**Pendeteksi *Code Clones* Otomatis**

**Menggunakan Pendekatan Token**

**Tugas Akhir**

**diajukan untuk memenuhi salah satu syarat**

**memperoleh gelar sarjana**

**dari Program Studi Informatika**

**Fakultas Informatika**

**Universitas Telkom**

**1301174018**

**Muhammad Haydar Maulana**

****

**Program Studi Sarjana Informatika**

**Fakultas Informatika**

**Universitas Telkom**

**Bandung**

**2021**

Pendeteksi Code Clones Otomatis Menggunakan Pendekatan Token

Muhammad Haydar Maulana1, Jati Hiliamsyah Husen2, Sri Widowati3

1,2,3Fakultas Informatika, Universitas Telkom, Bandung

1haydariday@students.telkomuniversity.ac.id, 2jatihusen@telkomuniversity.ac.id, 3sriwidowati@telkomuniversity.ac.id

#### Abstrak

Mendeteksi *code clones* merupakan kegiatan yang memakan banyak sumber daya. Semakin besar dan kompleks suatu sistem, semakin banyak kebutuhan sumber daya dan biaya yang diperlukan untuk mendeteksi *code clones*. Salah satu cara untuk meringankan sumber daya pada proses mendeteksi *code clones* adalah dengan otomasi proses mendeteksi *code clones*. Mendeteksi *code clones* otomatismenggunakan metode pendekatan token merepresentasikan *source codes* sebagai urutan token yang memungkinkan untuk mendeteksi *code clones* dengan struktur baris berbeda. Metode pendekatan token termasuk metode kelas berat karena memerlukan *language parser* khusus untuk setiap bahasa pemrograman berbeda. Penelitian ini bertujuan untuk membuat sebuah sistem pendeteksi *code clones* otomatis menggunakan metode pendekatan token untuk menganalisis dan membandingkan nilai *F1-score* dalam mendeteksi *code clones*. Penelitian ini dilakukan pada tiga *source codes* aplikasi yaitu *Apache*, *Postgre*, dan *Python* dengan hasil *F1-score* secara berturutan adalah “0,94”, “0,55”, “0,89”. Hasil penelitian ini mendapatkan F1-Score tertinggi dibandingkan dengan tiga *tools* pendeteksi *code clones* lain, termasuk Simcad, Nicad, dan MeCC. Implementasi sistem pendeteksi *code clones* pada penelitian ini menunjukan sistem dapat mendeteksi *code clones* dengan baik dibandingkan dengan tiga metode lain.

**Kata kunci :** code clones, sistem pendeteksi code clones otomatis, pendekatan token.

# **Abstract**

**Detecting code clones is a resource-consuming activity. The bigger and more complex a system is, the more resources and costs it takes to detect code clones. One way to reduce resources in the process of detecting code clones is to automate the process of detecting code clones. Detecting code clones automatically using the token approach method represents source codes as a sequence of tokens which allows detecting code clones with different line structures. The token approach method is a heavyweight method because it requires a special parser language for each different programming language. This study aims to create an automatic code clones detection system using the token approach method to analyze and compare the F1-score value in detecting code clones. This research was conducted on three application source codes, namely Apache, Postgre, and Python with the F1-score results being "0.94", "0.55", "0.89". The results of this study obtained the highest F1-Score compared to three other code clones detection tools, including Simcad, Nicad, and MeCC. The implementation of the code clones detection system in this study shows that the system can detect code clones better than the other three methods.**

**Keywords:** code clones, automatic code clones detection system, token approach

1. Pendahuluan

**Latar Belakang**

Menurut McCall, *maintainability* adalah besarnya usaha yang diperlukan untuk melokalisasi dan membetulkan kesalahan-kesalahan yang dapat ditemukan dalam program [1]. Sistem perangkat lunak yang dapat melakukan operasi *maintenance* dengan mudah memiliki kualitas maintainability yang bagus. Menurut observasi, 60%-70% dari seluruh *life cycle* perangkat lunak digunakan untuk pemeliharaan perangkat lunak dalam sisi sumber daya waktu, biaya, dan usaha [2]. Usaha untuk meningkatkan *software maintainability* cukup sulit dan dapat menghabiskan sebagian besar biaya total proyek.

