PROPOSAL TESIS

MEMODELKAN DEPENDENSI KEBUTUHAN BERDASARKAN DEPENDENSI DIAGRAM KELAS

Hernawati Susanti Samosir
5116201031

DOSEN PEMBIMBING
Daniel Oranova Siahaan, S. Kom, M. Sc., PDEng
19741123 200604 1 001

PROGRAM MAGISTER
BIDANG KEAHLIAN REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

LEMBAR PENGESAHAN

PROPOSAL TESIS

Judul : Memodelkan Dependensi Kebutuhan berdasarkan Dependensi Diagram Kelas

Oleh : Hernawati Susanti Samosir

NRP : 5116201031

Telah diseminarkan pada:

Tanggal : 04 Januari 2018
Tempat : Ruang IF 215

Mengetahui / menyetujui

Dosen Penguji:	Dosen Pembimbing:
1.	1.
Dr.Ir.Siti Rochimah, M.T	Daniel Oranovora Siahaan, S.Kom. PD.Eng
NIP. 19681002 199403 2 001	NIP. 19741123 200604 1 001
2.	2.
Dr.Eng.Chastine Fatichah, S.Kom, M.Kom	
NIP. 19751220 200112 2 002	
3.	
Dr.Eng.Darlis Heru Murti, S.Kom, M.Kom	
NIP. 19771217 200312 1 001	

[Halaman ini sengaja dikosongkan]

MEMODELKAN DEPENDENSI KEBUTUHAN BERDASARKAN DEPENDENSI DIAGRAM KELAS

Nama : Hernawati Susanti Samosir
NRP : 5116201031
Pembimbing : Daniel Oranovora Siahaan, S.Kom. PD.Eng

ABSTRAK

Kebutuhan dalam perangkat lunak merupakan elemen penting dalam pembangunan perangkat lunak. Tidak dapat dipungkiri jika satu kebutuhan memiliki keterkaitan dengan kebutuhan lainnya. Keterkaitan antar kebutuhan disebut dengan interdependensi kebutuhan. Interdependensi ini sangat penting dalam sebuah proses pengambilan keputusan dalam rekayasa kebutuhan, seperti dalam menajemen perubahan kebutuhan, perencanaan peluncuran versi dan manajemen kebutuhan. Terdapat beberapa penelitian sebelumnya tentang pemodelan dependensi antar kebutuhan, di antaranya visualisasi dependensi antar kebutuhan, analisis terhadap dampak perubahan pada perangkat lunak dengan menggunakan perubahan pada diagram kelas UML, dan prediksi *bug* berdasarkan dependensi antar kebutuhan. Penelitian tersebut menerima input berupa informasi dependensi kebutuhan. Dependensi kebutuhan diasumsikan telah dibangun oleh perekayasa kebutuhan.

Penelitian ini mengajukan suatu metode untuk membangun model dependensi kebutuhan. Suatu model dependensi kebutuhan dibangun berdasarkan relasi antara kebutuhan dan kelas dalam rancangan sistem serta informasi dependensi antar kelas. Pemetaan antara kebutuhan dan kelas serta ekstraksi dependensi kebutuhan dilakukan berdasarkan kemiripan semantik antara kebutuhan dan kelas. Suatu kelas dinyatakan merealisasikan suatu kebutuhan jika tingkat kemiripan keduanya melampaui suatu ambang batas tertentu.

Hasil keluaran yang diperoleh dari metode pembangunan dependensi kebutuhan perangkat lunak akan dibandingkan dengan hasil keluaran yang dilakukan oleh anotator. Tingkat reliabilitas dari metode diukur berdasarkan tingkat kesepakatan antara metode dan annotator menggunakan indeks statistik Gwet's AC1. Metode yang diajukan diharapkan dapat memiliki tingkat kesepakatan yang baik.

Kata kunci: interdependensi kebutuhan, dependensi kebutuhan, diagram kelas, *reliable, threshold*

[Halaman ini sengaja dikosongkan]

MODELING REQUIREMENT DEPENDENCY BASED ON CLASS DIAGRAM DEPENDENCY

Student's Name : Hernawati Susanti Samosir

Student ID : 5116201031

Supervisor : Daniel Oranovora Siahaan, S.Kom. PD.Eng

ABSTRACT

Software requirement is an important element in software development. It can not be denied if one requirement is related to other requirements. The interconnection between requirements is called interdependence of requirements. This interdependence is crucial in decision-making processes of requirement engineering, such as a change management requirement, a version launch plan, and a requirement management. There are several previous studies on dependency modeling among requirements. Those are dependency visualization between requirements, analysis of the impact of changes in software by using changes to UML class diagrams, and prediction of bugs based on dependencies between requirements. The study received input in the form of the information of requirement dependency. Requirement dependency is assumed to have been built by requirements engineers.

This research proposes a method for building dependency of requirement model. A dependency of requirement model is built on the relation between requirements and classes in the system design as well as class dependency information. The mapping between requirements and classes as well as the dependency extraction requirements is based on the semantic similarity between requirements and classes. A class is expressed to implement a requirement if the similarity level both exceeds a certain threshold.

The output obtained from the dependent software development method will be compared with the output performed by the annotator. The reliability level of the method is measured by the level of agreement between the method and the annotator using Gwet's AC1 statistical index. The proposed method is expected to have a good level of agreement.

Keywords: requirement, dependencies, interdependencies, threshold

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

BAB 1
PENDAHULUAN

Bab ini membahas tentang poin utama dalam pembuatan proposal penelitian yang meliputi latar belakang, perumusan masalah, tujuan, manfaat, kontribusi penelitian, dan batasan masalah.

1.1. Latar Belakang

Rekayasa kebutuhan perangkat lunak merupakan suatu rangkaian aktivitas yang terdiri dari elisitasi, spesifikasi, validasi, dan manajemen kebutuhan perangkat lunak. Serangkaian proses ini diharapkan menghasilkan suatu dokumen spesifikasi kebutuhan. Dokumen spesifikasi kebutuhan ini berisi sekumpulan pernyataan kebutuhan yang menspesifikasikan layanan dan kemampuan (fungsionalitas) yang disediakan oleh sistem dan karakteristik, batasan, properti, dan kualitas (non-fungsionalitas) dari sistem.

Sejumlah fungsionalitas dan non-fungsionalitas yang ada dalam suatu spesifikasi kebutuhan secara spesifik ditujukan untuk merealisasikan suatu kebutuhan dasar (*basic requirement*) dari seorang pengguna atau suatu eksternal obyek yang sedang menjalankan peran tertentu. Kebutuhan dasar tersebut menjadi tujuan (*goal*) seorang pengguna atau suatu eksternal sistem ketika berinteraksi dengan sistem yang dibangun. Jadi, ada pemetaan banyak ke banyak antara pernyataan kebutuhan dengan diagram kelas. Pemetaan ini merepresentasikan interdependensi antar kebutuhan dalam suatu spesifikasi kebutuhan.

Pada rekayasa kebutuhan, interdependensi kebutuhan diidentifikasi sebagai dasar yang penting dalam sebuah proses pengambilan keputusan dalam rekayasa kebutuhan, misalnya dalam manajemen perubahan kebutuhan, perencanaan peluncuran versi, dan manajemen kebutuhan [1]. Menurut Dahlstedt, interdependensi digunakan untuk menjelaskan relasi atau hubungan antara kebutuhan [2]. Kebutuhan bukanlah komponen yang dapat berdiri sendiri, akan tetapi kebutuhan selalu berhubungan dan berpengaruh pada kebutuhan lainnya dalam cara yang kompleks [1].

Suatu perubahan yang dilakukan pada sebuah kebutuhan dapat berpengaruh pada kebutuhan lainnya. Hal ini merupakan suatu hal yang tidak terelakkan. Ada beberapa alasan mengapa dependensi kebutuhan diperlukan. Pertama, dependensi kebutuhan dapat digunakan untuk mengidentifikasi seberapa luas dampak (kebutuhan lain yang ikut berubah) yang diakibatkan jika suatu kebutuhan berubah. Kedua, dengan diketahuinya dampak dari perubahan suatu kebutuhan terhadap kebutuhan lain, maka pihak pengembang dapat mengukur seberapa besar biaya yang diperlukan untuk melakukan perubahan tersebut. Terakhir, dalam pengembangan sistem rekomendasi kebutuhan, pengembang dapat mencari kebutuhan lain yang terkait dan memiliki ketergantungan kuat jika suatu kebutuhan terlebih dahulu diidentifikasi.

Oleh sebab itu, penelitian tentang interdependensi kebutuhan sangat dibutuhkan [1]. Interdependensi kebutuhan memberikan informasi bagaimana dependensi kebutuhan mempengaruhi aktivitas dalam rekayasa perangkat lunak dan bagaimana pengetahuan tentang interdependensi dapat memfasilitasi pengembangan perangkat lunak.

Terdapat sejumlah penelitian terkait pembangunan model dependensi kebutuhan. Pada paper Widiastuti, Siahaan, dijelaskan tentang visualisasi dependensi kebutuhan ke dalam LTS-RC (*Labelled Transition System for Requirement Change*). LTS-RC adalah LTS yang dikembangkan berguna dalam pemodelan perubahan kebutuhan perangkat lunak sehingga mempermudah *stakeholder* untuk mengamati alur perubahan kebutuhan beserta dampaknya. Metode ini dapat berperan dalam penyusunan strategi perubahan kebutuhan yang optimal [3]

Selanjutnya, Muller melakukan analisis terhadap dampak perubahan pada perangkat lunak dengan menggunakan perubahan pada diagram kelas UML [4]. Penelitian ini menggunakan informasi terkait dependensi antar kelas untuk memodelkan dampak perubahan. Jika suatu kelas mengalami perubahan, maka model yang dibangun dapat mengidentifikasi kelas-kelas mana dalam suatu diagram kelas yang juga akan terdampak. Namun, penelitian ini tidak menghubungkan dampak perubahan ini pada level kebutuhan.

Di sisi lain, Wang membangun model dependensi antar kebutuhan berdasarkan informasi frekuensi kemunculan *bug*. Setelah itu, dilakukan penyelidikan terhadap dependensi kebutuhan berkolaborasi dengan memprediksi *bug*, yang dapat memberikan perkiraan awal mengenai kualitas perangkat lunak. Akan tetapi identifikasi terhadap dependensi kebutuhan dilakukan secara manual [5].

Penelitian ini mengajukan suatu metode untuk memodelkan dampak perubahan kebutuhan terhadap kebutuhan lain dalam suatu proyek perangkat lunak. Model perubahan yang dibangun didasarkan informasi dependensi antar kelas yang didapatkan dari diagram kelas. Kelas yang ada pada diagram kelas dari suatu proyek perangkat lunak dipetakan kepada sekumpulan kebutuhan dari dokumen kebutuhan perangkat lunak terkait. Relasi ini didasarkan kepada kerangka kerja *rational unified process* dimana setiap kebutuhan memiliki sekumpulan kelas yang merealisasikan kebutuhan tersebut. Untuk itu, penelitian ini ditujukan untuk proyek perangkat lunak yang telah melewati satu tahapan perancangan.

1.2. Perumusan Masalah

Rumusan masalah yang dibahas dalam penelitian ini dapat dipaparkan sebagai berikut:

1. Bagaimana memetakan hubungan kebutuhan dan kelas pada diagram kelas?
2. Interdependensi apa saja yang dapat dimanfaatkan di diagram kelas untuk interdependensi kebutuhan?
3. Bagaimana mengkonversi suatu dependensi pada kelas menjadi suatu dependensi pada kebutuhan?

1.3. Tujuan

Tujuan penelitian yang ingin dicapai yaitu membangun metode untuk memodelkan dependensi kebutuhan berdasarkan *interdependensi* yang ada di dalam diagram kelas.

1.4. Manfaat

Penelitian ini diharapkan bermanfaat bagi pengembang perangkat lunak untuk mengetahui dampak perubahan sebuah kebutuhan terhadap kebutuhan yang lain dalam suatu proyek perangkat lunak sehingga pengembang dapat memprediksi besaran usaha untuk melakukan perubahan tersebut.

1.5. Kontribusi Penelitian

Kontribusi pada penelitian ini adalah membangun metode untuk memodelkan dependensi kebutuhan berdasarkan informasi interdependensi yang ada di diagram kelas.

1.6. Hipotesa

Hipotesa dari penelitian ini adalah dapatkah dependensi kelas dan kemiripan teks kebutuhan dan teks pada kelas untuk membuat dependensi antar kebutuhan?.

1.7. Batasan Masalah

Untuk memfokuskan permasalahan penelitian ini, batasan masalah yang ditentukan adalah sebagai berikut:

- Pernyataan kebutuhan dan diagram kelas dispesifikasikan dalam bahasa alamiah
- Bahasa alamiah yang digunakan adalah bahasa Inggris.

BAB 2
KAJIAN PUSTAKA

Bab ini merupakan pembahasan dari referensi terkait yang telah dilakukan dalam menyelesaikan permasalahan sesuai dengan uraian pada latar belakang. Bab ini diawali dengan menjabarkan setiap kajian pustaka yang berkaitan dengan topik penelitian.

2.1. Graph Dependensi

Terdapat sejumlah penelitian mengenai graph dependensi pemodelan [6][7][8][9][10]. Peneliti dari kelompok Pengolahan Bahasa Alami Stanford mengajukan sebuah graph ketergantungan kalimat [11]. Kemudian, peneliti mengusulkan deskripsi hubungan gramatikal dalam kalimat bahasa alami. Hal ini bertujuan meningkatkan pemahaman dan penggunaan yang dipahami serta efektif digunakan oleh orang tanpa keahlian linguistik yang ingin mengekstrak hubungan tekstual.

Pekerjaan pada graph dependensi juga digunakan dalam pengembangan perangkat lunak. Ye dan Hu [11] menggunakan ketergantungan antara fitur modul. Graph dependensi menentukan modul kombinasi yang efektif harus dikombinasikan untuk menghasilkan sebuah produk baru dengan fitur tertentu sembari menghindari potensi dependensi fitur yang saling bertentangan. Bouillard [6] mengusulkan sebuah model dependensi kompleks yang disederhanakan yang mampu menangani masalah *state-space explosion*. Model ini bertujuan memodelkan dan menganalisis masalah fungsional berbasis *Real-Time Calculus*.

