KAKAS BANTU PERHITUNGAN NILAI KOPLING MENGGUNAKAN *CONCEPTUAL COUPLING METRICS*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Laras Husna Aulia
NIM: 125150206111004

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

KAKAS BANTU PERHITUNGAN NILAI KOPLING MENGGUNAKAN *CONCEPTUAL COUPLING METRICS*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Laras Husna Aulia
NIM: 125150206111004

Skripsi ini telah diuji dan dinyatakan lulus pada
23 Agustus 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Fajar Pradana, S.ST., M.Eng	Bayu Priyambadha, S.Kom., M.Kom
NIP. 19871121 201504 1 004	NIP. 19820909 200812 1 004

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 23 Agustus 2016

Laras Husna Aulia

NIM: 125150206111004

KATA PENGANTAR

Dengan menyebut nama Allah Yang Maha Pengasih dan Penyayang. Segala puji bagi Allah Subhanahu wa ta'ala berkat limpahan nikmat dan rahmat karunia-Nya, penulis dapat menyelesaikan skripsi yang berjudul "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*".

Penyusunan skripsi ini tidak lepas dari bantuan semua pihak yang terus memberikan bimbingan, kritik, saran, motivasi dan dukungan serta doa yang melimpah. Oleh karena itu, penulis mengucapkan terima kasih sedalam-dalamnya kepada:

1. Bapak dan Mama tercinta, Bapak Dr.Ir.Tarmizi dan Mama Ir.Zuhroni selaku orang tua penulis, Kak Suluh Elman Swara, S.T.,M.T yang dengan sabar dan setia mengajarkan penulis, Kak Amalia Sukma Ridhani, S.Si, Adik Elmia Kharisma Arsyi, kakak sepupu Kak Masroni beserta segenap keluarga besar yang telah memberikan doa, mengajarkan banyak hal, menghibur di kala susah dan sedih serta dukungan penuh dalam penyusunan skripsi ini.
2. Bapak Fajar Pradana, S.ST.,M.Eng selaku pembimbing 1 dan Bapak Bayu Priyambadha, S.Kom.,M.Kom selaku pembimbing 2, atas segala kesabaran, kebaikan dalam membimbing, semangat serta motivasi dan waktu yang telah diluangkan serta kritik dan saran yang sangat berguna diberikan kepada penulis.
3. Bapak Adam Hendra Brata, S.Kom., M.T., M.Sc selaku penguji 1 dan Bapak Denny Sagita R., S.Kom., M.Kom selaku penguji 2 yang telah memberikan saran bagi penelitian ini.
4. Seluruh bapak dan ibu dosen Fakultas Ilmu Komputer Universitas Brawijaya atas segala bimbingan dan ilmu yang telah diajarkan kepada penulis.
5. Teman-teman seperjuangan skripsi Zaenab Asyahidatu Zahra, S.Kom, Mega Tricitanya, S.Kom, Fina Af Idatul Husna, S.Kom, Arsyad Mubarrok,S.Kom, Muhammad Ubaidillah, S.Kom, B.Findiarin Billyan, Sifasani Qalbina Fauzia, Wian Virgi, Nancy, Indah terima kasih atas dukungan, tidak lelah memberikan semangat dan motivasi serta berbagi suka duka dalam mengerjakan skripsi kepada penulis.
6. Mas Fredy Nendra Pranata, S.Kom, Mbak Elliya Lestari, S.Kom dan Mas Kenneth Setiawan, S.Kom atas skripsinya yang dapat menjadi bahan referensi.
7. Seluruh mahasiswa Program Studi Teknik Informatika khususnya angkatan 2012, kelas TIF J dan kelas TIF A.
8. Teman-teman kos yang selalu setia menemani dan memberi semangat tiada henti Angelica Natasya Putri, S.Pi, Markzy Brondy Laskmy, S.Kh, Nisa Tazkiyah, Mbak Restu, Mbak Sari dan Yulia Jasmine, S.H.,M.Kn serta ucuk kucing kesayangan keluarga besar GHF Family.
9. Keluarga besar *Gamping House Family* (GHF), Bapak dan Ibu Kusman selaku orangtua kedua penulis di Malang, Mbak Ega, Mbak Icha, Mbak Putri, Mbak Ida, Cak Ri dan Mas Ciung.

10. Ullum Pratiwi, Ayu Permatasari, S.Kom, Vitara dan seluruh keluarga besar TIF J 2012.

11. Seluruh civitas akademika Universitas Brawijaya yang secara tidak langsung memberikan kelancaran bagi penulis.

12. Seluruh keluarga besar tempat penulis menempuh studi yaitu keluarga besar TK PGRI Mataram, SDN 41 Mataram, SMPN 2 Mataram dan SMKN 2 Mataram.

13. Seluruh keluarga besar Taman Baru Mataram.

14. Semua pihak yang tidak dapat disebutkan satu per satu yang terlibat baik secara langsung maupun tidak langsung demi terselesaikannya skripsi ini.

Malang, 23 Agustus 2016

Penulis
husnaaulialaras@yahoo.co.id

ABSTRAK

Coupling merupakan salah satu parameter kualitas perangkat lunak. Kopling adalah suatu parameter untuk mengukur seberapa besar hubungan antar komponen dalam sebuah sistem. Untuk menentukan nilai kopling dapat menggunakan empat parameter conceptual coupling metrics antara lain *Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Method and a Class (CSMC)* , *Conceptual Similarity between two Classes (CSBC)* dan *Conceptual Coupling of Class (CoCC)*. Sebelum menghitung nilai kopling, parsing method harus dilakukan dengan menggunakan *spoon library*. Mencari kecocokan dokumen juga diperlukan dengan menggunakan suatu metode *information retrieval* yakni *Latent Semantic Indexing (LSI)*. Perhitungan akurasi sistem dengan cara melakukan suatu percobaan dengan cara memberikan tiga nilai toleransi yang berbeda yaitu 0.05, 0.1 dan 0.20. Pada saat memberikan ketiga nilai toleransi tersebut, hasilnya jika nilai toleransi sebesar 0.05 maka nilai akurasi sistem sebesar 24,83%. Jika nilai toleransi sebesar 0.1 maka nilai akurasi sistem sebesar 43,41%. Sedangkan, jika nilai toleransi sebesar 0.20 maka nilai akurasi yang diperoleh diatas 50% yaitu 69,78%.

