seis bibliotecas digitais para responder a quatro questões de pesquisa propostas, de acordo com os objetivos iniciais pretendidos, isto e, o estado da arte da clonagem de código. Assim, os resultados obtidos foram estruturados, descritos e analisados no decorrer da Seção 4 deste trabalho. Portanto, essa etapa está **concluída**.

2. **Estado da Arte - Linha de Produtos de Software (LPS):** nessa etapa foi realizada a revisão da literatura para identificar o estado da arte de Linhas de Produtos de Software. Nesse contexto, foi abordado características de produtos, variabilidades, desenvolvimento, evolução, e a clonagem de código em LPS.

3. **Proposta de Abordagem (Estado: em andamento) :** essa etapa concentrou-se na definição e elaboração de uma abordagem para detectar clones de código em Linhas de Produtos de Software orientadas a características. Após a pesquisa de abordagens de detecção de clones existentes, como resultados obtidos da etapa um (Clonagem de código), foi definida a proposta de uma abordagem híbrida de detecção de clones em LPS, composta pelos processos de análise de sequência de chamadas de métodos e análise estrutural de instruções. Tal abordagem contará com um apoio computacional a ser desenvolvido (*plug-in* (etapa 4) para a plataforma IDE Eclipse). Para essa abordagem, será utilizado *Program Dependence Graph* (PDG), uma representação intermediária do programa que utiliza notação de grafo para representar a dependência de vários objetos entre si. Após criar o grafo que representa a sequência de chamadas de métodos, é possível utilizar subgrafos a serem comparados para encontrar sequências de chamadas em comum. Para cada par de subgrafo comparado, a análise de cada chamada de método será feita pela verificação da assinatura (A) do método (M). Define-se assinatura de um método (A(M)) por:

**A(M)** = visibilidade + retorno + identificador + (tipo(s) + parâmetro(s)).

Logo, se A(M1) = A(M2), M1 e M2 são possíveis clones. Porém, alterações no identificador do método dificulta o tipo de análise proposta. Para resolver tal dificuldade, será utilizada a análise estrutural das instruções (atribuição, repetição, decisão e outros) dentro de cada método, onde é verificada a similaridade com base no grafo de instruções extraído. Nesse contexto, propriedades relacionadas a herança, isto é, sobrecarga e sobrescrita, não são considerados clones por possuírem conceitos específicos que não os classificam como tal. O estado atual dessa etapa é **em andamento**

4. ***Plug-in*:** nessa etapa, o objetivo é desenvolver um apoio computacional semi-automatizado para auxiliar a análise da abordagem proposta. Para isso, será utilizada a linguagem de programação Java para a implementação de um *plug-in* para a plataforma IDE Eclipse, que detecta clones em LPS. Logo, serão implementados métodos de funcionais para a abordagem.

5. **Análise dos Resultados:** nessa etapa, será feita a análise da abordagem proposta, utilizando os dados de saída da ferramenta para avaliação junto a outras ferramentas/abordagens. Tal análise será feita com base na comparação de desempenho e cobertura de detecção no código fonte.

# 3 LINHAS DE PRODUTOS DE SOFTWARE - ESTADO DA ARTE

## 3.1 Considerações iniciais

Uma Linha de Produtos de Software (LPS) é formada por um conjunto de sistemas de softwares que foram desenvolvidos com base em um código comum e buscam atingir um segmento de mercado específico (APEL et al., 2016). A gerência desse recurso é feita por meio de um modelo de árvore, operadores lógicos e restrições que podem ser implementados por tecnologias baseadas em composição (VALE et al., 2015).

Tendo em vista tais considerações, esse capítulo está estruturado da seguinte maneira. Na Seção 2.1 são descritas as considerações iniciais sobre linha de produtos de software e uma visão geral sobre o assunto. Na Seção 2.2 são abordadas características, variabilidades e comunalidades em LPS. A Seção 2.3 aborda a evolução de linha de produtos. A seção 2.4 introduz o ambiente de clonagem de linha de produtos. A seção 2.5 Abordam as considerações finais levantadas no desenvolvimento desse capítulo.

## 3.2 Características e Produtos

O termo Linha de Produtos de Software (LPS) surgiu da necessidade da indústria em promover adaptações a um produto, de acordo com as necessidades particulares de cada cliente (APEL et al., 2016). Isso possibilitou a variabilidade de produtos compondo uma família, gerada a partir de um conjunto de características em comum (LAGUNA; CRESPO, 2013b).

Para formar a base de uma linha de produção, são necessários artefatos e recursos, denominados ativos base (*core assets*), que incluem por exemplo componentes, modelo de domínio, requisitos, especificações, arquitetura e outros.

Quanto à estrutura, uma LPS é formada por engenharia de domínio, engenharia de aplicação e evolução da própria LPS. A engenharia de domínio é res-

ponsável pelo domínio de funcionalidades bem como a criação e manutenção do núcleo da LPS. Se tratando de engenharia de aplicação consiste basicamente no desenvolvimento do produto.

### 3.2.1 Variabilidades

O que diferem os produtos gerados em uma LPS uns dos outros, são as variabilidades acrescentadas a cada um deles. Tais variabilidades são responsáveis por acrescentar características específicas a cada produto.

A implementação de uma linha de produtos segue uma estrutura de árvore que representa o modelo de características (*feature model*). Os nós obrigatórios (obrigatoriedades) são acrescidos de características (*features*) variáveis (variabilidades) e as várias combinações possíveis dessas, possibilita a geração de vários produtos.

Variabilidades podem ser obrigatórias, opcionais ou alternativas. Nesse contexto, variabilidades obrigatórias estão presentes em todos os produtos da LPS. As opcionais pertencem a alguns produtos e as alternativas, dado um conjunto de características, possibilita a escolha de somente uma (COLANZI, 2014).

*Seção ainda sem resultados/em pesquisa como parte do cronograma com o orientador.*

## 3.3 Processo de Desenvolvimento

## 3.4 Evolução

*Seção ainda sem resultados/em pesquisa como parte do cronograma com o orientador.*

## 3.5 LPS Orientada a Características

## 3.6 Clonagem de código

Linhas de Produtos de Software foram propostas como uma abordagem melhor estruturada para a reutilização de artefatos de código-fonte entre um conjunto de sistemas com características semelhantes, conhecidos como família .

*Seção ainda sem resultados/em pesquisa como parte do cronograma com o orientador.*

## 3.7 Considerações finais

Este capítulo descreve o cenário e configurações de Linhas de Produtos de Software. As informações obtidas até aqui serão completadas e continuado o capítulo como próximo passo para o texto de pré projeto.

# 4 CLONAGEM DE CÓDIGO - ESTADO DA ARTE

## 4.1 Considerações iniciais

Com a crescente demanda de softwares para suprir às necessidades geradas pelo avanço da tecnologia, a atenção ao modo como esses softwares são desenvolvidos é essencial. Clones de código-fonte são fragmentos de código replicados e a inserção clones em softwares ocorre por diversos motivos, entre eles estão novas funcionalidades, copiar e colar, e reutilização de código para uma determinada finalidade(SOLANKI; KUMARI, 2016).. Nesse sentido, diversas ferramentas e abordagens foram desenvolvidas para detectar clones de código-fonte em softwares visando, principalmente, impedir que mudanças inconsistentes possam propagar erros e bugs, bem como auxiliar em manutenções necessárias.

De modo a explorar tal contexto de detecção de clones, esse capítulo descreve um estudo empírico sobre clonagem de código em softwares. Assim sendo, a estrutura deste capítulo está da seguinte forma. A Seção 3.2 está a introdução do assunto acerca de ferramentas e abordagens existentes de detecção de clones. Na Seção 3.3 são abordados os tipos de clones bem como as técnicas comumente utilizadas para detectar clones em softwares. Seção 3.4 conta com a descrição de como foi feito o processo de mapeamento sistemático e o que ele objetiva responder. Na Seção 3.5 contém os resultados obtidos e a discussão acerca destes. Na Seção 3.6 está a discussão. Na Seção 3.7 consta os trabalhos relacionados. Na Seção 3.8 estão as ameaças a validade do trabalho. E, por último, na Seção 3.8 estão as considerações finais.

## 4.2 INTRODUÇÃO

A clonagem de código é uma prática considerada comum e é introduzida por desenvolvedores principalmente na etapa de manutenção (FORDOS; TOTH, 2016)(DUALA-EKOKO; ROBILLARD, 2007). Clones surgem por vários moti-

vos, entre eles estão práticas de *copy and paste*, inserção de funções semelhantes e reutilização de código *ad-hoc* por programadores. A detecção de clones de código é essencial para prevenção da propagação de erros, correção de *bugs*, manutenção e gerencia de sistemas de software (PETERSEN R. FELDT; MATTSSON, 2008)(SOLANKI; KUMARI, 2016).

Clones podem ser idênticos ou possuir similaridades que também os classificam como tal. A similaridade pode ser classificada em dois grupos: i) sintática; e ii) semântica. Isso possibilitou a adoção da classificação de clones em quatro tipos, sendo os Tipos 1, 2 e 3 pertencentes ao primeiro grupo e o Tipo 4 pertencente ao segundo grupo (YUKI; HIGO; KUSUMOTO, 2017). Existem ferramentas desenvolvidas com o propósito de detectar clones em código. Essas ferramentas diferem principalmente em como fragmentos de código serão representados para serem comparados a outros fragmentos de código. Em geral, essa representação é por meio de *String*, *Token*, AST (*Abstract Syntax Tree*), PDG (*Program Dependence Graph*), Medidas e formas híbridas para chegar a uma abordagem final (SCHUGERL, 2011). Tal detecção pode ser feita analisando blocos de código, métodos, classes ou outra medida que determine, por exemplo, o tamanho do fragmento a ser analisado.