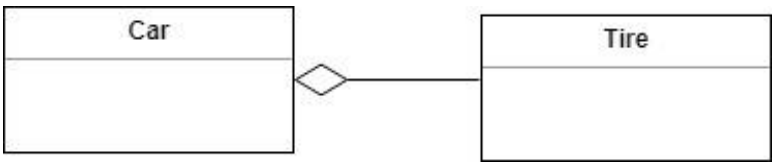
UML-Diagram.org memperkenalkan sebuah graph dependensi kelas sebagai hubungan yang diarahkan antara kelas dalam sebuah sistem [12]. Kemudian, diperkenalkan taksonomi dari dependensi antar kelas. Taksonomi mengklasifikasikan dependensi ke dalam 3 kelas dependensi, yaitu: penggunaan (*usage*), abstraksi (*abstraction*), dan penyebaran (*deployment*).

Hanya ada dua kelas dependensi yang dapat diekstraksi dari diagram penyebaran. Kelas dependensi usage terdiri dari 4 jenis dependensi, yaitu: *create, instantiate*, *call*, dan *send*. *Create* adalah ketergantungan antara kelas di mana keberadaan dari kelas aggregate bergantung pada keberadaan objek dari kelas asal.

Gambar 2.1 menggambarkan dependensi *create*. Sebuah objek dari kelas *Building* menciptakan sebuah objek kelas *Room*. Sebuah objek dari kelas *Room* hanya boleh ada ketika sebuah objek *Building* sudah ada sebelumnya. Jika sebuah objek dari *Building* dihapus, maka objek *Room* yang diciptakan oleh *Building* akan terhapus juga.

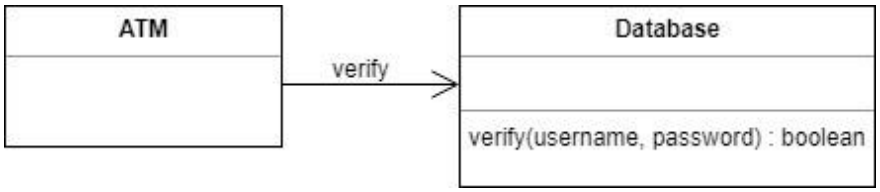
Gambar 2.1. Dependensi *create* antara 2 kelas

Instansiate adalah sebuah dependensi antara kelas dimana objek kelas memberi instansiasi sebuah *instance* dari kelas lain. Tidak seperti dependensi *create*, keberadaan objek *instantiate* tidak bergantung pada keberadaan objek asal kelas. Gambar 2.2 mengilustrasikan dependensi *instantiate*. Objek kelas *Car* memberi contoh pada objek kelas *Tire*.



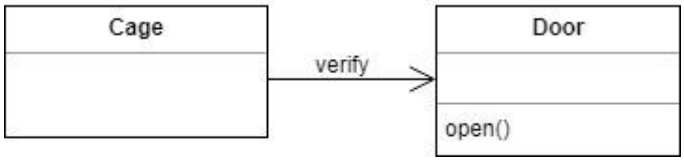
Gambar 2.2 Sebuah dependensi *instantiate* antara 2 kelas

Sebuah *call* adalah dependensi antara kelas di mana metode dari objek kelas dipanggil oleh sebuah objek dari kelas yang tergantung pada objek tersebut. Dependensi *call* bersifat sinkron. Artinya pemanggil membutuhkan nilai pengembalian dari si penerima. Gambar 2.3 menggambarkan dependensi *call*. Objek kelas ATM memanggil metode untuk memverifikasi di kelas *database*.



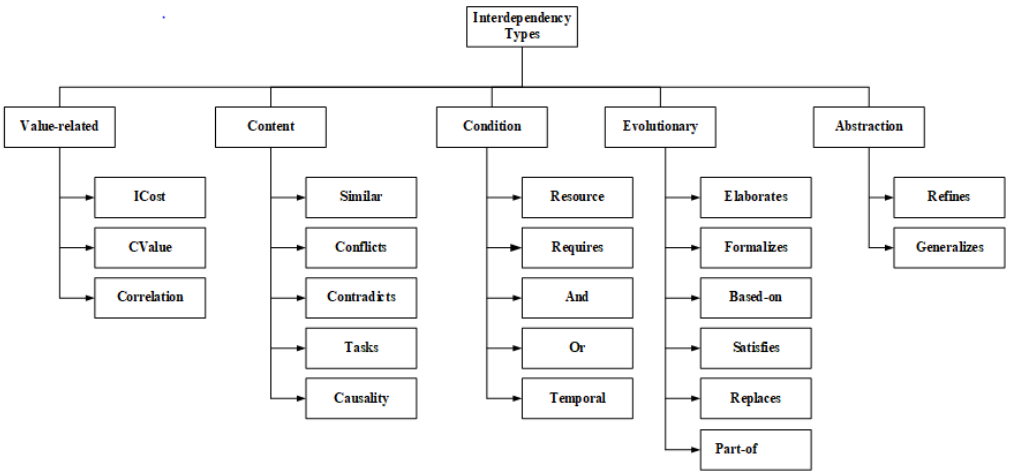
Gambar 2.3 Sebuah dependensi *call* antara 2 kelas

Send adalah dependensi antara kelas di mana metode objek kelas mengirim pesan ke objek dari kelas yang bergantung padanya. Dependensi *send* adalah asinkron, yang berarti pengirim tidak memerlukan kembalian nilai dari target. Gambar 2.4 mengilustrasikan dependensi *send*. Sebuah objek dari kelas *lift case* mengirim sebuah pesan ke sebuah objek dari kelas *door* untuk membuka *door*.



Gambar 2.4 Sebuah dependensi *send* antara 2 kelas

Interdependensi yang ada dalam diagram kelas digunakan sebagai dasar membangun interdependensi antar kebutuhan. Pada kebutuhan, terdapat tipe dari interdependensi kebutuhan yang didefinisikan oleh Robinson et al [1] seperti pada Gambar 2.5.



Gambar 2.5. Sebuah tipe model dari interdependensi kebutuhan

Interdependensi *value-related* adalah hubungan yang menjelaskan ketika kebutuhan mempengaruhi nilai pada kebutuhan lain. Jenis *value-related* terdiri dari 3 komponen di antaranya:

1. *ICost*: digunakan untuk menjelaskan bahwa satu kebutuhan mempengaruhi biaya dari implementasi kebutuhan lainnya
2. *CValue*: digunakan untuk menjelaskan bahwa satu kebutuhan mempengaruhi nilai dari kebutuhan lainnya untuk pelanggan. Tipe ini terdiri dari dalam nilai Positive dan Negatif
3. *Correlation*: digunakan untuk menjelaskan bahwa 1 kebutuhan mempengaruhi kepuasan dari kebutuhan lainnya. Tipe ini dapat terdiri dari korelasi positif, negatif, dan korelasi yang tidak ditentukan.

Interdependensi *content* (pada Gambar 2.5) digunakan untuk mengekspresikan hubungan konten dari kebutuhan. Jenis *content* ini terdiri dari 5 komponen, di antaranya:

1. *Similar*: digunakan untuk mengekspresikan hubungan kemiripan antara kebutuhan, contoh: kebutuhan duplikat atau kebutuhan *alternative*
2. *Conflict*: digunakan untuk menjelaskan bahwa pemenuhan satu persyaratan memiliki nilai negatif yang mempengaruhi persyaratan lain (tidak mewakili ketidakkonsistenan)
3. *Contradicts*: digunakan untuk mengekspresikan ketidakkonsistenan antara kebutuhan
4. *Tasks*: digunakan untuk mengekspresikan bahwa R1 menggambarkan tugas yang bergantung dari R2
5. *Causality*: digunakan untuk mengekspresikan bahwa R1 menjelaskan konsekuensi dari R2

Interdependensi *condition* (pada Gambar 2.5) digunakan untuk menyajikan *restriction* atau *constraint* yang ada pada setiap kebutuhan. Jenis *condition* ini terdiri dari 5 komponen, di antaranya:

1. *Resources* : digunakan untuk menjelaskan bahwa kebutuhan bergantung pada resource yang sama
2. *Requires*: digunakan untuk mengekspresikan bahwa satu kebutuhan tidak

dapat berfungsi tanpa kebutuhan yang lain, mis: R1 membutuhkan R2 agar dapat berfungsi, tetapi tidak sebaliknya

3. *And*: digunakan untuk mengekspresikan dua kebutuhan tidak dapat berfungsi tanpa satu sama lain, contoh: R1 tidak dapat berfungsi tanpa R2 dan R2 tidak dapat berfungsi tanpa R1
4. *Or*: digunakan untuk menjelaskan bahwa hanya satu dari dua kebutuhan yang bisa diimplementasikan, tetapi tidak keduanya
5. *Temporal*: digunakan untuk menjelaskan bahwa kebutuhan harus diimplementasikan pada urutan yang spesifik, seperti: R1 harus diimplementasikan sebelum R2

Interdependensi *evolutionary* (pada Gambar 2.5) digunakan untuk mengekspresikan evolusi dari kebutuhan, dengan mengidentifikasi kebutuhan mana yang telah menggantikan kebutuhan sebelumnya. Jenis *evolutionary* ini terdiri dari 6 komponen, di antaranya :

1. *Part-of* : digunakan untuk menjelaskan hubungan antara kebutuhan kompleks dan kebutuhan sederhana yang saling terkait dengan kebutuhan kompleks yang telah disederhanakan. Tautan ini digunakan untuk melihat bagaimana berbagai potongan kebutuhan cocok satu sama lain
2. *Replaces*: digunakan untuk menjelaskan bahwa kebutuhan tertentu telah digantikan dengan kebutuhan yang lain
3. *Satisfies*: digunakan untuk mengekspresikan hubungan antara dua kebutuhan, menjelaskan bahwa jika satu kebutuhan dipenuhi sebuah sistem, kebutuhan lainnya juga terpenuhi.
4. *Based_on*: digunakan untuk mengekspresikan bahwa satu kebutuhan dipengaruhi oleh definisi dari kebutuhan lain
5. *Formalises*: digunakan untuk menyatakan bahwa satu kebutuhan adalah versi yang lebih formal dari kebutuhan lain
6. *Elaborates*: digunakan untuk menyatakan bahwa kebutuhan dapat diuraikan oleh kebutuhan yang lain, artinya memberikan penjelasan lebih lanjut atau klarifikasi kebutuhan, contoh: deskripsi lebih komprehensif

Interdependensi *abstraction* (pada Gambar 2.5) digunakan untuk menyatakan hirarki dari kebutuhan. Jenis *abstraction* ini terdiri dari 2 komponen, di antaranya:

1. *Generalises*: digunakan untuk mengekspresikan bahwa sebuah kebutuhan menunjukkan sebuah generalisasi dari kebutuhan lain
2. *Refines*: digunakan untuk menunjukkan bahwa kebutuhan tertentu didefinisikan lebih detil dengan kebutuhan lain.

Carlshamre et al[1] membagi interdependensi kebutuhan menjadi 2 kategori, yaitu 1) berbasis fungsional mencakup *and*, *requires*, *temporal*, *icost*, dan *or* dan 2) berbasis nilai mencakup *cvalue* dan *icost*. Penjelasan dari interdependensi tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Interdependensi kebutuhan

No.	Tipe	Deskripsi
1.	*and* (R1 dan R2)	R1 mewajibkan R2 untuk berfungsi, dan R2 mewajibkan R1 untuk berfungsi
2.	*requires* (R1 *requires* R2)	R1 mewajibkan R2 agar dapat berfungsi, tapi tidak sebaliknya
3.	*temporal* (R1 *temporal* R2)	R1 harus diimplementasikan sebelum R2 atau sebaliknya
4.	*icost* (R1 *icost* R2)	R1 berdampak pada cost dari implementasi R2. Nilainya bisa negatif dan positif
5.	*or* (R1 OR R2)	Hanya salah satu dari R1 dan R2 dapat diimplementasikan

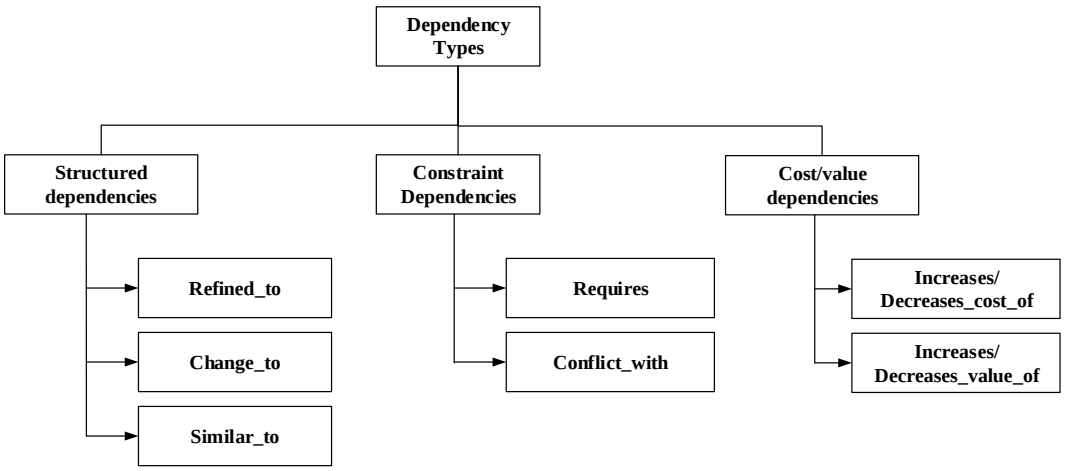
Robinson et al [1] juga menjelaskan lima jenis interaksi yang dapat dilihat sebagai tipe dasar dari hubungan kebutuhan. Kelima jenis interaksi tersebut dapat dilihat pada Tabel 2.2.

Tabel 2.2 Dasar dari hubungan kebutuhan

No	Tipe	Deskripsi
1.	STRUCTURE	R1 sama dengan R2
2.	RESOURCES	R1 dan R2 saling tergantung pada resources yang sama
3.	TASK	R1 menjelaskan dependent task dari R2
4.	CAUSALITY	R1 menjelaskan consequence dari R2
5.	TEMPORAL	R1 mempunyai temporal relation dari R2

Menurut Pohl [1], dependensi antar kebutuhan terdiri dari 10 tipe, yaitu: *refined_to, change_to, similar_to, requires, conflicts_with, increases_cost_of,*

decreases_cost_of, increases/decreases_value_of seperti yang digambarkan pada Gambar 2.6.



