PEMBANGUNAN LINK PENELUSURAN KEBUTUHAN FUNGSIONAL DAN KODE SUMBER BERDASARKAN KEDEKATAN SEMANTIK

Djoko Pramono¹, Siti Rochimah²

¹Jurusan Teknik Informatika, Universitas Brawijaya, Malang

²Jurusan Teknik Informatika, Fakultas Teknologi Informasi, ITS Surabaya

Email: ¹djoko.jalin@ub.ac.id, ²siti@its-sby.edu

Abstrak

Link penelusuran membantu pengembang dalam memahami perangkat lunak yang tidak memiliki dokumen perancangan yang lengkap seperti pada kode sumber terbuka. Dalam kode sumber, suatu *method* mewakili suatu aksi yang dilakukan untuk menyelesaikan suatu kebutuhan fungsional. Pada penelitian ini diusulkan suatu metode untuk membangun link penelusuran antara kebutuhan fungsional dan kode sumber menggunakan pemrosesan bahasa alami dan kedekatan semantik. Kedekatan semantik ditentukan berdasarkan WordNet dan metode perhitungan Wu-Palmer. Pemrosesan bahasa alami dilakukan pada semua deklarasi *method* dalam kode sumber. Selanjutnya dihitung kedekatan semantik antara kebutuhan fungsional dengan hasil pemrosesan bahasa alami untuk menentukan *method-method* yang merupakan link penelusuran. Metode ini membantu menemukan bagian mana dari kode sumber yang menjalankan suatu kebutuhan fungsional, serta meningkatkan keakuratan dalam membangun link penelusuran antara kebutuhan fungsional dan kode sumber.

Kata kunci : link penelusuran, kedekatan semantik, pemrosesan bahasa alami.

Abstract

Traceability link helps developers to understand the software that does not have a complete design document as in open source software. In the source code, a method represents an action performed to complete a functional requirement. This research proposed a method to build links between functional requirement and source code using natural language processing and semantic relatedness. Semantic relatedness is determined based on WordNet and the method of calculation by Wu-Palmer. Natural language processing is done on all method declarations in the source code. Further semantic similarity computed between functional requirement and the results of natural language processing to determine the method-method which is traceability link. This method helps locate which parts of the source code that runs a functional requirement, as well as improve the accuracy in building links between functional requirements and source code.

Keywords : *traceability link, semantic similarity, natural language processing.*

1. Pendahuluan

Penelusuran kebutuhan adalah kemampuan untuk menjelaskan dan mengikuti jalannya suatu kebutuhan dalam arah maju maupun mundur (Gotel, 1994). Link penelusuran membantu pengembang untuk mengerti hubungan antara dokumen-dokumen yang dihasilkan selama proses pengembangan perangkat lunak. Link penelusuran antara dokumen kebutuhan dan kode sumber sangat membantu dalam proses pengembangan dan pemeliharaan

perangkat lunak. Dalam proses pengembangan dan pemeliharaan perangkat lunak, pengembang melakukan perubahan pada kode sumber akan tetapi sering melupakan untuk memperbarui dokumen-dokumen yang menyertai kode sumber untuk mempersingkat waktu pengembangan karena dituntut oleh tenggat waktu yang singkat (Ali, 2011). Selama proses pengembangan, link penelusuran antara artifak-artifak perangkat lunak bisa menjadi tidak valid

karena tidak diperbarui sesuai dengan perubahan yang terjadi pada kode sumber perangkat lunak(Hayasi, 2010). Karena itu diperlukan suatu metode untuk membangun link penelusuran kebutuhan yang cepat dan akurat. Link penelusuran juga berguna jika pengembang menggunakan perangkat lunak yang tidak memiliki dokumen perancangan yang lengkap seperti pada kode sumber terbuka. Dalam proses pengembangan perangkat lunak kadang diperlukan perangkat lunak sumber terbuka. Banyak perangkat lunak sumber terbuka tidak menyertakan dokumen perancangan secara lengkap. Demikian juga pada proses pengembangan perangkat lunak menggunakan metode *agile*, tidak menyertakan dokumen pengembangan yang detail.