Diante da variedade de ferramentas, de técnicas, de métodos e de outras formas para detectar código clonado em sistemas de software, foi realizado um estudo exploratório, cujo resultado final obtido foi a identificação de 128 artigos que apresentam pesquisas sobre clonagem de código (ANEXO). Nesse estudo, foi utilizada a técnica Mapeamento Sistemático da Literatura (MSL), tendo em vista que os requisitos de pesquisa são menos rigorosos em relação à técnica Revisão Sistemática da Literatura (RSL), sendo o seu interesse em tendências de pesquisa (KITCHENHAM; BRERETON, 2010). Em MSL, não precisa realizar avaliação de qualidade, pois o seu resultado é um inventário de artigos sobre a área temática, mapeados para uma classificação (WIERINGA N. MAIDEN; ROLLAND,

2006), sendo uma visão geral do escopo da área que permite descobrir lacunas e tendências de pesquisa (PETERSEN R. FELDT; MATTSSON, 2008).

O restante desse trabalho está estruturado da seguinte maneira. Na Seção 2, o *background* sobre clonagem de códigos, tipos de clonagem e técnicas de detecção de clones é apresentado. Na Seção 3, é detalhado o processo de planejamento do estudo exploratório que inclui questões e estratégias de pesquisa e critérios de inclusão e de exclusão. Na Seção 4, são apresentados os resultados obtidos no estudo exploratório. Na Seção 5, são discutidos esses resultados. Na Seção 6, estão resumidos alguns trabalhos relacionados. Na Seção 7, estão descritas ameaças a validade. Na Seção 8, são apresentadas conclusões e sugestões de trabalhos futuros.

## 4.3 *Background*

A prática de "copiar e colar" trechos de código em outras partes do código é chamada de clonagem de código. Essa clonagem acontece em decorrência de vários fatores como reúso de código, necessidade de implementação de funções semelhantes, manutenção de sistemas legados ou acidentalmente por parte dos programadores (SOLANKI; KUMARI, 2016). À medida que o código aumenta de tamanho, a frequência com que a clonagem pode ocorrer tende a aumentar. Entretanto, estudos mostram que nem sempre o tamanho do código influencia na ocorrência de clones (TORRES; JUNIOR; FARIAS, 2017). Em sua maioria, tais ocorrências são relatadas em etapas de manutenção. Códigos clonados são classificados em quatro tipos acerca de sua similaridade sintática e semântica (GAUTAM; SAINI, 2016)(SOLANKI; KUMARI, 2016):

- **Tipo 1 (*exact clones*)**. Fragmentos de código idênticos com variações de espaço, guias, *layout* e comentários;

- **Tipo 2 (*renamed/parameterized clones*)**. Fragmentos de código com estrutura/sintaxe similar a outro(s) fragmento(s) de código, acrescidos de alterações em identificadores, literais, tipos, *layout* e comentários;

- **Tipo 3 (*near-miss clones*)**. Fragmentos de código acrescidos de declarações, inserções/eliminações e alterações em identificadores, literais, tipos e *layout*;

- **Tipo 4 (*semantic clones*)**. Fragmentos de código funcionalmente semelhantes, mas não possuem semelhança textual. Ou seja, são funções do código original implementadas com sintaxe diferente.

Como exemplo desses tipos de clonagem, pode-se analisar um trecho de código hipotético (Código 1) e outros 4 trechos clonados dele, que apresentam modificações que caracterizam esses tipos. No trecho do Código 2, nas linhas 1 a 4, estão realçadas (cor de fundo diferente) informações que caracterizam o **Tipo 1** de clonagem, sendo espaçamento, adição de comentário, espaçamento e *layout*, respectivamente.

```
1     ResultSet res = sist.executeQuery(query);
2     boolean ok;
3     for(ok = rs.first(); ok; ok = res.next()){
4     results.add(new Project(res.getInt(2)));
5     }
6     return results;
```

Código 4.1 – Trecho original

```
1     ResultSet res=sist.executeQuery(query);
2     boolean ok; //Controlador
3     for(ok = rs.first(); ok; ok=res.next())\{
4     results.add(new Project(res.getInt(2)));
5     return results;
```

Código 4.2 – Tipo 1 de Clonagem

No Código 3, as linhas 1, 2, 3, 5 e 7 possuem alterações em identificadores de variáveis e, na linha 4, há alteração no *layout*. Essas alterações caracterizam clonagem de código e classifica o Código 3 como **Tipo 2** de clonagem. No Código 4, o **Tipo 3** de clonagem pode ser identificado na linha 4, onde ocorre a eliminação da chamada do método `res.getInt(2)`. No Código 5, é possível verificar a mudança semântica no contexto de execução das instruções, onde o comando de repetição `for` foi subtituido pelo comando de repetição `do .. while`, sem alterações no resultado final.

```
1     ResultSet r = sist.executeQuery(query);
2     boolean ver;
3     for(ver = r.first(); ver; ver = r.next())
4     {
5     ok.add(new Project(res.getInt(2)));
6     }
7     return ok;
```

Código 4.3 – Tipo 2 de Clonagem

```
1     ResultSet res = sist.executeQuery(query);
2     boolean ok;
3     for(ok = rs.first(); ok; ok = res.next()){
4     results.add(new Project( );
5     }
6     return results;
```

Código 4.4 – Tipo 3 de Clonagem

```
1     ResultSet res = sist.executeQuery(query);
2     boolean ok;
3     if(!res.first());
4     return results;
5     do{
6     results.add(new Project(res.getInt(2)));
7     } while(res.next());
8     return results;
```

Código 4.5 – Tipo 4 de Clonagem

Para a classificação dos tipos de clonagem ser feita de maneira adequada, técnicas de detecção foram elaboradas, nas quais a variação da representação do

trecho de código a ser analisado é a principal diferença entre as técnicas mais comumente utilizadas (RATTAN; KAUR, 2016)(JANG; BRUMLEY, 2009):

- **Baseada em texto (*text-based*)**. Técnica que consiste na combinação de textos e de *strings* para encontrar candidatos a clones que diferem por meio de comentários e no *layout* do(s) fragmento(s) de código a ser(em) analisado(s). Utilizando essa técnica, é possivel identificar clones do **Tipo 1**, cujos exemplos de abordagens de detecção são extração de texto e comparações mais distantes;

- **Baseada em símbolos (*token-based*)**. O processo de análise léxica tem como resultado a produção de uma sequência de *tokens*. Esses *tokens* são utilizados como parâmetros para métodos de busca para encontrar possíveis clones. Essa técnica pode ser utilizada para detectar clones do **Tipo 2**;

- **Baseada em árvore (*tree-based*)**. *Abstract Syntax Tree* (AST) fornece uma abstração da análise sintática em forma de árvore. A AST é percorrida por subárvores semelhantes até encontrar possíveis clones sintáticos contidos nessas subárvores, usando alguma técninca de correspondência de árvores. AST é utilizada na detecção, principalmente, de clones do **Tipo 3**;

- **Baseada em grafo (*graph-based*)**. *Graphy Dependence Program* (PDG) é um grafo direcionado e funciona como abstração semântica. Com a utilização do isomorfismo de um subgráfico obtido de um sistema de software, é possível encontrar subgráfos semelhantes, sendo classificados como clones. Também, é capaz de preservar a semântica original do código, sendo comumente utilizada para encontrar clones do **Tipo 4**;

- **Baseada em híbrido (*hybrid-based*)**. Essa técnica consiste em combinar as técnicas anteriores, cujo objetivo é criar e/ou melhorar a detecção dos tipos de clone existentes.

Tais técnicas são implementadas de diversas formas. Para cada contexto de implementação, a(s) medida(s) a ser(em) utilizada(s) para comparação depende(m) da finalidade de utilização. Tamanho de fragmento, escopo de detecção, granularidade e escalabilidade são exemplos de medidas a serem utilizadas na implementação de ferramentas. Essas ferramentas detectam clones em várias linguagens e paradigmas, por exemplo, a ferramenta CCFinder (KAMIYA; KUSUMOTO; INOUE, 2002). Em sua maioria, são implementadas para detecção em linguagens orientada a objetos (e.g., as linguagens de programação JAVA e C++) (KAMIYA; KUSUMOTO; INOUE, 2002)(LIN et al., 2014)(YUAN; GUO, 2011) ou para o paradigma procedural (e.g., a linguagem de programação C. Entretanto, a detecção de clones não se limita a tais paradigmas/linguagens.

## 4.4 Planejamento do Estudo Exploratório

Nesta seção, é descrito como o estudo exploratório, utilizando a técnica Mapeamento Sistemático da Literatura, foi planejado, incluindo a descrição das questões de pesquisa, escopo de pesquisa , estratégia de busca e critérios de classificação.

### 4.4.1 Questões de pesquisa

Nesse trabalho, o objetivo é realizar o estudo exploratório de pesquisas existentes, que abordam a detecção de códigos clonados em sistemas de software. É importante considerar a abrangência de informações que tal tema retorna e possibilitar que, com a definição e o estabelecimento do tipo de informações coletadas, forneçam às pesquisas futuras o estado da arte acerca da clonagem de código. Para tanto, foram coletadas informações sobre técnicas, processos, métodos, procedimentos, metodologias e ferramentas disponíveis na literatura que detectam código clonado em sistemas de software. Assim, foram elaboradas e respondidas as se-

guintes questões de pesquisa:

**Q1:** Como é realizada a detecção de clones em código de sistemas de software?

**Justificativa**: Objetivo é mostrar o cenário de como são detectados clones em código. Assim sendo, identificar qual(is) ferramenta(s) é(são) utilizadas para identificar a existência de clones por parte dos programadores.

**Q2:** De que forma é realizada a detecção de clones nas abordagens que propõe técnicas, processos, métodos, procedimentos e metodologias para identificar clones?

**Justificativa**: O objetivo é coletar informações sobre o modo de abordar as formas de detecção de clones comuns às técnicas, processos, métodos, procedimentos e metodologias identificadas.

**Q3:** Quais tipos de clones são abordados nos artigos analisados?

**Justificativa**: O objetivo é identificar quais tipos de clonagem são identificados pelos autores e se existe realmente uma conveção da utilização desse termo, com base nos resultados obtidos.

**Q4:** Quais linguagens e paradigmas de programação são utilizados?

**Justificativa**: O objetivo é identificar as linguagens de programação comumente utilizadas para identificar clones e, com base nisso, quais paradigmas são predominantemente adotados pelos autores para realizar a detecção de clones em código.

