Jurnal Tugas Akhir Fakultas Informatika 12-2020

Otomasi Penelusuran Kebutuhan ke Kode Program menggunakan TF-IDF

Said R. K. Bahasyim 1, Sri Widowati 1, Jati H. Husen 1*

[1]School of Computing, Telkom University
Jl. Telekomunikasi no. 1, Kota Bandung, Jawa Barat 40257
[1]saidrahadi@student.telkomuniversity.ac.id, [1]sriwidowati@telkomuniversity.ac.id, *
jatihusen@telkomuniversity.ac.id

Abstrak

Salah satu faktor penting yang menentukan kualitas perangkat lunak adalah memastikan bahwa kebutuhan (requirement) dapat ditelusuri ke dalam kode program. Pada umumnya mengelola ketertelusuran kebutuhan dilakukan secara manual sehingga membutuhkan usaha yang besar dan ketelitian yang cukup tinggi. Salah satu metode yang dapat digunakan untuk penelusuran kebutuhan ke kode program adalah metode frekuensi-inverse document frequency (TF-IDF). Dalam penelitian ini metode TF-IDF diimplementasikan sebagai kakas, dan diujikan dengan beberapa kasus dari dataset RETRO.NET. Selanjutnya performansi metode TF IDF akan diukur berdasarkan 3 indikator yaitu accuracy, precision, dan recall. Beberapa modifikasi dilakukan pada implementasi metode TF-IDF untuk memberikan hasil yang lebih baik.

Kata kunci : Requirement traceability, TF-IDF, Information retrieval

Abstract

One of the important factors determining the quality of the software is ensuring that the requirements (requirements) can be traced into the program code. On the shot of interest is done manually, so it requires a lot of effort and high accuracy. One method that can be used to tracing program code requirements is the frequency-inverse document frequency (TF-IDF) method. In this research the TF-IDF method is implemented as a tool, and tested with several cases from the RETRO.NET dataset. Furthermore, the performance of the TF-IDF method will be based on 3 indicators accuracy, precision, and recall. Several modifications were made to the implementation of the TF-IDF method to provide better results.

Keywords: Requirement traceability, TF-IDF, Information retrieval

1. Pendahuluan

Latar Belakang

Tracing Requirements ke dalam *source code* merupakan aspek penting dari rekayasa perangkat lunak. *Traceability* memberikan informasi yang bermanfaat untuk menganalisis setiap biaya dan manfaat *requirements* yang diterapkan baik dalam tahap pengembangan dan pemeliharaan. Selain itu, traceability links menyediakan metode untuk *requirements* perangkat lunak agar sesuai dengan spesifikasi, kontrak, atau peraturan. Di tingkat organisasi, mengelola *links* tersebut bertindak sebagai investasi untuk meningkatkan kualitas produk, mengurangi biaya pemeliharaan, dan memfasilitasi penggunaan kembali[1, 2]. Namun, mengelola secara manual *requirements traceability* tersebut dapat menjadi tantangan.

Mengelola *traceability* secara manual sulit dicapai untuk beberapa tantangan. *Traceability* proyek cenderung menurun selama evolusi dan pemeliharaan perangkat lunak sebagai perubahan *source code*[3]. Artefak perangkat lunak, termasuk *requirements specification* perangkat lunak, cenderung ketinggalan zaman antara interval pendek *maintenance* dan evolusi, yang mengarah ke *links* yang salah antara *requirements* lama dan *source code* yang baru diterapkan. Selain itu, identifikasi manual dari *traceability links* rentan terhadap kesalahan manusia[4]. Kondisi ini menyebabkan diperlukannya otomatisasi dalam mengidentifikasi *traceability links*.

Salah satu cara untuk mengotomatiskan *traceability links* penelusuran adalah dengan menerapkan metode pengambilan informasi, seperti istilah *term frequency-inverse document frequency* (TF-IDF), antara requirements perangkat lunak dan *source code*. TF-IDF adalah statistik numerik yang mencerminkan betapa pentingnya sebuah kata bagi sebuah dokumen dalam suatu koleksi. TF-IDF adalah salah satu metode pencarian informasi yang menonjol karena sederhana dan efektif[5].