Beberapa metode penemuan kembali informasi(*information retrieval/IR*) digunakan untuk membangun link penelusuran antara dokumen kebutuhan dengan kode sumber, antara lain VSM(*vector space model*)(Antoniol, 2000), LSI(*latent semantic index*)(Marcus dan Meletic, 2003), JS(*Jensen Shannon*)(Abadi, 2008) dan LDA(*Latent Dirichlet Allocation*)(Oliveto, 2010). Pendekatan IR menghitung kemiripan tekstual antara kode sumber dengan dokumen kebutuhan, semakin tinggi kemiripan tekstual kemungkinan besar kedua artifak mengandung konsep yang sama dan dianggap saling berhubungan. Ali(2012) melakukan penelitian dengan membandingkan penemuan link penelusuran menggunakan metode IR pada beberapa studi kasus dan mendapatkan nilai *precision* dan *recall* yang rendah. Nilai *precision* dan *recall* dipengaruhi antara lain oleh kualitas dan besarnya kode sumber. Penelitian yang dilakukan Eaddy(2008) menggabungkan IR dengan jejak eksekusi untuk meningkatkan akurasi.

Penelitian menggunakan analisa *natural language processing*(NLP) pada kode sumber dilakukan oleh David

Sheperd(2006) untuk menemukan pasangan kata kerja(*verb*) dan kata benda(*direct object*) dalam deklarasi *method*. Penelitian yang dilakukan A. Mahmoud(2012) menambahkan dokumen eksternal yaitu Wikipedia sebagai referensi untuk menghitung nilai kedekatan semantik antar kata. Penelitian lain dilakukan Hayashi(2010) menggunakan ontologi sebagai domain pengetahuan untuk menentukan bobot hubungan antar *method* dalam kode sumber. Kelemahan metode tersebut adalah perlu dibangun ontologi secara manual untuk setiap domain permasalahan.

Dalam penelitian ini digunakan *abstract syntax tree*(AST) dari kode sumber dan proses pengurai(*parsing*) untuk mengidentifikasi setiap deklarasi *method* dalam kode sumber. Langkah berikutnya adalah melakukan ekstraksi menggunakan pemrosesan bahasa alami pada *method signature*. Selanjutnya menghitung kedekatan semantik antara hasil parsing *method signature* terhadap kebutuhan fungsional untuk menemukan link penelusuran. Penggunaan pemrosesan bahasa alam(*Natural Language Proceessing/NLP*) dan kedekatan semantik diharapkan bisa menemukan hubungan antara dua konsep yang berbeda berdasarkan suatu domain pengetahuan. Salah satu contoh domain pengetahuan yang digunakan dalam penelitian ini adalah WordNet(Fellbaum, 2005). Sedangkan untuk menentukan kedekatan semantik antar kata dapat menggunakan rumus yang diusulkan oleh Wu-Palmer(1994). Wu-Palmer menilai kedekatan dua kata menggunakan relasi dan kelompok yang sama(*least common subsumer, LCS*) dalam WordNet. WordNet digunakan sebagai domain pengetahuan untuk menentukan kedekatan semantik antara konsep dari kebutuhan dan kode sumber. Dari penelitian ini diharapkan akan menghasilkan link antara kebutuhan dengan *method* yang mengimplementasikan kebutuhan tersebut.

2. Tinjauan Pustaka

2.1. Penelitian Terkait Link Penelusuran

Penelitian link penelusuran antara dokumen dan kode sumber menggunakan metode IR berbasis LSI dilakukan oleh Marcus dan Meletic(2003). Penelitian ini menggunakan dua studi kasus perangkat lunak yaitu LEDA dan Albergate. Hasil penelitian pada studi kasus LEDA, hasil pengujian mendapatkan nilai *precision* sebesar 0.77 pada *recall* 0.59 sampai nilai *precision* 0.11 pada *recall* 1. Pada kasus Albergate, hasil pengujian mendapatkan nilai *precision* sebesar 0.44 pada *recall* 0.45 sampai nilai *precision* 0.16 pada *recall* 1. Pada kasus Albergate nilai *precision* lebih rendah karena dokumen yang digunakan dokumen kebutuhan yang sangat pendek dan tidak ada komentar dalam kode sumber.

Penelitian tentang pembangunan link penelusuran antara dokumen dan kode sumber menggunakan metode IR berbasis LDA dilakukan oleh Oliveto dkk(2010). Penelitian ini menggunakan dua studi kasus perangkat lunak dengan dataset EasyClinic dan eTour. Penelitian ini membandingkan dua studi kasus tersebut antara metode LDA dengan VSM, LSI dan JS. Hasilnya diperoleh bahwa nilai *precision* dan *recall* dari LDA tidak lebih baik dari tiga metode IR tersebut. Tetapi pada kasus eTour, LDA berhasil menemukan link benar yang tidak ditemukan oleh metode IR lainnya sebesar 10%.