As respostas das questões de pesquisa Q3 e Q4 estão diretamente ligadas às respostas obtidas nas questões de pesquisa Q1 e Q2.

### 4.4.2 Estratégia de Pesquisa

Após estabelecer as questões de pesquisa, foi possível definir palavras e/ou termos relevantes para obter resultados importantes e objetivos para o estudo exploratório. Como o foco está no contexto de clonagem de código, o termo "Clone de Código" é relevante. Para compor o contexto, foi necessário identificar os possíveis caminhos para chegar na resposta da questão de pesquisa Q1. Logo, ao decompor em termos tal questão, foram obtidas como resultado as variações das palavras-chave "técnicas", "processos", "métodos", "procedimentos", "metodologias" e "ferramentas" buscando abordar tal contexto. Além disso, "software" foi utilizada como a palavra-chave para a combinação final, pois representa o contexto de estudo. A combinação dessas palavras-chave com os conectores lógicos `AND` e `OR` forma a *string* de busca utilizada:

```
("code clones" OR "cloned code") AND (technique OR process OR
 method OR procedure OR methodology OR tool) AND (detect OR
           detection) AND software
```

Para responder às questões de pesquisa utilizando a *string* de busca, foram selecionadas as bibliotecas digitais ACM [1], EI Compendex[2], IEEE Xplorer[3], Science Direct[4], Scopus[5] e Springer Link[6]. Essas bibliotecas foram escolhidas por

[1] www.acm.org
[2] www.engineeringvillage.com
[3] http://ieeexplore.ieee.org/
[4] www.sciencedirect.com
[5] http://www.scopus.com
[6] www.link.springer.com

suportar (i) pesquisa avançada com utilização de palavras-chave, (ii) filtragem dos resultados por ano e área de publicação, (iii) filtragem por tipo de publicação e (iv) exportação do resultado da consulta em formato `BibTex` ou `Endnote`. O retorno inicial da pesquisa em cada biblioteca digital foi armazenado em bases de dados.

### 4.4.3 Critério de Inclusão/Exclusão

Na Tabela 4.1, é apresentada a quantidade de trabalhos resultantes nas bibliotecas digitais.

[CHEGAR NA UFLA E DESCREVER ISSO AQ]

Cada trabalho presente nas bases de dados passou por inspeções a fim de retirar os trabalhos caracterizados como não artigos (por exemplo, livros, normas e *table of contents*) retornados na busca (Apenas Artigos), o que resultou em bases de dados com trabalhos considerados artigos, os quais foram reunidos em uma base de dados para serem removidos os artigos duplicados (Resultado Final). Nessa remoção, foi considerada a quantidade de palavras-chave, ou seja, o artigo duplicado e armazenado na biblioteca digital com menor quantidade de palavras-chave foi removido. Os artigos restantes foram lidos de modo que o foco da leitura se deu em identificar itens relacionados às questões de pesquisa.

Tabela 4.1 – Quantidade de Artigos

| Bibliotecas Digitais | Resultado Inicial | Apenas Artigos | Resultado Final |
|---|---|---|---|
| ACM | 119 | 82 | 27 |
| EI COMPENDEX | 196 | 61 | 19 |
| IEEE | 135 | 131 | 49 |
| SCIENCE DIRECT | 35 | 34 | 2 |
| SCOPUS | 271 | 71 | 27 |
| SPRINGER LINK | 119 | 102 | 4 |
| Total | 875 | 481 | 128 |

Cabe ressaltar que três pesquisadores (Pesquisador A, Pesquisador B e Pesquisador C) foram envolvidos na obtenção dos artigos e foi realizado o seguinte procedimento:

1. O Pesquisador A executou a *string* de busca nas bibliotecas digitais selecionadas e documentou os resultados no sistema de software JabRef[7];

2. O Pesquisador A verificou e excluiu os trabalhos que não eram artigos e os artigos repetidos (com título, autores e resumo iguais). Na identificação de artigos repetidos, foram mantidos os artigos com palavras-chave que melhor descreviam o artigo;

3. Os artigos encontrados foram avaliados pelo Pesquisador A e pelo Pesquisador B, de maneira individual e separada, quanto ao atendimento aos critérios de inclusão e de exclusão. Essa avaliação foi realizada por meio da leitura do título, do resumo e das palavras-chave. Os artigos, cujas avaliações causaram dúvidas quanto a sua inclusão/exclusão por parte dos pesquisadores, foram incluídos. Os artigos foram documentados em uma lista de artigos incluídos e excluídos com justificativa para sua inclusão ou exclusão;

4. Realizou-se a interseção entre os artigos selecionados pelo Pesquisador A e pelo Pesquisador B, sendo esses artigos documentados (Interseção). Na ocorrência de algum desacordo sobre a inclusão ou a exclusão de artigos, os dois pesquisadores discutiram e resolveram. Em casos que não houve consenso, o artigo foi incluído. Os artigos excluídos foram documentados em uma lista de artigos excluídos com justificativa para sua exclusão;

5. O Pesquisador C avaliou os artigos exluídos e as justificativas de exclisão e os artigos presentes na Interseção. O resultado foi o conjunto de artigos resultantes do estudo exploratório.

[7] http://www.jabref.org/

## 4.5 Resultados

A finalização dos procedimentos e da seleção das informações necessárias, permite uma noção da diversidade de ferramentas e maneiras em comum de detecção de clones nas abordagens que propõe técnicas, processos, métodos , procedimentos e metodologias acerca do estado da arte da detecção de clones em sistemas de software. A detecção de clones é abordada de diversas maneiras e por várias ferramentas que implementam essas abordagens, ambas identificadas nos 128 artigos obtidos após aplicar os critérios de inclusão/exclusão. Nesta seção, são apresentadas a análise das características gerais dos artigos resultantes do estudo exploratório, bem como respostas para as questões de pesquisa levantadas.

### 4.5.1 Características Gerais dos Trabalhos Selecionados

Como mencionado na seção 3, foram utilizadas 6 bibliotecas digitais de artigos científicos para a realização do estudo exploratório. Na Tabela 4.1, é apresentada a quantidade final de 128 artigos como resultado para análise, sendo 27 artigos (21,09%) na ACM, 19 artigos (14,84%) na EI COMPENDEX, 49 artigos (38,28%) na IEEE, 2 artigos (1,56%) na SCIENCE DIRECT, 27 artigos (21,09%) na SCOPUS e 4 artigos (3,12%) na SPRINGER LINK. Foi considerado o período que engloba, aproximadamente, 18 anos de publicações (de 2001 a 2018) como mostra o gráfico da figura 4.1. A quantidade de artigos publicados acerca do tema "Detecção de Código Clonado", aparentemente, tende a crescer (gráfico da figura 4.1) com o decorrer dos anos. A maior concentração dessas publicações foram encontradas no ano de 2017, com o total de 17 artigos.

Sobre a quantidade de pesquisadores, foram identificados 397 pesquisadores no tema "Detecção de Clonagem de Código", sendo os mais atuantes os pesquisadores `S. Kusumoto` com 7 artigos, `C. K. Roy` com 6 artigos, `Y. Higo` com 5 artigos e R. K. Tekchandani com 5 artigos. Cabe ressaltar a presença de dois grupos de pesquisadores que têm publicado juntos. Um desses grupos é com-

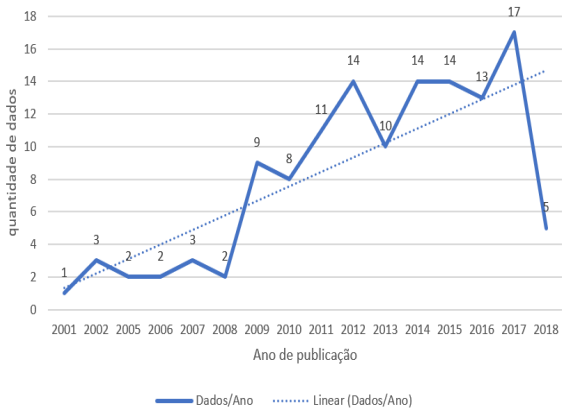


Figura 4.1 – Quantidade ao longo dos Anos

posto por cinco pesquisadores Nguyen, H. A., Nguyen, T. N., Nguyen, T. T., Pham, N. H. e Al-Kofahi, J. M. com 4 artigos (ClemanX: Incremental clone detection tool for evolving software, Clone-Aware Configuratioan Management, Scalable and incremental clone detection for evolving software e Accurate and Efficient Structural Characteristic Feature Extraction for Clone Detection. Outro grupo é composto por dois pesquisadores Kanmani, S. e Kodhai, E. com 4 artigos (Detection of Type-1 and Type-2 Code Clones Using Textual Analysis and Metrics, Method-level code clone detection through LWH (Light Weight Hybrid) approach, Method-level incremental code clone detection using hybrid approach e Method-level code clone detection for Java through hybrid approach).

Os 128 artigos selecionados no estudo exploratório foram publicados em 84 veículos distintos de divulgação científica, sejam *journals*, *workshops*, conferências e simpósios. Os quatro veículos com mais quantidade de artigos são Symposium on Applied Computing (SP) com 5 artigos publicados (5,85%), Asia-Pacific Software Engineering Conference (APSEC) com 5 artigos publicados (5,85%), International Conference on Software Engineering

(ICSE) com 6 artigos publicados (7,14%) e International Workshop on Software Clones (IWSC) com 8 artigos publicados (9,52%) (Tabela 4.2).

Tabela 4.2 – Principais Veículos de Publicação

| Veículos de Divulgação Científica | Qtde | Artigos |
|---|---|---|
| Asia-Pacific Software Engineering Conference (APSEC) | 5 | A18, A07, A28, A97, A100 |
| International Conference on Software Engineering (ICSE) | 6 | A13, A33, A43, A65, A80, A11 |
| International Workshop on Software Clones (IWSC) | 8 | A20, A27, A44, A73, A82, A87, A96, A98 |
| Symposium on Applied Computing (SP) | 5 | A19, A26, A45, A78, A49 |

### 4.5.2 Resposta às Questões de Pesquisa

Em resposta a questão de pesquisa Q1

| Como é realizada a detecção de clones em código de sistemas de software? |
|---|

a detecção de clones é feita por meio de abordagens e feramentas que implementam essas abordagens. Foram identificadas 52 ferramentas (Tabela 4.3). Cada ferramenta utiliza uma representação de fragmentos a serem analisados/considerados clone (Tabela 4.3), sendo *Token*, Árvore (ou AsT) e Grafo as representações que possuem mais quantidade de ferramentas.

