****

**UNIVERSIDADE FEDERAL DO CEARÁ**

**CENTRO DE TECNOLOGIA**

**DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA**

**ENGENHARIA DE TELEINFORMÁTICA**

**FRANCISCO ANDERSON DE ALMADA GOMES**

**IMPLEMENTAÇÃO DE TÉCNICAS PARA A ANÁLISE DE SIMILARIDADE ENTRE CÓDIGOS-FONTE BASEADAS EM CONTAGEM DE ATRIBUTOS ESTRUTURAIS E SEMÂNTICOS**

**FORTALEZA**

**2014**

FRANCISCO ANDERSON DE ALMADA GOMES

IMPLEMENTAÇÃO DE TÉCNICAS PARA A ANÁLISE DE SIMILARIDADE ENTRE CÓDIGOS-FONTE BASEADAS EM CONTAGEM DE ATRIBUTOS ESTRUTURAIS E SEMÂNTICOS

Monografia apresentada ao Curso de Engenharia de Teleinformática do Departamento de Engenharia de Teleinformática da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Engenheiro de Teleinformática.

Orientador: Prof. Dr. José Marques Soares.

FORTALEZA

2014

FRANCISCO ANDERSON DE ALMADA GOMES

IMPLEMENTAÇÃO DE TÉCNICAS PARA A ANÁLISE DE SIMILARIDADE ENTRE CÓDIGOS-FONTE BASEADAS EM CONTAGEM DE ATRIBUTOS ESTRUTURAIS E SEMÂNTICOS

Monografia apresentada ao Curso de Engenharia de Teleinformática do Departamento de Engenharia de Teleinformática da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Engenheiro de Teleinformática.

Orientador: Prof. Dr. José Marques Soares.

Aprovada em \_\_\_/\_\_\_/\_\_\_\_\_\_.

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. José Marques Soares (Orientador)

Universidade Federal do Ceará (UFC)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. Giovanni Cordeiro Barroso

Universidade Federal do Ceará (UFC)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me. Wellington Wagner Ferreira Sarmento

Universidade Federal do Ceará (UFC)

Ao grande mestre, Sobral.

**AGRADECIMENTOS**

*“A inovação distingue os líderes dos seguidores.”*

(Steve Jobs)

**RESUMO**

**Palavras-chave:**

**ABSTRACT**

**Keywords:**

**LISTA DE ILUSTRAÇÕES**

Figura 1 – legenda ....... 2

**LISTA DE TABELAS**

Tabela 1 – legenda ..... 3

**LISTA DE ABREVIATURAS E SIGLAS**

ABNT Associação Brasileira de Normas Técnicas

**SUMÁRIO**

[**1 INTRODUÇÃO** 1](#_Toc397617166)

[**1.1** **Contextualização** 1](#_Toc397617167)

[**1.2** **Motivação** 1](#_Toc397617168)

[**1.3** **Objetivos** 1](#_Toc397617169)

[**1.4** **Estrutura do documento** 2](#_Toc397617170)

[**2 FUNDAMENTAÇÃO TEÓRICA** 3](#_Toc397617171)

[**2.1 Técnicas de plágio** 3](#_Toc397617172)

[**2.2 Algoritmos e Ferramentas para análise de similaridade** 5](#_Toc397617173)

**1 INTRODUÇÃO**

* 1. **Contextualização**

O acompanhamento de alunos em turmas numerosas de disciplinas voltadas ao aprendizado de técnicas de programação é uma dificuldade encontrada pelos professores e/ou monitores, que buscam usar a tecnologia para facilitar o gerenciamento de turmas grandes. Um problema encontrado em laboratórios de programação é a cópia total ou parcial de soluções entre colegas. Em cenários de turmas numerosas, a detecção deste tipo de conduta se torna bastante complicada.

A análise de similaridade em códigos-fonte é objetivo de estudo de diversos trabalhos e algumas ferramentas foram desenvolvidas para auxiliar na avaliação.