Kata Kunci: kopling, conceptual coupling metrics, Latent Semantic Indexing (LSI), parsing method

ABSTRACT

Coupling is the one of software quality measurement. Coupling is a paramater that measuring how big the relationship between components of a system. To measuring value of coupling we can use conceptual coupling metrics. Conceptual coupling metrics consist of four parameters, there are Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Method and a Class (CSMC) , Conceptual Similarity between two Classes (CSBC) and Conceptual Coupling of Class (CoCC). We must do parsing method using spoon library first before calculate coupling value.To match the documents using information retrieval need Latent Semantic Indexing (LSI). Calculation of accuracy is doing by three way provides different tolerance values consist of 0.05, 0.1 and 0.2. If tolerance value is 0.05 then the accuracy value is 24,83%. If tolerance value is 0.1 then the accuracy value is 43,41%. Meanwhile, If tolerance value is 0.2 then the accuracy value is up to 50%, it is 69,78%.

Keywords: kopling, conceptual coupling metrics, Latent Semantic Indexing (LSI), parsing method

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Konsep pembuatan baik analisis, perancangan hingga pengujian perangkat lunak semakin berkembang setiap waktu. Tujuan penggunaan konsep tersebut agar perangkat lunak bernilai ekonomi, terpercaya dan efisien. Perangkat lunak yang tidak memiliki konsep pengembangan yang baik pada akhirnya sering tidak digunakan karena tidak seperti yang diinginkan pelanggan, tidak mudah digunakan kembali (*non-reusable*) atau bahkan karena masalah non-teknis seperti ketidakmampuan pengguna memahami sistem sehingga pengguna enggan untuk mengubah cara kerja dari manual ke otomatis. Oleh sebab itu, konsep perancangan perangkat lunak harus ditelaah lebih dalam (Rosa, 2013).

Konsep pengembangan perangkat lunak menggunakan dua pendekatan yaitu pendekatan terstruktur dan pendekatan berorientasi objek. Pendekatan berdasarkan objek menganggap sistem berupa objek-objek virtual yang saling berkaitan dan objek tersebut merefleksikan objek-objek yang ada di dunia nyata. Perangkat lunak yang menerapkan konsep berorientasi objek akan melakukan perancangan yang berpusat pada objek-objek yang berelasi tersebut. Objek tersebut terdiri dari beragam struktur data dan perilaku seperti yang dimiliki suatu entitas (Melian, 2011).

Konsep *Object Oriented Programming* (OOP) dapat dimanfaatkan untuk mengukur keterkaitan hubungan antar *class*. Pengukuran ini sangat diperlukan untuk mendapatkan perancangan desain perangkat lunak yang baik dan implementasi perangkat lunak yang baik sehingga menghasilkan produk perangkat lunak berkualitas tinggi. Tidak ada perangkat lunak yang salah. Hanya ada perangkat lunak yang baik atau tidak tergantung pada perancangan sistem apakah sudah dilakukan dengan baik atau belum.

Salah satu parameter kualitas perangkat lunak adalah kopling. Kopling adalah suatu parameter untuk mengukur seberapa besar hubungan antar komponen dalam sebuah sistem (Poshyvanyk, 2006). Kopling juga dikaitkan dengan parameter untuk pemeliharaan perangkat lunak. Pengukuran kopling dapat dilakukan dengan berbagai cara, salah satunya adalah dengan menggunakan satu set langkah baru perhitungan nilai kopling yang disebut dengan *conceptual coupling*. Dimana pada konsep ini pengukuran dilakukan dengan cara mengambil informasi semantik yang dibagi antar elemen pada kode program (Poshyvanyk, 2006). Semantik berarti memperhatikan kata berdasarkan kata yang sama maupun makna kata yang sama. Dari hasil analisis kode program tersebut lalu didapatkan nilai kopling. Selanjutnya, nilai kopling ini berguna untuk mengetahui seberapa besar tingkat kopling antar komponen. Semakin rendah tingkat kopling antar komponen maka semakin baik hasil rancangannya . Jika ternyata diperoleh hasil bahwa nilai kopling tinggi maka bukan hanya implementasi kode programnya saja yang harus diperbaiki namun perancangan class diagram pada tahap perancangan juga harus diperbaiki.

Salah satu hal yang melatarbelakangi penelitian ini ialah tidak adanya pengukuran kopling yang jelas dalam mengembangkan sebuah perangkat lunak. Pengukuran tinggi rendahnya parameter kopling sering membuat ambigu ketika para pengembang ingin mengukur ketergantungan antar komponen pada saat perancangan telah selesai dilakukan. Selama ini, besaran nilai kopling masih diukur dengan cara perkiraan saja sehingga tidak ada besaran nilai kopling yang jelas. Penelitian ini akan merancang dan membuat sebuah kakas bantu yang akan membantu pengembang perangat lunak dalam mengukur nilai kopling secara otomatis berdasarkan implementasi program dengan menggunakan metrik *conceptual coupling*.

1.2 Rumusan masalah

Berdasarkan uraian latar belakang yang telah dikemukakan diatas, maka dapat dirangkum permasalahan pada penelitian ini yaitu sebagai berikut:

1. Bagaimana cara mengidentifikasi *class* yang saling berkaitan untuk mengukur besaran nilai kopling berdasarkan hubungan antar *method* di dalam *class*-*class* tersebut ?
2. Bagaimana cara menentukan *conceptual coupling metrics* berdasarkan pada method dalam sebuah kumpulan dokumen *source code* program ?
3. Bagaimana tingkat akurasi dari hasil perhitungan sistem ?

1.3 Tujuan

Tujuan dari penelitian ini antara lain sebagai berikut:

1. Dapat menghitung dan mengukur secara otomatis tingkat kopling pada tahap implementasi perangkat lunak yaitu source code / kode program.
2. Mengetahui kualitas perancangan *class* yang sudah diimplementasikan menggunakan *conceptual coupling metrics*.
3. Mengetahui tingkat akurasi penerapan metrik *conceptual coupling* melalui analisis kode program untuk memperbaiki kualitas perancangan *class*.

1.4 Manfaat

Penelitian ini diharapkan dapat memberikan manfaat umum antara lain:

1. Membantu *developer* dalam membuat perancangan dengan baik dan menghasilkan perangkat lunak berkualitas tinggi.
2. Membantu penelitian-penelitian selanjutnya yang dapat meningkatkan tingkat akurasi dalam pengembangan di bidang ilmu Rekayasa Perangkat Lunak (RPL).
3. Memberikan referensi untuk perkembangan di bidang ilmu Rekayasa Perangkat Lunak (RPL).