Gambar 2.6 Model dependensi Pohl

Dari antara 10 elemen tersebut hanya ada 5 tipe dependensi yang relevan digunakan, di antaranya:

1. "*similar_to*" lebih tepat dari dari "*similar*". "*Similar*" berarti "2 objek yang sama". "*Similar*_to" berarti 1 kebutuhan yang dinyatakan adalah sama atau *overlap* dengan 1 atau lebih kebutuhan.
2. "*Precondition*" sama dengan "*Requires*". Pada kasus ini, keduanya digunakan untuk menjelaskan relasi control proses. Pada *P-dependency* dan *D-dependency*, "*precondition*" dan "*require*" digunakan untuk menjelaskan hubungan hirarki antara 2 kebutuhan. "*Precondition*" adalah Yang paling dependensi dalam modul individu dan antara modul
3. "*Refines*" adalah sama dengan "*Refined*_to". Keduanya digunakan untuk menjelaskan hubungan di antara kebutuhan pada level yang sama. "*Refines*" berarti sebuah kebutuhan didefinisikan dengan lebih detail oleh kebutuhan lain. "*Refined*_to" berarti "sebuah kebutuhan tingkat atas disempurnakan dengan sejumlah persyaratanyang lebih spesifik. Pada model *D-dependency* dan P-dependency, kedua hubungan ini menjelaskan struktur hirarki dari kebutuhan. Pada kasus ini, 2 relasi ini menjelaskan hubungan antara 2

kebutuhan pada level yang sama. Contohnya pada 1 modul fungsi, "s*et user permissions*" menjelaskan "hanya *user* yang diijinkan yang bisa melihat/mengedit/menghapus data"

4. "*Constraint*" digunakan untuk menjelaskan hubungan antara kebutuhan non-fungsional dan fungsional. Link *constraint* digunakan untuk "menghubungkan constraint dengan objek tertentu". Tetapi apa "*a constraint*" tidak secara jelas dijelaskan pada model *P-dependency*. Pada kasus ini, "*constraint*" digunakan untuk menjelaskan relasi antara kebutuhan non-fungsional dan fungsional.
5. "*Satisfied*" digunakan untuk menjelaskan sejenis hubungan *constraint* antara kebutuhan dan bukan hubungan evolusioner seperti yang dijelaskan pada model *P-dependency*. "Satisfies" mengekspresikan "jika objek target dicapai dengan sistem final, objek sumber juga akan dipenuhi". Contoh: pada sistem PDA, ada 1 kebutuhan "*sign off screen* (dengan menangkap tandatangan) " dan pada sistem desktop, ada 1 kebutuhan "penilai harus submit penilaian mereka untuk autorisasi setelah selesai"

2.2. Interdependensi diagram kelas

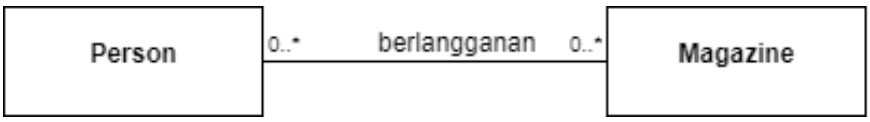
Pada rekayasa kebutuhan perangkat lunak terdapat istilah yang dikenal dengan *cohesion* dan *coupling*. *Cohesion* dan *coupling* merupakan konsep dasar dalam perancangan dan rekayasa perangkat lunak. Kohesi bertujuan mengukur seberapa kuat keterikatan fungsi-fungsi dalam modul program, sedangkan *coupling* bertujuan mengukur seberapa banyak modul-modul yang saling bergantung pada modul lainnya.

Diagram kelas merupakan salah satu pemodelan yang menggambarkan relasi yang menghubungkan suatu kelas dengan kelas lainnya. Adapun beberapa contoh interdependensi dari kelas diagram adalah sebagai berikut:

1. Asosiasi/ *Associations*

Asosiasi menunjukkan 2 elemen model yang memiliki tipe hubungan umum. Salah satu contoh dari penggunaan asosiasi adalah Person berlangganan Magazine. Tanda 0..* menandakan kuantitas dari relasi antara Person dan Magazine. Person bisa saja tidak berlangganan Magazine, tetapi banyak Person

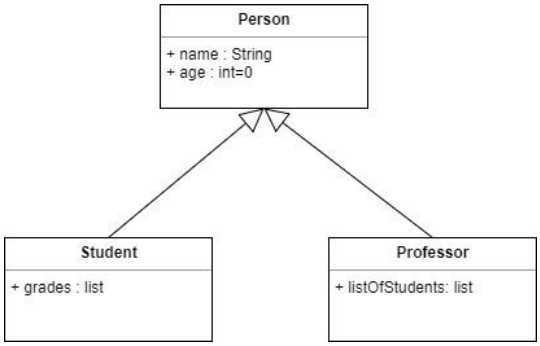
bisa berlangganan banyak Magazine. Contoh relasi asosiasi *Person* dan *Magazine* disajikan pada Gambar 2.7.



Gambar 2.7 *Association* antara kelas *Person* dan *Magazine*

2. Generalisasi/ *Generalizations*

Generalisasi menunjukkan warisan. Generalisasi mengimplikasi suatu kelas mewarisi karakter kelas lain. *Superclass* (*base class*) dalam hubungan generalisasi juga dikenal sebagai "*parent*", *superclass*, *base class*, atau *base type*. Contoh relasi *generalization* dari Student dan Professor disajikan pada Gambar 2.8.



Gambar 2.8 Generalizations dari kelas *Students* dan *Professor*

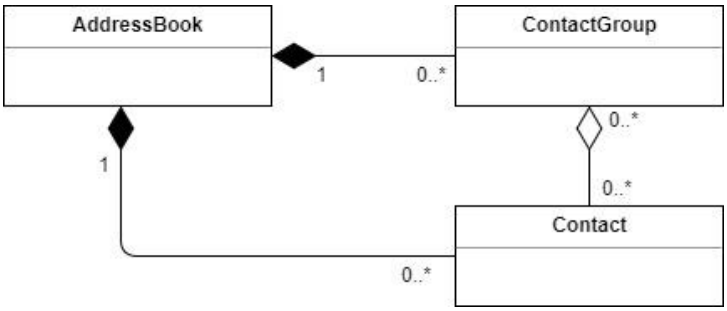
3. Agregasi/*Aggregations*

Agregasi digunakan untuk menggambarkan elemen yang terdiri dari komponen yang lebih kecil. Agregasi digambarkan dengan simbol diamond. Ada 2 jenis dari agregasi, di antaranya:

a) *Strong aggregation* atau *composite aggregation* ditunjukkan dengan simbol diamond dengan warna hitam dan komposisi satu per satu. Jika *parent/composite aggregation* dihapus, maka kelas yang merupakan bagian dari kelas itu akan terhapus juga. Akan tetapi, bagian dari kelas itu dapat dihapus dari *compostion* tanpa harus menghapus semua composition. Contoh: ketika ***AddressBook*** dihapus maka ***contactGroup*** dan ***Contact*** juga

akan terhapus. Dari gambar juga diketahui bahwa 1 ***AddressBook*** bisa tidak memiliki ***ContactGroup*** atau 1 ***AddressBook*** bisa memiliki banyak ***ContactGroup***.

b) *Weak aggregation* ditunjukkan dengan simbol *diamond* yang isinya kosong. Relasi *weak aggregation* mendeskripsikan jika salah satu kelas dihapus tidak akan berpengaruh pada kelas lainnya. Contoh *aggregations* pada kelas ***Address Book*** dapat dilihat pada Gambar 2.9. ***ContactGroup*** adalah grup virtual dari ***contact***. Apabila ***ContactGroup*** dihapus, maka tidak ada ***Contact*** yang dihapus. Gambar ini juga menjelaskan bahwa 1 ***AddressBook*** bisa saja tidak memiliki **Contact** atau 1 **AddressBook** bisa memiliki banyak **Contact.**



Gambar 2.9. *Aggregations* pada kelas *AddressBook*

2.3. Penelitian Sebelumnya dan penelitian yang diusulkan

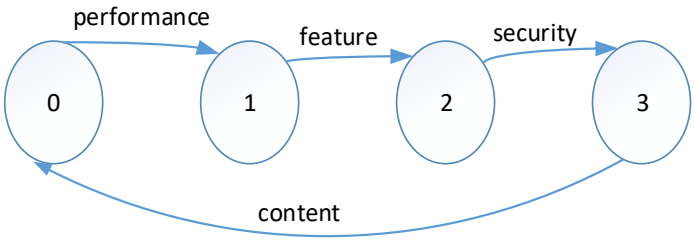
Pada penelitian sebelumnya, kebutuhan merupakan hal yang menjadi dasar dalam pengembangan perangkat lunak. Beberapa penelitian tersebut dapat dijadikan sebagai rujukan untuk usulan metode dalam penelitian ini. Beberapa penelitian terkait model dependensi kebutuhan, dapat dilihat pada subbab berikutnya.

2.3.1. Visualisasi dependensi antar kebutuhan

Menurut Widiastuti, Siahaan[3], terdapat sebuah metode untuk visualisasi dependensi antar kebutuhan yang disebut LTS-RC (*Labelled Transition System for Requirement Change*). Metode ini digunakan untuk memodelkan perubahan kebutuhan berdasarkan LTS. LTS ini dikenal sebagai model yang efektif dalam mengambarkan perubahan perilaku sistem.

LTS-RC adalah LTS yang berguna dalam pemodelan perubahan kebutuhan perangkat lunak sehingga mempermudah *stakeholder* untuk mengamati alur perubahan kebutuhan beserta dampaknya. Metode ini dapat berperan dalam penyusunan strategi perubahan kebutuhan yang optimal.

Sebagai contoh, LTS-RC memodelkan komponen perubahan kebutuhan perangkat lunak sistem informasi FRS *Online* sederhana: FRS *Online* ({0,1,2},{*performance*, *feature*, *security*, *content*}, {(0, *performance*, 1), (1, *feature*, 2), (2, *security*, 3), (3, *content*, 0)},0)



Gambar 2.10 Pemodelan perubahan kebutuhan dengan LTS-RC

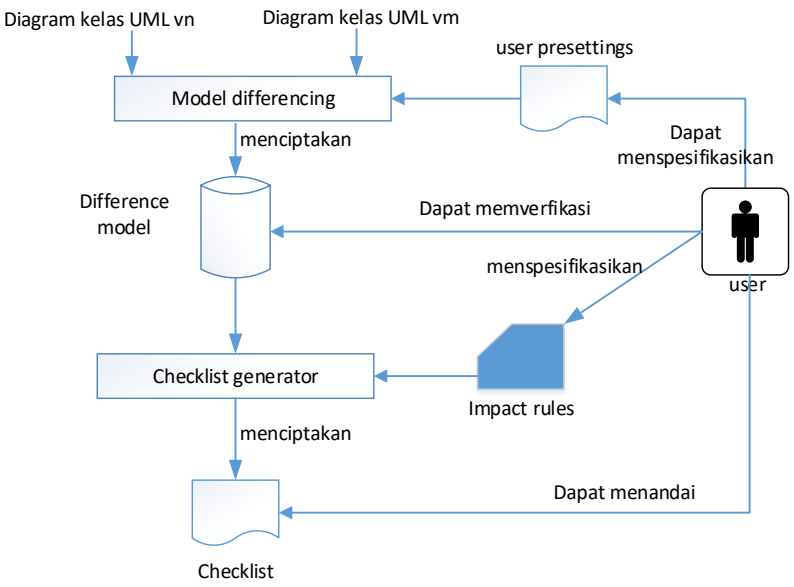
Gambar 2.10 merepresentasikan LTS-RC secara grafis. Angka yang digambarkan dalam *state* menunjukkan urutan *state*. *State* diberi nomor bilangan bulat dengan nol menunjukkan *inisial state*. Transisi dimulai dari *inisial state*. Berawal dari *state* 0, tindakannya adalah melakukan perubahan *performance* dan kemudian berlanjut pada *state* 1, pada *state* 1, tindakannya adalah melakukan perubahan *feature* dan kemudian menuju *state 2*, pada *state* 2, tindakannya adalah melakukan perubahan *security* akhirnya pada *state* 3, perubahan kebutuhan kembali ke *inisial state* dengan melakukan perubahan *content*.

2.3.2. Analisis dampak perubahan perangkat lunak pada diagram kelas

Menurut Muller [4], analisis terhadap dampak perubahan pada perangkat lunak dilakukan dengan menggunakan perubahan pada diagram kelas UML. Pendekatan yang digunakan adalah pendekatan berbasis model untuk analisis dampak yang mana *impact rule*-nya ditentukan oleh DSL (*Domain Specific Language*). Perubahan diagram kelas UML diidentifikasi secara otomatis dengan menggunakan *model differencing*.

Pendekatan terhadap analisis dampak terdiri dari 2 langkah utama, yaitu identifikasi perbedaan model dan pembuatan *checklist* berdasarkan perbedaan ini. Namun, penelitian ini tidak menghubungkan dampak perubahan ini pada level kebutuhan. Dua langkah utama tersebut disajikan pada Gambar 2.11. Penjelasan secara rinci dua langkah utama adalah:

- *Model differencing* : model ini menentukan perbedaan antara 2 model yang harus didefinisikan pengguna sebagai input. Pengguna dapat memverfikasi kebenaran perbedaan ini dan mengintegrasikan *user presetting*, jika pengguna ingin memperbaiki perbedaan yang dilaporkan. Jika *user presetting* ada, *user presetting* akan diperhitungkan dalam proses model *differencing*. Hasil dari langkah ini adalah model perbedaan yang berisi semua perbedaan. Pembuatan *checklist:* model perbedaan dilalui oleh generator *checklist* untuk membuat *checklist*. Setiap perbedaan dari model perbedaan dilewatkan pada *impact rule* yang tersedia. Kemudian, setiap *impact rule* menganalisis perbedaan dan membuat daftar petunjuk pada langkah pengembangan lebih lanjut, jika perbedaannya dianggap relevan.

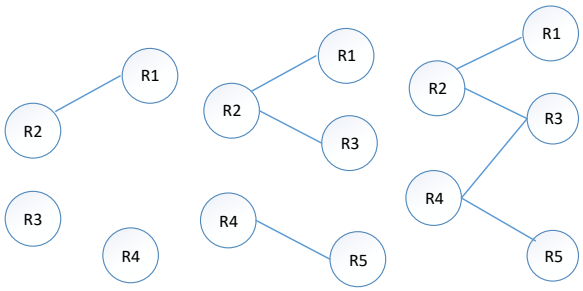


Gambar 2.11 Gambaran umum pendekatan terhadap analisis dampak

2.3.3. Prediksi *bugs* berdasarkan dependensi antar kebutuhan

Menurut Wang [5], pembangunan model dependensi antar kebutuhan dilakukan berdasarkan informasi frekuensi kemunculan *bug*. Setelah itu, penyelidikan terhadap dependensi kebutuhan berkolaborasi dengan memprediksi *bug* yang dapat memberikan perkiraan awal mengenai kualitas perangkat lunak. Pada penelitian ini, pembentukan jaringan dependensi kebutuhan dilakukan setelah mengindentifikasi dependensi kebutuhan.