Topik dan Batasannya

Penelitian ini menitikberatkan pada implementasi metode TF-IDF untuk identifikasi *traceability link* antara *requirement* dengan *source code* pada perangkat lunak. Hasil dari perhitungan TF-IDF terhadap beberapa kasus dan dataset selanjutnya akan dianalisis, sebagai dasar melakukan modifikasi terhadap tahapan dalam *requirements traceability* antara *requirement* dengan *source code* pada perangkat lunak. Untuk mencapai tujuan tersebut, penulis menerapkan metode dan mengujinya menggunakan kumpulan data. Untuk evaluasi metode TF-IDF.

Tujuan

Tujuan yang tertera digunakan untuk memandu penulis dalam penelitian, yang dimana tujuannya seperti berikut:

- Bagaimana cara melakukan recovery antara requirement dengan source code dengan menggunakan TF-IDF?
- Mengukur accuracy, precision, dan recall dari metode TF-IDF yang digunakan.

Penelitian penulis berkontribusi pada badan pengetahuan rekayasa perangkat lunak, terutama bidang pengetahuan teknik *requirements*, untuk memahami cara mengelola *requirement traceability* dengan lebih efisien.

Organisasi Tulisan

Sisa dari makalah ini dibangun sebagai berikut: Bab 2 menjelaskan pengetahuan kita tentang teori latar belakang dan pekerjaan terkait penelitian ini. Bab 3 menjelaskan penelitian yang telah penulis lakukan. Bab 4 menyajikan hasil penelitian penulis dan membahasnya. Akhirnya, penulis menyimpulkan penelitian ini di Bab 5 dengan beberapa karya mendatang.

2. Studi Terkait

2.1 *Software Requirements Traceability Recovery*

Traceability adalah kemampuan untuk melacak informasi dari dua artefak yang berbeda[6, 7]. *Traceability* sangat penting, terutama untuk proyek perangkat lunak besar, karena menyediakan informasi tentang hubungan antara dua artefak dari fase rekayasa perangkat lunak yang berbeda[8]. Informasi tersebut sangat penting dalam analisis dampak perubahan untuk aktivitas pemeliharaan dan evolusi.

Hubungan dua artefak dalam *traceability* disebut sebagai *traceability links*. *Link* tersebut dapat diidentifikasi dengan menggunakan metode ortogonal, di mana pengambilan informasi merupakan bagian dari[8]. *Traceability recovery* mengacu pada kemampuan mendapatkan kembali *traceability links* dari dua artefak perangkat lunak yang sudah ada.

2.2 *Term Frequency – Inverse Document Frequency (TF-IDF)*

Term frequency-inverse document frequency (TF-IDF) adalah statistik numerik yang menunjukkan betapa pentingnya sebuah kata bagi sebuah dokumen dalam sebuah koleksi[9, 10]. Dalam penelitian ini kata-kata pada *requirements*, sedangkan dokumen merupakan singkatan dari *source code file*. TF-IDF terdiri dari dua persamaan *term frequency* (TF) dan *inverse document frequency* (IDF). *Term Frequency* merupakan istilah yang ditentukan oleh jumlah kemunculan sebuah kata dalam dokumen[11, 12]. yang dihitung dengan rumus berikut:

$$TF = \frac{Total\ appearance\ of\ a\ word\ in\ document}{Total\ words\ in\ a\ document} \quad (1)$$

Inverse document frequency adalah ukuran kesamaan kata di semua dokumen yang ada. IDF dapat dihitung menggunakan rumus berikut:

$$IDF = log\frac{All\ Document\ Number}{Document\ Frequency} \quad (2)$$

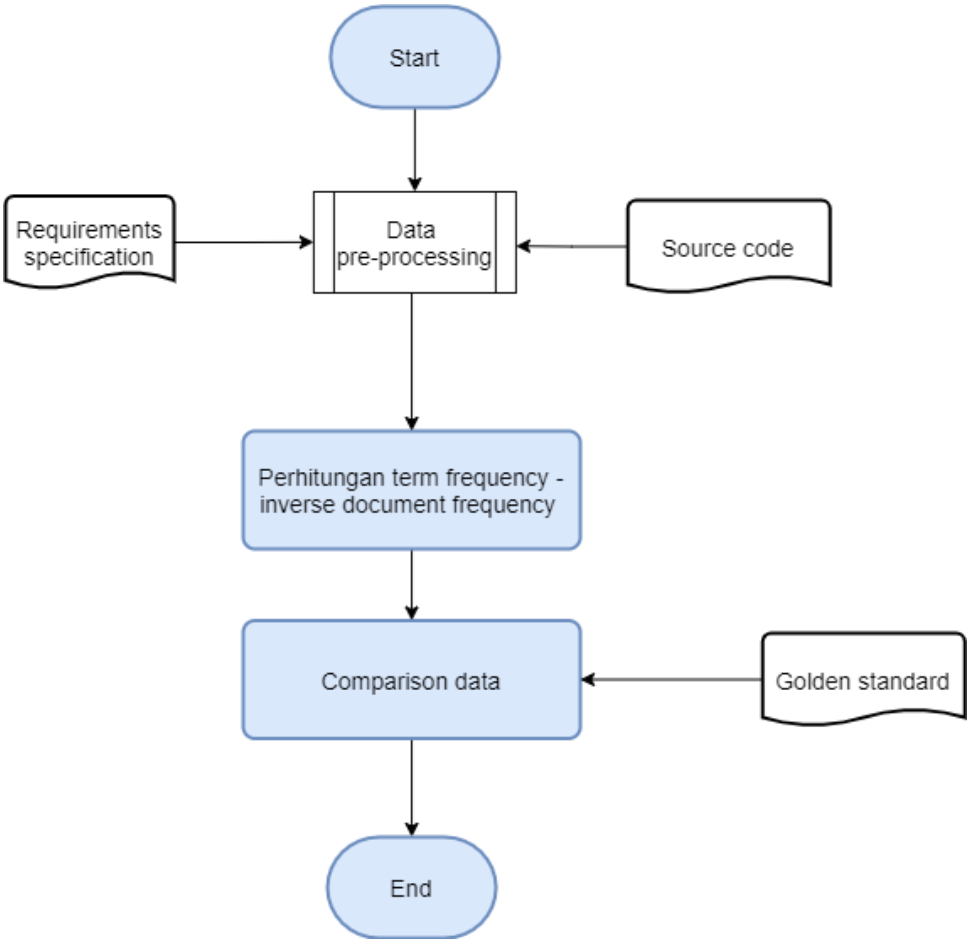
Terakhir, baik TF maupun IDF dihitung untuk mencari nilai TF-IDF dengan menggunakan rumus berikut:

$$TF - IDF = TF\ X\ IDF \quad (3)$$

Nilai TF-IDF berkisar dari 0 hingga 1, di mana 0 menunjukkan sebuah kata tidak memiliki kepentingan sementara 1 menunjukkan sebuah kata memiliki kepentingan total di dalam dokumen.

3. Sistem yang Dibangun

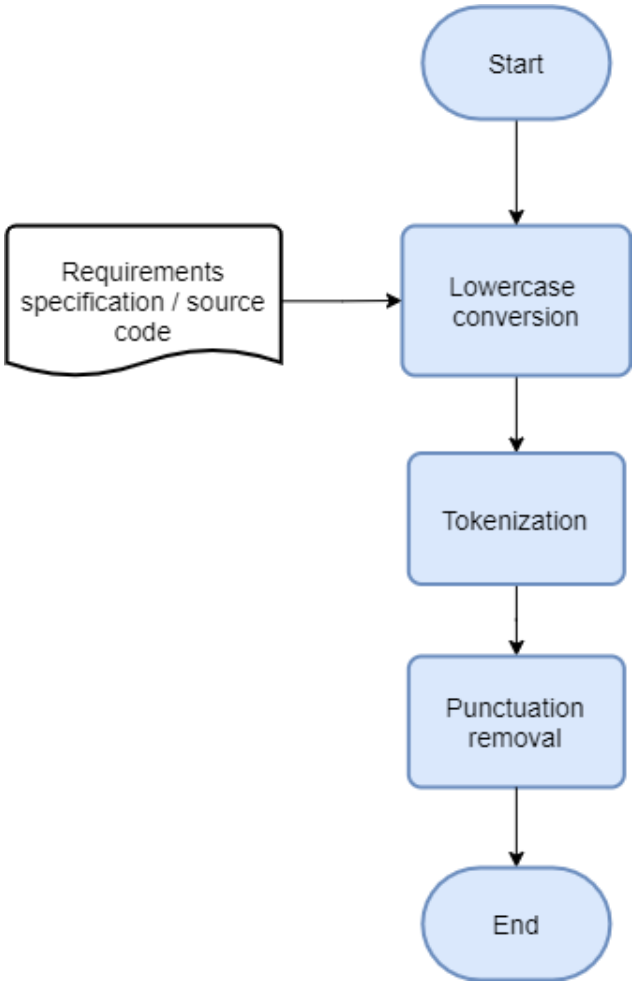
Penelitian penulis bertujuan untuk mengevaluasi metode TF-IDF sebagai otomatisasi untuk *recovering traceability* antara *requirements* perangkat lunak dan *source code*. Untuk mencapai ini, penulis mengimplementasikan metode TF-IDF dan mengeksekusinya dengan dataset RETRO.NET[13] untuk menemukan accuracy, precision, dan recall. Gambaran umum pendekatan penulis dapat dilihat pada gambar 1.