Em resposta a questão de pesquisa Q2

| De que forma é realizada a detecção de clone nas abordagens que propõe técnicas, processos, métodos, procedimentos e metodologias para identificar clones? |
|---|

foram identificadas várias formas de detectar clones. Tais formas foram agrupadas em 26 técnicas (Tabela 4.4) que se baseiam na representação do código para detectar clones, sendo a técnica baseada em Árvore (AST e Distância) a que possui mais quantidade de artigos que a utiliza. Ao todo, 124 artigos apresentaram a detecção de clones baseadas nos tipos listados na tabela 4.4 (os artigos A91, A116, A119 e A123 não apresentaram). Mesmo que possuam algum tipo de interface implementada, algoritmos e outros, 71 trabalhos (55,42%) não possuem nome para as abordagens descritas pelo próprios autores para referenciá-las. Por esse motivo, os resultados são em relação ao tipo de representação (baseada em) utilizada para detecção de clones. Não foi identificada a técnica abordada em três artigos (A12, A88, A123).

Em resposta a questão de pesquisa Q3,

| Quais tipos de clones são abordados nos artigos analisados? |
|---|

quanto aos tipos de clones, todos foram listados e relacionados na Tabela 4.5 e, quando o tipo não estava explicitamente identificado, havia referência somente à detecção de clones semânticos ou sintáticos. Foram identificadas 155 ocorrências de abordagens de detecção de clones, sendo 37 artigos detectam clones do Tipo 1, 37 artigos detectam clones do Tipo 2, 44 artigos detectam clones do Tipo 3, 11 artigos detectam clones do Tipo 4, 14 artigos detectam clones semânticos e 12 artigos detectam clones sintáticos.

Em resposta a pergunta Q4,

| Quais linguagens e paradigmas de programação são utilizados? |
|---|

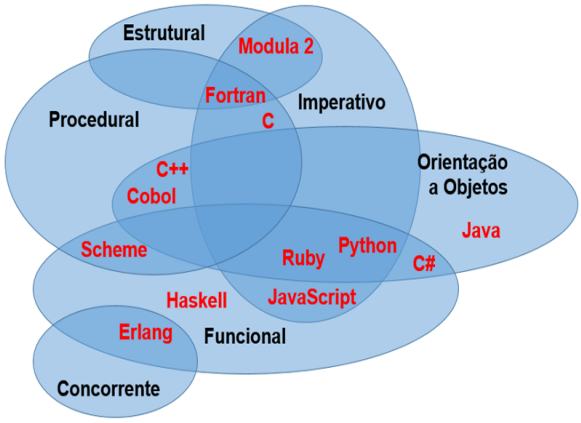


Figura 4.2 – Paradigmas de Programação Identificados

foram identificadas 13 linguagens de programação (Tabela 4.6). A quantidade de vezes em que elas apareceram nos artigos está distribuidas em Python (1), JavaScript (1), C# (12), Java (62), C++ (21), C (43), Modula 2 (1), Scheme (1) , Cobol (3), Erlang (3), Haskell (1), Ruby (2) e Fortran (1). Essas linguagens pertencem a 6 paradigmas de programação (Figura 4.2): i) estrutural; ii) imperativo; iii) funcional; iv) orientado a objetos; v) procedural; e vi) concorrente.

## 4.6 Discussão

Clones de código são detectados por abordagens e ferramentas, sendo implementadas de maneiras diferentes. Porém, é possível identificar algumas maneiras de compará-los, por exemplo, o modo em que o código a ser dito “clone” é representado para identificação e os tipos de clones identificados. Os trabalhos selecionados no estudo exploratório possuem intervalo de publicação de 18 anos (de 2001 a 2018), inclusive.

Quanto à quantidade de veículos de divulgação científica identificados (84 *journals*, *workshops*, conferências e simpósios), pode ser considerada alta e mostra a diversidade de contextos/áreas da Engenharia de Software, onde são abordadas a detecção de clones, por exemplo, segurança, inteligência artificial, engenharia

reversa, *data mining* e computação aplicada. É observável que a maior concentração de artigos foram apresentados no `International Workshop on Software Clones (IWSC)`, cujo objetivo é "reunir pesquisadores e profissionais para avaliar o estado atual da pesquisa, discutir problemas comuns e direções emergentes (como a detecção de clones em modelos de software, análise de clones em reengenharia para reutilização, análise de clones em evolução de software e detecção de clones em direitos autorais e plágio)"[8].

Foram reunidas abordagens e suas bases de representação similar em tipos diferentes. Alguns desses tipos são comumente encontrados na literatura como árvores e grafos, cujos resultados foram obtidos em maior quantidade, e outras são representações relativamente novas, pois apresentam uma abordagem com uma base de identificação completamente diferente das usuais, por exemplo, *wavelets*, métodos formais e análise concólica, as quais apresentam bom desempenho e escalabilidade na detecção de clones complexos comparados a outras abordagens. Entretanto, algumas dessas técnincas não são explicadas adequadamente pelos autores e/ou passaram por poucas avaliações.

As ferramentas facilitam a identificação de clones para os programadores. Elas comumente oferecem interfaces para visualização dos clones identificados, sejam elas gráficas ou não. Além disso, conseguem identificar clones de vários tipos e isso os beneficia, pois identificá-los manualmente é cansativo, dispendioso e propenso a falhas. Nem todas essas ferramentas de detecção são explícitas quanto a isso ou garantem encontrar um tipo de clone específico, o que acaba por generalizar termos como clones semânticos e sintáticos. Tal convenção pode gerar confusão quanto aos tipos identificados, pois não existe uma exatidão dos tipos que abrange cada termo.

Quanto aos tipos identificados nota-se divergência entre os autores quanto à utilização ou não da classificação usual que ditam 4 tipos de clones relacionados

[8] https://iwsc2018.github.io/

às similaridades semântica e sintática. Com isso, alguns autores preferem utilizar a nomenclatura convencional de separação, sendo os tipos de clones diferidos por similaridades sintáticas e semânticas.

Mesmo com a variedade de linguagens de programação existentes, é perceptível que, em clonagem de código, destaca-se a utilização de programas desenvolvidos em JAVA (40%), C (27,92%) e C++ (13,63%). Tais valores podem estar relacionados à popularidade de utilização da linguagem em diversas outras áreas de pesquisa. Isso também revela o porquê da maioria das abordagens existentes empregarem suas técnicas em ambientes voltados ao desenvolvimento utilizando o paradigma orientado a objetos e procedural.

Também, pode-se perceber, no decorrer da leitura, que os artigos são comumente aplicados em contextos industriais e acadêmicos. No contexto industrial, a detecção de clones em sistemas de software trata principalmente da realização de testes nesses sistemas, visando às melhorias de manutenção. No contexto acadêmico, relativamente maior que o industrial, são destacadas pesquisas buscando o desenvolvimento de ferramentas que proporcionam mais escalabilidade e desempenho. Outros aspectos comumente abordados juntamente com a detecção de clones são (i) refatoração de códigos clonados (com o objetivo de manutenção) e (ii) necessidade/propensão a gerência de clones.

Relatados, em sua maioria, no conteúdo dos artigos analisados, clones de código têm impacto direto na manutenção de sistemas de software. A detecção por meio de ferramentas e abordagens busca facilitar a sua identificação pelos programadores, a fim de reduzir propagações de erros disseminados por clones não identificados, bem como facilitar inserções/alterações/remoções de funções em sistemas de software.

## 4.7 Trabalhos Relacionados

Em um trabalho ROY; CORDY, realizam um estudo do estado da arte, criando um *survey* sobre detecção de clones. Assim sendo, em parte desse trabalho, a literatura sobre clones de código é abordada, bem como a identificação de ferramentas e de técnicas de detecção de clones, descrevendo o processo. Neste trabalho, foi relatado quais ferramentas e abordagens existem, utilizando investigação da literatura bem como características relacionadas à linguagem e ao paradigma de programação.

Em outro trabalho GAUTAM; SAINI, são classificados técnicas de detecção de clones e tipos de clones com pesquisas que abrangem, por típo de técnincas, períodos diferentes de publicação, em até, no máximo, o ano de 2012. Neste trabalho, são identificadas essas abordagens no período de 2001 a 2018.

Em outro trabalho KAPDAN; AKTAS; YIGIT, é tratado o problema de identificação de clones estruturais, considerando abordagens de detecção de clones baseadas em medidas, utilizando pesquisa sobre o estado da arte. Neste trabalho, são identificadas ferramentas/abordagens que utilizam medidas, mas não especificá-las.

## 4.8 Ameaças a validade

As ameaças a validade deste estudo exploratório são importantes para determinar o nível de confiança dos dados obtidos.

***String* de busca**. A *string* de busca pode conter falhas ou estar incompleta para o contexto, o que pode gerar alteração no resultado obtido nas buscas nas bibliotecas digitais utilizadas.

**Obtenção de material**. A *string* de busca pode conter falhas, a falta de termos ou excesso deles pode comprometer a qualidade dos dados.

**Conteúdo analisado**. Vários trabalhos analisados não fundamentam completamente (ou nenhuma vez) como/com base em que são as técnicas de detecção utilizadas bem como os Tipos de clones identificados pela ferramenta/abordagem criada. Dados acerca da linguagem e paradigma de programação utilizada também é uma lacuna a ser preenchida com informações não contidas nos trabalhos selecionados.