Penelitian ini menggunakan identifikasi dependensi kebutuhan secara manual untuk membangun landasan yang kokoh. Terdapat empat atau tiga praktisi yang terlibat dalam proses konstruksi jaringan dependensi kebutuhan untuk proyek eksperimental. Contoh dari jaringan dependensi kebutuhan digambarkan pada Gambar 2.12.

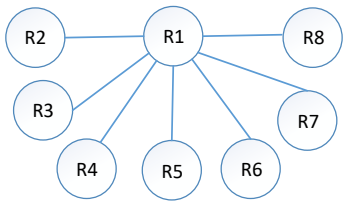


Gambar 2.12 Contoh koneksi jaringan dependensi kebutuhan[5]

Dari Gambar 2.12 dapat diketahui bahwa semakin banyak koneksi, maka bug yang akan muncul akan semakin banyak. R menandakan Kebutuhan. Sedangkan garis (*edge*) dari 1 kebutuhan ke kebutuhan lainnya menandakan dependensi antar kebutuhan, seperti *pre-condition*, *similar_to*, *constraint.*

Di sisi lain, penelitian ini juga menjelaskan tentang sentralitas, peneliti mengasumsikan bahwa semakin tinggi koneksi dan sentralitas, maka cenderung semakin rentan terhadap *bug* perangkat lunak.

Gambar 2.13 menunjukkan bahwa adanya dependensi yang kuat antara R1 (Kebutuhan 1) dengan R2, R3, R4, R5, R6, R7, R8. Hal ini menandakan bahwa sentralisasi dependensi kebutuhan ini tergolong tinggi. Hal ini memungkinkan tingkat kemuculan *bug* pada perangkat lunak adalah tinggi.



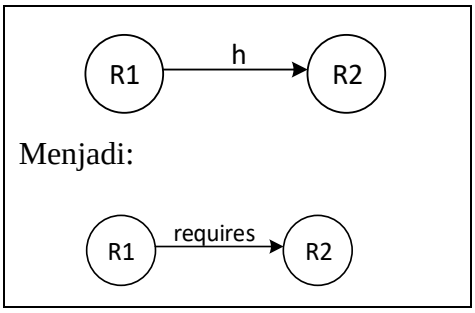
Gambar 2.13 Contoh jaringan dependensi kebutuhan dengan sentralisasi

2.3.4. Membangun model dependensi kebutuhan berdasarkan dependensi diagram kelas

Metode yang diusulkan ini adalah metode baru yang diperoleh dari dependensi yang ada pada diagram kelas. Diagram kelas ini sangat diperlukan dalam pembangunan perangkat lunak berbasis *object-oriented.* Desain diagram kelas ini dapat digunakan sebagai prasyarat dalam pemodelan dependensi kebutuhan. Akan tetapi, jika dilihat dari penelitian sebelumnya, belum ada dibahas tentang pemodelan dependensi kebutuhan berdasarkan dependensi diagram kelas dan pada dokumen SKPL (Spesifikasi Kebutuhan Perangkat Lunak) juga belum ada pencatatan informasi dependensi kebutuhan.

Pencatatan ini berguna dalam pengindentifikasian biaya terhadap pengembangan perangkat lunak pada tahap pengumpulan kebutuhan. Tidak hanya itu, pencatatan ini bisa memprediksi kebutuhan-kebutuhan apa saja yang akan berdampak jika suatu kebutuhan diubah. Oleh sebab itu, fokus dari penelitian ini adalah memodelkan dependensi kebutuhan berdasarkan dependensi diagram kelas. Salah satu contoh dari pemodelan kebutuhan berdasarkan dependensi diagram kelas dapat dilihat pada Gambar 2.14.

Pada Gambar 2.14 dapat diketahui bahwa Kebutuhan 1 (R01) mempunyai dependensi h (*strong aggregation*) terhadap Kebutuhan 2 (R02). Dependensi ini diperoleh dari dependensi diagram kelas. Kemudian dilakukan pendefinisian dependensi kebutuhan terhadap dependensi diagram kelas. Berdasarkan pendefinisian dependensi tersebut, maka diperoleh informasi h (*strong aggregation*) didefinisikan sebagai *requires*. Sehingga R1 memiliki dependensi *requires* terhadap R2.



Gambar 2.14 Contoh dependensi kebutuhan berdasarkan dependensi diagram kelas

2.3.5. Posisi penelitian yang diusulkan terhadap penelitian sebelumnya

Untuk mengetahui posisi penelitian penelitian yang diusulkan dengan penelitian sebelumnya, maka dilakukan perbandingan antara penelitian tersebut berdasarkan segi fokus penelitian, input dan output dari penelitian. Posisi penelitian yang dibandingkan dapat dilihat pada Tabel 2.3.

Tabel 2.3. Posisi penelitian yang diusulkan terhadap penelitian sebelumnya.

No.	Pembanding	Metode penelitian			
		Widiastuti, Siahaan	**Muller**	**Wang**	**Metode usulan**
1.	Fokus penelitian	Visualisasi dependensi antar kebutuhan	Analisis terhadap dampak perubahan pada perangkat lunak dengan menggunakan perubahan pada diagram kelas UML	Prediksi *bugs* berdasarkan dependensi antar kebutuhan.	Membangun model dependensi kebutuhan berdasarkan dependensi diagram kelas
2.	Input	Dependensi kebutuhan	Dependensi kebutuhan, diagram kelas	Dependensi kebutuhan, kelas dan *bugs*	Diagram kelas

No.	Pembanding	Metode penelitian			
		Widiastuti, Siahaan	**Muller**	**Wang**	**Metode usulan**
3.	Output	Visualisasi dependensi antar kebutuhan	Kelas-kelas yang berubah jika suatu kelas berubah	Kelas-kelas yang mungkin terkena *bug*	Model dependensi kebutuhan

2.4. Pendefinisian pasangan dependensi kebutuhan dan dependensi diagram kelas

Dari studi literatur yang telah dibahas sebelumnya, maka penulis menganalisis pemetaan antara dependensi kebutuhan dan diagram kelas, seperti yang ditunjukkan pada Tabel 2.4.

Tabel 2.4. Pasangan dependensi kebutuhan dan dependensi diagram kelas

No.	Pasangan	
	Dependensi kebutuhan	**Dependensi diagram kelas**
1.	AND (R1 dan R2)	*Implements*
2.	REQUIRES (R1 *requires* R2)	*strong aggregation*
3.	TEMPORAL (R1 *temporal* R2)	*uses, strong aggregation*
6.	ICOST (R1 *icost* R2)	-
7.	OR (R1 *or* R2)	-

Dari beberapa pasangan dependensi kebutuhan dan dependensi kelas diagram, nantinya akan dipilih pasangan dependensi mana yang akan diimplementasikan disesuaikan dengan dataset yang diperoleh dan kebutuhan utama sistem.

2.5. Metode Penghitungan Kemiripan

Penghitungan kemiripan dapat dilakukan dengan berbagai metode, seperti:

1) *Levenshtein Distance* [14]

Levenshtein distance (jarak *Levenshtein*) disebut juga *edit distance* ditemukan oleh ilmuan asal Rusia bernama *Vladimir Levenshtein* pada tahun

1963. Jarak *Levenshtein* adalah angka terkecil dari penyisipan, penghapusan, dan substitusi yang dibutuhkan untuk mengubah suatu *string* menjadi *string* lainnya. Jarak *levenshtein* sering juga disebut dengan operasi minimum untuk melakukan pengubahan satu *string* ke *string* lain.

Contoh hasil penggunaan jarak *Levenshtein*, yaitu *string* "thesis" dan "tesis" memiliki *distance* 1 karena hanya perlu dilakukan satu operasi saja untuk mengubah satu *string* ke *string* yang lain. Pada kasus kedua *string* di atas, *string* "thesis" dapat menjadi "tesis" hanya dengan melakukan satu operasi, yaitu penghapusan karakter "h". Penghitungan *online* dari *Levenshtein distance* dapat dilakukan dengan mengakses https://planetcalc.com/1721/. Pengukuran similaritas antara 2 kata dapat juga dilakukan dengan menggunakan jarak *Lavensthein*. Rumus similaritas *Levenshtein* dijabarkan pada Persamaan 2.1.

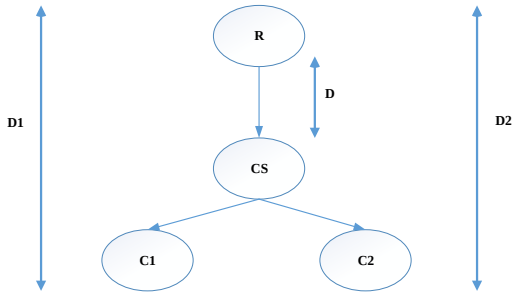
$$Sim = 1 - \frac{dis}{MaxLength} \tag{2.1}$$

Persamaan 2.1 menunjukkan bahwa dis = jarak *Levenshtein*, $MaxLength$ adalah jumlah huruf pada kata terpanjang. Pengukuran similarity untuk 2 kata di atas yaitu *thesis* dan tesis (yang sudah diketahui jarak *Levenshtein* (dis) = 1, dan nilai *maxLength* adalah 6 dari kata *thesis*) adalah

$$Sim = 1 - \frac{dis}{MaxLength} \qquad = 1 - \frac{1}{6} = 0.83$$

2) ***Wu*** dan ***Palmer*** [15]

Prinsip perhitungan kemiripan didasarkan pada metode penghitungan tepi *root* seperti yang ditunjukkan pada Gambar 2.16. Terdapat *root*/simpul akar, kemudian C1 dan C2 merepresentasikan 2 elemen yang akan dihitung nilai kemiripannya. Prinsip kemiripan ini didasarkan pada jarak (D1 dan D2) yang memisahkan induk terdekat (*common ancestor/CS*) dari C1 dan C2 dari simpul *root*. Rumus dari kemiripan ***Wu*** dan ***Palmer*** ditunjukkan pada Persamaan 2.2.

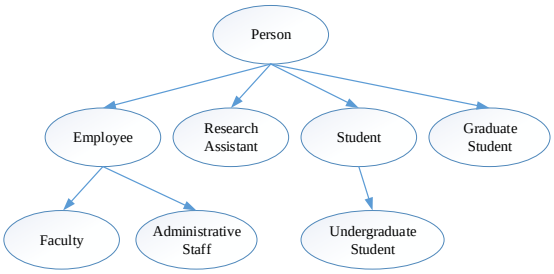


Gambar 2.15 Representasi dari perhitungan ***Wu*** dan ***Palmer***

$$SimWP = \frac{2\ X\ D}{(D1+D2)} \quad (2.2)$$

Dari Persamaan 2.2, dapat diketahui, bahwa D1 dan D2 = *Distance* (jarak) pemisah antara C1 dan C2 dari node *root* sedangkan D= *Distance* (jarak) yang memisahkan *closest common ancestor* (CS) atau node terdekat antara node C1 dan C2 dari *node* R.

Salah satu contoh penggunaan *Wu* dan *Palmer* dengan taksonomi yang ada pada Gambar 2.17. Menghitung similaritas berdasarkan dari C1 dan C2. C1 adalah *Person* dan C2 adalah *Research Assistant*



Gambar 2.16 Contoh representasi *tree Wu* dan *Palmer*

Sim (*Person, Research Assistant*) = 2*D/(D1+D2)

= 2*1/(1+2)

= 2/3

= 0.67

3) *Cosine similarity* [16]

Cosine similarity mengukur kemiripan antara dua dokumen atau teks. Pada *cosine similarity* dokumen atau teks dianggap sebagai *vector*. *Cosine similarity* dihitung dengan rumus pada Persamaan 2.3.

$$Cosine\ similarity\ = \frac{A.B}{|A||B|} = \frac{\sum_{i=1}^{n} Ai\ X\ Bi}{\sqrt{\sum_{i=1}^{n} (Ai)^2}\ X\ \sqrt{\sum_{i=1}^{n} (Bi)^2}} \quad (2.3)$$

Dari Persamaan 2.3, A adalah Vektor A, yang akan dibandingkan kemiripannya, B adalah Vektor B, yang akan dibandingkan kemiripannya. A.B = *dot product* antara vektor A dan vektor B. |A||B| = *cross product* antara panjang vektor |A| dan panjang vektor |B|. Contoh *cosine similarity* dapat dilihat pada Tabel 2.5.

Tabel 2.5 Kemunculan kata pada kebutuhan dan kelas

Kebutuhan dan kelas	token unik pada kebutuhan dan kelas							
	Patron	**library**	**manage**	**account**	**number**	**history**	**Open**	**state**
R1	1	1	1	1	0	0	0	0
C1	0	0	0	1	1	1	1	1

Hasil dari pengukuran nilai *cosine similarity* dapat dilihat pada Persamaan 2.4.

$$= \frac{\sum_{i=1}^{n} Ri\ X\ Ci}{\sqrt{\sum_{i=1}^{n} (Ri)^2}\ X\ \sqrt{\sum_{i=1}^{n} (Ci)^2}} \quad (2.4)$$

$$= \frac{(1*0)+(1*0)+(1*0)+(1*1)+(0*1)+(0*1)+(0*1)+(0*1)}{\sqrt{1^2+1^2+1^2+1^2+0^2+0^2+0^2+0^2}+\sqrt{0^2+0^2+0^2+1^2+1^2+1^2+1^2+1^2}}$$

$$= \frac{1}{4.47} = 0.22$$

4) *Greedy Algorithms*

Algoritma ini adalah algoritma yang mengikuti pemecahan masalah heuristik untuk membuat pilihan optimal secara lokal pada setiap tahap dengan tujuan menemukan optimal global [17]. Algoritma *greedy* dikenal cukup sederhana. Algoritma ini biasanya digunakan untuk menemukan rute terpendek dalam graf. Menurut ***Al-Khiaty*** dan ***Ahmed*** [18], algoritma *greedy* dapat dilakukan dengan cara berikut ini:

1. Mengambil nilai similaritas tertinggi pada *cell* matriks kebutuhan dan kelas, S_{mxn},

2. Menghapus *cell* pada matriks S_{mxn}, yang berada pada kolom dan baris yang sama dengan nilai similaritas tertinggi tersebut
3. Jika masih ditemukan nilai similaritas tertinggi berikutnya, kembali ke langkah pertama (1) ,
4. Jika tidak ditemukan lagi nilai similaritas tertinggi, maka dihitung nilai similaritas dari kebutuhan dan kelas dengan menggunakan Persamaan 2.5

$$\text{Sim } S_{mxn}, = \frac{2\ X\ (\sum_{i=1}^{\min[m][n]} maks\ token Sim|m_i||n_i|)}{m+n} \quad (2.5)$$

Persamaan 2.5 menjelaskan bahwa pada similaritas S_{mxn}. m adalah jumlah token pada kebutuhan sedangkan n adalah jumlah token pada kelas. Nilai i menandakan iterasi pada token yang ada di kebutuhan maupun kelas atau dinotasikan dengan i={1…n} atau i={1..m}. Nilai similaritas pada matriks S_{mxn} diperoleh dengan cara mengalikan jumlah nilai similaritas maksimum dari token i pada m dan token i pada n yang berada pada *cell* S_{mxn}, yang jumlah kolomnya paling kecil dan dikalikan dua. Hasil perkalian tersebut dibagi dengan jumlah *token* yang ada pada kebutuhan dan kelas. Contoh dari penghitungan similaritas S_{mxn} dapat dilihat pada Tabel 2.6.