Gambar 1. Alur kerja Metode Penelitian

3.1 Data Pre-Processing

Diperlukan *pre-processing* data sebelum pemrosesan TF-IDF agar dapat memproses *requirements* dan *source code* dengan baik. Penulis mengikuti teknik *pre-processing* seperti pada[14]. Penerapan teknik pra-pemrosesan penulis dilakukan untuk *requirements* dan *source code*. Alur kerja keseluruhan dari pra-pemrosesan diilustrasikan pada gambar 2. Semua teknik *pre-processing* yang digunakan dalam penelitian ini diimplementasikan melalui pustaka Natural Language ToolKit (NLTK) dari Python. Sumber-sumber yang dirujuk tidak menjelaskan proses *pre-processing* yang dilakukan [15, 16, 17], Oleh karena itu pada penelitian ini digunakan *pre-processing* yang umum digunakan seperti pada penelitian Fakry et. al. [14]



Gambar 2. Alur Kerja Pra-Pemrosesan

3.1.1 *Lowercase conversion*

Lowercase conversion bertujuan untuk memastikan bahwa semua kata ditulis dalam huruf kecil. Teknik ini mengubah semua kata yang menggunakan huruf besar menjadi huruf kecil. Teknik pra-pemrosesan ini diperlukan untuk memastikan bahwa tidak ada pemeriksaan kesamaan yang diabaikan karena perbedaan kapitalisasi.

3.1.2 *Tokenization*

Tokenization adalah proses untuk mengubah string menjadi potongan yang lebih kecil yang disebut token. *Tokenization* membagi string kalimat menjadi beberapa token kata. Teknik ini digunakan untuk menyusun data menjadi bentuk yang dapat diolah dengan metode TF-IDF, dalam proses *tokenization* ini terdapat dua proses yang berbeda.

3.1.3 *Tokenization* berbeda tahapan

Proses *tokenization* disini sama halnya dengan tahapan pertama terlihat pada gambar 2 bagian kiri yang merupakan proses tahapan dasar yang penulis ikuti seperti pada[14]. *Tokenization* pada gambar 2 bagian kanan terlihat bahwa proses tokenization ditukar tahapan dengan *lowercase conversion*, gambar 2 bagian kanan merupakan proses yang penulis rancang sendiri agar mendapatkan hasil yang berbeda.

3.1.3 *Removing punctuation*

Dalam teknik ini, penulis menghapus semua tanda baca dan simbol lainnya di setiap token. Teknik *pre-processing* ini diterapkan karena tanda baca dan simbol *Unicode* lainnya dapat menimbulkan masalah

pada proses TF-IDF, terutama untuk *source code* yang memuat simbol dalam jumlah yang signifikan. Teknik ini menghapus tanda baca seperti titik, koma, dan simbol lain dari setiap token.

3.2 Implementation of Term Frequency – Inverse Document Frequency (TF-IDF) Method

Implementasi TF-IDF dilakukan untuk memudahkan percobaan. Penulis menerapkan metode TF-IDF dengan Python menggunakan Google Colab. URL *source code* implementasi penulis dapat ditemukan di repositori yang dinyatakan dalam data dan ketersediaan program komputer.