## 4.9 Considerações Finais

Clones de código são trechos de código idênticos ou similares que têm alta propensão à propagação de erros, *bugs* e a geração de inconsistências em sistemas de software. Se um trecho é modificado por algum motivo e seu clone não, o processo de manutenção tende a tornar-se mais trabalhoso e massante, principalmente, tratando-se de sistemas de software mais complexos. Logo, ferramentas e abordagens relacionadas à detecção de clones são construídas visando minimizar esses problemas e auxiliar os programadores.

Esse capítulo do trabalho apresentou um estudo exploratório utilizando a técnica Mapeamento Sistemático de Literatura sobre a detecção de códigos clonados, no nível de código, em sistemas de software. Foram analisados e coletados dados de 128 artigos relacionados ao tema para responder as questões de pesquisa. Esse resultado foi obtido por um processo detalhado de investigação/exploratório descrito na Seção 3.

Algumas informações foram identificadas a respeito das características gerais identificadas nos artigos selecionados. Em relação às bibliotecas digitais, em ordem descrescente, a quantidade de artigos analisados estão concentradas na IEEE (38,28%), ACM (21,09%), SCOPUS (21,09%), EI COMPENDEX (14,84%), SPRINGER LINK(3,12%) e SCIENCE DIRECT (1,56%). Com relação ao ano de publicação, foi abrangido o período de 2001 a 2018, onde a maior concentração de publicações foi no ano de 2017. Quanto aos autores, quatro autores principais

destacaram-se com publicações superiores/iguais a cinco artigos. Além disso, foram identificados dois grupos de pesquisadores que têm publicado em conjunto; um grupo com cinco pesquisadores que publicaram quatro artigos juntos e outro grupo com dois pesquisadores que publicaram quatro artigos juntos. Também, foi contabilizado ao todo 84 veículos de divulgação de trabalhos científicos diferentes, categorizando a diversificação de aplicação da detecção de clones.

Como resposta à questão de pesquisa Q1, como resultado foram encontradas 52 ferramentas implementadas baseadas em 12 tipos diferentes. Para a questão de pesquisa Q2, o total de técnicas utilizadas para encontrar cada tipo de clone quantificam 26 modos diferentes. Foram encontrados os quatro tipos de clone ao final do estudo das ferramentas e das abordagens analisadas, respondendo a questão de pesquisa Q3. Para a questão de pesquisa Q4, foram identificadas 13 linguagens de programação diferentes e 6 paradigmas.

Como trabalhos futuros, é sugerida a pesquisa sobre as ferramentas desenvolvidas para gerenciar a detecção de clones, visto que pode facilitar a utilização pelos programadores. Além disso, pesquisar as medidas utilizadas em cada abordagem para melhor compreendê-las.

Tabela 4.3 – Ferramentas Identificadas

| **Ferramenta** | **Baseada em** | **Referência** |
|---|---|---|
| CCCD | Análise Concólica | A72 |
| CLORIFI | Análise Concólica | A57 |
| Asta | Árvore | A23 |
| CCLearner | Árvore | A79 |
| CCR | Árvore | A99 |
| ClemanX | Árvore | A17 |
| Clever | Árvore | A35 |
| CReN | Árvore | A42 |
| Deckard | Árvore | A11 |
| KLON | Árvore | A121 |
| LICCA | Árvore | A4 |
| RefactoreRL | Árvore | A58 |
| MultiDup | Árvore e Medidas | A1 |
| CloneDR | AST | A75 |
| SHAPE | Extração de Procedimento Amorfo | A22 |
| CCSharp | Grafo | A38 |
| ClemanX | Grafo | A33, A125 |
| CloneWorks | Grafo | A25 |
| CMCD | Grafo | A18 |
| DyCLINK | Grafo | A36 |
| GemScan | Grafo | A125 |
| SCVD | Grafo | A50 |
| Scopio | Grafo e Híbrida | A81 |
| DebCheck | Híbrida | A21 |
| PC Detector | Híbrida | A95 |
| SynTex | Híbrida | A44 |
| Hanni | Macros | A30 |
| HeapAbsCC | Manipulação de *Heap* | A60 |
| gCad | Mapeamento de Função | A103 |
| Clone Miner | Medidas | A94 |
| SDD | Medidas | A6 |
| XIAO | Medidas | A9, A27 |
| JCC | *Pipeline* | A111 |
| CCDemon | Recomendação Interativa | A55 |
| BinClone | *Token* | A89 |
| BOREAS | *Token* | A70 |
| CCFinder | *Token* | A7, A13, A14 |
| CCFinderSW | *Token* | A97 |
| Ctcompare | *Token* | A24 |
| FRISC | *Token* | A63 |
| IDCCD | *Token* | A100 |
| LSC Miner | *Token* | A124 |
| ReDeBug | *Token* | A78 |
| RTF | *Token* | A41 |
| SaCD | *Token* | A62 |
| ScalClone | *Token* | A53 |
| SHINOBI | *Token* | A66 |
| SourcererCC | *Token* | A90 |
| SourcererCC-1 | *Token* | A90 |
| VUDDY | *Token* | A49 |
| Simone | Não Informado | A123 |
| VFDTECT | Não Informado | A88 |
| **Total de Ferramentas** | | 52 |

Tabela 4.4 – Técnincas Detecção de Código Clonado

| Técnica Baseada em | Métodos mais Utilizados | Quantidade | Referência |
|---|---|---|---|
| Texto | Verificação de *string* | 1 | A92 |
| Token | Comparação de *tokens* e distância | 19 | A7, A13, A14, A24, A32, A41, A49, A53, A62, A63, A66, A70, A78, A88, A89, A90, A97, A100, A124 |
| Árvore | AST e Distância | 27 | A1, A4, A11, A17, A23, A28, A31, A34, A35, A37, A39, A42, A45, A47, A48, A58, A61, A67, A73, A75, A79, A83, A93, A99, A112, A121, A125, A126 |
| Grafo | PDG, Vetores e Distância | 14 | A18, A25, A33, A36, A38, A43, A50, A54, A64, A76, A80, A81, A102, A125 |
| Híbrida | AST, PDG e Medidas | 19 | A2, A8, A9, A10, A16, A19, A21, A44, A46, A74, A95, A98, A101, A106, A107, A108, A110, A115, A127 |
| Métrica | Tamanho, Complexidade, Escalabilidade, etc | 19 | A1, A3, A6, A15, A27, A29, A40, A56, A77, A82, A87, A92, A94, A105, A109, A114, A115, A122, A128 |
| Macro | Macros | 1 | A30 |
| Procedimento Amorfo | Procedimentos Amorfos | 1 | A22 |
| Análise Concólica | Teste Simbólico | 3 | A26, A57, A72 |
| Recomendação Interativa | Recomendação Interativa | 1 | A55 |
| Semântica Baseada na Web | Web Semântica | 5 | A5, A20, A84, A85, A86 |
| Processo Hierárquico Analítico | Tomada de Decisões Complexas | 1 | A51 |
| Fechamento Transitivo | Fecho Transitivo | 1 | A52 |
| Análise de Crescimento | Análise de Crescimento | 1 | A59 |
| Manipulação de *Heap* | *Heap* | 1 | A60 |
| Métodos Formais | Formalismo Matemático | 2 | A65, A96 |
| Limiar | Limite | 1 | A68 |
| PALEX | PALEX | 1 | A69 |
| *Wavelets* | Decompor e Descrever ou Representar outra Função/Dados | 1 | A71 |
| Mapeamento de Função | Mapear Funções em Estruturas de Dados | 1 | A103 |
| Aprendizado de Máquina | Aprendizado de Máquina | 1 | A104 |
| *Pipeline* | Ações Paralelas | 1 | A111 |
| Controle de *Statements* | Controle de Fluxo de *Statements* | 1 | A113 |
| Chave de Redução | Elemento Principal de Redução | 1 | A117 |
| Índice | Índice de Referência | 1 | A118 |
| Mineração de Conjunto | Itens Frequentemente Ponderados | 1 | A120 |
| **Total de Técnicas** | | | 26 |

Tabela 4.5 – Tipos de Clones Identificados

| Tipo | Quantidade | Referência |
|---|---|---|
| 1 | 37 | A6, A8, A14, A16, A18, A24, A26, A30, A32, A34, A38, A39, A45, A46, A49, A59, A62, A71, A77, A79, A81, A82, A92, A96, A97, A98, A99, A103, A105, A106, A107, A108, A109, A113, A118, A120. |
| 2 | 37 | A6, A8, A14, A18, A24, A26, A31, A34, A38, A39, A43, A45, A46, A49, A53, A59, A62, A71, A77, A79, A81, A82, A83, A91, A92, A96, A97, A98, A99, A103, A105, A106, A107, A108, A113, A118, A120 |
| 3 | 44 | A1, A6, A8, A2, A8, A14, A16, A18, A19, A22, A25, A26, A29, A34, A37, A38, A39, A41, A42, A43, A45, A46, A47, A63, A68, A70, A71, A73, A74, A75, A79, A81, A82, A85, A90, A94, A97, A99, A03, A105, A106, A107, A113, A117, A120 |
| 4 | 11 | A2, A14, A20, A26, A38, A39, A46, A72, A74, A79, A97, A113 |
| Semântico | 14 | A4, A11, A28, A80, A84, A86, A87, A93, A95, A102, A112, A119, A124, A126 |
| Sintático | 12 | A10, A60, A69, A78, A83, A100, A101, A110, A115, A116, A21, A127 |
| **Total** | 155 | |