Penelitian lain dilakukan oleh David Shepherd dkk(2007) menggunakan *Natural Language Processing*(NLP). NLP digunakan untuk menemukan *verb*/kata kerja dan *noun*/kata benda dalam kode sumber yang saling berhubungan. Pasangan kata kerja dan kata benda, V-DO<verb/kata kerja, Direct Object/kata benda> dan lokasinya dalam kode sumber disimpan dalam suatu daftar. Ketika manusia memasukkan queri berupa kata

kerja, aplikasi akan memberikan saran berupa pilihan kata benda berdasar pasangan V-DO yang sudah dibangun dari kode sumber.

2.2. Pemrosesan Bahasa Alami

Pemrosesan bahasa alami atau *Natural Language Processing* (NLP) adalah sebuah bidang dalam ilmu komputer dan bahasa yang berkaitan dengan sistem temu kembali (*information retrieval/IR*) dan berkonsentrasi pada interaksi antara bahasa manusia dan komputer. Salah satu alat yang digunakan dalam pemrosesan bahasa alami adalah *Stanford NLP*. *Stanford NLP* adalah sistem yang dikembangkan oleh *The Stanford NLP Group* dengan fokus penelitian pada *sentence understanding*, *probabilistic parsing* dan *tagging*, *biomedical information extraction*, *grammar introduction*, *word sense disambiguation*, dan *automatic question answering* (nlp.stanford.edu, 2013). NLP dalam penelitian ini digunakan untuk melakukan *tagging* dan lemmatisasi. *Part of Speech* (POS) *tagger* adalah proses perangkat lunak untuk membaca teks dalam bahasa alami dan memberikan label dengan menggunakan *Part of Speech* (POS) seperti kata kerja, kata benda, kata sifat dll. Lemmatisasi bertujuan menghasilkan bentuk dasar dari suatu kata.

2.3. Kedekatan Semantik

Langkah-langkah utama dalam mengukur kedekatan semantik adalah dengan memperkirakan keterkaitan kata secara semantik dengan memanfaatkan sejumlah besar domain pengetahuan leksikal dan menggunakan teknik statistik untuk memanfaatkan semua kemungkinan hubungan yang berkontribusi pada kesamaan konsep. Domain pengetahuan biasanya tersedia dalam sumber-sumber eksternal termasuk pengetahuan berbasis linguistik (*linguistic knowledge base/LKB*) seperti WordNet(Fellbaum, 1998) dan

pengetahuan berbasis kolaboratif (*collaborative knowledge base/CKB*). WordNet merupakan kamus (database leksikal) online untuk bahasa inggris. WordNet mendefinisikan sebuah kata menjadi sebuah konsep yang diorganisasi kedalam sejumlah sinonim set(*synset*)(Miller dkk, 1990). *Synset* memiliki beberapa macam hubungan dengan *synset* lainnya. Metode-metode untuk menghitung kedekatan semantik antar kata menggunakan WordNet dikategorikan dalam Tabel 1.

Tabel 1. Metode-Metode Kemiripan Semantic(Shridara dkk, 2006)

Penulis	Kategori
Leacock dan Chadrow	Path based
Wu dan Palmer	Path based
Jiang dan Conrath	Information content based
Resnik	Information content based
Lin	Information content based
Banerjee dkk	Gloss based

Wu-Palmer(1994) mendefinsikan kedekatan semantik sebagai jarak antara dua konsep dalam struktur WordNet dan jarak *Least Common Subsumer(LCS)*.

Gambar 1. Ilustrasi Kemiripan Kata (Wu dan Palmer, 1994)

Untuk menghitung kemiripan kata seperti pada ilustrasi Gambar 1, dengan rumus Wu-Palmer ditunjukkan pada Rumus 1. Nilai kemiripan antar dua kalimat Sen_a dan Sen_b dapat dihitung menggunakan Rumus 4 yang diusulkan oleh Ming Che Lee(2011).