Tabel 2.6 Kebutuhan dan kelas setelah pra-pemrosesan

Kode ID	**Kebutuhan/kelas setelah pra-pemrosesan**	**Token**
F01	Kebutuhan	*Patron// library// manage// account//*
C01	Kelas	*Book// ISBN// name// subject// overview// publisher// publication// date//*

Langkah-langkah pengerjaan algoritma *Greedy* pada data yang ada di Tabel 2.6 adalah:

1. Pada awalnya, matriks similaritas $S_{m\times n}$ diinisialisasi yang ditunjukkan pada Tabel 2.7. $S_{m\times n}$ adalah matriks similaritas antara C01 dan F01. Nilai m adalah jumlah token dalam C01 dan n adalah jumlah token pada F01.

Tabel 2.7 Inisialisasi matriks $S_{m\times n}$

Kode ID	**Kelas/Atribut**	**FO1 (Patron Library manage account)**			
		Patron (1)	**Library (2)**	**Manage (3)**	**Account (4)**
C01	book (1)				
	ISBN (2)				
	Name (3)				
	Subject (4)				
	Overview (5)				
	Publisher (6)				
	Publication (7)				
	Date (8)				

2. Pada matriks $S_{m\times n}$, setiap indeks pada matriks $S_{m\times n}$ token pada C01 adalah i sedangkan indeks token pada F01 adalah j. Sehingga dinotasikan i = {1..m} sedangkan j = {1..n}. Untuk setiap S_{ij} pada matriks $S_{m\times n}$, dihitung nilai similaritas antara token i pada C01 dan token j pada F01 dengan menggunakan ***Wu*** dan ***Palmer*** atau *Lavensthein*. *Lavensthein* digunakan apabila nilai similaritas kata yang diperoleh dari ***Wu*** dan ***Palmer*** tidak ditemukan (nilai -1). Tabel 2.8 menunjukkan $S_{m\times n}$yang sudah dihitung nilai similaritasnya.
3. Mengambil nilai similaritas tertinggi dari matriks $S_{m\times n}$. Tabel 2.8 menunjukkan bahwa nilai similaritas tertinggi pertama berada pada baris 7 (*publication)* dan kolom 2 (*library)* yaitu 0.56. Setelah itu, semua *cell* nilai similaritas yang berada pada baris dan kolom yang sama dengn nilai similaritas tertinggi tersebut dicoret.

Tabel 2.8 Pencarian nilai similaritas tertinggi pertama dari matriks $S_{m\times n}$

Kode ID	Kelas/Atribut	FO1 (Patron Library manage account)			
		Patron (1)	Library (2)	Manage (3)	Account (4)
C01	Book (1)	0.38	0.52	0.00	0.12
	ISBN (2)	0.00	0.00	0.00	0.14
	Name (3)	0.14	0.13	0.50	0.31
	Subject (4)	0.15	0.14	0.00	0.50
	Overview (5)	0.13	0.13	0.00	0.43
	Publisher (6)	0.12	0.11	0.00	0.25
	Publication (7)	0.40	0.56	0.00	0.13
	Date (8)	0.14	0.13	0.33	0.31

4. Jika masih terdapat nilai similaritas tertinggi berikutnya pada matrik $S_{m\times n}$, ulangi langkah 3 dan 4,
5. Pada kasus ini, masih ada nilai tertinggi pada matriks $S_{m\times n}$. Nilai tertinggi kedua yang diperoleh adalah 0.50 yang berada pada baris 4 dan kolom 4. Kemudian semua *cell* pada matriks $S_{m\times n}$ yang berada pada baris dan kolom yang sama dengan nilai similaritas tertingi tersebut dicoret. Nilai similaritas tertinggi kedua dapat dilihat pada Tabel 2.9

Tabel 2.9 Pencarian nilai similaritas tertinggi kedua dari matriks $S_{m\times n}$

Kode ID	Kelas/Atribut	FO1 (Patron Library manage account)			
		Patron (1)	Library (2)	Manage (3)	Account (4)
C01	Book (1)	0.38	0.52	0.00	0.12
	ISBN (2)	0.00	0.00	0.00	0.14
	Name (3)	0.14	0.13	0.50	0.31
	Subject (4)	0.15	0.14	0.00	0.50
	Overview (5)	0.13	0.13	0.00	0.43
	Publisher (6)	0.12	0.11	0.00	0.25
	Publication (7)	0.40	0.56	0.00	0.13
	Date (8)	0.14	0.13	0.33	0.31

6. Pada matriks $S_{m\times n}$, pengecekan nilai tertinggi dilakukan kembali pada matriks $S_{m\times n}$. Nilai similaritas yang ketiga tertinggi adalah 0.50 yang berada pada baris 3 dan kolom 3 kemudian semua *cell* yang berada pada baris dan

kolom yang sama dengan nilai similaritas tersebut dicoret dapat diihat pada Tabel 2.10.

Tabel 2.10 Pencarian nilai similaritas tertinggi ketiga dari matriks $S_{m \times n}$

Kode ID	Kelas/Atribut	FO1 (Patron Library manage account)			
		Patron (1)	Library (2)	Manage (3)	Account (4)
C01	Book (1)	0.38	0.52	0.00	0.12
	ISBN (2)	0.00	0.00	0.00	0.14
	Name (3)	0.14	0.13	0.50	0.31
	Subject (4)	0.15	0.14	0.00	0.50
	Overview (5)	0.13	0.13	0.00	0.43
	Publisher (6)	0.12	0.11	0.00	0.25
	Publication (7)	0.40	0.56	0.00	0.13
	Date (8)	0.14	0.13	0.33	0.31

7. Pengecekan nilai tertinggi selanjutnya dilakukan pada matriks $S_{m \times n}$ yaitu 0.38 sebagai nilai similaritas tertinggi keempat yang berada pada baris 1 dan kolom 1 dapat dilihat pada Tabel 2.11.

Tabel 2.11. Pencarian nilai similaritas tertinggi keempat dari matriks $S_{m \times n}$

Kode ID	Kelas/Atribut	FO1 (Patron Library manage account)			
		Patron (1)	Library (2)	Manage (3)	Account (4)
C01	Book (1)	0.38	0.52	0.00	0.12
	ISBN (2)	0.00	0.00	0.00	0.14
	Name (3)	0.14	0.13	0.50	0.31
	Subject (4)	0.15	0.14	0.00	0.50
	Overview (5)	0.13	0.13	0.00	0.43
	Publisher (6)	0.12	0.11	0.00	0.25
	Publication (7)	0.40	0.56	0.00	0.13
	Date (8)	0.14	0.13	0.33	0.31

8. Apabila sudah tidak ditemukan lagi nilai similaritas berikutnya, maka nilai similaritas dihitung dengan menggunakan Persamaan 2.6.

$$\text{Sim } S_{mxn}, = \frac{2\,X\,(\sum_{i=1}^{\min[m][n]} maks\ tokenSim|m_i||n_i|)}{m+n} \quad (2.6)$$

$$= \frac{2\,X\,(0.56+0.50+0.50+0.38)}{(8+4)}$$

$$= \frac{1.94}{12} = 0.32$$

2.6. Perbandingan antara metode penghitungan kemiripan

Perbandingan antara metode kemiripan ini dilakukan untuk mengolah penghitungan kemiripan teks pada informasi kelas dan pernyataan kebutuhan. Tabel 2.11 menunjukkan karakteristik ke-4 metode berdasarkan jenis kemiripan dan jenis data. Berdasarkan jenis kemiripan, metode ***Levenshtein*** dan ***Wu*** dan ***Palmer*** fokus pada jenis kemiripan sintaktik dari dua kata.

Tabel 2.12 Perbandingan antara metode penghitungan kemiripan

No.	Informasi dan Fitur	Metode penghitungan kemiripan			
		Levenshtein Distance	**Wu dan Palmer**	**Cosine similarity**	**Greedy Algorithm**
1.	jenis kemiripan	sintaktik	sintaktik	sintaktik dan semantik	sintaktik dan semantik
2.	jenis data	kata	Kata	kalimat	Kalimat

2.7. Gwet's AC1

Gwet's AC1 adalah salah satu metode yang digunakan untuk mengukur indeks kesepakatan antara dua pengamat. Gwet's AC1 menunjukkan pendekatan yang lebih dapat diandalkan dibandingan dengan Cohen Kappa [20]. Penghitungan nilai AC1, hasil dari observasi pengamat dimasukkan dalam matrik 2x2 seperti yang ditunjukkan pada Tabel 2.12. Penulisan Hasil Pengamatan dapat dilihat pada Tabel 2.12.

Tabel 2.13 Tabel Penulisan Hasil Pengamatan

Pengamat 1	**Pengamat 2**		
	Ya	**Tidak**	**Total**
Ya	A	B	B1=A+B
Tidak	C	D	B2=C+D
Total	A1=A+C	A2=B+D	N

Pada Tabel 2.12 terdapat dua pengamat yang mengklasifikasikan N subjek ke dua kemungkinan kategori. Dua kategori dilabelkan sebagai "ya" dan "tidak". A diklasifikasikan oleh dua pengamat sebagai ya. B diklasifikasikan oleh pengamat 1 sebagai ya dan tidak oleh pengamat 2. D diklasifikasikan oleh dua pengamat sebagai tidak. C diklasifikasikan oleh pengamat 1 sebagai tidak dan ya oleh pengamat 2. B1 dan B2 menunjukkan jumlah subjek yang diklasifikasikan pada setiap kategori oleh pengamat 2. A1 dan A2 juga menunjukkan jumlah subjek yang diklasifikasikan pada setiap kategori oleh pengamat 1. Perhitungan AC1-statistic ditunjukkan pada Persamaan 2.7. Perhitungan AC1-statistic melibatkan perhitungan kesepakatan yang terobservasi (*P*) dan probabilitas *chance-agreement* ($e(\gamma)$). Rumus kesepakatan yang terobservasi dan probabilitas *chance-agreement* ditunjukkan pada Persamaan 2.8 dan Persamaan 2.9.

AC1-statistic, $AC1 = \frac{P - e(\gamma)}{1 - e(\gamma)}$ (2.7)

Kesepakatan yang terobservasi, $P = \frac{A+D}{N}$ (2.8)

Probabilitas *chance-agreement*, $e(\gamma) = 2P_1(1 - P_1)$ (2.9)

Pada Persamaan 2.8, A merupakan banyaknya data yang dikelompokkan ke dalam kategori "Ya" oleh kedua pengamat. Kemudian D merupakan banyaknya data yang dikelompokkan ke dalam kategori "tidak" oleh kedua pengamat. Pada

persamaan 2.9, $P_1 = \frac{(A1+B1)/2}{N}$ merepresentasikan perkiraan kemungkinan seorang pengamat (1 atau 2) mengelompokkan data ke dalam kategori "Ya". A_1 dan B_1 masing-masing adalah jumlah pengamat 1 atau 2 mengelompokkan data ke dalam kategori "Ya", sedangkan N adalah jumlah data. Tabel 2.13 berikut merupakan interpretasi nilai dari Gwet AC1 yang mengukur indeks kesepakatan antar dua pengamat [21].

Tabel 2.14 Interpretasi Nilai Gwet AC1

Index Kappa	**Proporsi Kesepakatan**
< 0	Rendah
0.01 - 0.20	Sedikit
0.21 - 0.40	Cukup
0.41 - 0.60	Sedang
0.61 - 0.80	Substansial
0.81 – 1	Hampir sempurna

Berikut adalah contoh perhitungan *AC1-static*:

Terdapat dua orang pengamat yang dimintai pendapatnya terhadap 100 topik penelitian, apakah topik tersebut diminati atau tidak. Hasil pengamatan terlihat seperti Tabel 2.14.