*Code clones* merupakan segmen kode di dalam suatu *source file* yang identik atau memiliki kemiripan antara satu sama lain [3, 4]. Semakin banyak *code clones*  pada suatu sistem perangkat lunak maka semakin banyak *Line of Codes* pada suatu sistem perangkat lunak yang harus dipelihara oleh seorang *developer*. Oleh karena itu, salah satu faktor yang dapat menyebabkan performansi *maintainability* rendah pada suatu sistem perangkat lunak adalah *code clones*. Ketika sebuah sistem mempunyai *code clones sub-system* yang diciptakan oleh duplikasi kode dengan sedikit modifikasi, seorang *developer* harus berhati-hati memodifikasi seluruh *sub-system* yang lain ketika *software fault* ditemukan [5]. *Code clones* dalam *source codes* menyebabkan kesulitan memodifikasi suatu program. *Code clones* dapat diciptakan akibat penggunaan kembali kode dengan “*copy-and-paste*” atau dengan mengulangi porsi kode secara sengaja oleh seorang *developer* [4, 6].

Terdapat banyak tipe *code clones* yang dapat dideteksi dalam suatu *source files* yaitu *code clones* dari salinan yang sangat identik tanpa modifikasi (*type-1*), *code clones* dari salinan identik secara sintaksis (*type-2*) dan *code clones* dari salinan kode dengan sedikit modifikasi (*type-3*) [4]. Tahap yang penting dalam menghilangkan ataupun mengkoreksi *code clones* yaitu tahap mendeteksi *code clones*. Mendeteksi *code clones* dapat membantu mencari *bugs*, menentukan perbaikan *bugs* yang tidak konsisten dan menemukan redundansi di dalam kode [3].

Mendeteksi *code clones* merupakan kegiatan yang memakan waktu banyak karena seorang *developer* harus mengkonfirmasi atau membandingkan seluruh baris kode pada source files secara manual [5]. Diperlukan usaha lebih dan banyak sumber daya untuk mendeteksi *code clones*. Semakin besar dan kompleks suatu sistem perangkat lunak, semakin banyak kebutuhan sumber daya waktu dan biaya yang diperlukan untuk mendeteksi *code clones*.

Salah satu cara untuk meringankan sumber daya pada proses mendeteksi *code clones* adalah dengan otomasi proses mendeteksi *code clones* [4]. Pada penelitian ini, sistem pendeteksi code clones yang dirancang menggunakan metode perbandingan token. Keunggulan metode perbandingan token yaitu metode ini dapat merepresentasikan *source codes* sebagai urutan token yang memungkinkan untuk mendeteksi clones dengan struktur baris berbeda [5]. Sistem pendeteksi *code clones* penelitian ini akan menghasilkan token yang berurutan dari *source codes* inputan melalui analisis leksikal atau *tokenization*. Kami berharap penelitian ini dapat berkontribusi untuk menambah pemahaman tentang software enggineering dan menambah kelebihan dan kekurangan dalam metode ini.

**Rumusan Masalah**

Pada penelitian ini didefinisikan pertanyaan-pertanyaan penelitian (PP) sebagai berikut:

* PP1. Berapakah nilai *F1-score* dari sistem pendeteksi *code clones* otomatis menggunakan metode pendekatan token?
* PP2. Bagaimana performa nilai *F1-score* dari sistem pendeteksi *code clones* otomatis menggunakan metode pendekatan token dibandingkan dengan metode lain?

**Tujuan**

Penelitian ini bertujuan untuk membuat sebuah sistem pendeteksi code clones otomatis menggunakan metode pendekatan token untuk menganalisis dan membandingkan nilai *F1-score* dalam mendeteksi *code clones*. Penelitian ini berkontribusi pada Badan Ilmu Rekayasa Perangkat Lunak.

**Topik dan Batasannya**

Penelitian ini menggunakan metode pendekatan token untuk mendeteksi *code clones* pada *source codes* level fungsi. Metode pembanding pada penilitian ini adalah Simcad [7], Nicad [8], dan MeCC [9]. Dataset dan *golden standard* menggunakan hasil penelitian Krutz [3] tersedia pada website[[1]](#footnote-2).

**Organisasi Tulisan**

Setelah Pendahuluan, dilanjutkan dengan bagian ke-2 Studi Terkait yang berisi teori/studi/literatur yang mendukung (terkait erat) dengan penelitian yang dikerjakan. Pada bagian ke-3 Sistem yang Dibangun yaitu menjelaskan rancangan dan sistem atau produk yang dihasilkan. Dilanjutkan bagian ke-4 Evaluasi yaitu Hasil Pengujian dan Analisis Hasil Pengujian. Dan diakhiri pada Bagian ke-5 Kesimpulan yang memuat kesimpulan dan Saran (*Future Work*)

1. Studi Terkait

Penelitian Terkait

Penelitian terkait dengan sistem pendeteksi *code clones* otomatis merupakan studi yang masih aktif diteliti. Salah satunya adalah sistem pendeteksi *code clones* otomatis dengan model pendekatan Baker menggunakan metode pendekatan token [10]. *Source code*s direpresentasikan menjadi urutan token dari baris-baris kode dengan analisis leksikal. Sistem ini akan membandingkan urutan token secara baris antar baris dengan menggunakan *suffix tree* untuk mendeteksi *code clones*. Metode Baker dapat mendeteksi code clone meskipun syntax-nya berbeda.