$$\text{sim}(c1, c2) = \frac{2 \times N3}{N1 + N2 + (2 \times N3)} \quad (1)$$

$$NC_{ab} = \left(\frac{\overrightarrow{NV_{sen_a}} \cdot \overrightarrow{NV_{sen_b}}}{|\overrightarrow{NV_{sen_a}}| \times |\overrightarrow{NV_{sen_b}}|} \right)^2$$

$$= \left(\frac{\sum_{i=1}^{S_{N_a} \cup S_{N_b}} NV_{sen_a_i} \times NV_{sen_b_i}}{\sqrt{NV_{sen_a_i}^2} \times \sqrt{NV_{sen_b_i}^2}} \right)^2 \quad (2)$$

$$VC_{ab} = \left(\frac{\overrightarrow{VV_{sen_a}} \cdot \overrightarrow{VV_{sen_b}}}{|\overrightarrow{VV_{sen_a}}| \times |\overrightarrow{VV_{sen_b}}|} \right)^2$$

$$= \left(\frac{\sum_{i=1}^{S_{N_a} \cup S_{N_b}} VV_{sen_a_i} \times VV_{sen_b_i}}{\sqrt{VV_{sen_a_i}^2} \times \sqrt{VV_{sen_b_i}^2}} \right)^2 \quad (3)$$

$$\text{Sim}_{a,b} = (\zeta NC_{ab} \times (1 - \zeta) VC_{ab}) \quad (4)$$

dengan :

NC_{ab} :Kemiripan semantik antara vektor kata benda(*noun*) kalimat Sen_a dan kalimat Sen_b .

VC_{ab} :Kemiripan semantik antara vektor kata kerja(*verb*) kalimat Sen_a dan kalimat Sen_b .

VV_a :Vektor kata kerja(*verb*) kalimat Sen_a .

NV_a :Vektor kata benda(*noun*) kalimat Sen_a .

$\text{Sim}_{a,b}$:Nilai kemiripan antara kalimat Sen_a dan Sen_b .

ζ :Konstanta pengali yang menunjukkan perbandingan kontribusi kata benda dan kata kerja dalam menentukan kemiripan.

Dalam rumus diatas didefinisikan bahwa suatu kalimat dibagi menjadi satu kelompok kata kerja(S_{V_a}) dan satu kelompok kata benda (S_{N_a}). Sedangkan *base space* kata kerja adalah gabungan dari kelompok kata kerja dari kedua kalimat. *Base space* kata benda adalah gabungan dari kelompok kata benda dari kedua kalimat. Vektor kata kerja $|VV|$ merupakan vektor yang terbentuk dari kelompok kata kerja terhadap *base space* kata kerja. Vektor kata kerja dapat dihitung dari kemiripan maksimal masing-masing kata dalam kelompok kata kerja kalimat Sen_a terhadap *base space* kata kerja menggunakan Persamaan 1. Demikian

juga vektor kata benda $|NV|$ dapat dihitung dari kemiripan maksimal masing-masing kata dalam kelompok kata benda kalimat Sen_a terhadap *base space* kata benda.

3. Arsitektur Sistem

Arsitektur sistem secara keseluruhan ditunjukkan pada Gambar 2. Masukan yang diperlukan dalam proses ini adalah kode sumber dan kebutuhan fungsional. Urutan langkah kerja dari sistem yang dibangun pertama adalah membentuk kumpulan kata dalam bahasa alami dari *method signature* dalam kode sumber. Kumpulan kata dihasilkan melalui preproses kode sumber dan proses pengolahan bahasa alami. Kemudian menghitung kemiripan antara *method signature* dengan kalimat-kalimat kebutuhan fungsional. Untuk melakukan ekstraksi *method signature* dari kode sumber perlu dilakukan analisa statik untuk membentuk *abstract syntax tree*(AST) dari kode sumber.

Gambar 2. Arsitektur Sistem

3.1. Preproses Kode Sumber

Preproses dilakukan terhadap nama *package*, nama *class*, dan *method signature*. *Method signature* terdiri dari nama *method*, tipe data kembalian dari *method* dan parameter dalam suatu *method*. Langkah pertama dalam preproses adalah melakukan pemisahan kata(*token splitting*) berdasarkan garis

bawah(*underscore*), *CamelCase*, *MixedCase*. Langkah kedua adalah lemmatisasi yakni mengembalikan kata ke akar katanya. Langkah berikutnya adalah menghilangkan *stopwords*.