Tabela 4.6 – Linguagens de Programação Identificadas

| **Linguagens de Programação** | **Qtde** | **Artigos** |
|---|---|---|
| Python | 1 | A73 |
| JavaScript | 1 | A4 |
| C# | 12 | A1, A9, A19, A23, A24, A27, A29, A31, 45, A59, A91, A105 |
| Java | 62 | A2, A4, A7, A8, A10, A11, A15, A18, A19, A23, A25, A26, A28, A29, A32, A33, A35, A36, A37, A39, A42, A43, A44, A45, A47, A52, A55, A57, A62, A64, A65, A68, A69, A70, A71, A74, A75, A79, A82, A84, A85, A91, A92, A94, A96, A98, A99, A100, A101, A106, A107, A108, A109, A110, A111, A112, A113, A117, A118, A119, A120, A126 |
| C++ | 21 | A3, A7, A8, A9, A15, A16, A27, A31, A40, A49, A50, A52, A62, A66, A57, A78, A80, A81, A94, A110, A113 |
| C | 43 | A3, A4, A7, A9, A11, A15, A19, A21, A22, A25, A26, A27, A29, A31, A34, A37, A38, A40, A45, A49, A50, A52, A60, A62, A63, A66, A69, A75, A76, A77, A78, A80, A81, A83, A85, A92, A104, A106, A108, A113, A116, A118, A121 |
| Modula 2 | 1 | A4 |
| Scheme | 1 | A4 |
| Cobol | 3 | A7, A15, A30 |
| Erlang | 3 | A46, A58, A61 |
| Haskell | 1 | A48 |
| Ruby | 2 | A69, A91 |
| Fortran | 1 | A121 |
| **Número total de linguagens** | | 13 |

## ANEXO - ARTIGOS IDENTIFICADOS NO ESTUDO EXPLORATÓRIO

Tabela 4.7 – Artigos Selecionados

| ID | Título | Autores(as) | Fonte |
|---|---|---|---|
| A1 | Detection of near-miss clones using metrics and abstract syntax trees | Vishwachi; Gupta, Sonam | EI COMPEDEX |
| A2 | Semantic Clone Detection Using Machine Learning | Sheneamer, A.; Kalita, J. | IEEE |
| A3 | Identifying clones in the Linux kernel | Casazza, G.; Antoniol, G.; Villano, U.; Merlo, E.; Penta, M. Di | IEEE |
| A4 | LICCA: A tool for cross-language clone detection | Vislavski, T.; RakiÄ‡, G.; Cardozo, N.; Budimac, Z. | IEEE |
| A5 | An approach to identify duplicated web pages | Lucca, G. A. Di; Penta, M. Di; Fasolino, A. R. | IEEE |
| A6 | SDD: High Performance Code Clone Detection System for Large Scale Source Code | Lee, Seunghak; Jeong, Iryoung | ACM |
| A7 | On detection of gapped code clones using gap locations | Ueda, Y.; Kamiya, T.; Kusumoto, S.; Inoue, K. | EI COMPEDEX |
| A8 | Clone detection using time series and dynamic time warping techniques | Abdelkader, M.; Mimoun, M. | IEEE |
| A9 | XIAO: Tuning Code Clones at Hands of Engineers in Practice | Dang, Yingnong; Zhang, Dongmei; Ge, Song; Chu, Chengyun; Qiu, Yingjun; Xie, Tao | ACM |
| A10 | To enhance the code clone detection algorithm by using hybrid approach for detection of code clones | Roopam; Singh, G. | IEEE |
| A11 | DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones | Jiang, L.; Misherghi, G.; Su, Z.; Glondu, S. | IEEE |
| A12 | A Code clone oracle | Krutz, Daniel E.; Le, Wei | EI COMPEDEX |
| A13 | Detecting Differences Across Multiple Instances of Code Clones | Lin, Yun; Xing, Zhenchang; Xue, Yinxing; Liu, Yang; Peng, Xin; Sun, Jun; Zhao, Wenyun | ACM |
| A14 | CCFinder: a multilinguistic token-based code clone detection system for large scale source code | Kamiya, T.; Kusumoto, S.; Inoue, K. | IEEE |
| A15 | Software clone detection using clustering approach | Joshi, Bikash; Budhathoki, Puskar; Woon, Wei Lee; Svetinovic, Davor | EI COMPEDEX |
| A16 | Hybridizing Evolutionary Algorithms and Clustering Algorithms to Find Source-code Clones | Sutton, Andrew; Kagdi, Huzefa; Maletic, Jonathan I.; Volkert, L. Gwenn | ACM |
| A17 | Scalable and incremental clone detection for evolving software | Nguyen, Tung Thanh; Nguyen, Hoan Anh; Al-Kofahi, Jafar M.; Pham, Nam H.; Nguyen, Tien N. | EI COMPEDEX |
| A18 | CMCD: Count Matrix Based Code Clone Detection | Yuan, Y.; Guo, Y. | IEEE |
| A19 | Is Code Cloning in Games Really Different? | Al-omari, Farouq; Roy, Chanchal K. | ACM |
| A20 | Scalable Clone Detection Using Description Logic | Schugerl, Philipp | ACM |
| A21 | DebCheck: Efficient Checking for Open Source Code Clones in Software Systems | Cordy, J. R.; Roy, C. K. | IEEE |
| A22 | SPAPE: A semantic-preserving amorphous procedure extraction method for near-miss clones | Bian, Yixin; Koru, Gunes; Su, Xiaohong; Ma, Peijun | SCIENCE |
| A23 | Clone detection via structural abstraction | Evans, William S.; Fraser, Christopher W.; Ma, Fei | EI COMPEDEX |
| A24 | Ctcompare: Code clone detection using hashed token sequences | Toomey, W. | SCIENCE |
| A25 | CloneWorks: A Fast and Flexible Large-scale Near-miss Clone Detection Tool | Svajlenko, Jeffrey; Roy, Chanchal K. | ACM |
| A26 | Examining the Effectiveness of Using Concolic Analysis to Detect Code Clones | Krutz, Daniel E.; Malachowsky, Samuel A.; Shihab, Emad | ACM |
| A27 | Code Clone Detection Experience at Microsoft | Dang, Yingnong; Ge, Song; Huang, Ray; Zhang, Dongmei | ACM |
| A28 | Instant Code Clone Search | Lee, Mu-Woong; Roh, Jong-Won; Hwang, Seung-won; Kim, Sunghun | ACM |
| A29 | Fast and Flexible Large-scale Clone Detection with CloneWorks | Svajlenko, Jeffrey; Roy, Chanchal K. | ACM |
| A30 | Detection of Code Clones in Software Generators | Lillack, Max; Bucholdt, Christian; Schilling, Daniela | ACM |
| A31 | Phoenix-based Clone Detection Using Suffix Trees | Tairas, Robert; Gray, Jeff | ACM |
| A32 | A Scalable and Accurate Approach Based on Count Matrix for Detecting Code Clones | Yuan, Yang | ACM |
| A33 | ClemanX: Incremental clone detection tool for evolving software | Nguyen, T. T.; Nguyen, H. A.; Pham, N. H.; Al-Kofahi, J. M.; Nguyen, T. N. | IEEE |
| A34 | Tree-pattern-based Duplicate Code Detection | Lee, Hyo-Sub; Doh, Kyung-Goo | ACM |
| A35 | Clone-Aware Configuration Management | Nguyen, T. T.; Nguyen, H. A.; Pham, N. H.; Al-Kofahi, J. M.; Nguyen, T. N. | IEEE |
| A36 | Code Relatives: Detecting Similarly Behaving Software | Su, Fang-Hsiang; Bell, Jonathan; Harvey, Kenneth; Sethumadhavan, Simha; Kaiser, Gail; Jebara, Tony | ACM |
| A37 | Detecting Code Clones with Gaps by Function Applications | Matsushita, Tsubasa; Sasano, Isao | ACM |
| A38 | CCSharp: An Efficient Three-Phase Code Clone Detector Using Modified PDGs | Wang, M.; Wang, P.; Xu, Y. | IEEE |
| A39 | Deep Learning Code Fragments for Code Clone Detection | White, Martin; Tufano, Michele; Vendome, Christopher; Poshyvanyk, Denys | ACM |
| A40 | Constructing Universal Version History | Chang, Hung-Fu; Mockus, Audris | ACM |
| A41 | Efficient Token Based Clone Detection with Flexible Tokenization | Basit, Hamid Abdul; Jarzabek, Stan | ACM |
| A42 | CReN: A Tool for Tracking Copy-and-paste Code Clones and Renaming Identifiers Consistently in the IDE | Jablonski, Patricia; Hou, Daqing | ACM |
| A43 | Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets | Chen, Kai; Liu, Peng; Zhang, Yingjun | ACM |
| A44 | A Hybrid Approach (Syntactic and Textual) to Clone Detection | Funaro, Marco; Braga, Daniele; Campi, Alessandro; Ghezzi, Carlo | ACM |
| A45 | IDE-based Real-time Focused Search for Near-miss Clones | Zibran, Minhaz F.; Roy, Chanchal K. | ACM |
| A46 | Clone Detection and Removal for Erlang/OTP Within a Refactoring Environment | Li, Huiqing; Thompson, Simon | ACM |
| A47 | Practical Language-independent Detection of Near-miss Clones | Cordy, James R.; Dean, Thomas R.; Synytskyy, Nikita | ACM |
| A48 | Clone Detection and Elimination for Haskell | Brown, Christopher; Thompson, Simon | ACM |
| A49 | VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery | Kim, S.; Woo, S.; Lee, H.; Oh, H. | IEEE |
| A50 | SCVD: A new semantics-based approach for cloned vulnerable code detection | Zou, Deqing; Qi, Hanchao; Li, Zhen; Wu, Song; Jin, Hai; Sun, Guozhong; Wang, Sujuan; Zhong, Yuyi | EI COMPEDEX |
| A51 | Implementation of analytical hierarchy process in detecting structural code clones | Aktas, Mehmet S.; Kapdan, Mustafa | EI COMPEDEX |
| A52 | Code clone genealogy detection on e-health system using Hadoop | Tekchandani, Rajkumar; Bhatia, Rajesh; Singh, Maninder | EI COMPEDEX |
| A53 | Scalable code clone search for malware analysis | Farhadi, Mohammad Reza; Fung, Benjamin C. M.; Fung, Yin Bun; Charland, Philippe; Preda, Stere; Debbabi, Mourad | EI COMPEDEX |
| A54 | Scalable and accurate detection of code clones | Sargsyan, S.; Kurmangaleev, Sh.; Belevantsev, A.; Avetisyan, A. | EI COMPEDEX |
| A55 | Clone-based and interactive recommendation for modifying pasted code | Lin, Yun; Peng, Xin; Xing, Zhenchang; Zheng, Diwen; Zhao, Wenyun | EI COMPEDEX |
| A56 | Structural Code Clone Detection Methodology Using Software Metrics | Aktas, Mehmet S.; Kapdan, Mustafa | EI COMPEDEX |
| A57 | CLORIFI: Software vulnerability discovery using code clone verification | Li, Hongzhe; Kwon, Hyuckmin; Kwon, Jonghoon; Lee, Heejo | EI COMPEDEX |
| A58 | Identifying code clones with refactorerl | Fordos, Viktoria; Toth, Melinda | EI COMPEDEX |
| A59 | A novel approach to effective detection and analysis of code clones | Rajakumari, K. E.; Jebarajan, T. | IEEE |
| A60 | Modular Heap Abstraction-Based Code Clone Detection for Heap-Manipulating Programs | Dong, L.; Wang, J.; Chen, L. | IEEE |
| A61 | Incremental clone detection and elimination for Erlang programs | Li, Huiqing; Thompson, Simon | EI COMPEDEX |
| A62 | A novel detection approach for statement clones | Shi, Qing Qing; Zhang, Li Ping; Meng, Fan Jun; Liu, Dong Sheng | IEEE |
| A63 | Folding repeated instructions for improving token-based code clone detection | Murakami, Hiroaki; Hotta, Keisuke; Higo, Yoshiki; Igaki, Hiroshi; Kusumoto, Shinji | EI COMPEDEX |
| A64 | Incremental Code Clone Detection: A PDG-based Approach | Higo, Y.; Yasushi, U.; Nishino, M.; Kusumoto, S. | IEEE |
| A65 | Clone detection through process algebras and Java Bytecode | Santone, Antonella | EI COMPEDEX |
| A66 | SHINOBI: A Tool for Automatic Code Clone Detection in the IDE | Kawaguchi, S.; Yamashina, T.; Uwano, H.; Fushida, K.; Kamei, Y.; Nagura, M.; Iida, H. | IEEE |