3.3 Evaluation Dataset

Untuk evaluasi, penulis menguji penerapan metode TF-IDF penulis dengan dataset RETRO.NET[13]. Dataset RETRO.NET terdiri dari 66 *requirements specifications* dan 118 *source code files* yang ditulis dalam bahasa C # untuk platform .NET. Dataset RETRO.NET juga menyediakan matriks *traceability* yang dapat digunakan sebagai standar emas untuk mengukur accuracy, precision, dan recall atas implementasi TF-IDF penulis dalam penelitian ini.

3.4 Evaluation

Evaluasi dalam penelitian penulis melakukan dengan konstruksi *confusion matrix* dengan membandingkan *traceability matrix* dalam dataset RETRO.NET dengan hasil implementasi penulis. Konstruksi *confusion matrix* terdiri dari 4 kriteria: *true positive* (TP), *true negative* (TN), *false positive* (FP), dan *false negative* (FN). Kriteria tersebut selanjutnya digunakan untuk menghitung *accuracy, precision, dan recall* dengan menggunakan rumus-rumus berikut[18]:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} * 100\% \quad (3)$$

$$Precision = \frac{TP}{FP+TP} * 100\% \quad (4)$$

$$Recall = \frac{TP}{FN+TP} * 100\% \quad (5)$$

Accuracy, precision, dan recall yang dihasilkan kemudian akan digunakan untuk menganalisis keberhasilan metode. Penulis juga akan mengevaluasi setiap hasil *pre-processing* untuk melihat apakah ada teknik *pre-processing* yang berdampak negatif pada hasil.

4. Evaluasi

4.1 *Results*

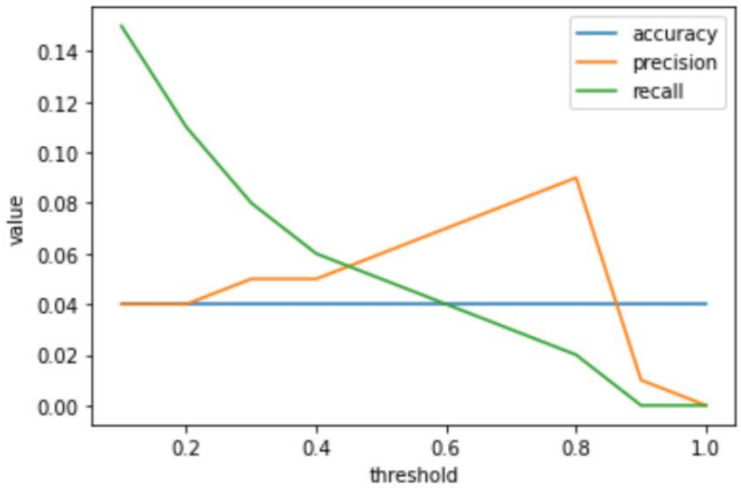
Tabel I menunjukkan hasil penelitian penulis. *Threshold* adalah nilai minimum TF-IDF agar *traceability links* dianggap sebagai hasil positif. Sel diwarnai dengan warna biru untuk nilai tertinggi, abu-abu untuk nilai terendah, dan hijau untuk nilai konsisten di seluruh *threshold*. Nilai *precision* tertinggi diperoleh pada sekitar nilai *threshold* 0,8 dengan 0,09 sedangkan yang terendah berada pada *threshold* 1 dengan 0. Nilai *recall* tertinggi pada nilai *Threshold* batas 0,1 dengan 0,1 dan diperoleh 0 pada nilai *threshold* batas 0,9. dan 1. Terakhir, keakuratannya sama di seluruh *threshold* yang berbeda dengan nilai 0,04.