1.5 Batasan masalah

Berdasarkan latar belakang seperti yang sudah dibahas sebelumnya, agar pembahasan tidak keluar dari pokok permasalahan maka diperlukan pembatasan masalah sebagai berikut:

1. Aplikasi yang dibuat hanya dapat melakukan *parsing* terhadap dokumen yang berisi *souce code* / kode program.
2. Penelitian hanya pada perancangan di model SDLC (*System Development Life Cycle*) *Waterfall*.
3. Penilaian kualitas rancangan *class* sebelumnya dihasilkan dari analisis implementasi kode program. Dari analisis implementasi ini didapatkan hubungan antar *method-method* dalam *class-class* tersebut dimana dalam satu kali eksekusi program maksimal terdapat sebanyak empat dokumen *source code* program yang dianalisis dalam perhitungan.
4. Relasi antar *class* yang digunakan hanya sebatas adanya kata yang sama dan atau bermakna sama yang sama-sama digunakan oleh *class-class* tersebut.
5. Dokumen *source code* yang dapat dianalisis adalah dokumen *source code* dengan bahasa *java*.
6. Implementasi program menggunakan bahasa *Java*.
7. Library yang digunakan dalam melakukan parsing dokumen source code adalah *spoon library.*
8. Metode pengindeksan kemiripan antar dokumen kode program menggunakan metode *Latent Semantic Indexing (LSI).*
9. Library yang digunakan untuk perhitungan pembobotan serta perankingan dokumen *source code* adalah *Jama Matrix Package.*
10. Parameter kopling pada penelitian ini menggunakan *coupling metrics Conceptual Coupling of a Class (CoCC).*

1.6 Sistematika pembahasan

Gambaran secara umum pembahasan dari keseluruhan isi penulisan untuk setiap bab adalah sebagai berikut:

BAB I PENDAHULUAN

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, serta sistematika penulisan.

BAB II LANDASAN KEPUSTAKAAN

Membahas mengenai kajian pustaka serta teori penunjang yang berhubungan dengan dasar teori tentang konsep dasar Rekayasa Perangkat Lunak (RPL), konsep dasar pendekatan berorientasi objek, konsep dasar *Object Oriented System Analysis and Design (OOAD)*, konsep dasar *Object Oriented Programming (OOP)*, konsep kebutuhan dalam tahapan pengembangan perangkat lunak, konsep dasar pemodelan, konsep dasar diagram *Unified Modelling Languange* (UML), konsep *use case diagram,* konsep *use case*

scenario, konsep *sequence diagram,* konsep *class diagram,* konsep pengujian perangkat lunak seperti pengujian *white box* dan *black box*, konsep pengujian akurasi perhitungan, konsep dasar *System Development Life Cycle* (SDLC), konsep dasar SDLC *waterfall,* konsep dasar implementasi perangkat lunak, konsep dasar pemrograman, konsep dasar bahasa pemrograman java, konsep dasar *library spoon*, konsep dasar *Latent Semantic Indexing (LSI)*, konsep dasar *library JAMA Matrix Package*, konsep dasar kopling, konsep dasar konseptual kopling dalam perangkat lunak, konsep dasar *information retrieval,* dan konsep dasar *conceptual coupling metrics* seperti *Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Methods and a Class (CSMC), Conceptual Similarity between Two Classes (CSBC), Conceptual Coupling of a Class (CoCC).*

BAB III METODOLOGI

Membahas metode yang diterapkan pada penelitian ini seperti, studi literatur, perancangan perangkat lunak, implementasi perangkat lunak, serta pengujian dan analisis.

BAB IV REKAYASA KEBUTUHAN

Bab ini membahas tentang analisis kebutuhan sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics.*

BAB V PERANCANGAN DAN IMPLEMENTASI

Membahas tentang penjelasan secara umum hingga detail dari perancangan perangkat lunak. Uraian perancangan sistem ini meliputi perancangan data mengenai masukkan *(input)* dan keluaran *(output)* sistem, perancangan proses mengenai bagaimana sistem akan bekerja dengan proses-proses tertentu, maupun perancangan antarmuka dalam desain dan implementasi yang akan digunakan dalam pembuatan aplikasi ini. Menjelaskan implementasi dari perangkat lunak sesuai dengan perancangan sistem yang telah dibangun.

BAB VI PENGUJIAN

Menjelaskan proses dan hasil dari pengujian sistem yang telah berhasil diimplementasikan dan memastikan bahwa program telah sesuai dengan perancangan dan disertai analisis data perhitungan serta akurasi perhitungannya.

BAB VII PENUTUP

Memaparkan kesimpulan yang didapat dari uraian pembahasan bab-bab sebelumnya, serta saran yang bermanfaat dalam pengembangan penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas tinjauan pustaka yang meliputi kajian pustaka dan dasar teori yang mendasari penelitian. Kajian pustaka membahas tentang penelitian-penelitian serupa sebelumnya. Dasar teori adalah pembahasan teori yang diperlukan untuk menyusun penelitian yang diusulkan.

2.1 Pengukuran Nilai Kopling

Penelitian yang dibahas yaitu penelitian Gui Gui dan Paul D.Scott dengan judul "*Ranking reusability of software components using coupling metrics*" menjelaskan bahwa ada teknik pengukuran baru untuk menentukan besaran nilai kopling pada rancangan perangkat lunak agar memudahkan penggunaan kembali sistem tersebut menggunakan komponen java yang diambil dari mesin pencari. Gui Gui dan Paul D. Scott mendapatkan bukti bahwa dengan mempertimbangkan tiga hal yaitu antara lain menghitung hubungan kopling tidak langsung, mengukur sejauh mana dua kelas jika digabungkan dan terakhir dengan cara mempertimbangkan kompleksitas fungsional *class*. Langkah-langkah dalam menentukan sejauh mana tingkat *reusability* komponen java tersebut digunakan adalah pertama dengan cara mengidentifikasi komponen-komponen fungsionalitas yang diperlukan. Langkah kedua dengan cara menilai kedua class tersebut apakah terdapat ketidakcocokan dan class tersebut mudah diadaptasi untuk sistem yang lebih kompleks (Scoot, 2007).

Kelas A dianggap berhubungan ke kelas lain jika kelas A mengakses satu atau lebih variabel dari klas lain atau setidaknya memanggil salah satu *method* dari klas lain (Scoot, 2007).