Langkah berikutnya adalah melakukan ekstraksi kata dari *method signature*. Tujuan utama dalam ekstraksi kata ini adalah menemukan aksi atau operasi yang dilakukan suatu *statement* dalam kode sumber. Proses ini memanfaatkan POS tagger untuk menentukan kata kerja dan kata benda dalam *method signature*. Suatu deklarasi *method* terdiri dari *method modifier*, *method return type*, nama *method* dan parameter. Untuk mengekstraksi kata dari suatu *method signature* digunakan aturan sebagai berikut :

1. Tipe data primitif seperti *int*, *long*, *char* dan tipe data bawaan kompiler akan diabaikan dalam proses ekstraksi kata dalam *method signature*. Jenis data bawaan kompiler seperti *String*, *Map*, *List* tidak akan digunakan dalam proses ekstraksi kata.
2. Dalam proses ekstraksi kata, tidak hanya nama *method* yang digunakan, tetapi ditambahkan juga nama *class* pada kode sumber memberikan kontribusi dalam menentukan konteks dari aksi yang dilakukan oleh suatu *method*.
3. Jika terdapat kata yang sama dalam satu *method signature*, hanya diambil satu kali kata kemunculan kata tersebut dan mengabaikan kemunculan berikutnya.
4. Proses ekstraksi kata dari deklarasi *method* tidak menyertakan *return type* dan *modifier* dari suatu *method*.
5. Parameter suatu *method* juga diikutsertakan jika tidak ada kata benda dalam nama *method*.
6. Dalam kode sumber sering digunakan nama *method* yang sudah menjadi standard dan maknanya secara tekstual tidak sesuai dengan kebutuhan fungsionalnya. Nama-nama *method*

tersebut misalnya *actionPerformed*, *run*, *execute*, *fire*. Nama *method* ini dihilangkan atau diabaikan.

7. Dalam bahasa pemrograman java ada beberapa kata yang banyak digunakan dalam nama *method* atau nama *class* yang menjadi standard penamaan dalam design pattern seperti *listener*, *event*, *action*, *adapter*, *handler*, *singleton*, *factory*. Kata-kata tersebut dihilangkan dari *method signature*.

3.2. Preproses Kebutuhan Fungsional

Preproses kebutuhan fungsional diawali dengan memisahkan kalimat. Akhir kalimat ditandai dengan tanda titik(.), tanda tanya(?) atau tanda seru(!). Kemudian dari masing-masing kalimat akan dihilangkan karakter *non-alphanumeric*. Selanjutnya dilakukan pemisahan kata berdasarkan *white space*(spasi).

Langkah berikutnya adalah melakukan lemmatisasi dan *part of speech tagger* terhadap kata-kata yang sudah dipisah. Proses lemmatisasi dan Part of Speech Tagger menggunakan *library* dari Stanford CoreNLP. Stanford CoreNLP menggunakan proses *pipeline* pengolahan bahasa alami berbasis anotasi (Manning,2014). Langkah berikutnya dalam preproses adalah menghilangkan *stopwords*.

3.3. Perhitungan Kemiripan Semantik

Hubungan semantik dihitung antara kalimat dalam kebutuhan dengan kata-kata yang dihasilkan dari pemrosesan bahasa alami terhadap *method* dalam kode sumber. Perhitungan kemiripan semantik dilakukan menggunakan Persamaan 4 untuk menghitung jarak antara *method signature* dan kebutuhan fungsional.

4. Skenario pengujian

Proses pengujian dilakukan dengan mengimplementasikan alur kerja yang diusulkan dan dijelaskan pada metodologi. Dalam penelitian ini dikembangkan plugin

eclipse untuk membantu mempercepat proses pengujian. Plugin dikembangkan untuk membentuk dan melakukan penelusuran terhadap AST dan mengambil deklarasi *method* dalam semua *class*.

Tabel 2. Keterangan Dataset Pengujian

Nama Aplikasi	Jml. method	Respositori
Itrust	2698	https://sourceforge.net/projects/itrust/
Gantt Project	5292	https://github.com/bardsoft/ware/ganttproject

Pengujian juga dilakukan menggunakan metode yang diusulkan pada penelitian sebelumnya yaitu menggunakan *Latent Semantic Analysis*(LSA) (Marcus,2003) dan *Latent Dirichlet Allocation*(LDA)(Oliveto,2010). Hasil pengujian dengan LSA dan LDA akan dibandingkan dengan hasil pengujian menggunakan metode yang diusulkan.

4.1. Dataset Pengujian

Dataset untuk uji coba terdiri dari kode sumber dalam bahasa pemrograman java dan kebutuhan fungsional dari perangkat lunak tersebut. Perangkat lunak yang akan dijadikan dataset dipilih perangkat lunak kode sumber terbuka.