Tabela 4.8 – Artigos Selecionados

**Continuação da Tabela 7**

| ID | Título | Autores(as) | Fonte |
|---|---|---|---|
| A67 | Cross-language clone detection | Kraft, Nicholas A.; Bonds, Brandon W.; Smith, Randy K. | EI COMPEDEX |
| A68 | Threshold-free code clone detection for a large-scale heterogeneous Java repository | Keivanloo, I.; Zhang, F.; Zou, Y. | IEEE |
| A69 | Code clone detection using parsing actions | Lazar, F. M.; Banias, O. | IEEE |
| A70 | Boreas: an accurate and scalable token-based approach to code clone detection | Yuan, Y.; Guo, Y. | IEEE |
| A71 | Code clone detection using wavelets | Karus, S.; Kilgi, K. | IEEE |
| A72 | CCCD: Concolic code clone detection | Krutz, D. E.; Shihab, E. | IEEE |
| A73 | An execution-semantic and content-and-context-based code-clone detection and analysis | Kamiya, T. | IEEE |
| A74 | An efficient code clone detection model on Java byte code using hybrid approach | Raheja, K.; Tekchandani, R. K. | IEEE |
| A75 | Code clone detection using decentralized architecture and code reduction | Patil, R. V.; Joshi, S. D.; Shinde, S. V.; Ajagekar, D. A.; Bankar, S. D. | IEEE |
| A76 | LLVM-based code clone detection framework | Avetisyan, A.; Kurmangaleev, S.; Sargsyan, S.; Arutunian, M.; Belevantsev, A. | IEEE |
| A77 | Selecting a set of appropriate metrics for detecting code clones | Bansal, G.; Tekchandani, R. | IEEE |
| A78 | ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions | Jang, J.; Agrawal, A.; Brumley, D. | IEEE |
| A79 | CCLearner: A Deep Learning-Based Clone Detection Approach | Li, L.; Feng, H.; Zhuang, W.; Meng, N.; Ryder, B. | IEEE |
| A80 | Scalable detection of semantic clones | Gabel, M.; Jiang, L.; Su, Z. | IEEE |
| A81 | Code Clone Detection on Specialized PDGs with Heuristics | Higo, Y.; Kusumoto, S. | IEEE |
| A82 | A technique to detect multi-grained code clones | Yuki, Y.; Higo, Y.; Kusumoto, S. | IEEE |
| A83 | Code syntax-comparison algorithm based on type-redefinition-preprocessing and rehash classification | Cui, B. and Guan, J. and Guo, T. and Han, L. and Wang, J. and Ju. Y. | SCOPUS |
| A84 | WSIM: Detecting Clone Pages Based on 3-Levels of Similarity Clues | Jung, W.; Wu, C.; Lee, E. | IEEE |
| A85 | Gapped code clone detection with lightweight source code analysis | Murakami, H.; Hotta, K.; Higo, Y.; Igaki, H.; Kusumoto, S. | IEEE |
| A86 | Clone detection meets Semantic Web-based transitive closure computation | Keivanloo, I.; Rilling, J. | IEEE |
| A87 | Towards slice-based semantic clone detection | Alomari, H. W.; Stephan, M. | IEEE |
| A88 | VFDETECT: A vulnerable code clone detection system based on vulnerability fingerprint | Liu, Z.; Wei, Q.; Cao, Y. | IEEE |
| A89 | BinClone: Detecting Code Clones in Malware | Farhadi, M. R.; Fung, B. C. M.; Charland, P.; Debbabi, M. | IEEE |
| A90 | SourcererCC and SourcererCC-I: Tools to Detect Clones in Batch Mode and during Software Development | Saini, V.; Sajnani, H.; Kim, J.; Lopes, C. | IEEE |
| A91 | Code clone detection using parsing actions | Maeda, K. | IEEE |
| A92 | Detection of Type-1 and Type-2 Code Clones Using Textual Analysis and Metrics | Kodhai, E.; Kanmani, S.; Kamatchi, A.; Radhika, R.; Saranya, B. V. | IEEE |
| A93 | Semantic code clone detection using parse trees and grammar recovery | Tekchandani, R.; Bhatia, R. K.; Singh, M. | IEEE |
| A94 | A Data Mining Approach for Detecting Higher-Level Clones in Software | Basit, H. A.; Jarzabek, S. | IEEE |
| A95 | Implementing a 3-way approach of clone detection and removal using PC Detector tool | Mahajan, G.; Bharti, M. | IEEE |
| A96 | A novel approach based on formal methods for clone detection | Cuomo, A.; Santone, A.; Villano, U. | IEEE |
| A97 | CCFinderSW: Clone Detection Tool with Flexible Multilingual Tokenization | Semura, Y.; Yoshida, N.; Choi, E.; Inoue, K. | IEEE |
| A98 | Scalable code clone detection and search based on adaptive prefix filtering | Nishi, Manziba Akanda; Damevski, Kostadin | IEEE |
| A99 | Tree-pattern-based clone detection with high precision and recall | Lee, H.-S. and Choi, M.-R. and Doh, K.-G. | SCOPUS |
| A100 | Interface Driven Code Clone Detection | Patil, R.V. and Joshi, S.D. and Shinde, S.V. and Khanna, V. | SCOPUS |
| A101 | Method-level code clone detection for Java through hybrid approach | Kodhai, E. and Kanmani, S. | SCOPUS |
| A102 | Non-trivial software clone detection using program dependency graph | Gautam, P. and Saini, H. | SCOPUS |
| A103 | An automatic framework for extracting and classifying near-miss clone genealogies | Saha, R.K. and Roy, C.K. and Schneider, K.A. | SCOPUS |
| A104 | Code clones detection using machine learning technique: Support vector machine | Jadon, S. | SCOPUS |
| A105 | Generic code Cloning method for detection of Clone code in software Development | Haque, S.M.F. and Srikanth, V. and Reddy, E.S. | SCOPUS |
| A106 | Method-level code clone detection through LWH (Light Weight Hybrid) approach | Kodhai, Egambaram and Kanmani, Selvadurai | SPRINGER LINK |
| A107 | Code clone detection using coarse and fine-grained hybrid approaches | Sheneamer, A. and Kalita, J. | SCOPUS |
| A108 | Method-level incremental code clone detection using hybrid approach | Kodhai, E. and Kanmani, S. | SCOPUS |
| A109 | A measurement of similarity to identify identical code clones | Mythili, S.S. and Sarala, S. | SCOPUS |
| A110 | An effective approach using dissimilarity measures to estimate software code clone | Patil, R.V. and Joshi, S.D. and Shinde, S.V. and Khanna, V. | SCOPUS |
| A111 | Enhancing generic pipeline model for code clone detection using divide and conquer approach | Mubarak-Ali, A.-F. and Syed-Mohamad, S. and Sulaiman, S. | SCOPUS |
| A112 | Semantic code clone detection for Internet of Things applications using reaching definition and liveness analysis | Tekchandani, Rajkumar and Bhatia, Rajesh and Singh, Maninder | PRINGER LINK |
| A113 | Structural similarity detection using structure of control statements | Sudhamani, M. and Rangarajan, L. | SCOPUS |
| A114 | A metric space based software clone detection approach | Li, Z. and Sun, J. | SCOPUS |
| A115 | An iterative, metric space based software clone detection approach | Li, Z. and Sun, J. | SCOPUS |
| A116 | A hybrid technique in pre-processing and transformation process for code clone detection | Mubarak Ali, A.-F. and Sulaiman, S. | SCOPUS |
| A117 | Code process block based reduction technique for software code clone detection | Patil, R.V. and Joshi, S.D. and Shinde, S.V. and Khanna, V. | SCOPUS |
| A118 | Index-based code clone detection: Incremental, distributed, scalable | Hummel, B. and Juergens, E. and Heinemann, L. and Conradt, M. | SCOPUS |
| A119 | An improved method for tree-based clone detection in Web Applications | Li, C. and Sun, J. and Chen, H. | SCOPUS |
| A120 | Detection of recurring clones using weighted frequent itemset mining | Mythili, S. and Sarala, S. | SCOPUS |
| A121 | KLONOS: Similarity-based planning tool support for porting scientific applications | Ding, W. and Hsu, C.-H. and Hernandez, O. and Chapman, B. and Graham, R. | SCOPUS |
| A122 | An effective software clone detection using distance clustering | Gayathri Devi, D. and Punithavalli, M. | SCOPUS |
| A123 | Models are code too: Near-miss clone detection for Simulink models | Alalfi, M.H. and Cordy, J.R. and Dean, T.R. and Stephan, M. and Stevenson, A. | SCOPUS |
| A124 | An efficient new multi-language clone detection approach from large source code | Rehman, S.U. and Khan, K. and Fong, S. and Biuk-Aghai, R. | SCOPUS |
| A125 | Accurate and Efficient Structural Characteristic Feature Extraction for Clone Detection | Nguyen, Hoan Anh and Nguyen, Tung Thanh and Pham, Nam H. and Al-Kofahi, Jafar M. and Nguyen, Tien N. | SPRINGER LINK |
| A126 | Tree slicing: Finding intertwined and gapped clones in one simple step | Akhin, M. and Itsykson, V. | SPRINGER LINK |
| A127 | Detect functionally equivalent code fragments via k-nearest neighbor algorithm | Kong, D. and Su, X. and Wu, S. and Wang, T. and Ma, P. | SCOPUS |
| A128 | Parallel code clone detection using MapReduce | Sajnani, H. and Ossher, J. and Lopes, C. | SCOPUS |