TABLE I
EVALUATION RESULTS

THRESHOLD	ACCURACY	PRECISION	RECALL
0.1	0.04	0.04	0.15
0.2	0.04	0.04	0.11
0.3	0.04	0.05	0.08
0.4	0.04	0.05	0.06
0.5	0.04	0.06	0.05

0.6	0.04	0.07	0.04
0.7	0.04	0.08	0.03
0.8	0.04	0.09	0.02
0.9	0.04	0.01	0.0
1	0.04	0.0	0.0

Nilai *accuracy, precision, dan recall* di seluruh *threshold* yang berbeda ditunjukkan pada gambar 3. Nilai *accuracy* konsisten terlepas dari nilai *threshold*-nya. *Precision* menunjukkan nilai positif dengan kenaikan nilai *threshold* hingga nilai *threshold* 0,8 sebelum menurun hingga mencapai nol pada threshold 1. Terakhir, *recall* menunjukkan nilai negatif dengan kenaikan nilai *threshold* batas.



Gambar 3. Hasil

4.2 *Discussion*

Berdasarkan nilai *accuracy, precision, dan recall* tersebut, terlihat bahwa implementasi TF-IDF penulis menunjukkan hasil yang kurang baik. Nilai *accuracy* pada 0,04 sangat rendah untuk metode yang akan dianggap sebagai solusi yang dapat diandalkan untuk mendapatkan *traceability* antara *requirements* perangkat lunak dan *source code*. Dengan pemikiran tersebut, penulis perlu menganalisis hasil penulis lebih lanjut untuk menemukan penyebab dari hasil yang buruk.

Berdasarkan analisis lebih lanjut setelah evaluasi, penulis menemukan bahwa ada penyebab kegagalan dengan urutan *pre-processing*. Urutan penulis mengikuti implementasi *pre-processing* untuk jenis dokumen umum. Sementara *pre-processing* ini berfungsi untuk *requirements*, itu tidak berfungsi untuk *source code*. Karena *source code* mengikuti gaya penulisan yang berbeda dari dokumen umum, urutan *pre-processing* yang berbeda diperlukan untuk dapat mengimplementasikan TF-IDF dalam *source code*. Contoh kegagalan *pre-processing* dalam source code ditunjukkan pada Tabel II.

TABLE II
SAMPLE OF TOKENIZING FAILURE

No	Sample from dataset	Desired result	Actual result
1	applyResultFilter	[“apply”, ”result”, ”filter”]	[“applyresultfilter”]
2	WarningLevel	["warning", "level"]	["warninglevel"]
3	getPluginName	[“get”, ”plugin”, ”name”]	[“getpluginname”]
4	applyResultFilter	[“apply”, ”result”, ”filter”]	[“applyresultfilter”]
5	ComponentModel	["component"," model"]	["componentmodel"]

Kegagalan yang ditunjukkan dalam penelitian ini disebabkan oleh cara penulisan metode dan atribut dalam C #. Karena metode dan atribut dengan dua kata atau lebih ditandai dengan huruf besar dan bukan

spasi, teknik *Tokenization* penulis tidak dapat membaginya dengan benar. Metode *Tokenization* yang dapat memfasilitasi gaya pengkodean seperti itu diperlukan untuk membuat *source code* dengan benar. Selain itu, dengan melakukan konversi huruf kecil sebelum *Tokenization*, penulis menghapus kapitalisasi, yang berfungsi sebagai tanda untuk memisahkan string menjadi token. Beberapa penyesuaian tentang bagaimana urutan setiap teknik *pre-processing* diterapkan juga diperlukan.

Penemuan tersebut menimbulkan implikasi yang menantang. Karena bahasa pemrograman yang berbeda dapat memiliki gaya yang berbeda, teknik *Tokenization* yang berbeda diperlukan untuk memfasilitasi setiap gaya pemrograman. Penelitian untuk bahasa dan gaya pemrograman yang berbeda diperlukan untuk mencapai itu.

Penyebab lain yang mungkin adalah perbedaan kata-kata berhenti dalam *source code*. Sintaks pemrograman dapat menurunkan nilai IDF karena akan meningkatkan jumlah kata di dalam dokumen tetapi tidak memberikan informasi yang lebih signifikan. Kamus kata berhenti untuk setiap bahasa mungkin diperlukan untuk dapat menghapus sintaks tersebut secara efektif. Namun, penulis membutuhkan penelitian lebih lanjut sebelum penulis dapat memvalidasi penyebab ini.