Penelitian selanjutnya adalah penelitian yang dilakukan oleh Jitender Kumar Chhabra dan Anshu Parashar (2011) dengan judul *conference* "*Clustering dynamic class coupling data using k-mean and cosine similarity measure to predict class reusability pattern*". Teknik pengelompokan data dapat diterapkan untuk mengetahui tingkat ketergantungan class pada perancangan class diagram. Komponen perangkat lunak yang memiliki karakteristik yang sama dan dipadukan dengan pola kopling yang sering terjadi. Pengelompokan dilakukan dengan cara menghitung nilai bobot tf-idf dan *cosine similarity*. Data kopling tersebut masing-masing akan diwakili dalam ruang vektor N-dimensi. Selanjutnya frekuensi kelas kopling dan frekuensi kelas terbalik dihitung menggunakan tf-idf. Kemudian, untuk menentukan kelompok kelas digunakan *k-mean clustering* dan *cosine similarity*. Pada akhirnya, hasil klusterisasi tersebut diberi peringkat untuk mengetahui komponen-komponen yang dapat digunakan kembali (Chhabra,dkk. 2011).

Penelitian ketiga yang dibahas adalah penelitian yang dilakukan oleh Elliya Lestari dengan judul skripsi "Perhitungan kualitas desain *object-oriented* menggunakan *matrix similarity-based class cohesion* (SCC) pada kelas kohesi berbasis web". Lestari (2015) menjelaskan bahwa untuk menghitung tingkat kualitas *cohesion* desain *object oriented* diperlukan metric pengukuran *similarity*

based class cohesion. Untuk mendapatkan bukti bahwa sebuah *class* memiliki interaksi dan hubungan antar elemen, Lestari (2015) menggunakan metrik kohesi. Lestari (2015) mengukur keterkaitan *method* dengan variabel dalam satu *class* (Lestari, 2015).

Penelitian keempat adalah penelitian yang dilakukan oleh Denys Poshyvanyk dan Andrian Marcus dengan judul "The Conceptual Coupling Metrics for Object-Oriented Systems". Poshyvanyk (2006) melakukan penelitian perhitungan nilai *conceptual coupling* berbasis desktop menggunakan bahasa C++ berdasarkan pada source code yang di *parsing,* perhitungan *information retrieval* yaitu Latent Semantic Indexing *(LSI)* untuk pemodelan dan analisis informasi semantik menggunakan *Latent Semantic Analysis (LSA)* dan menggunakan perhitungan *coupling metrics*. Poshyvanyk (2006) menjelaskan bahwa untuk menentukan nilai kopling dapat menggunakan empat parameter conceptual coupling metrics antara lain *Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Method and a Class (CSMC) , Conceptual Similarity between two Classes (CSBC)* dan *Conceptual Coupling of Class (CoCC)*. Langkah-langkah penelitian tersebut pertama, source code di parsing ke sistem. Langkah kedua, pembuatan *method mappings* lalu disimpan di dalam *corpus builder*. Penyimpanan *methods* ke dalam sebuah *corpus* bertujuan untuk melakukan *indexing information retrieval*. Langkah ketiga, pembuatan perhitungan *information retrieval Latent Sematic Indexing (LSI)* yang terdiri dari sekumpulan *semantik space*. Langkah terakhir, menghitung nilai kopling.

2.2 Dasar Teori

Pada bagian ini akan dibahas mengenai teori-teori yang berkaitan dengan penelitian ini. Teori yang berkaitan diantaranya adalah teori tentang konsep dasar Rekayasa Perangkat Lunak (RPL), konsep dasar pendekatan berorientasi objek, konsep dasar *Object Oriented System Analysis and Design (OOAD)*, konsep dasar *Object Oriented Programming (OOP)*, konsep kebutuhan dalam tahapan pengembangan perangkat lunak, konsep dasar pemodelan, konsep dasar diagram *Unified Modelling Languange* (UML), konsep *use case diagram,* konsep *use case scenario,* konsep *sequence diagram,* konsep *class diagram,* konsep pengujian perangkat lunak seperti pengujian *white box* dan *black box*, konsep pengujian akurasi perhitungan, konsep dasar *System Development Life Cycle* (SDLC), konsep dasar SDLC *waterfall,* konsep dasar implementasi perangkat lunak, konsep dasar pemrograman, konsep dasar bahasa pemrograman java, konsep dasar *library spoon*, konsep dasar *Latent Semantic Indexing (LSI)*, konsep dasar *library JAMA Matrix Package*, konsep dasar kopling, konsep dasar *conceptual coupling* dalam perangkat lunak, konsep dasar *information retrieval,* dan konsep dasar *conceptual coupling metrics* seperti *Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Methods and a Class (CSMC), Conceptual Similarity between Two Classes (CSBC), Conceptual Coupling of a Class (CoCC).*

2.2.1 Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak (RPL) adalah teknik pembuatan perangkat lunak dengan berdasarkan kepada prinsip rekayasa untuk menghasilkan produk perangkat lunak yang efisien, berkualitas tinggi dan memiliki nilai ekonomis. Rekayasa Perangkat Lunak (RPL) juga merupakan disiplin ilmu mulai dari analisis, perancangan, implementasi hingga perawatan sebuah perangkat lunak atau *software* agar *software* tersebut dapat terus terpakai. Relayasa perangkat lunak fokus pada pengembangan perangkat lunak hingga perawatan perangkat lunak. Dalam penerapanannya, rekayasa perangkat lunak harus memenuhi kriteria sebagai berikut:

1. *Maintainability* yaitu perangkat lunak tersebut harus dapat terus dipelihara dan dirawat terhadap seiringnya perkembangan teknologi dan lingkungannya.
2. *Dependability* dan *robust* yakni proses bisnis pada perangkat lunak tersebut harus bisa diandalkan dan mampu menerima perubahan.
3. Perangkat lunak harus efisien baik dari segi penggunaan maupun sumber daya.
4. *Usability* yaitu perangkat lunak mampu digunakan sesuai dengan kebutuhan pengguna bukan kebutuhan pengembang.

Rekayasa perangkat lunak fokus pada "*what*" yang berarti pihak pengembang harus mencari tahu dan menganalisis informasi apa saja yang dibutuhkan untuk membangun perangkat lunak yang diinginkan *customer* dan informasi apa saja yang harus diproses, perangkat lunak dengan kinerja seperti apa yang diinginkan *customer* hingga kriteria validasi kebutuhan sistem yang dibutuhkan sistem (Rosa,2013).