# 5 ABORDAGEM

# 6 FERRAMENTA

# 7 AVALIAÇÃO

# 8 AMEAÇA A VALIDADE

# 9 TRABALHOS RELACIONADOS

Nesta seção são é descritas contextos de aplicação de detecção de clones em softwares em conjunto ou não com linhas de produtos de software, de modo a descrever a relação deste trabalho com outros já propostos.

Em um trabalho ((PAIVA; FIGUEIREDO, 2016)), há a proposta de desenvolvimento e de avaliação de um novo método de detecção de clone de código baseado em sequências similares de chamada de métodos. A ferramenta desenvolvida para detecção (`McSheep` é comparada com a ferramenta PMD[1] para detectar clones não identificados. Participaram de uma pesquisa 25 desenvolvedores para avaliar o método e mais de 90% deles concordam com sua validade para detecção de clones. O resultado da comparação das ferramentas mostra que ambas podem ser utilizadas de forma híbrida, pois existe um conjunto interseção de casos detectados e, para cada ferramenta, um conjunto exclusivo.

Em outro trabalho ((SCHULZE; APEL; KäSTNER, 2011)), há descrição de razões para existência de clones relacionados à programação orientada a características e como lidar com eles. Para isso, aplicar refatorações em clones sintáticos para remoção de clones em LPS's é a abordagem proposta, visando à permanência dos benefícios de programação orientada a características como mais coesão e reutilização de códigos de características. Os resultados foram: i) cerca de 10% a 15% do código das dez LPS's analisadas são clones; ii) entre 9% a 12% desses podem ser refatorados com base na sintaxe; iii) mais ou menos 9% dos clones são relacionados a POC e 2% a 3% relacionados a POO; e iv) em todas as etapas de análise, a quantidade de clones é maior em LPS's desenvolvidas do zero do que em LPS's decompostas de sistemas legados.

Assim como os trabalhos citados, essa investigação propõe a detecção de clones de código em código. Mas, há diferenças a serem consideradas, tais como,

[1] PMD é uma ferramenta de análise que varre código Java e dispõe da implementação da análise CPD (Copy/Paste Detector), que detecta código duplicado.

a abordagem de como é feita a detecção de clones em paradigmas diferentes (programação orientada a objetos e programação orientada a características), propor uma abordagem, (semi)automatizar a abordagem proposta (apoio computacional) para detectar clones diretamente na LPS e analisar os resultados da abordagem com relação à manutenibilidade.

# 10 CONSIDERAÇÕES FINAIS

# 11 CONCLUSÃO

# REFERÊNCIAS

APEL, S. et al. **Feature-oriented software product lines**. [S.l.]: Springer, 2016.

COLANZI, T. E. Uma abordagem de otimização multiobjetivo para projeto arquitetural de linha de produto de software. 2014.

CORRÊA, H. L.; CORRÊA, C. A. **Administração de Produção E Operações: Manufatura E Serviços: Uma Abordagem Estratégica .** [S.l.]: Editora Atlas SA, 2000.

DUALA-EKOKO, E.; ROBILLARD, M. P. Tracking code clones in evolving software. In: **29th International Conference on Software Engineering (ICSE'07)**. [S.l.: s.n.], 2007. p. 158–167. ISSN 0270-5257.

FORDOS, V.; TOTH, M. Identifying code clones with refactorerl. **Acta Cybernetica**, v. 22, n. 3, p. 553 – 571, 2016. ISSN 0324721X. Clone detection algorithms;Code clone;Identifying code;Maintenance cost;Source codes;Tools and techniques;.

GAUTAM, P.; SAINI, H. Various code clone detection techniques and tools: a comprehensive survey. In: SPRINGER. **International Conference on Smart Trends for Information Technology and Computer Communications**. [S.l.], 2016. p. 655–667.

JANG, J.; BRUMLEY, D. Bitshred: Fast, scalable code reuse detection in binary code (cmu-cylab-10-006). **CyLab**, p. 28, 2009.

KAMIYA, T.; KUSUMOTO, S.; INOUE, K. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. **IEEE Transactions on Software Engineering**, v. 28, n. 7, p. 654–670, Jul 2002. ISSN 0098-5589.

KAPDAN, M.; AKTAS, M.; YIGIT, M. On the structural code clone detection problem: a survey and software metric based approach. In: SPRINGER. **International Conference on Computational Science and Its Applications**. [S.l.], 2014. p. 492–507.

KITCHENHAM, D. B. B.; BRERETON, P. The value of mapping studies-a participant-observer case study. In: **14th International Conference on Evaluation and Assessment in Software Engineering, British Computer Society**. [S.l.: s.n.], 2010. p. 25–33.

LAGUNA, M. A.; CRESPO, Y. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. **Science of Computer Programming**, Elsevier, v. 78, n. 8, p. 1010–1034, 2013.

LAGUNA, M. A.; CRESPO, Y. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. **Science of Computer Programming**, v. 78, n. 8, p. 1010 – 1034, 2013. ISSN 0167-6423. Special section on software evolution, adaptability, and maintenance & Special section on the Brazilian Symposium on Programming Languages.

LIN, Y. et al. Detecting differences across multiple instances of code clones. In: **Proceedings of the 36th International Conference on Software Engineering**. New York, NY, USA: ACM, 2014. (ICSE 2014), p. 164–174. ISBN 978-1-4503-2756-5.

MORESI, E. et al. Metodologia da pesquisa. **Brasília: Universidade Católica de Brasília**, v. 108, p. 24, 2003.

PAIVA, A. M.; FIGUEIREDO, E. **ON THE DETECTION OF CODE CLONE WITH SEQUENCE OF METHOD CALLS**. Dissertação (Masters Thesis) — University of Minas Gerais, Belo Horizonte, may 2016.

PETERSEN R. FELDT, S. M. K.; MATTSSON, M. Systematic mapping studies in software engineering. In: **12th International Conference on Evaluation and Assessment in Software Engineering**. [S.l.: s.n.], 2008. p. 1.

RATTAN, D.; KAUR, J. Systematic mapping study of metrics based clone detection techniques. In: **Proceedings of the International Conference on Advances in Information Communication Technology & Computing**. New York, NY, USA: ACM, 2016. (AICTC '16), p. 76:1–76:7. ISBN 978-1-4503-4213-1. Disponível em: <http://doi.acm.org/10.1145/2979779.2979855>.

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

SCHUGERL, P. Scalable clone detection using description logic. In: **Proceedings of the 5th International Workshop on Software Clones**. New York, NY, USA: ACM, 2011. (IWSC '11), p. 47–53. ISBN 978-1-4503-0588-4.

SCHULZE, S.; APEL, S.; KäSTNER, C. Code clones in feature-oriented software product lines. **ACM SIGPLAN Notices**, v. 46, n. 2, p. 103–112, 2011. Cited By 2. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-79951743431&doi=10.1145%2f1942788.1868310&partnerID=40&md5=f151bb4f4d206bf539e21a1b3fc92e58>.

SOLANKI, K.; KUMARI, S. Comparative study of software clone detection techniques. In: IEEE. **Management and Innovation Technology International Conference (MITicon), 2016**. [S.l.], 2016. p. MIT–152.

TORRES, J. J. B.; JUNIOR, M. C. R.; FARIAS, M. A. D. F. Procedural x oo: A corporative experiment on source code clone mining. In: . Porto, Portugal: [s.n.], 2017. v. 2, p. 395 – 402. Closed source;Code clone detection;Experimental software engineering;Mining software repositories;Object oriented software;Private educations;Proprietary software;Similarity threshold;.

VALE, G. et al. Criteria and guidelines to improve software maintainability in software product lines. In: **2015 12th International Conference on Information Technology - New Generations**. [S.l.: s.n.], 2015. p. 427–432.

WIERINGA N. MAIDEN, N. M. R.; ROLLAND, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. In: . [S.l.: s.n.], 2006. p. 102—-107.

YUAN, Y.; GUO, Y. Cmcd: Count matrix based code clone detection. In: **2011 18th Asia-Pacific Software Engineering Conference**. [S.l.: s.n.], 2011. p. 250–257. ISSN 1530-1362.

YUKI, Y.; HIGO, Y.; KUSUMOTO, S. A technique to detect multi-grained code clones. In: **2017 IEEE 11th International Workshop on Software Clones (IWSC)**. [S.l.: s.n.], 2017. p. 1–7.

## APÊNDICE A – O que são apêndices

Um apêndice é um suporte elucidativo e ilustrativo do texto principal. Sua função é agrupar elementos que são úteis à compreensão do texto e que, no entanto, podem ser apresentados à parte sem prejuízo à compreensão. É útil para a apresentação de modelagens, diagramas extensos, listagens de código-fonte de programas e demais elementos que o autor julgar necessário à complementação do tema abordado no texto principal.