Sistem pendeteksi *code clones* otomatis dengan model pendekatan Kamiya dkk. juga merepresentasikan *source codes* dengan urutan token [5]. Namun, model pendekatan Kamiya membandingkan token antar token tanpa menghiraukan baris kode. Model Kamiya menerapkan tranformasi kode dengan peraturan yang telah mereka definisikan untuk mengoptimasi efisiensi komputasi.

Model pendekatan Ducasse dkk. membandingkan seluruh baris antar baris pada *source codes* secara tekstual [11]. Model pendekatan ini tidak melalui proses *parsing* sehingga *language independent*. Untuk menambah efisiensi komputasi, baris-baris kode pada *source codes* dipartisi menggunakan fungsi *hash strings*. Hasil proses pendeteksi *code clones* divisualisasikan dengan *dotplot*, setiap titik merupakan baris yang terdeteksi sebagai pasangan *code clones*.

*Software Maintainability*

*Maintainability* dapat didefinisikan sebagai kemampuan suatu perangkat lunak untuk memperbaiki kesalahan-kesalahan dan dalam memodifikasi atau yang dapat ditemukan dalam program [1]. Dengan kata lain *maintainability* mengukur kemudahan perangkat lunak dalam beradaptasi untuk memenuhi kebutuhan dan ekspektasi pengguna. ISO 25010 menyebutkan bahwa *maintainability* merepresentasikan derajat efektivitas dan efisiensi suatu sistem dapat dimodifikasi untuk memperbaharui, mengkoreksi atau mengadaptasi [12]. Terdapat beberapa sub-faktor kualitas pada faktor kualitas maintainability untuk dijadikan sebagai aspek penilaian kualitas :

* *Modularity*, ukuran dari suatu komponen program yang dapat diubah dengan memiliki efek minim kepada komponen lain pada program.
* *Reusability*, aset-aset yang dapat digunakan pada sistem lain pada suatu program.
* *Modifiability*, efektivitas suatu perangkat lunak dalam penambahan dan perbaikan fitur tanpa mengurangi kualitas dan menambah error/bug.
* *Testability*, efektivitas suatu perangkat lunak dalam ketersediaan kriteria testing yang dapat dijalankan oleh sistem.

|  |
| --- |
|  |
| Gambar . Contoh *Code Clones* |

*Code Clones*

*Code clones* merupakan segmen kode di dalam suatu *source files* yang identik atau memiliki kemiripan antara satu sama lain [3]. *Code clones* adalah redundansi dalam kode yang dapat menambah beban pekerjaan maintenance [4]. Semakin banyak *code clones*  pada suatu sistem perangkat lunak maka semakin banyak *Line of Codes* pada suatu sistem perangkat lunak yang harus dipelihara oleh seorang *developer*. Gambar 1 merupakan contoh dari *code clones* pada level fungsi.

*Code clones* memiliki tiga tipe yang dapat dibandingkan berdasarkan basis teks program yang disalin [4]. Tipe-tipe *code clones* tersebut yaitu :

* Tipe 1 merupakan salinan yang sama dari segmen kode tanpa ada modifikasi, kecuali perbedaan spasi dan komentar.
* Tipe 2 merupakan salinan dengan struktur sintaks, hanya ada perbedaan pada nama variabel, tipe dan function identifier.
* Tipe 3 merupakan salinan dari segmen kode dengan sedikit modifikasi struktur.

Akar munculnya *code clones* adalah dari *programmers* yang melakukan *forking*, *templating* dan *customization* [4]. *Forking* adalah menduplikasi yang digunakan untuk bootstrap pengembangan solusi serupa, dengan harapan bahwa evolusi kode akan terjadi secara independen, setidaknya dalam jangka pendek. *Templating* digunakan sebagai metode untuk langsung menyalin perilaku dari kode yang ada, tetapi mekanisme abstraksi yang sesuai tidak tersedia dalam jangka pendek. *Customization* terjadi ketika kode yang ada saat ini tidak cukup memenuhi serangkaian persyaratan yang baru.