4.2. Pengujian ITrust

ITrust adalah aplikasi medis yang memiliki fitur antara lain memberikan fasilitas pada pasien untuk memantau riwayat kesehatan. Pengujian ini menggunakan kode sumber itrust versi 21.0. Pengujian untuk menghitung nilai kemiripan antara kode sumber dengan kebutuhan fungsional menggunakan metode wu-palmer dan Persamaan 4. Dengan menggunakan nilai alpha sebesar 0.4, 0.5 dan 0.6. Nilai *precision* hasil pengujian dengan nilai alpha 0.5 ditampilkan pada Tabel 3 dan nilai *recall* hasil pengujian ditampilkan pada Tabel 4.

Tabel 3. Nilai *Precision* Hasil Pengujian Itrust

kebutuhan fungsional	treshhold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.137	0.175	0.2	0.2	0.219	0.545
2	0.423	0.5	0.5	0.5	0.524	0.875
3	0.081	0.152	0.071	0.333	0.333	1
4	0.227	0.222	0.167	0	0	0
5	0.09	0.123	0.154	0.267	0.333	0
6	0.229	0.229	0.444	0.333	0	0
7	0.5	0.667	1	1	1	1
8	0.053	0.185	0.333	0.6	0.667	1
9	0.027	0.032	0.065	0.222	0.667	0.667
10	0.19	0.444	0.286	0.286	0.333	0
11	0.03	0.053	0.111	0.2	0.25	1
12	0.1	0.106	0.159	0.212	0.091	0.182
13	0.25	0.429	0.6	0.6	0.667	1
14	0.088	0.136	0.14	0.153	0.164	0.27
15	0.167	0.222	0.333	0.429	0.5	0
16	0.889	0.875	1	1	1	0
rata-rata	0.218	0.284	0.348	0.396	0.422	0.471

Sebagai pembandingan dilakukan percobaan terhadap dataset yang sama menggunakan LDA dan LSA. Hasil nilai *precision* dari percobaan menggunakan LSA dan LDA ditunjukkan pada Tabel 5 dan nilai *recall* ditunjukkan pada Tabel 6.

Tabel 4. Nilai *Recall* Hasil Pengujian Itrust

kebutuhan fungsional	treshhold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.778	0.778	0.778	0.778	0.778	0.667
2	0.647	0.647	0.647	0.647	0.647	0.412
3	0.438	0.313	0.063	0.063	0.063	0.063
4	0.714	0.286	0.143	0	0	0
5	0.867	0.667	0.4	0.267	0.267	0
6	0.857	0.571	0.286	0.036	0	0
7	0.4	0.4	0.4	0.2	0.2	0.2
8	0.625	0.625	0.5	0.375	0.25	0.25

9	0.429	0.286	0.286	0.286	0.286	0.286
10	0.667	0.667	0.333	0.333	0.333	0
11	0.4	0.4	0.2	0.2	0.2	0.2
12	0.816	0.579	0.474	0.368	0.079	0.053
13	0.429	0.429	0.429	0.429	0.286	0.143
14	0.824	0.706	0.706	0.647	0.588	0.588
15	0.588	0.471	0.353	0.353	0.176	0
16	0.727	0.636	0.636	0.636	0.636	0
rata-rata	0.638	0.529	0.415	0.351	0.299	0.179

Tabel 5. Nilai *Precision* Hasil Pengujian Itrust dengan metode LSA dan LDA

	treshhold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.15	0.046	0.058	0.021	0.022	0.016
LDA	0.284	0.305	0.292	0.298	0.23	0.093

Tabel 6. Nilai *Recall* Hasil Pengujian Itrust dengan metode LSA dan LDA

	treshhold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.149	0.102	0.086	0.047	0.047	0.023
LDA	0.265	0.236	0.21	0.19	0.159	0.048

Hasil pengujian menggunakan metode LDA menghasilkan nilai *precision* sebesar 0.305 pada *recall* 0.23. Pada kasus ini nilai tersebut tercapai pada iterasi 2000 dengan jumlah topik 100 dan ambang batas 0.75. Metode LSA menghasilkan *precision* sebesar 0.15 pada *recall* 0.15. Pada kasus Itrust, metode LDA menghasilkan *precision* dan *recall* lebih baik dari pada LSA. Perbandingan *precision* antara metode kemiripan dalam penelitian ini dengan LDA ditunjukkan pada grafik Gambar 3 dan perbandingan nilai *recall* pada Gambar 4.