2.2.2 Pendekatan berorientasi objek

Objek adalah suatu entitas yang memiliki satu set atribut beserta operasi. Objek tersebut dapat dihubungkan dengan objek lain dengan cara saling menukar informasi, mengirim pesan ke objek lain, menerima pesan dari objek lain dan memanggil operasi atau *method* yang dimiliki objek lain. Objek-objek tersebut diciptakan melalui sebuah klas objek dimana klas tersebut merupakan *template* atau *blue print.* Sebuah objek dapat diinstantiasi melalui *class* (Sommerville, 2003).

Pendekatan berorientasi objek menggabungkan data dan proses secara bersamaan dalam bentuk paket, sehingga sangat memudahkan pengembang dalam mengelola paket tersebut. Hal ini juga menjadi kelebihan dari pendekatan ini, yaitu kode program menjadi lebih mudah untuk digunakan kembali. Pendekatan berorientasi objek memperkenalkan beberapa istilah baru yaitu kelas, objek, hubungan antar kelas, dan lain-lain (Widodo, 2011).

Class adalah abstraksi dari kumpulan objek yang memiliki batasan, semantik yang sama dan spesifikasi fitur yang sama. Jika berbicara

mengenai program komputer, objek berupa entitas yang memiliki tiga karakteristik dasar antara lain, keadaan, identitas dan perilaku. Setiap objek yang tercipta dari sebuah *class* bisa saja tidak selalu memiliki perilaku yang sama tetapi berbeda-beda sesuai dengan keadaan (Widodo, 2011).

2.2.3 Object Oriented System Analysis and Design (OOAD)

Tahap analisis sistem membantu seorang analis sistem melihat dan memperhatikan kebutuhan sistem dengan lebih seksama. Kebutuhan-kebutuhan dalam sistem dimasukkan ke dalam sebuah dokumentasi kebutuhan yang nantinya harus dipenuhi sistem. Proses analisis sistem dan desain sistem sering dilakukan secara bersama-sama. Hal ini menguntungkan bagi pihak pengembang maupun pengguna karena sering ditemukan pengguna yang kesulitan dalam menyampaikan kebutuhan mereka (Rosa,2013). Analisis dalam hal sistem berorientasi objek dapat diartikan sebagai proses pengembangan berorientasi objek berdasarkan domain permasalahan. Objek direpresentasikan sebagai entitas dan operasi yang berhubungan dengan domain permasalahan (Sommerville, 2003).

Desain sistem adalah penentuan arsitektur sistem secara keseluruhan yang terdiri atas satu kesatuan komponen proses, perangkat lunak yang digunakan, perangkat keras yang digunakan, aktor, sistem lain dan komunikasi yang terjadi di antara komponen-komponen tersebut dimana proses ini harus memenuhi kebutuhan sistem yang ada (Shumate, 1992). Sebelum melakukan desain sistem, kebutuhan bisnis yang telah didapatkan dari wawancara terhadap pengguna pada tahap analisis kebutuhan dialihkan menjadi sebuah kebutuhan sistem yang menggambarkan detail kebutuhan sistem secara teknis. Hal ini sangat berguna bagi pembangunan sistem agar sesuai dengan kebutuhan yang telah teridentifikasi sebelumnya.

Perancangan berorientasi objek fokus pada hubungan antara objek-objek dengan berbagai solusi masalah yang diimplementasikan sesuai dengan kebutuhan sistem. Fokus perancangan berorientasi objek bukan operasi maupun fungsi melainkan objek. Dalam tahap ini, dapat ditemui kasus dimana objek yang telah dibuat perlu ditambahkan objek lagi agar menemukan solusi dari masalah. Objek-objek tersebut di definisikan sebagai class. Objek-objek tersebut memiliki hak akses pribadi. Hak akses pribadi tersebut berarti informasi yang dimiliki objek tersebut hanya diketahui dan boleh diakses oleh objek itu sendiri. Objek berhak mengatur hak aksesnya apakah boleh diakses oleh objek lain atau tidak. Objek dapat menyembunyikan seluruh informasinya atau sebagian (Sommerville, 2003). Dari analisis dan perancangan berorientasi objek maka sistem dapat diimplementasikan menggunakan metode pemrograman berorientasi objek dimana bahasa pemrograman yang digunakan adalah bahasa pemrograman objek seperti *java* yang memiliki fasilitas lengkap dalam mendefinisikan *object class*.

2.2.4 Kebutuhan

Kebutuhan adalah sekumpulan pernyataan sederhana yang harus dipenuhi oleh sistem. Kebutuhan juga dapat diartikan sebagai karakteristik yang harus dimiliki oleh sistem (Dennis,dkk. 2006). Selama proses analisis berlangsung, kebutuhan ditulis dari perspektif bisnis dan fokus pada apa yang akan dilakukan sistem. Pada tahap perancangan, kebutuhan dibahas secara lebih teknis dan pada tahap perancangan ini kebutuhan akan menggambarkan bagaimana sistem diimplementasikan. Dari sudut pandang para pengembang, kebutuhan diserbut dengan kebutuhan sistem atau *system requirements.*

Kebutuhan dibagi menjadi dua jenis, yaitu kebutuhan fungsional dan kebutuhan non-fungsional. Kebutuhan fungsional adalah segala kebutuhan yang berhubungan langsung dengan proses-proses yang ada dalam sistem atau informasi yang dibutuhkan dalam sistem (Dennis,dkk. 2006).Contoh kebutuhan fungsional salah satunya adalah sistem harus mampu mencari ketersediaan barang dalam inventaris. Kebutuhan fungsional mendefinisikan seluruh fungsi yang harus dimiliki oleh sistem.

Kebutuhan non-fungsional adalah sekumpulan perilaku yang harus dimiliki oleh sistem, seperti *performance* dan *usability* (Dennis,dkk. 2006).Salah satu contoh kebutuhan non-fungsional seperti sistem ini mampu diakses melalui beragam *web browser*.