Berdasarkan percobaan lebih lanjut didapatkan hasil berupa, bahwa tahapan *pre-processing* implementasi yang sudah diperbarui tidak benar-benar menghasilkan nilai *accuracy*, *precision*, dan *recall* yang memuaskan. Hasil yang didapatkan sudah mengubah permasalahan awal pada *pre-processing* tersebut pada bagian *tokenizing* ada beberapa kata yang tidak benar-benar terpisah dalam *source code* yang tertulis, setelah dicoba lebih lanjut dan diperbaiki pada bagian tersebut ternyata nilai *accuracy, precision,* dan *recall* tidak menghasilkan perbedaan nilai yang signifikan, contoh perbaikan seperti Tabel III.

TABLE III
SAMPLE OF REPAIRING TOKENIZING

No	Sample from dataset	Desired result	Actual result
1	applyResultFilter	[“apply”, ”result”, ”filter”]	[“apply”, “result”, “filter”]
2	WarningLevel	["warning", "level"]	["warning", "level"]
3	getPluginName	[“get”, ”plugin”, ”name”]	[“get”, ”plugin”, ”name”]
4	applyResultFilter	[“apply”, ”result”, ”filter”]	[“apply”, ”result”, ”filter”]
5	ComponentModel	["component"," model"]	["component"," model"]

Percobaan lebih lanjut, mendapatkan hasil baru dimana penulis menambahkan tahapan baru berupa *stemming* dan menghasilkan nilai accuracy, precision, recall yang tidak beda jauh dari sebelumnya. Dalam proses stemming yang penulis uji coba penulis menggunakan *library*, dalam *library* tersebut penulis menemukan proses stemming terdapat masalah didalam library tersebut, contoh hasil stemming seperti table IV. Penulis mengambil kesimpulan bahwa proses tersebut tidak berubah banyak dikarenakan factor yang sama pada *pre-processing* data mentah dari source code kurang memadani. Percobaan lainnya berupa mencoba data yang berbeda dan tidak terlalu kompleks. Penulis mendapatkan hasil yang baik dan bagus berupa ketiga nilai indikator berupa 100%, hal ini didapatkan dari *source code* yang dipunyai tidak terlalu kompleks berupa keseluruhan fungsionalitas program hanya didalam satu jenis yang membuat nilai *requirements* yang benar hanya menunjuk satu *source code* program saja.

TABLE IV
SAMPLE OF STEMMING PROCESS

No	Sample from dataset	Desired result	Actual result
1	Encoding	[encode]	[encod]
2	Software	[software]	[softw]
3	Reserved	[reserve]	[reserv]
4	License	[license]	[licens]
5	Hope	[hope]	[hop]

4.3 *Threat to Validity*

Ancaman utama validitas penelitian ini adalah kualitas implementasi TF-IDF penulis. Karena implementasi yang buruk dapat menyebabkan hasil yang salah, ini mungkin tidak mencerminkan kualitas

metode TF-IDF dengan benar. Validasi konstan program yang dikembangkan dengan pihak eksternal dilakukan selama penelitian untuk meminimalkan ancaman ini.

5. Kesimpulan

Berdasarkan temuan penulis, penulis menyimpulkan bahwa TF-IDF tidak dapat diterapkan secara langsung untuk *recovering traceability* antara *requirements* perangkat lunak dan *source code*. Kesimpulan ini didasarkan pada rendahnya nilai *accuracy, precision, dan recall*. Dari percobaan lanjut penulis, beberapa modifikasi yang sudah dilakukan pada perubahan proses *tokenizing* tidak benar benar memberikan perbedaan hasil yang siginifikan. Dari analisis penulis selanjutnya, beberapa modifikasi pada alur kerja penulis saat ini diperlukan, terutama dalam *pre-processing*. Karakteristik khusus dari *source code*, seperti penamaan metode dan atribut, menyebabkan teknik *tokenizing* tidak berfungsi sebagaimana mestinya.

Di masa depan, kita perlu melihat lebih jauh ke teknik *tokenizing* yang memenuhi karakteristik bahasa dan gaya pemrograman yang berbeda. Penulis juga perlu melihat lebih jauh temuan penulis tentang bagian *source code* yang tidak perlu yang menghambat *recovery process*.