Tabel 2.15 Data Pengamatan Contoh Kasus Kappa

Pengamat 1	**Pengamat 2**		
	Ya	**Tidak**	**Total**
Ya	95	5	100
Tidak	45	5	50
Total	140	10	150

P = $\frac{95+5}{150}$ = 0.67

Jika dihitung menggunakan indeks Cohen Kappa statistik maka didapat nilai indeks Kappa sebagai berikut :

$e(K) = \left(\frac{100}{150}\right) x \left(\frac{140}{150}\right) + \left(\frac{50}{150}\right) x \left(\frac{10}{150}\right) = 0.645$

Kappa = (0.67 – 0.645) / (1-0.645) = 0.070

Nilai indeks Cohen Kappa yang dihasilkan pada data pengamatan Tabel 2.14 sangatlah rendah yaitu 0.070, hal ini disebabkan oleh kesepakatan antara dua pengamat sangat rendah. Sedangkan nilai indeks Gwet's AC1 dapat diketahui dengan perhitungan berikut.

$$e(\gamma) = 2\left(\frac{100+140}{2 \times 150}\right)\left(1 - \frac{100+140}{2 \times 150}\right) = 0.32$$

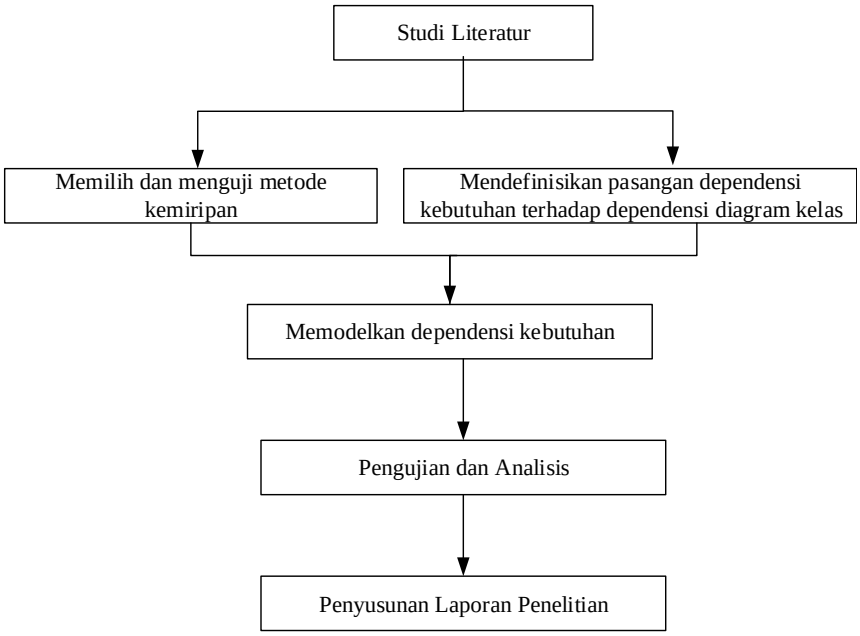
AC1 = (0.67 - 0.32) / (1 - 0.32) = 0.514

Akan tetapi, jika dihitung menggunakan nilai indeks Gwet's AC1 statistik maka hasil yang diperoleh yaitu sebesar 0.514, dimana lebih konsisten dan dapat diandalkan jika dibandingkan dengan Cohen Kappa statistik.

[Halaman ini sengaja dikosongkan]

BAB 3
METODOLOGI PENELITIAN

Penelitian ini dilakukan dengan menerapkan sekumpulan tahapan. Sekumpulan tahapan tersebut ada di dalam prosedur penelitian. Prosedur penelitian dimulai dengan studi literatur, memilih metode kemiripan yang sesuai dan mengujinya. Setelah itu, dilakukan pendefinisian pasangan dependensi kebutuhan dan dependensi diagram kelas. Tahapan berikutnya adalah memodelkan dependensi kebutuhan. Kemudian, dilakukan pengujian kesepakatan antara kerangka kerja dan tiga orang ahli di bidang rekayasa perangkat lunak. Pengujian tersebut dilakukan untuk mengukur nilai reliabilitas antara metode yang diusulkan dengan pengetahuan ahli (*anotator*). Keseluruhan prosedur tersebut dapat dilihat pada Gambar 3.1. Pada subbab berikutnya disajikan penjelasan untuk setiap langkah pada diagram alur penelitian tersebut.



Gambar 3.1 Diagram Alur Penelitian

3.1. Studi Literatur

Pada subbab ini, dijelaskan penggalian ilmu tentang topik penelitian dependensi kebutuhan, dependensi diagram kelas, dan metode kemiripan melalui pengumpulan berbagai literatur seperti jurnal, buku-buku dan sumber lainnya. Berdasarkan studi literatur tersebut, penelitian ini mencakup:

1. Pengumpulan dataset berupa daftar kebutuhan dan diagram kelas dari proyek yang sudah ada.
2. Menghitung kemiripan antara kebutuhan dan kelas pada diagram kelas

3.2. Memilih dan menguji metode kemiripan

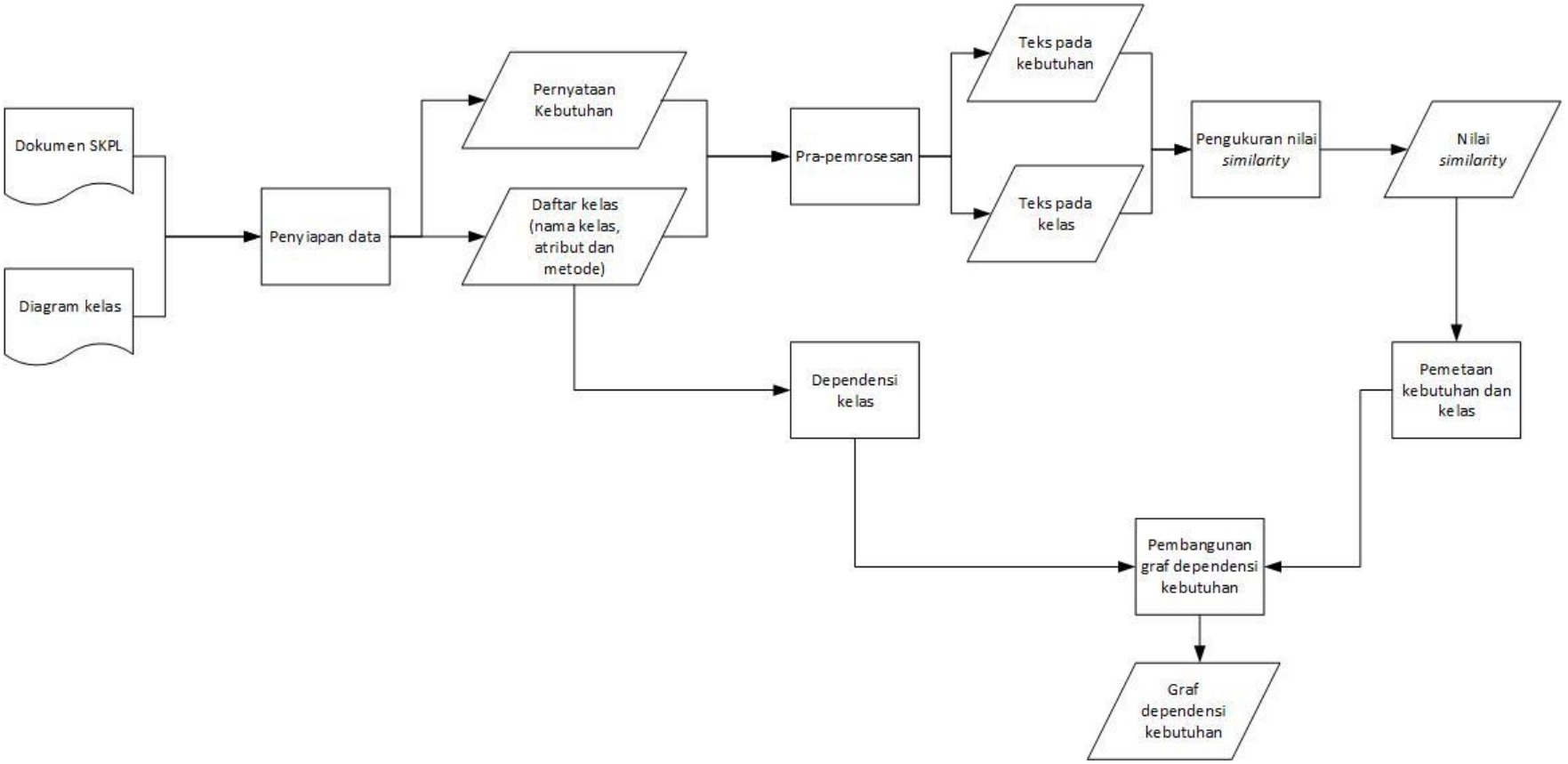
Pada tahap ini dilakukan studi literatur tentang metode-metode *kemiripan* yang ada dan melihat hasil keakuratannya kemudian melakukan pengujian terhadap *dataset* pada metode *kemiripan* yang sudah dipilih. Kriteria yang digunakan pada penelitian ini adalah cakupan data yang diolah, penerapan, performansi dan fleksibilitas penggunaan metode.

3.3. Mendefiniskan pasangan dependensi kebutuhan terhadap dependensi kelas pada diagram kelas

Pada tahap ini dilakukan studi literatur tentang dependensi kebutuhan dan juga analisis terhadap keterkaitan relasi antar kebutuhan dengan relasi antar kelas pada diagram kelas.

3.4. Memodelkan dependensi kebutuhan

Setelah melakukan studi literatur dan pengujian terhadap dataset, maka akan diperoleh hasil pemodelan dependensi kebutuhan berdasarkan dependensi diagram kelas. Secara umum, gambaran tahapan yang digunakan untuk memperoleh model dependensi kebutuhan seperti pada Gambar 3.2.



Gambar 3.2 Metode untuk memodelkan dependensi kebutuhan

Gambar 3.2 menjelaskan metode untuk memodelkan dependensi kebutuhan terdiri dari beberapa tahapan umum, yaitu:

1. Menyiapkan data kebutuhan dan diagram kelas,
2. Pemetaan kebutuhan dan kelas,
3. Menyediakan dependensi berdasarkan dependensi kelas,
4. Membuat model dependensi kebutuhan.

Pada subbab berikutnya, disajikan penjelasan setiap tahapan untuk memodelkan dependensi kebutuhan.

3.4.1 Menyiapkan data kebutuhan dan diagram kelas

Data kebutuhan diambil dari dokumen SKPL (Spesifikasi Kebutuhan Perangkat Lunak). Data kebutuhan ini terdiri dari perrnyataan kebutuhan dan diagram kelas. Sebagai contoh pada system *library*, F adalah singkatan dari Fungsionalitas seperti yang ditunjukkan pada Tabel 3.1.

Tabel 3.1 Daftar pernyataan kebutuhan

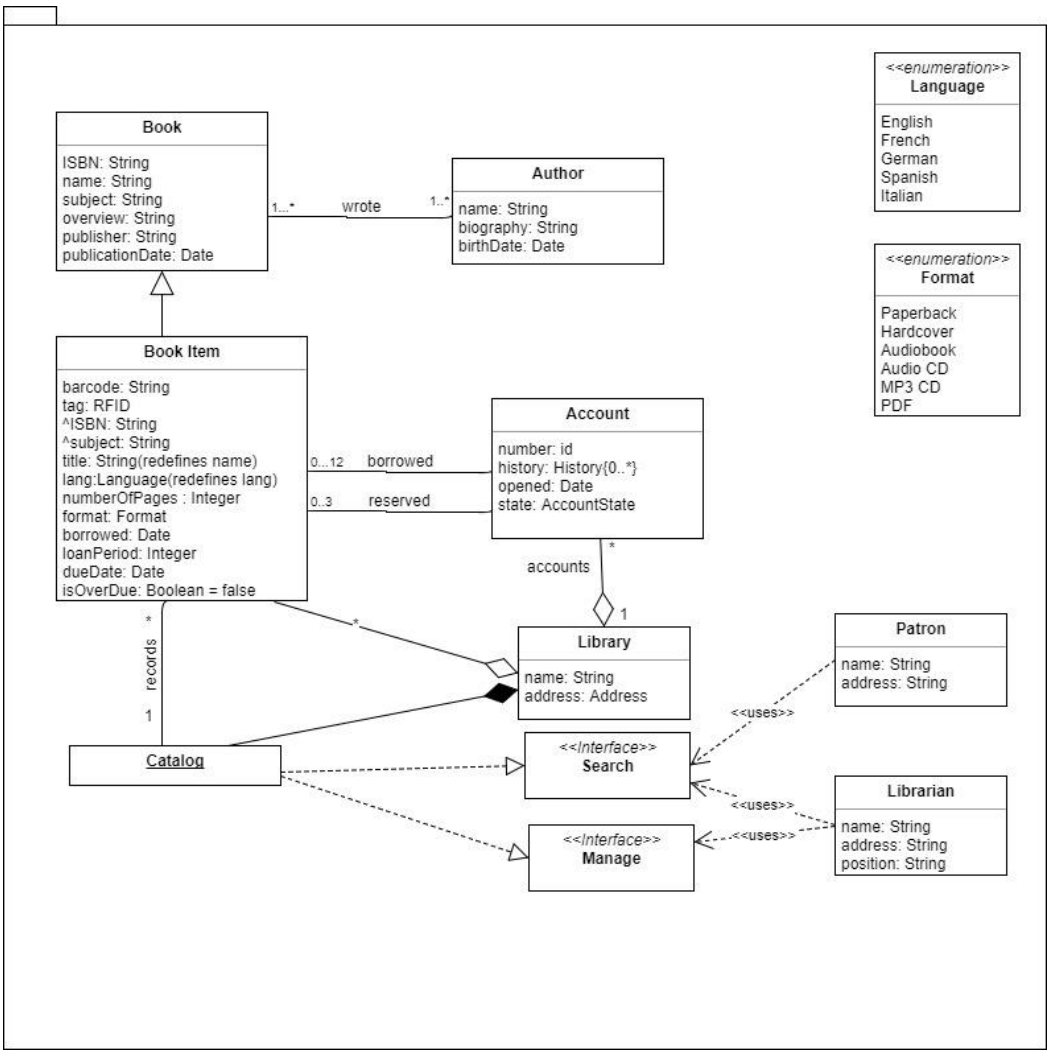
Kode ID	Pernyataan Kebutuhan
F01	*Patron or Library can manage account*
F02	*Patron or Library can search catalog*
F03	*Patron or Library can reserve book item*
F04	*Library can renew book item*
F05	*Patron can provide feedback*

Merujuk Gambar 3.3, dapat diuraikan kelas mana saja yang merupakan bagian dari daftar kebutuhan. Pada sistem *library* di atas, terdapat 10 kelas, di antaranya 8 kelas utama dan 2 kelas *interface*. Kedelapan kelas utama tersebut adalah: ***Book***, ***Author***, ***Book Item***, ***Account***, ***Library***, ***Catalog***, ***Patron***, ***Librarian***, ***Account*** dan ***Library***. Dua kelas *interface* tersebut adalah ***Search*** dan ***Manage***. Terdapat 2 tipe kelas bentukan (*enumeration*) yang memuat atribut bertipe konstan, yaitu ***Language*** dan ***Format***.