2.2.5 Pemodelan

Pemodelan adalah proses pembuatan representasi abstrak dalam bentuk objek dari sesuatu yang nyata. Pembuatan abstraksi objek ini menggunakan aturan tertentu. Pemodelan berguna untuk meminimalisasi kegagalan serta resiko yang dapat terjadi sewaktu-waktu. Pemodelan juga berguna untuk memudahkan pemahaman bagi pengguna dan pengembang sehingga diharapkan mampu menyamakan pemikiran mengenai sistem yang akan dibangun walaupun bahasa teknis sudah digunakan pada tahap ini. Pemodelan sangat berguna untuk tahap berikutnya karena dengan pemodelan sistem informasi yang akan dibangun terlihat secara rinci, terencana dan terorganisir dengan sangat baik. Model ini pula dapat meramalkan bagaimana sistem nantinya akan bekerja (Rosa, 2013).

Abstraksi pemodelan perangkat lunak antara lain (Rosa, 2013):

1. Fokus sistem berada pada proses yang ada pada model.
2. Spesifikasi sistem tertera secara abstrak.
3. Dapat berupa spesifikasi lengkap dari sebuah sistem yang final.
4. Dapat terdiri dari spesifikasi umum dan spesifikasi khusus.
5. Spesifikasi sistem tertera secara sebagian atau keseluruhan.

2.2.6 Unified Modelling Languange (UML)

Unified Modelling Languange (UML) adalah suatu bahasa pemodelan sistem yang berdasarkan *object*. Pemodelan berfungsi sebagai penyederhana masalah sehingga mudah dipahami dan dimengerti oleh manusia (Adi, 2010). Diagram juga dapat berfungsi sebagai media penjelas berbagai macam ide desain perangkat lunak dan membantu pemahaman sebuah proses bisnis (Fowler, 2004).

UML dibagi menjadi tiga kategori (Rosa, 2013):

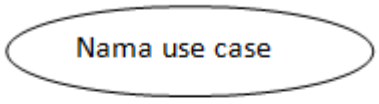
1. *Structure diagrams* adalah kumpulan model diagram statis sistem.
2. *Behavior diagrams* adalah kumpulan diagram perilaku sistem serta perubahan sistem.
3. *Interaction diagrams* adalah kumpulan diagram interaksi yang menghubungkan dengan sistem luar atau sistem dengan subsistem.

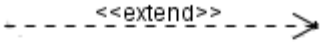
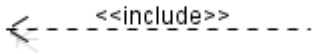
2.2.7 Use Case Diagram

Use case diagram adalah pemodelan berorientasi objek yang direpresentasikan dalam bentuk diagram. Fungsi *use case diagram* yaitu untuk menggambarkan perilaku sistem informasi khususnya interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibangun. Selain itu, *use case diagram* juga berguna untuk mengetahui fitur apa saja yang ada pada sistem. Penamaan *use case diagram* harus sederhana, jelas dan mudah dipahami. Berikut komponen-komponen yang ada dalam *use case diagram* (Rosa, 2013):

Tabel 2.1 Komponen Use Case Diagram

Sumber : (Rosa, 2013)

Simbol	Deskripsi
Use case 	Simbol ini menggambarkan unit-unit yang bertukar pesan antara aktor dengan sistem. Penamaan *use case* harus menggunakan kata kerja.
Aktor 	Aktor dapat berupa orang, sistem lain atau proses yang berinteraksi dengan sistem secara langsung. Penamaan aktor menggunakan kata benda.
Asosiasi/*association* ———	Garis lurus ini menggambarkan asosiasi yakni komunikasi antara aktor dengan *use case*.
Ekstensi/*extend*	Relasi *use case* yang menandakan

	bahwa *use case* tersebut dapat berdiri sendiri (*stand-alone)* walau tanpa *use case* tambahan. Sifatnya lebih opsional, dapat digunakan atau tidak.
Generalisasi 	Relasi generaliasasi antara dua *use case* dimana salah satu *use case* lebih umum dari yang lainnya. Arah panah menunjukkan bahwa *use case* tersebut menjadi generalisasi *use case* yang dihubungkan dengan simbol ini (umum ke khusus).
Include 	Relasi *include* ini menggambarkan bahwa *use case* yang terhubung dengan relasi ini memerlukan *use case* lain untuk menjalankan fungsinya atau *use case* lain menjadi syarat untuk menjalankan fungsinya. *Use case* yang menggunakan relasi ini wajib menggunakan *use case* tambahannya tersebut. Terdapat dua persepsi yang beredar di kalangan pengembang perangkat lunak mengenai *include*: 1. Relasi *include* tersebut berarti menggambarkan *use case* akan selalu dipanggil saat *use case* tambahan dijalankan. 2. Relasi *include* menggambarkan *use case* selalu memeriksa kembali apakah *use case* yang induk telah dijalankan sebelum *use case* tambahan dijalankan. Kedua persepsi diatas dapat digunakan tergantung pada keadaan dan kebutuhan.

2.2.8 Sequence Diagram

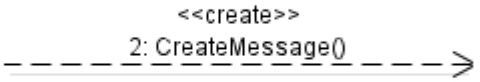
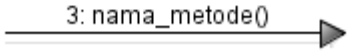
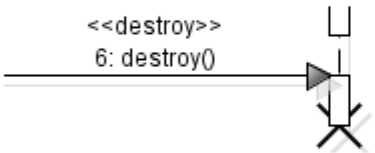
Sequence diagram adalah diagram yang menggambarkan perilaku objek pada *use case* dengan melibatkan waktu hidup objek dan pesan atau *message* yang dikirim dan diterima antar objek. Masing-masing *use case* memiliki satu *sequence diagram.* Pembuatan sequence diagram juga

berdasarkan *use case scenario.* Hal pertama yang dilakukan sebelum membuat *sequence diagram* ialah menentukan objek-objek yang terlibat dalam *use case* tersebut dan menentukan *method-method* apa saja yang dimiliki *class* yang nantinya diinstantiasi menjadi objek tersebut (Rosa, 2013). Simbol-simbol dalam *sequence diagram* antara lain sebagai berikut (Rosa, 2013):

Tabel 2.3 Komponen *Sequence Diagram*

Sumber: (Rosa, 2013)