*String Matching Algorithm*

Konsep suatu algoritma *string matching* adalah mencari satu atau beberapa pola string yang terdapat pada suatu teks yang banyak [13]. Algoritma string matching memindai teks menggunakan *window teks* sebagai pemeriksa kemunculan pola dengan membandingkan karakter window dengan karakter pola. Setelah mencapai pola yang cocok atau setelah tidak cocok, *windows* akan bergeser ke pola selanjutnya. Proses mencocokan dan mengeser *windows* akan terus diulang hingga akhir ujung teks.

* + 1. Algoritma *Ratcliff/Obershelp*

Algoritma *string matching* ini menghasilkan nilai persentase yang menunjukan *kesamaan* diantara dua string yang berbeda [14, 15]. Berikut adalah rumus dari algoritma *Ratcliff/Obershelp* :

()

Simbol dan merupakan *string* yang dibandingkan. merupakan jumlah karakter yang sama pada kedua pasangan *string*. Proses pencocokan dari algoritma ini yaitu :

* Pertama menemukan sub string terpanjang yang memiliki kesamaan dari string S1 dan S2 yang di sebut anchor. Nilai dari Km bertambah berdasarkan panjang dari anchor.
* Kemudian bagian yang tersisa dari string sebelah kiri dan kanan dari anchor harus diperiksa sebagai string-string yang baru (dengan kata lain mengulangi step 1).
* Proses tersebut diulangi sampai semua karakter dari string S1 dan S2 dianalisa.

Metode Pendekatan Token

Metode pendekatan token yaitu metode untuk membandingkan kode dengan mengubah *source codes* menjadi token [4]. Metode ini bertujuan untuk membandingkan secara baris-per-baris dari kode yang sudah ditransformasikan menjadi urutan - urutan token menggunakan sebuah *token parser*. *Token parser* yang digunakan bergantung dengan bahasa pemrograman yang dijadikan model [6]. Metode pendekatan token termasuk metode kelas berat karena memerlukan *language parser* khusus untuk setiap bahasa pemrograman berbeda. Tujuan dari transformasi kode menjadi urutan token adalah untuk mendeteksi code clones yang memiliki sintaks berbeda tetapi memiliki makna yang sama [5]. Mewakili kode sebagai urutan token memungkinkan untuk mendeteksi code clones dengan perbedaan struktur baris, yang tidak dapat dideteksi metode perbandingan baris.

1. Sistem yang Dibangun

Pada penelitian ini, sistem yang dirancang memiliki tujuan untuk mendeteksi *code clones* pada level fungsi. Hasil *code clones* yang terdeteksi sistem akan dievaluasi dan akan dibandingkan dengan hasil tools pendeteksi *code clones* penelitian lain, yaitu Simcad [7], Nicad [8], dan MeCC [9] . Sistem memiliki enam tahap utama yaitu *input source codes*, *Data Pre-Processing*, *Function Searching*, *Tokenization*, dan *Evaluation*. Gambar 2 merupakan alur dari sistem pendeteksi *code clones* secara otomatis menggunakan pendekatan token.

|  |
| --- |
| Gambar Flowchart Sistem |
|  |

*Input Source Codes*

Tahap *Input Source Codes* bertujuan untuk membaca file *source codes* sebagai dataset yang akan dideteksi *code clones* oleh sistem. Gambar 3 merupakan contoh source codes yang dapat diterima sistem. Pada penelitian ini, dataset yang digunakan berasal dari hasil penelitian Krutz [3]. Dataset berupa *source codes* berbahasa C dari aplikasi Apache[[2]](#footnote-3), Postgre[[3]](#footnote-4), dan Python[[4]](#footnote-5). Pada penelitian Krutz juga terdapat 1536 pasangan fungsi untuk dideteksi apakah terdapat *code clones* atau tidak terdapat *code clones*.



Gambar Contoh *Source Codes*

*Data Pre-Processing*

Tahap *data pre-processing* bertujuan untuk proses tokenisasi tidak mengubah kode yang tidak penting dan proses pendeteksi *code clones* agar berjalan lebih baik. Teknik *pre-processing* yang digunakan dalam penilitian ini diimplementasikan melalui pustaka *Natural Language ToolKit*[[5]](#footnote-6) (NLTK). Gambar 4 adalah alur dari proses *data pre-processing*. Terdapat dua tahap penting dalam *data pre-processing* yaitu *remove* *white spaces* dan *remove comments*.