Gambar 3. Grafik Perbandingan Nilai Precision Itrust

Gambar 4. Grafik Perbandingan Nilai Recall Itrust

Gambar 5. Grafik Nilai Interpolasi Itrust

Gambar 5 menunjukkan kurva interpolasi *precision recall* hasil pengujian. Metode penelusuran menggunakan Sim menghasilkan nilai *precision* tertinggi 0.47 pada *recall* 0.18 tapi nilai *precision*-nya terus menurun dengan meningkatnya *recall*.

4.3. Pengujian Gantt Project

Gantt Project adalah aplikasi pengelolaan proyek sederhana yang ditulis menggunakan bahasa pemrograman java dan bersifat kode terbuka. Fitur yang dimiliki meliputi fungsi dasar pengelolaan proyek seperti gantt chart, penjadwalan dan pengelolaan sumber daya manusia. Pada penelitian ini digunakan kode sumber versi 2.7.1.

Kode sumber Gantt Project terdiri dari 520 class dengan jumlah method sebanyak 5292 method. Nilai *precision* hasil pengujian dengan nilai alpha 0.4 ditampilkan pada Tabel 7 dan nilai *recall* hasil pengujian ditampilkan pada Tabel 8.

Tabel 7. Nilai *Precision* Hasil Pengujian Gantt Project

kebutuhan fungsional	threshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.018	0.03	0.06	0.053	0.097	0.104
2	0.178	0.271	0.579	1	1	0
3	0.031	0.071	0.211	0.24	0.208	0.238
4	0.292	0.35	0.579	0.889	0.889	1
5	0.009	0.019	0.03	0.088	0.241	0
6	0.025	0.058	0.273	0.5	0	0
rata-rata	0.092	0.133	0.289	0.462	0.406	0.224

Tabel 8. Nilai *Recall* Hasil Pengujian Gantt Project

kebutuhan fungsional	threshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.588	0.588	0.529	0.353	0.353	0.294
2	0.533	0.433	0.367	0.333	0.2	0
3	0.846	0.769	0.615	0.462	0.385	0.385
4	0.95	0.7	0.55	0.4	0.4	0.3
5	0.421	0.421	0.368	0.368	0.368	0
6	0.6	0.6	0.6	0.6	0	0
rata-rata	0.656	0.585	0.505	0.419	0.284	0.163

Hasil nilai *precision* dari percobaan menggunakan LSA dan LDA ditunjukkan pada Tabel 9 dan nilai *recall* ditunjukkan pada Tabel 10. Hasil pengujian menggunakan metode LDA menghasilkan

nilai *precision* sebesar 0.078 pada *recall* 0.372. Pada kasus ini nilai tersebut tercapai pada iterasi 2000 dengan jumlah topik 100 dan ambang batas 0.70. Metode LSA menghasilkan *precision* sebesar 0.167 pada *recall* 0.014.

Tabel 9. Nilai *Precision* Hasil Pengujian Gantt Project dengan Metode LSA dan LDA

	threshold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.159	0.125	0.139	0.167	0.167	0
LDA	0.078	0.068	0.062	0.051	0.02	0.02

Tabel 10. Nilai *Recall* Hasil Pengujian Gantt Project dengan Metode LSA dan LDA

	threshold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.028	0.014	0.014	0.014	0.008	0
LDA	0.372	0.298	0.25	0.127	0.05	0.042

Pada kasus Gantt Project, metode LSA menghasilkan nilai *precision* lebih tinggi daripada LDA. Tetapi pada titik potong 0.95 LSA tidak menemukan *traceability* link, nilai *precision* yang dihasilkan 0. Nilai *recall* LSA sangat kecil, nilai *recall* tertinggi 0.028. Perbandingan *precision* antara metode kemiripan dalam penelitian ini dengan LDA ditunjukkan pada grafik Gambar 6 dan perbandingan nilai *recall* pada Gambar 7. Kurva interpolasi *precision recall* hasil pengujian ditunjukkan pada Gambar 8. Metode penelusuran menggunakan Sim menghasilkan nilai *precision* 0.46 sampai pada rentang *recall* 0.41. Sedangkan LDA hanya mencapai nilai *precision* 0.08 sampai rentang *recall* 0.37. Dalam kasus Gantt Project kemiripan *method signature* memberikan hasil *precision* dan *recall* lebih baik daripada LDA.