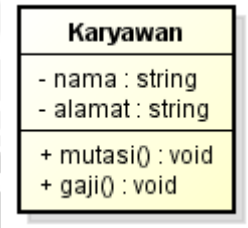
Simbol	Deskripsi
Aktor 	Aktor dapat berupa orang atau sistem lain yang berada pada luar sistem yang dibuat. Penamaan menggunakan kata benda.
Garis hidup/*lifeline*	Simbol garis putus-putus ini merepresentasikan waktu hidup sebuah objek.
Objek 	Menggambarkan objek yang mengirim dan menerima pesan.
Waktu aktif 	Simbol ini menandakan bahwa objek berada pada kondisi aktif dan berinteraksi dengan objek lainnya. Waktu aktif ini menandakan bahwa ada suatu tahapan atau proses yang dilakukan di dalamnya. Hanya

	objek yang memiliki waktu aktif. Aktor tidak memiliki waktu aktif.
Pesan/*message* tipe *create* 	Simbol ini menandakan bahwa ada suatu objek yang membuat objek lain. Tanda panah selalu mengarah pada objek yang dibuat.
Pesan/*message* tipe *call* 	Simbol ini menandakan bahwa objek tersebut memanggil *method* dari objek lain atau *method* yang dimiliki oleh dirinya sendiri. Tanda panah mengarah pada objek yang memiliki *method* yang dipanggil. *Method* tersebut harus ada dan sesuai dengan yang ada pada *class diagram*.
Pesan/*message* tipe *return* 	Simbol ini menandakan bahwa suatu objek telah menjalankan *method* dan objek tersebut memberikan suatu kembalian ke objek tertentu. Tanda panah mengarah kepada objek yang menerima kembalian.
	ke objek tertentu. Tanda panah mengarah kepada objek yang menerima kembalian.
Pesan/*message* tipe *destroy* 	Simbol ini menandakan bahwa suatu objek menghentikan *lifeline* atau garis hidup objek lain. Tanda panah mengarah kepada objek yang diakhiri. Simbol ini dapat digunakan ketika ada *message create* yang harus diakhiri.

2.2.9 Class Diagram

Klas objek dapat direpresentasikan dalam bentuk persegi panjang dan dibagi menjadi tiga bagian, yaitu bagian atas berisi nama klas, bagian tengah berisi atribut objek dan bagian bawah berisi operasi atau *method-method* objek. Atribut dan operasi yang ada pada *class* tersebut harus yang berkaitan erat dengan objek yang akan diinstantiasi dari *class*

tersebut (Sommerville, 2003). Berikut contoh dari *class diagram* (Sommerville, 2003):



Gambar 2.1 Contoh *Class Diagram*
Sumber: (Sommerville, 2003)

2.2.10 Class

Class adalah *blueprint* atau *template* secara umum untuk mendefinisikan dan membuat *instance* secara spesifik atau bahkan untuk membuat banyak objek. Setiap objek terhubung dengan *class.* Seperti contohnya, semua objek pasien yang menangkap informasi mengenai pasien akan memanggil *class* Pasien karena objek pasien saling berbagi banyak atribut dan *method* (Dennis,dkk. 2006).

Sebuah objek merupakan instantiasi dari sebuah class. Objek dapat berupa orang, tempat, kejadian atau sesuatu yang dapat menangkap informasi. Jika ingin membuat sebuah sistem rumah sakit maka *class* yang dibutuhkan adalah *class* Dokter, *class* Pasien dan *class* Perjanjian. Pasien yang spesifik seperti Budi, Nina, Anto adalah sebuah *instance* atau objek dari *class* Pasien (Dennis,dkk. 2006).

2.2.11 Method

Method mengimplementasikan tingkah laku objek. *Method* berupa aksi yang dilakukan oleh objek. *Method* seperti fungsi atau prosedur dalam bahasa pemrograman terstruktur. *Messages* akan mengirimkan informasi kepada objek untuk dapat menjadi *trigger* akan method dijalankan. *Message* secara esensial berupa fungsi atau prosedut yang memanggil dari satu objek ke objek lain. Salah satu contohnya, jika ada seorang pasien baru yang belum terdaftar pada rumah sakit tersebut maka sistem akan mengirim sebuah *insert message* kepada objek pasien. Objek pasien akan menerima *message* atau instruksi dan melakukan apa yang diperintahkan untuk memasukkan pasien baru tersebut ke dalam sistem (Dennis,dkk. 2006).

2.2.12 Pengujian Perangkat Lunak

Pengujian adalah tahapan untuk menganalisis dan mengetahui apakah perangkat lunak terdapat kesalahan atau tidak. Hal ini sangat penting sebelum perangkat lunak benar-benar disebarluaskan kepada pengguna. Pengujian juga berguna untuk mengevaluasi kembali hasil akhir perangkat lunak agar sesuai dengan perancangan yang telah dilakukan sebelumnya (Rouf, 2012).

Tujuan pengujian perangkat lunak sebagai berikut (Rouf, 2012):

1. Mencari dan mengevaluasi kesalahan pada perangkat lunak dengan cara menjalankan program.
2. Pengujian yang baik harus menggunakan kasus uji dengan tingkat peluang besar terjadi kesalahan *(error)* yang belum diketahui.
3. Pengujian yang baik mampu menemukan kesalahan program yang belum diketahui.
4. Pengujian yang baik dan dapat dikatakan berhasil adalah ketika penguji menemukan banyak kesalahan. Dari kesalahan-kesalahan tersebut pengembang mampu mengevaluasi ulang perangkat lunak mereka. Inilah fungsi pengujian perangkat lunak, bukan berarti pada saat program dieksekusi penguji hanya dapat memastikan tidak terjadi kesalahan pada saat program dijalankan melainkan masih banyak komponen-komponen baik di dalam maupun di luar program yang mungkin saja memiliki kesalahan.

Terdapat empat langkah-langkah secara umum pengujian perangkat lunak, antara lain sebagai berikut (Dennis,dkk. 2006):

1. Unit Test

Unit Test fokus pada pengujian terhadap unit-unit yang ada pada program tersebut seperti menguji modul-modul yang merepresentasikan fungsi spesifik yang dapat diuji. Pengujian ini berguna untuk memastikan bahwa modul-modul atau program berfungsi sesuai dengan spesifikasi program. *Unit Testing* dilakukan sesudah *programmer* mengimplementasikan program dengan menggunakan bahasa pemrograman. Hasil implementasi tersebut berupa *code*. *Code* itulah yang akan diuji menggunakan beberapa kasus uji untuk memastikan tidak ada kesalahan dalam *code* dalam mengimplementasikan modul-modul. Orang yang menguji dalam *unit test* biasanya *system analyst* atau terkadang *programmer* yang mengembangkan unit tersebut (Dennis,dkk. 2006).

2. Integration Test

Integration Test merupakan suatu proses pengujian hubungan modul-modul dalam program yang harus bekerja bersama-sama tanpa adanya kesalahan *(error)* (Dennis,dkk. 2006).