Referensi

[1] K. Wiegers and J. Beatty, *Software Requirements (Third Edition)*. 2013.

[2] J. H. Husen and R. R. Riskiana, "Alat Pengukur Keatomikan Kebutuhan Perangkat Lunak Berbasis Kemajemukan Kalimat," *Techno.Com*, vol. 18, no. 3, pp. 203–213, Aug. 2019, doi: 10.33633/tc.v18i3.2383.

[3] P. Mäder, O. Gotel, and I. Philippow, "Enabling automated traceability maintenance through the upkeep of traceability relations," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5562 LNCS, pp. 174–189, doi: 10.1007/978-3-642-02674-4_13.

[4] R. Tsuchiya, T. Kato, H. Washizaki, M. Kawakami, Y. Fukazawa, and K. Yoshimura, "Recovering traceability links between requirements and source code in the same series of software products," in *ACM International Conference Proceeding Series*, 2013, pp. 121–130, doi: 10.1145/2491627.2491633.

[5] J. Ramos, "Using TF-IDF to Determine Word Relevance in Document Queries," *Urol. Clin. North Am.*, vol. 2, no. 1, pp. 29–48, 2003.

[6] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Traceability recovery using numerical analysis," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2009, pp. 195–204, doi: 10.1109/WCRE.2009.14.

[7] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model traceability," *IBM Systems Journal*, vol. 45, no. 3. IBM Corporation, pp. 515–526, 2006, doi: 10.1147/sj.453.0515.

[8] N. Fitria A. and N. Aini, "EVALUASI PENDEKATAN PEMBANGUNAN TRACEABILITY LINK DALAM EVOLUSI PERANGKAT LUNAK," *JUTI J. Ilm. Teknol. Inf.*, vol. 11, no. 2, p. 43, Jul. 2013, doi: 10.12962/j24068535.v11i2.a10.

[9] H. Christian, M. P. Agus, and D. Suhartono, "Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF-IDF)," *ComTech Comput. Math. Eng. Appl.*, vol. 7, no. 4, p. 285, Dec. 2016, doi: 10.21512/comtech.v7i4.3746.

[10] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, "Interpreting TF-IDF term weights as making relevance decisions," *ACM Trans. Inf. Syst.*, vol. 26, no. 3, pp. 1–37, Jun. 2008, doi: 10.1145/1361684.1361686.

[11] S. Albitar, S. Fournier, and B. Espinasse, "An effective TF/IDF-based text-to-text semantic similarity measure for text classification," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8786, pp. 105–114, Oct. 2014, doi: 10.1007/978-3-319-11749-2_8.

[12] S. Qaiser and R. Ali, "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents," *Int. J. Comput. Appl.*, vol. 181, no. 1, pp. 25–29, 2018, doi: 10.5120/ijca2018917395.

[13] J. H. Hayes, A. Dekhtyar, and J. Payne, "The requirements tracing on target (RETRO).NET dataset," in *Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018*, Oct. 2018, pp. 424–427, doi: 10.1109/RE.2018.00054.

[14] Fakry Adi Permana, "Analisis dan Implementasi Vector Space Model dengan Tiga Pendekatan Term Weighting pada Pembangunan Ensiklopedia Kosa Kata Al Qur'an," 2020.

[15] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using IR methods for labeling source code artifacts: Is it worthwhile?," in *IEEE International Conference on Program Comprehension*, 2012, pp. 193–202, doi: 10.1109/icpc.2012.6240488.

[16] Z. Zhang, Y. Lei, J. Xu, X. Mao, and X. Chang, "TFIDF-FL: Localizing faults using term frequency-

inverse document frequency and deep learning," *IEICE Trans. Inf. Syst.*, vol. E102D, no. 9, pp. 1860–1864, Sep. 2019, doi: 10.1587/transinf.2018EDL8237.

[17] G. Scanniello and A. Marcus, "Clustering support for static concept location in source code," in *IEEE International Conference on Program Comprehension*, 2011, pp. 1–10, doi: 10.1109/ICPC.2011.13.

[18] R. Kohavi and F. Provost, "Glossary of Terms: Special Issue on Applictions of Macine Learning and the Knowledge Discovery Process," *Mach. Learn.*, vol. 30, pp. 271–274, 1998.