Pada diagram kelas Gambar 3.3, terdapat relasi generalisasi (antara kelas ***Book*** dan kelas ***Book Item***), asosiasi *borrowed* dan *reserved* (antara kelas ***Book***

Item dan kelas ***Account***. Setiap akun yang memesan buku dapat dilakukan selama 3 bulan, sedangkan untuk akun yang meminjam buku dapat dilakukan selama 12 bulan), asosiasi *wrote* antara kelas ***Book*** dan ***Author*** (setiap ***author*** bisa menulis 1 atau banyak buku dan banyak ***author*** bisa menulis banyak buku),

Kelas *library* terdiri dari kelas ***Catalog*** dan kelas *library* terdiri dari banyak kelas ***Book Item***. Kelas ***Catalog*** memiliki relasi asosiasi *record* dengan kelas ***Book Item*** yang berarti bahwa kelas ***Catalog*** meyimpan banyak ***Book Item***. Kelas Catalog mengimplementasikan 2 kelas *interface*, yaitu kelas ***Search*** dan kelas ***Manage***. Kelas ***Patron*** menggunakan kelas *interface* ***Search*** sedangkan ***Librarian*** dapat menggunakan 2 kelas *interface*, yaitu kelas ***Search*** dan ***Manage***. Representasi diagram kelas tersebut, dapat dilihat pada Gambar 3.3.



Gambar 3.3 Diagram kelas *library*

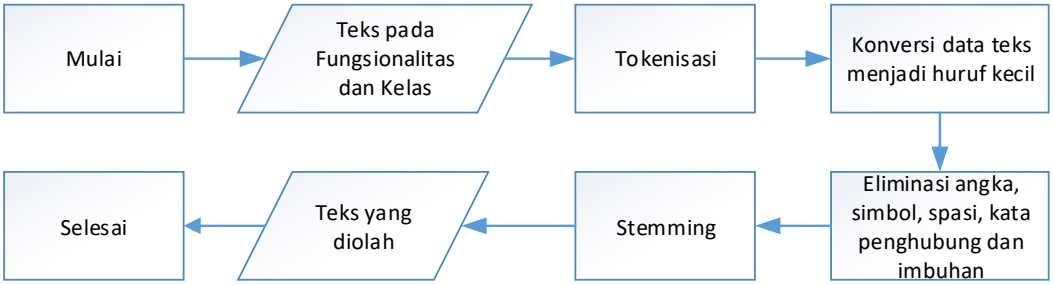
Kemudian dilakukan pemetaan dari fungsionalitas yang tersedia sebelumnya, terhadap setiap kelas pada diagram kelas. Pemetaan dari setiap kelas pada diagram kelas akan ditampilkan pada Tabel 3.2.

Tabel 3.2 Pemetaan dari setiap kelas pada diagram kelas

Kode ID	Kelas	Atribut	Metode	Keterangan
CO1	*Book*	*ISBN, name, subject, overview, publisher, publicationDate*	-	Kelas utama
C02	*BookItem*	*Barcode, tag, ISBN, subject, title, lang, numberOfPages, format, borrowed, loanPeriod, dueDate, isOverDue*	-	Kelas utama
CO3	*Author*	*Name, biography, birthdate*	-	Kelas utama
CO4	*Account*	*Number, history, opened, state*	-	Kelas utama
C05	*Library*	*Name, address*	-	Kelas utama
CO6	*Patron*	*Name, address*	-	Kelas utama
C07	*Librarian*	*Name, address, position*		Kelas utama
C08	*Catalog*	-	-	Kelas utama
C09	*Search*	-	-	Kelas *interface*
C10	*Manage*	-	-	Kelas *interface*

Proses yang dilakukan pada tahap ini adalah pra-pemrosesan. Tujuan dari pra-pemrosesan adalah mengubah data masukan berupa teks pada pernyataan kebutuhan dan teks pada informasi diagram kelas menjadi format yang sesuai untuk dianalisis selanjutnya. Langkah-langkah dalam pra-pemrosesan data mencakup membersihkan data untuk menghilangkan *noise* [22].

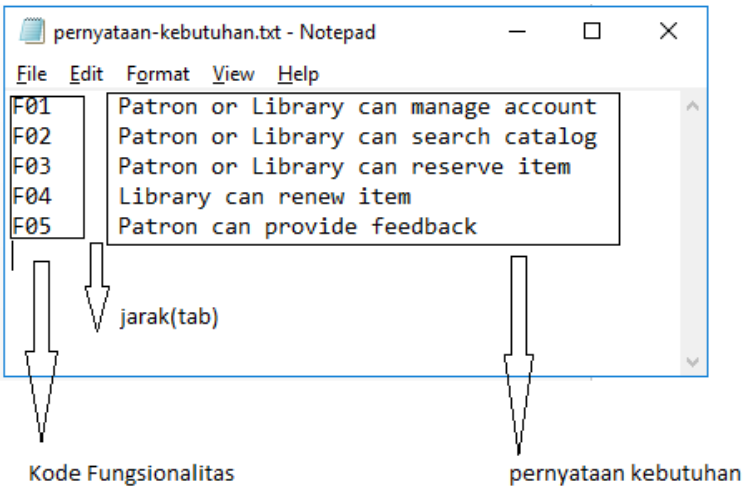
Data teks, umumnya harus diolah terlebih dahulu. Pengolahan pada teks memerlukan beberapa tahapan. Elemen-elemen yang tidak dibutuhkan dalam teks seperti simbol, tanda baca, spasi, kata penghubung (*stopwords*) dan imbuhan akan dihilangkan. Tahapan-tahapan dalam proses pengolahan data teks tersebut diperlukan untuk membersihkan data teksnya, sehingga data teks tersebut dapat diproses dan juga dianalis ke tahap selanjutnya.



Gambar 3.4 Tahapan Pra-pemrosesan teks

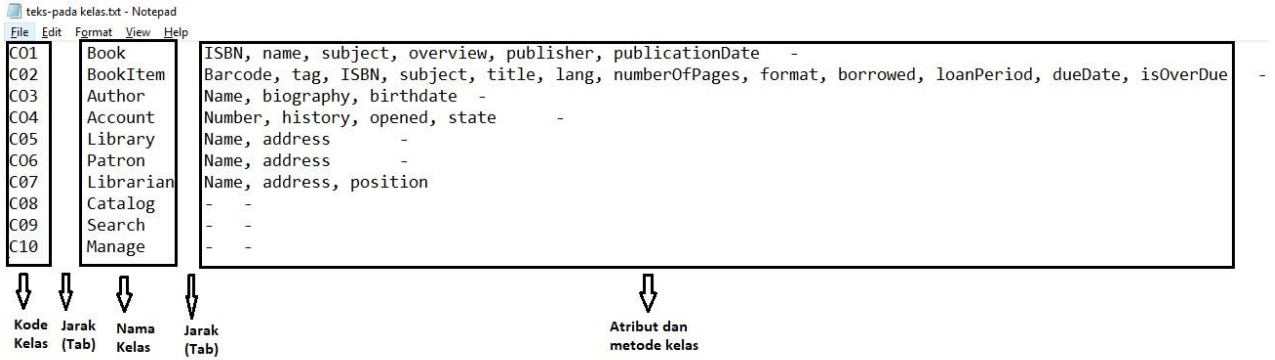
Pada umumnya pra-pemrosesan teks terdiri dari beberapa tahap seperti pada Gambar 3.4. Pertama, teks akan dipecah menjadi kumpulan kata atau disebut juga tokenisasi. Seluruh huruf pada kumpulan kata tersebut diubah menjadi huruf kecil. Selanjutnya, angka, simbol, dan kata penghubung akan dihilangkan dari kumpulan kata tersebut. Tahap terakhir adalah penghilangan imbuhan dari setiap kata (*stemming*). Sehingga hanya kata-kata penting yang tersisa di dalam data teks.

Teks yang dimaksud sebagai data input terdiri dari 2 jenis teks, yaitu teks untuk pernyataan kebutuhan dan teks informasi diagram kelas yang mencakup kode kelas, nama kelas, atribut dan metodenya. Teks pernyataan kebutuhan, disimpan dalam sebuah *file* berformat txt yang berisi kumpulan pernyataan kebutuhan, seperti contoh pada Gambar 3.5.



Gambar 3.5 Teks pernyataan kebutuhan

Teks pada diagram kelas juga disimpan dalam sebuah *file* berformat txt. Dari daftar kelas yang telah disediakan sebelumnya, dilakukan pemisahan teks berdasarkan kode kelas, nama kelas, atribut kelas dan metode kelas yang merujuk pada Gambar 3.6.



Gambar 3.6 Teks pada kelas (kode kelas, nama kelas, atribut dan metode)

Adapun contoh implementasi dari setiap proses pada pra-pemrosesan data fungsionalitas 1 (F01) "*Patron or Library can manage account*", adalah sebagai berikut:

1) Tokenisasi. Pada tahap tokenisasi, akan dipecah menjadi Patron//or//Library//can//manage//account
2) Konversi data teks menjadi huruf kecil. Pada tahap konversi data teks menjadi huruf kecil, akan berubah menjadi:

patron//or//library//can//manage//account

3) Eliminasi angka, simbol, spasi, kata penghubung dan imbuhan. Pada tahap eliminasi angka, simbol, spasi, kata penghubung dan imbuhan, akan berubah menjadi :

   patron//library//manage//account

4) *Stemming.* Pada tahap *stemming*, akan berubah menjadi:

   patron//library//manage//account

Setelah pra-pemrosesan dilakukan pada Fungsionalitas dan kelas, maka hasilnya dapat dilihat pada tabel 3.3.

Tabel 3.3. Hasil data kebutuhan dan kelas setelah pra-pemrosesan

No	Kode Kebutuhan	Data Kebutuhan	Kode kelas	Data Kelas
1.	R01	*patron library manage account*	C01	*Book ISBN name subject, overview publisher publication date*
2.	R02	*patron library search catalog*	C02	*book item barcode tag isbn subject title lang numberofpages format borrowed loanperiod duedate isoverdue*
3.	R03	*patron library reserve book item*	C03	*author name biography birthdate*
4.	R04	*library renew item*	C04	*account number history opened state*
5.	R05	*patron provide feedback*	C05	*library name address patron name address*
6.	-	-	C06	*librarian name address position*
7.	-	-	C07	*catalog*
8.	-	-	C08	*search*
9.	-	-	C09	*manage*

3.4.2 Pemetaan kebutuhan dan kelas

Pertama kali, dibuat matriks dengan jumlah m kolom * n baris, n baris menandakan jumlah kelas, sedangkan m kolom menandakan jumlah kebutuhan. Setiap informasi kelas yang ada pada diagram kelas (termasuk kode kelas, nama kelas, atribut dan metode), akan dipetakan terhadap setiap fungsionalitas yang ada pada sistem *library*. Kemudian nilai similaritas dari setiap teks pada informasi diagram kelas, akan dipetakan terhadap teks pada daftar kebutuhan. Contohnya:

Pemetaan antara Kelas 01 (CO1) dan Kebutuhan 01 (F01). Maka teks yang ada pada kelas dan kebutuhan dimasukkan dalam bentuk matriks seperti pada Tabel 3.4.

Tabel 3.4 Nilai similaritas antara C01 dan F01

Kode ID	**Kelas/Atribut**	**FO1 (Patron Library manage account)**			
		patron	**library**	**manage**	**account**
C01	book	0.38	0.52	0.00	0.12
	ISBN	0.00	0.00	0.00	0.14
	Name	0.14	0.13	0.50	0.31
	Subject	0.15	0.14	0.00	0.50
	Overview	0.13	0.13	0.00	0.43
	Publisher	0.12	0.11	0.00	0.25
	Publication	0.40	0.56	0.00	0.13
	Date	0.14	0.13	0.33	0.31

Nilai similaritas dari setiap kolom (teks pada Kebutuhan) dan baris (teks pada Kelas), diperoleh dengan menggunakan rumus metode penghitungan kemiripan *Wu* dan *Palmer* (http://ws4jdemo.appspot.com/), kemudian hasil akhir nilai similaritas dari C01 dan F01 diperoleh dengan menggunakan *Greedy Algorithm* (merujuk pada sub bab 2.5 metode penghitungan kemiripan). Hasil perhitungan nilai similaritas tersebut terdapat pada Persamaan 3.1.

$$\text{Sim } S_{mxn}, = \frac{2\ X\ (\sum_{i=1}^{\min[m][n]} maks\ tokenSim|m_i||n_i|)}{m+n} \quad (3.1)$$

$$= \frac{2\ X\ (0.56+0.50+0.50+0.38)}{(8+4)}$$

$$= \frac{1.94}{12} = 0.32$$

Dengan menggunakan rumus similaritas tersebut, maka diperoleh hasil matriks dari C01 dan F01 adalah 0.32. Nilai pemetaaan C01 dan F01 dimasukkan ke dalam matriks yang tersedia pada Tabel 3.5. Nilai similaritas untuk kelas dan kebutuhan berikutnya, dihitung dengan cara yang sama pada C01 dan F01, sehingga diperoleh hasil seperti pada tabel yang sama. Pada contoh kasus ini,

digunakan *threshold* >= 0.40 sebagai nilai ambang batas dalam menentukan dependensi antara kelas dan fungsionalitas. Tabel 3.5 menunjukkan dependensi yang terbentuk antara kelas dan kebutuhan.

Tabel 3.5 Pemetaan nilai similaritas teks pada kelas dan Fungsionalitas

Kode ID	**F01**	**F02**	**F03**	**F04**	**F05**
C01	0.32	0.33	**0.43**	**0.44**	0.18
C02	0.21	0.22	0.30	0.28	0.10
C03	**0.56**	0.27	0.37	0.35	0.29
C04	**0.42**	0.25	0.21	0.30	0.21
C05	**0.54**	0.36	**0.46**	**0.53**	0.30
C06	**0.44**	0.37	**0.46**	0.36	0.39
C07	**0.47**	0.28	**0.40**	**0.42**	0.31
C08	0.20	**0.40**	0.32	0.38	0.18
C09	0.11	**0.40**	0.11	0.13	0.08
C10	**0.40**	0.16	0.07	0.09	0.14

Hasil pemetaan yang terbentuk dengan *threshold* 0.40 adalah seperti pada Tabel 3.6. Tabel ini menjelaskan suatu Fungsionalitas, yang direalisasikan oleh kelas tertentu. Tanda centang (∨), mengandung arti direalisasikan. F01 direalisasikan oleh C03, C04, C05, C06, C07 dan C10. F02 direalisasikan oleh C08 dan C09. F03 direalisasikan oleh C01, C05, C06, C07. F04 direalisasikan oleh C01, C05 dan C07.

Tabel 3.6 Pemetaan nilai similaritas teks pada kelas dan Fungsionalitas

Kode ID	**F01**	**F02**	**F03**	**F04**	**F05**
C01			✓	✓	
C02					
C03	✓				
C04	✓				
C05	✓		✓	✓	
C06	✓		✓		
C07	✓		✓	✓	
C08		✓			
C09		✓			
C10	✓				

3.4.3 Membuat dependensi diagram kelas

Pada tahapan ini, dilakukan pemetaan antara kelas sumber (*source*) ke kelas tujuan (*destination*). Relasi antar kelas sumber dan kelas tujuan ini diambil dari diagram kelas. Hasil pemetaan dari setiap kelas yang berelasi dengan kelas lain yang ditampilkan pada Tabel 3.7 akan dipetakan lagi terhadap fungsionalitas yang tersedia pada sistem.