Diagram

Description automatically generated

Gambar Flowchart Data Pre-Processing

* + 1. *Remove White Spaces*

Tahap ini berfokus terhadap penghapusan indentasi di setiap baris source codes dan spasi tambahan yang tidak diperlukan. Tahap ini memiliki tujuan untuk menghapus data yang dapat merusak pada proses transformasi token sehingga data mudah untuk diproses pada tahap tokenisasi. Proses ini menghasilkan *source codes* tanpa indentasi seperti pada Gambar 5.



Gambar Contoh *Source Codes* Hasil Proses *Remove White Spaces*

* + 1. *Remove Comments*

Proses ini dilakukan untuk menghapus partikel redundan seperti komentar pada *source codes*. Tujuan proses ini adalah mengurangi data yang akan ditransformasikan menjadi token agar proses mendeteksi *code clones* berjalan lebih lancar. Pada bahasa pemrograman C terdapat dua tipe komentar yang dihapus oleh sistem, yaitu :

* Pertama, komentar yang diawali dengan simbol “//”.
* Kedua, komentar yang diawali dengan simbol “/\*” dan diakhiri dengan “\*/”.

Komentar tipe pertama menghapus saat simbol tersebut ditemukan hingga baris baru ditemukan, menghapus seluruh string yang ada setelah symbol tersebut ditemukan. Komentar tipe kedua menghapus ketika simbol “/\*” ditemukan hingga simbol “\*/” ditemukan, menghapus seluruh string diantara kedua simbol tersebut. Proses ini menghasilkan *source codes* tanpa komentar seperti pada Gambar 6.



Gambar *Source Codes* Hasil Proses *Remove Comments*

*Function Searching*

Tahap *function searching* berfungsi untuk mencari seluruh fungsi dan prosedur dalam *source codes* untuk dicatat menjadi sebuah kamus - kamus fungsi. Sistem akan mencatat nama file, nama fungsi, dan kode fungsi untuk setiap fungsi yang ditemukan pada *source codes*. Pada penelitian ini, sistem mendeteksi *code clones* pada level fungsi sehingga setiap fungsi akan dibandingkan dengan fungsi lain. Terdapat peraturan untuk mencari dan mencatat fungsi dan prosedur :

* Pertama, diawali dengan tipe data atau “void” dalam pada suatu baris.
* Kedua, terdapat simbol “(“ dan diakhiri dengan simbol “)”. Jika tidak ada simbol tersebut maka bukan termasuk fungsi melainkan variabel. Sistem akan mencatat string sebelum tanda “(“ sebagai nama fungsi tersebut.
* Ketiga, terdapat simbol “{“ dan diakhiri dengan simbol “}”. Sistem akan mencatat seluruh kode diantara kedua symbol tersebut sebagai kode fungsi tersebut.

Jika ketiga dari peraturan tersebut terpenuhi maka sistem akan menambahkan fungsi tersebut ke sebuah list yang menyimpan seluruh fungsi. Tahap ini menghasilkan list seluruh fungsi yang bernama *list of functions*.

*Tokenization*

Pada tahap ini, sistem mengganti seluruh kode fungsi menjadi urutan – urutan token. Tujuan dari tahap ini adalah untuk memecah kode – kode menjadi beberapa partikel [16]. Transformasi kode menjadi token berguna untuk menyamakan porsi kode dengan variabel yang berbeda nama sehingga dapat terdeteksi sebagai clone. Aturan tranformasi kode menjadi token :

* Seluruh *identifier variabel*, tipe data, konstanta, dan nomor menjadi token dengan lambang “T”
* Partikel *language construct* (for, to, do, while, if, else, dsb) menjadi token dengan lambang masing – masing partikel.
* Lambang matematika dan tanda baca menjadi token dengan lambang simbol masing - masing.

Gambar 7 merupakan contoh hasil tranformasi kode menjadi beberapa urutan token. Proses ini menghasilkan list dari seluruh fungsi dengan kode fungsi yang sudah ditransformasikan menjadi token.



Gambar *Source Codes* Hasil Proses *Tokenization*

*Inter-Functions Similarity Calculation*

Tahap *inter-functions similarity calculation* berfungsi untuk menghitung nilai kemiripan (*similarity*) antar dua pasangan fungsi yang berbeda. Dikarenakan kode fungsi sudah menjadi token yang berurutan, sistem menghitung nilai kemiripan kedua pasangan fungsi menggunakan algoritma *ratcliff/obershelp*. Algoritma *ratcliff/obershelp* yang digunakan dalam penelitian ini diimplementasikan melalui pustaka *textdistance*[[6]](#footnote-7). Proses ini menghasilkan list seluruh pasangan fungsi dengan nilai kemiripanya masing – masing.