* 1. **Motivação**

Apesar da grande quantidade de ferramentas disponíveis, poucas delas são capazes de identificar de maneira eficaz todas as semelhanças léxicas e semânticas entre pares de códigos.

* 1. **Objetivos**

Tendo em vista as dificuldades apresentadas por essas ferramentas, esse trabalho propõe uma ferramenta que extrai as características-chave únicas da linguagem e o processamento dessas para criação de uma métrica que define o nível de similaridade entre dois códigos-fonte e a comparação dos resultados encontrados com as outras ferramentas de análise de similaridade em códigos-fonte.

Os objetivos específicos são:

* Efetuar um estudo teórico sobre as principais técnicas de detecção de plágio em código-fonte e estudar os resultados dos principais algoritmos relacionados.
* Analisar técnicas de normalização de códigos-fonte que melhorem o desempenho dos algoritmos de análise sintática no contexto de analise de similaridade.
* Introduzir métricas de complexidade a ferramenta.
  1. **Estrutura do documento**

O trabalho está organizado em cinco capítulos. O presente capítulo descreve uma breve introdução ao tema, contextualizando o assunto abordado, a motivação, os objetivos e as contribuições deste trabalho.

No Capítulo 2 são abordados os principais conceitos teóricos necessários para compreensão deste trabalho. Entre eles estão os principais conceitos relativos a técnicas de plágio, ao funcionamento de algoritmos e de ferramentas para análise de similaridade e detecção de plágio, as técnicas de normalização e as métricas de complexidade.

No Capítulo 3 apresenta a ferramenta desenvolvida, bem como o método de conformidade que foi desenvolvido para a análise dos resultados de múltiplas ferramentas.

No Capítulo 4, apresentam-se os resultados obtidos com códigos gerados manualmente.

Por fim, no Capítulo 5 são realizadas as considerações finais, bem como as possíveis melhorias a serem consideradas em trabalhos futuros.

**2 FUNDAMENTAÇÃO TEÓRICA**

Neste capítulo, são apresentados os principais conceitos teóricos necessários para compreensão deste trabalho. A Seção 2.1 apresenta os meios mais comuns empregados para realizar o plágio. A Seção 2.2 apresenta os fundamentos da detecção de plágio em código-fonte. A Seção 2.3 apresenta os algoritmos usados na detecção de plágio. A Seção 2.4 mostra as ferramentas mais comuns para análise de similaridade. Na Seção 2.5, as técnicas de normalização são abordadas. Por fim, as métricas de complexidade são discutidas na Seção 2.6.

**2.1 Técnicas de plágio**

Nos primeiros laboratórios de programação é fácil encontrar códigos com nomes de variáveis que são pouco usuais e idênticos, revelando certa ingenuidade na tentativa da cópia. Com o passar das aulas de programação, são apresentados problemas mais complexos, também se tornando mais difícil identificar o nível de similaridade entre pares de código dos alunos, mas não são esperadas modificações muito elaboradas em códigos plagiados.

As modificações utilizadas na cópia de um código-fonte podem ser bastante simples, como mudar comentários e nomes de variáveis, ou atingir níveis maiores complexidade, no caso de alterar uma estrutura de controle ou utilizar comandos similares (e.g., *for* no lugar de *while*), o que requer um conhecimento maior da linguagem de programação.

Faidhi e Robinson estão entre os primeiros autores a caracterizar essas modificações, ilustrando-as conforme a Figura 2.1, que mostra, de dentro para fora, o aumento da dificuldade na detecção do plágio (FAIDHI e ROBINSON, 1987):

L0: Corresponde a códigos sem modificação;

L1: Corresponde a modificação de comentários no código;

L2: Corresponde a alteração de identificadores (e.g., os nomes das variáveis);

L3: Corresponde a mudança das posições de variáveis (e.g., tornar uma variável global em local);

L4: Equivale a modificações de combinação de procedimentos (e.g., trocar trechos de código por funções);

L5: Equivale a alterações nas instruções (e.g., substituir um operador por outro similar);

L6: Equivale a mudança na lógica de controle (e.g., *if* por *switch*).