Gambar 6. Grafik Perbandingan Nilai Precision Gantt Project

Gambar 7. Grafik Perbandingan Nilai Recall Gantt Project

Gambar 8. Kurva Interpolasi Precision dan Recall Gantt Project

4.4. Analisa dan Pembahasan

Pada kebutuhan fungsional yang memiliki kalimat dengan kata pelengkap atau objek tidak langsung (*Indirect Object*) seperti "Add resource assignment to Task", dimana "task" merupakan kata pelengkap, memiliki nilai kemiripan yang rendah karena parsing pada *method* hanya menghasilkan kata kerja dan objek (*Direct Object/DO*). Pada kalimat paralel seperti "an email will be sent after the data is stored", sistem tidak bisa menemukan link penelusuran. Kalimat paralel memiliki kata penghubung, dua kata benda dan dua kata

kerja. Parsing *method signature* hanya menghasilkan satu kata kerja dan satu kata benda sehingga nilai kemiripannya sangat kecil.

5. Kesimpulan dan Saran

Beberapa kesimpulan yang dapat ditarik dari hasil pengerjaan penelitian ini adalah hasil pengujian menunjukkan metode kemiripan *method signature* memberikan hasil nilai *precision* dan *recall* lebih baik daripada metode LSA dan LDA. Saran-saran untuk penelitian mendatang antara lain hasil pengujian pada beberapa kebutuhan fungsional tidak menemukan link penelusuran karena kecilnya nilai kemiripan. Salah satu penyebabnya karena struktur kalimat kebutuhan fungsional seperti kalimat paralel dan adanya objek tidak langsung(*Indirect Object*) dalam kalimat. Perlu dilakukan parsing struktur kalimat kebutuhan fungsional untuk meningkatkan akurasi.

Referensi

- [1] Abadi, A., Nisenson, M., Simionovici, Y. (2008), "A traceability technique for specifications", *Proceedings of 16th IEEE International Conference on Program Comprehension*, hal. 103-112.
- [2] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.(2000), "Tracing Object-Oriented Code into Functional Requirements", *Proceedings of 8th International Workshop on Program Comprehension (IWPC'00)*, hal. 79-87.
- [3] Eaddy, M., Aho, A., Antoniol, G., Guéhéneuc, Y.-G. (2008), "Cerberus: Tracing requirements to source code using information retrieval dynamic analysis and program analysis", *Proceedings of 16th IEEE International Conference on Program Comprehension*, hal. 53-62.
- [4] Fellbaum, C. (1998), "WordNet: An Electronic Lexical Database", MIT Press, Cambridge.
- [5] Gotel O., Finkelstein, A. (1994), "An analysis of the requirements traceability problem", *Proceedings of 1st International Conference on Requirements Engineering*, hal. 94-101.
- [6] Hayashi, S., Yoshikawa, T., Saeki, M. (2010), "Sentence-to-Code Traceability Recovery with Domain Ontologies," *Proceedings of 2010 Asia Pacific Software Engineering Conference*, hal. 385-394.
- [7] Mahmoud, A., Nan Niu, Songhua Xu (2012), "A semantic relatedness approach for traceability link recovery", *proceedings of 20th International Conference on Program Comprehension*, Passau, hal. 183 – 192.
- [8] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. "The Stanford CoreNLP Natural Language Processing Toolkit", *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.
- [9] Marcus, A., Maletic, J.I. (2003), "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", *Proceeding International Conf. on Software Engineering*, hal. 125-135.
- [10] Miller, G.A. (1995), "WordNet: A Lexical Database for English", *Communications of the ACM*, Vol. 38, No. 11, hal.39-41.
- [11] Ming Che Lee (2011), "A novel sentence similarity measure for semantic-based expert systems", *Expert Systems with Applications*, Vol. 38, Issue 5, hal 6392-6399.
- [12] Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A. (2010), "On the equivalence of information retrieval

- methods for automated traceability link recovery", *Proceedings of Int. Conf. Program Comprehension*, hal. 68-71.
- [13] Shepherd, D., Fry, Z.P., Hill, E., Pollock, L., Vijay-Shanker, K. (2007), "Using natural language program analysis to locate and understand action-oriented concerns", *Proceeding of 6th International Conference on Aspect-Oriented Software Development(AOSD)*, hal. 212-224.
- [14] Sridhara, G., Hill, E., Pollock, L., Vijay-Shanker, K. (2008), "Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools", *proceedings of 16th IEEE International Conference on Program Comprehension*, Amsterdam, hal. 123 – 132.
- [15] Wu, Z., & Palmer, M. (1994), "Verbs semantics and lexical selection", *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, hal. 133-138.