*Format Output*

Pada tahap ini, sistem mengeluarkan seluruh informasi tentang pasangan fungsi, rasio kemiripan (*similarity*) pasangan fungsi, dan *threshold*. *Treshold* adalah parameter nilai batas bawah yang digunakan untuk menentukan nilai *true* atau *false* dari suatu rasio. Nilai rasio kemiripan (*similarity*) pasangan fungsi yang bernilai lebih besar atau sama dengan parameter nilai *threshold* mendapatkan nilai *true*, dengan kata lain disebut *code clones*. Sebaliknya, nilai rasio kemiripan (*similarity*) pasangan fungsi yang bernilai lebih kecil daripada parameter nilai *threshold* mendapatkan nilai *false*. Pada penelitian ini, diberikan nilai 0.1 untuk parameter *threshold* yang terkecil kemudian bertambah dengan nilai 0.1 secara konsisten hingga mencapai nilai 1, sehingga terdapat 10 model untuk setiap pasangan fungsi. Penamaan function yang dikeluarkan sistem ditambah dengan nama file sebagi prefix yang dipisah oleh tanda “..”, misalkan fungsi A terdapat pada file bernama X sehingga penamaan fungsi menjadi “X..A()”. Tabel 1 adalah contoh keluaran dari tahap *format output*.

Tabel Contoh Hasil Proses *Format Output*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Pasangan Fungsi** | **Similarity** | **Threshold** | | | | | | |
| **>=0.1** | **>=0.2** | **>=0.3** | **…** | **>=0.8** | **>=0.9** | **>=1** |
| 1 | X..A()-  Y..B() | 0.81 | F | F | F | … | T | T | T |
| 2 | X..A()-  Y..C() | 0.95 | F | F | F | … | F | T | T |

*Evaluation*

Tahap *evaluation* merupakan tahap terakhir pada sistem yang bertujuan untuk menguji performansi sistem dengan mengkalkulasikan *F1 scores*. Pada hasil penelitian Krutz [3], ditemukan sebanyak 66 *code clones* yang dijadikan sebagai *golden standard* untuk menghitung *F1 scores* pada penelitian ini. Sebelum menghitung *F1 scores*, sistem membuat *confusion matrix* untuk menghitung *precision*, dan *recall*. Tabel 2 adalah format *confusion matrix*. Terdapat empat kategori hasil dalam confusion matrix : *true positive* (TP) jika pasangan fungsi *code* *clones* yang ditemukan oleh sistem merupakan *code clones* yang aktual, *true negative* (TN) jika pasangan fungsibukan *code clones* yang ditemukan oleh sistem merupakan pasangan fungsi biasa, *false positive* (FP) jika pasangan fungsi *code clones* yang ditemukan oleh sistem bukan merupakan *code clones*, dan *false negative* (FN) jika pasangan fungsi bukan *code clones* ditemukan oleh sistem yang sebetulnya *code clones* actual.

Tabel *Confusion Matrix*

|  |  |  |  |
| --- | --- | --- | --- |
|  | | ***Predicted*** | |
| ***Clones*** | ***Not Clones*** |
| ***Actual*** | ***Clones*** | *True Positive* (TP) | *False Negative* (FN) |
| ***Not Clones*** | *False Positive* (FP) | *True Negative* (TN) |

Setelah mendapatkan hasil jumlah keempat kategori dalam *confusion matrix*, *precision* () dan *recall* () dapat dicari menggunakan rumus :

()

Rata – rata dari *precision* dan *recall* dapat dikatakan sebagai *F-score* [17] :

(3)

Sehingga *F1-score* dapat dihitung dengan rumus :

()

1. Evaluasi

Hasil Penelitian

Pada penelitian ini, diberikan nilai 10% untuk parameter *threshold* yang terkecil kemudian bertambah nilai 10% secara konsisten hingga mencapai nilai 100%, sehingga terdapat 10 model untuk setiap tipe dataset. Setiap model akan dievaluasi dengan dihitung nilai *F1 Score*. Model yang memiliki nilai *F1 Score* tertinggi merupakan model yang paling optimal. Hasil evaluasi *F1 Scores* penelitian ini ditunjukan di Tabel 3.