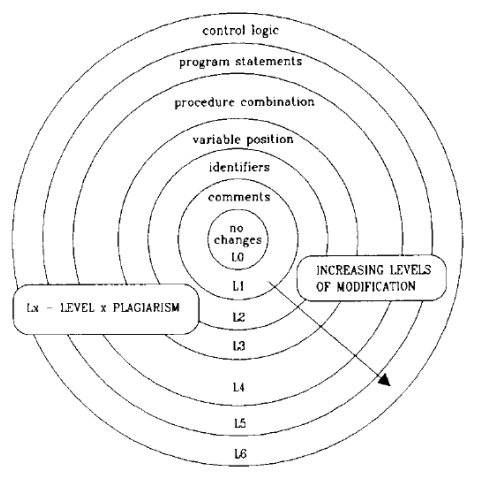


Figura 2.1: Componentes da complexidade na detecção de plágio.

Fonte: FAIDHI e ROBINSON

De acordo com Whale (1990), as técnicas de modificações mais empregadas são:

i. Alteração de comentários e/ou formatação;

ii. Modificação de nomes de identificadores;

iii. Alteração da ordem de operandos e expressões;

iv. Alteração de tipos de dados;

v. Substituição de expressões por equivalentes;

vi. Adição de instruções redundantes ou variáveis;

vii. Alteração na ordem de instruções que não alteram o funcionamento;

viii. Alteração das estruturas de loop;

ix. Alteração das estruturas das instruções de seleção;

x. Substituição de chamadas a funções pelo respectivo conteúdo;

xi. Adicionar instruções que não influenciam o fluxo do programa;

xii. Combinação de código copiado com código original.

Para Mozgovoy (2006), as transformações para esconder o plágio são:

i. Alteração de comentários;

ii. Alteração de espaços em branco e layout;

iii. Modificação de nomes de identificadores;

iv. Reordenação de blocos de código;

v. Reordenação de instruções dentro de blocos de códigos;

vi. Alteração na ordem de operadores/operandos em expressões;

vii. Mudança de tipos de dados;

viii. Adição de instruções redundantes ou variáveis;

ix. Substituição de estruturas de controle por equivalentes;

x. Substituição da chamada a uma função pelo conteúdo da mesma.

Verifica-se que não existe uma grande variação entre as principais técnicas defendidas pelos autores. Para este trabalho, foram reunidas algumas dessas técnicas para a elaboração de um conjunto de códigos que representem as principais modificações. As ações utilizadas para a produção dos códigos são apresentadas na subseção 4.1.1.

**2.2 Detecção de plágio em código-fonte**

Esta seção descreve as principais metodologias empregadas na detecção de plágio, a história das ferramentas, técnicas, algoritmos e ferramentas utilizadas na detecção de plágio.

**2.2.1. Contagem de atributos e comparação de estruturas**

Segundo Ji, Woo e Cho (2007), duas metodologias para detecção de plágio em código-fonte são utilizadas: contagem de atributos e comparação de estruturas. Contagem de atributos é uma metodologia que extrai e calcula informações (e.g., a freqüência de *keywords*). De acordo com Kleiman (2007) e Cornic (2008), essa abordagem é eficiente apenas para casos em que ocorreram poucas tentativas de alteração no código. Segundo Verco e Wise (2006), os primeiros sistemas automatizados para a detecção de plágio em código-fonte utilizavam a técnica de contagem de atributo em suas comparações. Segundo Ji, Woo e Cho (2007), a metodologia baseada em comparação de estruturas subdividem os códigos -fonte em pequenas estruturas, e essas são posteriormente comparadas usando métricas diversas. Essa abordagem torna possível a detecção plágio em trechos de código, algo que não ocorre na contagem de atributos. Em Verco e Wise (1996) é revelada a maior eficiência dos sistemas que usam comparação de estruturas em relação aos sistemas baseados em contagem de atributos.