Tabel 3.7 Relasi antar kelas pada diagram kelas

		Kelas Tujuan									
Kelas sumber	tujuan / sumber	C01/ Book	C02/ BookItem	C03/ Author	C04/ Account	C05/ Library	C06/ Patron	C07/ Librarian	C08/ Catalog	C09/ Search	C10 /Manage
	C01/Book										
	C02/BookItem	**s**									
	C03/Author	**c**									
	C04/Account		**c**								
	C05/Library		**c**		**c**				**h**		
	C06/Patron									**d**	
	C07/Librarian									**d**	**d**
	C08/Catalog		**c**							**i**	**i**
	C09/Search										
	C10/Manage										

Keterangan:

s: specializes
h: has (strong aggregation)
c: contain (weak aggregation)
u:uses
i: implements

3.4.4 Membuat model dependensi kebutuhan

Setelah diperoleh hasil relasi antar kelas pada diagram kelas, maka kelas tujuan akan dipetakan terhadap daftar Fungsionalitas berdasarkan dependensi kelas. Tabel 3.8 merepresentasikan pemetaan dependensi antara suatu Fungsionalitas dengan Fungsionalitas yang lain, di antaranya F01 berelasi *strong aggregation* dengan F02. F01 berelasi *weak aggregation* dengan F03 dan F04. F03 dan F04 mempunyai relasi yang sama ke F01, yaitu *weak aggregation* dan *uses*. F03 dan F04 mempunyai relasi yang sama ke F02, yaitu *strong aggregation* dan *uses*.

Tabel 3.8.Model dependensi antar kebutuhan

Fungsionalitas Tujuan (destination)						
Fungsionalitas Sumber (source)	Fungsionalitas	F01	F02	F03	F04	F05
	F01		**h**	**c**	**c**	
	F02					
	F03	**c,u**	**h,u**			
	F04	**c,u**	**h,u**			
	F05					

Tabel 3.8 menunjukkan bahwa relasi antar Fungsionalitas berdasarkan dependensi kelas. Contohnya: Dari tabel yang tersedia, diketahui bahwa relasi F01 terhadap F02 adalah "h" (*strong aggregation*). Relasi "*strong aggregation*" diperoleh dari tahapan berikut ini:

1. Dari tabel 3.6 diketahui bahwa F1 direalisasikan oleh C03, C04, C05, C06, C07 dan C10 atau F1 = {C03, C04, C05, C06, C07, C10},
2. Salah satu Fungsionalitas yang digunakan adalah F01 direalisasikan oleh C05. Kemudian pada tabel 3.7 diketahui bahwa C05 memiliki relasi "c/*weak aggregation*" terhadap C02, C04,C08,
3. Pada tabel 3.6 diketahui bahwa C02 tidak direalisasikan oleh Fungsionalitas manapun, C04 direalisasikan oleh Fungsionalitas 1 (F01), C08 direalisasikan oleh Fungsionalitas 2 (F02) sehingga diperoleh hasil bahwa F01 berelasi "h (*strong aggregation*)" terhadap F02.

Deskripsi dari penjelasan tabel 3.8 lebih rinci disajikan pada Tabel 3.9. Tabel ini merepresentasikan dependensi antar kebutuhan yang diperoleh

berdasarkan dependensi antar kelas pada diagram kelas. Relasi *weak aggregation* tidak dimasukkan dalam Tabel 3.9 karena tidak ada pendefinisian *weak aggregation* dengan dependensi antar kebutuhan.

Tabel 3.9 Relasi fungsionalitas berdasarkan relasi antar kelas

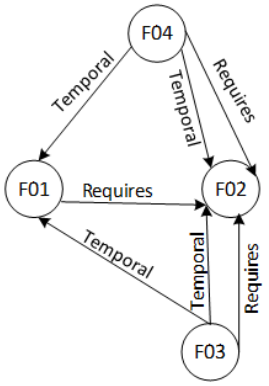
No.	Fungsionalitas sumber	Relasi	Fungsionalitas tujuan
1.	F01	*strong aggregation*	F02
2.	F03	*Uses*	F01, F02
3.	F04	*Uses*	F01, F02
4.	F03	*strong aggregation*	F02
5.	F04	*strong aggregation*	F02

Setelah diperoleh hasil dependensi kebutuhan berdasarkan dependensi antar kelas, maka dilakukan pendefinisian pasangan dependensi kebutuhan terhadap dependensi kelas pada diagram kelas. Hasil pemetaan dependensi kebutuhan berdasarkan dependensi diagram kelas dapat dilihat pada Tabel 3.10.

Tabel 3.10 Tabel dependensi fungsionalitas.

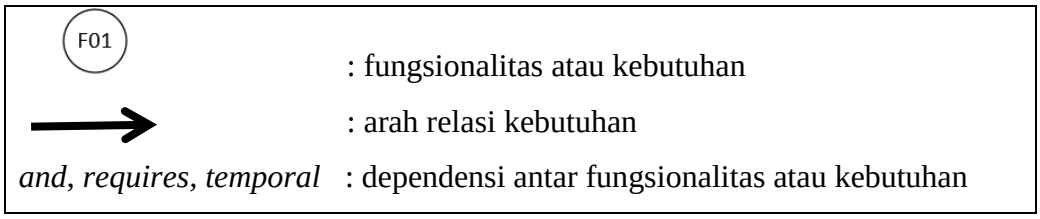
No.	Fungsionalitas sumber	Relasi	Fungsionalitas tujuan
1.	F01	*requires, temporal*	F02
2.	F03	*Temporal*	F01, F02
3.	F04	*Temporal*	F01, F02
4.	F03	*requires, temporal*	F02
5.	F04	*requires, temporal*	F02

Dari hasil dependensi kebutuhan tersebut, dapat digambarkan suatu model dependensi kebutuhan berupa graf, seperti pada Gambar 3.7. Graf dependensi kebutuhan, terdiri dari fungsionalitas sumber dan fungsionalitas tujuan. Dari graph tersebut, dapat diketahui model dependensi antar kebutuhan yang terbentuk pada studi kasus sistem *library*.



Gambar 3.7 Graf model dependensi kebutuhan

Model dependensi yang diperoleh dari gambar sebelumnya adalah dalam bentuk graf. Graf terdiri dari node asal, tujuan dan arah. Penjelasan simbol yang digunakan pada graf model dependensi kebutuhan, dapat dilihat pada Gambar 3.8.



Gambar 3.8. Model dependensi kebutuhan

3.5. Pengujian dan Analisis

Tujuan pengujian adalah untuk menjawab apakah metode yang diusulkan dapat diandalkan seperti seorang ahli dalam membangun graf ketergantungan antar kebutuhan berdasarkan kelas diagram. Uji coba ini, dilakukan terhadap beberapa pernyataan kebutuhan perangkat lunak dan informasi kelas pada diagram kelas. Dalam penelitian ini, kuisioner akan disebarkan kepada tiga orang ahli. Para ahli ini akan berperan sebagai annotator. Mereka akan memberi anotasi kepada setiap pasangan kebutuhan dan kelas yang dianggap merealisasikan kebutuhan terkait. Selain itu, annotator juga memberi anotasi kepada pasangan kebutuhan yang saling terkait, beserta dengan tipe dependensinya. Ketiga ahli ini minimal memiliki pengalaman bekerja di bidang rekayasa kebutuhan perangkat lunak atau mengampu mata kuliah terkait rekayasa perangkat lunak. Kehandalan metode yang diusulkan diukur dengan menghitung tingkat kesepakatan antara

metode dan para ahli. Perhitungan tingkat kesepakatan ini didasarkan pada metode kappa statistic, yaitu *Gwet's AC1*. Metode akan diperlakukan sebagai salah satu ahli yang jawabannya akan dibandingkan kesamaannya dengan jawaban dari para ahli.

Sebelum dianalisis, maka hasil anotasi para annotator akan diuji tingkat integritasnya. Jika tingkat integritasnya sudah baik, maka proses pembandingan dengan metode yang diusulkan dapat dilakukan. Adapun tiga jenis skenario uji coba yang dilakukan pada penelitian ini. Pertama, kesepakatan dua annotator. Pada skenario ini, jawaban anotator akan diagregasi. Anotasi yang dinyatakan benar adalah anotasi yang dipilih oleh mayoritas annotator. Hasil anotasi ini kemudian dibandingkan dengan hasil analisis dari metode. Kedua, kesepakatan tiga annotator. Pada skenario ini, hanya pasangan kebutuhan-kelas yang ketiga annotator menjawab sama saja yang diuji. Hasil dari anotasi ini kemudian dibandingkan dengan hasil analisis dari metode. Ketiga, kesepakatan antar ahli. Pada skenario ini, akan dihitung tingkat kesepatan antar anotator dan tingkat kesepakatan antara metode dengan masing-masing annotator. Skenario ini hendak melihat apakah metode yang diusulkan kurang obyektif, seobyektif, atau lebih obyektif dari sesama annotator. Jika tingkat kesepatan antara metode dan annotator lebih tinggi dibandingkan tingkat kesepakatan antar annotator, berarti metode yang diusulkan lebih obyek dari para ahli. Jika sebaliknya, berarti metode yang diusulkan tidak seobyektif dari para ahli.

3.6. Laporan Penelitian

Dokumentasi dan laporan, merupakan tahap akhir dari penelitian ini. Dimana seluruh proses penelitian dan pengujian didokumentasikan dalam bentuk laporan.

3.7. Jadwal Penelitian

Tabel 3.11 Rencana Jadwal Kegiatan Penelitian

No.	Kegiatan	Bulan																			
		Agustus 2017				September 2017				Oktober 2017				November 201			Desember 2017-Mei 2018				
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Studi Literatur																				
2	Memilih dan menguji metode kemiripan																				
3	Mendefinisikan pasangan dependensi kebutuhan dan dependensi kelas pada diagram kelas																				
4	Memodelkan dependensi kebutuhan																				
5	Pengujian dan analisis																				
6	Penyusunan laporan penelitian																				

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] Å. G. Dahlstedt, "Requirements Interdependencies – a Research Framework," no. July, 2001.

[2] A. G. Dahlstedt and A. Persson, "Requirements Interdependencies : State of the Art and Future Challenges," *Eng. Manag. Softw. Requir.*, pp. 95–116, 2005.

[3] M. Widiastuti and D. Siahaan, "Labelled Transition System For Requirement Change (Lts-Rc): Pemodelan Perubahan Kebutuhan Perangkat Lunak Berdasarkan Labelled Transition System," in *Prosiding Seminar Nasional Manajemen Teknologi VI*, 2008, p. C11.2-C11.9.

[4] K. Müller and B. Rumpe, "A Model-Based Approach to Impact Analysis Using Model Differencing," *Proc. 8th Int. Work. Softw. Qual. Maintainab.*, vol. 65, no. February, pp. 1–15, 2014.

[5] J. Wang and Q. Wang, "Analyzing and predicting software integration bugs using network analysis on requirements dependency network," *Requir. Eng.*, 2016.

[6] M. Widiastuti and I. T. Faculty, "Mapping The Impact Of Requirement Changes Using (LTS-RC)," pp. 315–319, 2008.

[7] W. Chen, M. Zhang, and H. Li, "Utilizing Dependency Language Models for Graph-based Dependency Parsing Models," *Proc. 50th Annu. Meet. Assoc. Comput. Linguist. (Volume 1 Long Pap.*, no. July, pp. 213–222, 2012.

[8] M. P. Robillard and G. C. Murphy, "Concern graphs," *Proc. 24th Int. Conf. Softw. Eng. - ICSE '02*, p. 406, 2002.

[9] M. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," *20090110 Httpnlp Stanford*, vol. 40, no. September, pp. 1–22, 2010.

[10] M. Zhang, W. Chen, X. Duan, and R. Zhang, "Improving graph-based dependency parsing models with dependency language models," *IEEE Trans. Audio, Speech Lang. Process.*, vol. 21, no. 11, pp. 2313–2323, 2013.

[11] W. Chen, J. Kazama, K. Uchimoto, and K. Torisawa, “Exploiting subtrees in auto-parsed data to improve dependency parsing,” *Comput. Intell.*, vol. 28, no. 3, pp. 426–451, 2012.

[12] “UML dependency is directed, supplier-client relationship which shows that some element requires other model elements.” .

[13] M. Widiastuti and D. Siahaan, “(Lts-Rc): Pemodelan Perubahan Kebutuhan Perangkat Lunak Berdasarkan Labelled Transition System,” *Pros. Semin. Nas. Manaj. Teknol. VII*, p. C-11, 2008.

[14] I. Bagus and K. Surya, “Implementasi Algoritma Levenshtein Pada Sistem Pencarian Judul Skripsi / Tugas Akhir,” pp. 46–53.

[15] I. Measure, F. O. R. Taxonomy, B. On, and E. Counting, “A N Ew S Imilarity Measure For Taxonomy Based On,” vol. 3, no. 4, pp. 23–30, 2012.

[16] S. Christina, “Kinerja Cosine Similarity Dan Semantic Similarity Dalam Pengidentifikasian Relevansi Nomor Halaman Pada Daftar Indeks Istilah,” vol. 2014, no. Sentika, 2014.

[17] “Winter Semester , 2011 Greedy algorithm,” pp. 3–4, 2011.

[18] M. Al-Khiaty and M. Ahmed, “UML Class Diagrams: Similarity Aspects and Matching,” *Lect. Notes Softw. Eng.*, vol. 4, no. 1, 2016.

[19] T. Ruotsalo and E. Mäkelä, “A comparison of corpusbased and structural methods on approximation of semantic relatedness in ontologies,” *Int. J. Semant. Web Inf. Syst.*, vol. 5, no. 4, pp. 39–56, 2009.

[20] K. Gwet, “Kappa Statistic is not satisfactory for assessing the extent of agreement between raters,” *Stat. Methods Inter-Rater Reliab. Assessmen*, no. 1, pp. 1–5, 2002.

[21] J. R. Landis and G. G. Koch, “The Measurement of Observer Agreement for Categorical Data Data for Categorical of Observer Agreement The Measurement,” vol. 33, no. 1, pp. 159–174, 1997.

[22] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. 2005.