Tabel Hasil Evaluasi *F1 Score*s Penelitian *Code Clone*s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Threshold** | **F1 Score** | | |
| **Apache** | **Postgre** | **Python** |
| 1 | 10% | 0,1 | 0,09 | 0,05 |
| 2 | 20% | 0,1 | 0,1 | 0,06 |
| 3 | 30% | 0,12 | 0,1 | 0,08 |
| 4 | 40% | 0,16 | 0,12 | 0,13 |
| 5 | 50% | 0,23 | 0,15 | 0,22 |
| 6 | 60% | 0,44 | 0,22 | 0,38 |
| 7 | 70% | 0,7 | 0,32 | 0,69 |
| 8 | 80% | 0,9 | 0,48 | **0,89** |
| 9 | 90% | **0,94** | **0,55** | 0,42 |
| 10 | 100% | 0,83 | 0,37 | 0 |

Hasil evaluasi *F1 Score*s tertinggi terdapat pada threshold dengan nilai 80%, pada dataset Python, dan 90%, pada dataset Apache dan Postgre. Nilai *F1 Score* tertinggi pada dataset Apache, Postgre dan Python ditandai dengan *underline* dengan nilai secara berurutan adalah “0,94”, “0,55” dan “0,89”.

Analisis Hasil Pengujian

Hasil evaluasi *F1 Scores* pada penelitian ini dibandingkan dengan hasil *F1 Scores*  tools pendeteksi *code clones* Nicad, MeCC dan Simcad. Tabel 4 merupakan hasil perbandingan *F1 Scores*. Kolom Token menggambarkan nilai *F1 Scores* dari metode yang diangkat pada penelitian ini. Penelitian ini memberikan hasil evaluasi *F1 Scores* tertinggi pada dataset Apache, Postgre dan Python dibanding dengan *tools* pendeteksi *code clones* lain dengan nilai secara berurutan “0,94”, “0,55” dan “0,89”.

Tabel Perbandingan Hasil Evaluasi F1 Score *Code Clones* Tertinggi Setiap Dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **F1 Score** | | | |
| **Nicad** | **MeCC** | **Simcad** | **Token** |
| **Apache** | 0,91 | 0,91 | 0,7 | **0,94** |
| **Postgre** | 0,21 | 0,15 | 0 | **0,55** |
| **Python** | 0,63 | 0,58 | 0,68 | **0,89** |

Diskusi Penelitian

Pendeteksi code clones menggunakan pendekatan token hanya melihat dari sisi kemiripan struktur/bentuk dari sebuah *LoC* atau fungsi, tidak melihat proses *LoC* atau fungsi tersebut. Tabel 5 memberikan contoh kasus dimana struktur fungsi \_outfieldselect() serupa dengan struktur pengkodeans fungsi \_outinnerindexscaninfo(), tetapi bukan termasuk *code clones*. Pasangan fungsi pada Tabel 5 bukan termasuk pasangan *code clones* karena kode menjalankan proses berbeda. Pada proses fungsi \_outfieldselect() memanggil fungsi WRITE\_INT\_FIELD() sedangkan proses fungsi \_outinnerindexscaninfo() memanggil WRITE\_BOOL\_FIELD() yang merupakan proses berbeda. Sistem pendeteksi *code clones* dengan menggunakan metode pendekatan token mendeteksi kasus pada Tabel 5 sebagai *code clones* sehingga menurunkan nilai *F1-score*. Pada *source codes* dataset Postgre terdapat banyak kasus seperti pada Tabel 5 sehingga nilai *F1-score* memiliki nilai terkecil dari dataset lain.

Tabel Kasus False Positive

|  |  |  |
| --- | --- | --- |
| **Dataset** | Postgre | |
| **Nama Files** | outfuncs | outfuncs |
| **Nama Fungsi** | \_outfieldselect() | \_outinnerindexscaninfo() |
| **Source Codes** | static void  \_outFieldSelect(StringInfo str, FieldSelect \*node)  {  WRITE\_NODE\_TYPE("FIELDSELECT");  WRITE\_NODE\_FIELD(arg);  WRITE\_INT\_FIELD(fieldnum);  WRITE\_OID\_FIELD(resulttype);  WRITE\_INT\_FIELD(resulttypmod);  } | static void  \_outInnerIndexscanInfo(StringInfo str, InnerIndexscanInfo \*node)  {  WRITE\_NODE\_TYPE("INNERINDEXSCANINFO");  WRITE\_BITMAPSET\_FIELD(other\_relids);  WRITE\_BOOL\_FIELD(isouterjoin);  WRITE\_NODE\_FIELD(cheapest\_startup\_innerpath);  WRITE\_NODE\_FIELD(cheapest\_total\_innerpath);  } |
| **Tokenized Source Codes** | T T  T ( T T, T  \* T )  {  T ( T  ) ;  T ( T ) ;  T ( T ) ;  T ( T ) ;  T ( T ) ;  } | T T  T ( T T , T  \* T )  {  T ( T  ) ;  T ( T ) ;  T ( T ) ;  T ( T ) ;  T ( T ) ;  } |
| **Rasio Similarity** | 100% | |