**2.2.2 História dos sistemas de detecção de plágio em código-fonte**

O primeiro sistema de detecção de plágio foi desenvolvido por OTTENSTEIN (1976), que utilizou a contagem de atributos para identificar similaridades entre códigos-fonte escritos em FORTRAN. Esse sistema de detecção de plágio usou as métricas de Halstead (1972), em que era considerado plágio se o valor delas fossem idênticos.

Robinson e Soffa (1980) desenvolveram um sistema de detecção de plágio também para a linguagem FORTRAN chamado Instructional Tool for Program ADvising (ITPAD). ITPAD realizava três procedimentos: primeiro fazia-se uma análise léxica onde eram calculadas 14 métricas, depois analisava a estrutura do programa por meio de gráficos de fluxo e, por último, interpretava-se os resultados obtidos.

Donaldson, Lancaster e Sposato (1981) criaram o primeira ferramenta de detecção de plágio usando comparação de estruturas e comparação de atributos.

Whale (1990) desenvolveu um sistema de detecção de plágio chamado Plague, em que gera perfis dos programas de entrada. Esses perfis são sequências de *tokens* compostos de informações estruturais. Os perfis de estrutura semelhante são combinados e suas sequências de tokens comparadas para encontrar subsequências comuns.

Wise (1993) criou o algoritmo Greedy-String-Tilin (GST) usado para combinar sequências de tokens. Esse algoritmo foi usado na ferramenta YAP3 que também foi desenvolvida pelo autor.

Prechelt, Malpohl e Phlippsen (2000) apresentaram um sistema chamado JPlag. A ferramenta transforma os programas em sequências de tokens e compara as sequências usando o algoritmo Running-Karp-Rabin Greedy-String-Tiling (RKRGST).

MOSS (Measure of Software Similarity) (MOSS, 2012) é disponibilizado na Universidade da Califórnia. Seu desenvolvimento foi iniciado em 1994. Esse sistema é baseado no algoritmo Winnowing (Schleimer, 2003) e possui código fechado.

**2.2.3 Técnicas para detecção de plágio**

**2.2.4 Algoritmos usados na detecção de plágio**

Os algoritmos utilizados na análise de similaridade (detecção de plágio) variam de acordo com à lógica de comparação. Kleiman (2007) apresentou alguns algoritmos para a detecção de plágio em códigos-fonte, os principais são: *Winnowing*, *Greedy String Tiling* e *Running Karp-Rabin Greedy String Tiling*. Esses algoritmos são descritos nas próximas subseções.

**2.2.4.1 *Winnowing***

O *Winnowing* é um algoritmo que tem como objetivo melhorar a eficiência do processo de comparação de documentos com base em assinatura única por documento. Segundo Schleimer et. al (2003), esse algoritmo obtém uma assinatura para cada documento de forma que essa assinatura possa ser usada para identificá-lo e detectar similaridade.

Esse algoritmo utiliza o conceito de k-gramas. Conforme Kleiman (2007), os k-gramas de uma cadeia S são as sub-cadeias de comprimento k contiguas e sobrepostas da cadeia S. Os 3-gramas da frase “o gato morreu" são: ‘oga’, ‘gat’, ‘ato’, ‘tom’, ‘omo’, ‘mor’, ‘orr’, ‘rre’, ‘reu’, por exemplo. O processo de obtenção da assinatura inicia-se com a divisão do texto em k-gramas. Na sequência, cada k-grama é representado por um valor numérico, e, por fim, um subconjunto desses valores é obtido da peneiração do super-conjunto de todos os valores para ser a assinatura do documento.

**2.2.2 *Greedy String Tiling***

A

**2.2.5 Ferramentas utilizadas para análise de similaridade**