1. Kesimpulan

Hasil nilai *F1 Score* pada penelitian ini bernilai tinggi dibanding dengan *tools* pendeteksi *code clones* Nicad, MeCC, Simcad. Oleh karena itu, implementasi sistem pendeteksi *code clones* pada penelitian ini cukup akurat dalam mendeteksi *code clones*. Saran untuk penelitian berikutnya yaitu mengembangkan sistem pendeteksi *code clones* yang mampu mendeteksi pasangan fungsi *code clones* dengan mengobservasi proses atau hasil dari fungsi tersebut.

**Daftar Pustaka**

|  |  |
| --- | --- |
| [1] | R. S. Pressman, "BAB 14 Konsep-konsep Kualitas," in *Rekayasa Perangkat Lunak: Pendekatan Praktisi, Edisi 7*, Yogyakarta, Penerbit ANDI, 2012, pp. 482-504. |
| [2] | R. Malhotra and C. Anuradha, "Software Maintainability: Systematic Literature Review and Current Trends," *International Journal of Software Engineering and Knowledge Engineering,* vol. 26, pp. 1221-1253, 2016. |
| [3] | D. E. Krutz and W. Le, "A Code Clone Oracle," *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014,* 2014. |
| [4] | T. Mens and S. Demeyer, "Chapter 2 Identifying and Removing Software Clones," in *Software evolution*, Berlin, Springer, 2010, pp. 15-36. |
| [5] | T. Kamiya, S. Kusumoto and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *Software Engineering, IEEE Transactions on,* vol. 28, pp. 654-670, 08 2002. |
| [6] | O. Onuoha, "Detecting Code Clones: A review," *arXiv preprint arXiv:1605.02661,* 2016. |
| [7] | M. S. Uddin, C. K. Roy and K. A. Schneider, SimCad: An extensible and faster clone detection tool for large scale software systems, 2013 21st International Conference on Program Comprehension (ICPC), 2013, pp. 236-238. |
| [8] | C. K. Roy and J. R. Cordy, NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization, 2008 16th IEEE International Conference on Program Comprehension, 2008, pp. 172-181. |
| [9] | H. Kim, Y. Jung, S. Kim and K. Yi, MeCC: memory comparison-based clone detector, 2011 33rd International Conference on Software Engineering (ICSE), 2011, pp. 301-310. |
| [10] | B. S. Baker, "A program for identifying duplicated code," *Computing Science and Statistics,* 1992. |
| [11] | S. Ducasse, M. Rieger and S. Demeyer, "A language independent approach for detecting duplicated code," *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360),* pp. 109-118, 1999. |
| [12] | "Maintainability," [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/57-maintainability. [Accessed 20 November 2020]. |
| [13] | K. Al-Khamaiseh and S. ALShagarin, "A Survey of String Matching Algorithms," *International Journal of Engineering Research and Applications,* vol. 4, pp. 144-156, August 2014. |
| [14] | Y. L. Joane, A. Sinsuw and A. Jacobus, "Rancang Bangung Aplikasi Deteksi Kemiripan Dokumen Teks," *E-Journal Teknik Informatika,* vol. 11, no. 1, 2017. |
| [15] | L. W. Aritonang, "Rancang Bangun Aplikasi Deteksi Kemiripan Dua Gambar Menggunakan Algoritma Ratcliff/Obershelp," *Journal of Computer System and Informatics (JoSYC),* vol. 1, no. 3, pp. 192-198, 2020. |
| [16] | F. M. Barcala, J. Vilares, M. A. Alonso, J. Grana and M. Vilares, "Tokenization and proper noun recognition for information retrieval," *Proceedings. 13th International Workshop on Database and Expert Systems Applications,* pp. 246-250, 2002. |
| [17] | C. Goutte and E. Gaussier, "A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation," *Lecture Notes in Computer Science,* vol. 3408, pp. 345-359, 04 2005. |

1. http://www.se.rit.edu/~dkrutz/cloneoracle/ [↑](#footnote-ref-2)
2. http://www.apache.org [↑](#footnote-ref-3)
3. http://www.postgresql.org [↑](#footnote-ref-4)
4. http://www.python.org [↑](#footnote-ref-5)
5. https://www.nltk.org/ [↑](#footnote-ref-6)
6. https://pypi.org/project/textdistance/ [↑](#footnote-ref-7)