3. System Testing

Pengujian perangkat lunak bertujuan untuk menguji keseluruhan sistem agar dapat bekerja secara bersama-sama tanpa terjadi kesalahan (*error)* (Dennis,dkk. 2006).

4. Acceptance Testing

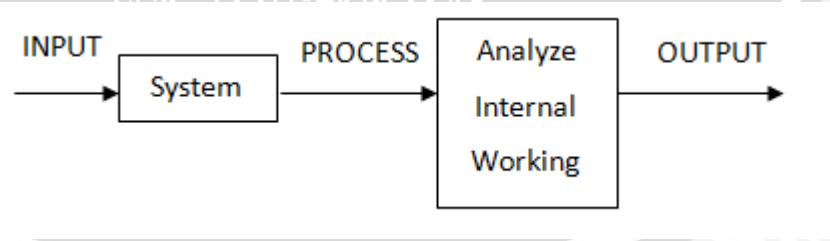
Menguji apakah program mampu diterima dengan baik oleh pengguna. Tujuan utama para pengembang dalam membangun sebuah perangkat lunak ialah sistem komplit, sistem mampu memenuhi kebutuhan bisnis sesuai yang dikembangkan dan mampu diterima oleh pengguna. *Acceptance testing* dapat dilakukan dengan dua langkah. Pertama, *alpha testing* dimana pengguna menguji sistem menggunakan data yang dibuat-buat. Kedua, *beta testing* dimana pengguna menguji sistem dengan cara mulai menggunakan sistem tersebut menggunakan *real data* dan memperhatikan apakah terjadi kesalahan atau tidak pada saat sistem digunakan (Dennis,dkk. 2006).

2.2.13 Kasus Uji (Test Case)

Untuk melakukan pengujian perangkat lunak, diperlukan sejumlah kasus uji dalam proses pengujiannya. Kasus uji dengan kriteria yang baik adalah kasus uji yang mampu memiliki peluang tinggi dalam menemukan kesalahan dalam perangkat lunak. Perancangan *test case* ini dilakukan sebelum pengujian dilaksanakan. Dengan begitu, waktu dan usaha yang dikeluarkan untuk menguji perangkat lunak menjadi minimum (Rouf, 2012). Perancangan *test case* dapat dilakukan dengan dua cara yaitu sebagai berikut (Rouf, 2012):

1. *White Box Testing*

Kasus uji dibuat berdasarkan struktur kontrol dari prosedur yang diperoleh dari perancangan (Pressman, 2010).



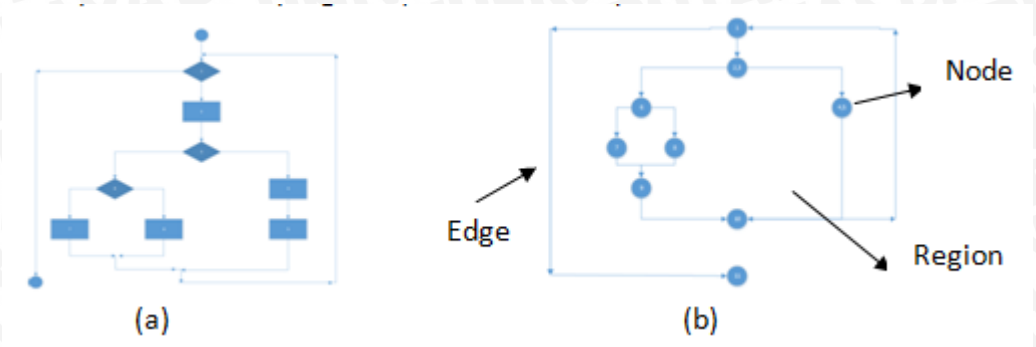
Gambar 2.2 Sistem Kerja Pengujian *White Box*

Sumber : (Rouf, 2012)

1.1 Basis Path Testing

Pengujian *basis path* termasuk ke dalam kategori pengujian *white box.* Penguji perangkat lunak dapat menghitung kompleksitas siklomatis *(cyclomatic complexity)* dari diagram alir *(flowchart)* dan menggunakan diagram alir ini sebagai acuan untuk menentukan basis

prosedur jika program tersebut dijalankan. *Cyclomatic complexity* adalah suatu metrik yang digunakan untuk mengukur tingkat aliran kompleksitas suatu program (Pressman, 2010).



Gambar 2.3 Perubahan *flowchart* (a) ke bentuk *flowgraph* (b)
Sumber: (Pressman, 2010)

Diagram alir *(flowchart)* diubah menjadi graph alir *(flowgraph)* untuk menghitung kompleksitas siklomatis. *Flowgraph* terdiri dari *node* dan *edge*. Beberapa ketentuan dalam menghitung kompleksitas program sebagai berikut:

1. Jumlah region *flowgraph* sesuai dengan hasil dari perhitungan rumus kompleksitas siklomatis.
2. Kompleksitas siklomatis atau yang dituliskan dengan simbol V(G) memiliki rumus V(G) = E – N + 2, dimana E adalah jumlah *edge* dan N adalah jumlah *node.*
3. Kompleksitas siklomatis, V(G) = P + 1, dimana P merupakan predikat *node* yakni *node* yang memiliki percabangan atau kondisi.
4. Dari nilai hasil rumus V(G) dicari jalur independen sebanyak nilai V(G). Dari jalur independen inilah penguji dapat menentukan kasus uji *(test case).* Setelah menemukan *test case,* maka proses pengujian perangkat lunak dapat langsung dikerjakan sesuai dengan kasus uji yang diperoleh.

2. Black-box Testing

Pengujian *black-box testing* berguna untuk mengetahui apakah semua fungsi dalam perangkat lunak sudah berjalan sesuai dengan kebutuhan fungsional yang telah didefinisikan sebelumnya. Pengujian ini dilakukan pada tahap akhir pengujian perangkat lunak. Dalam *black-box testing* pengujian yang dilakukan jenisnya bernama pengujian validasi. Pengujian validasi fokus pada sudut pandang pengguna dan keluaran *(output)* dari sistem apakah sudah sesuai dengan kebutuhan pengguna dan kebutuhan fungsional atau tidak. Pengujian dikatakan berhasil jika semua fungsi yang ada dalam perangkat lunak dapat dikenali oleh

pengguna. Validasi dilakukan dengan cara menyediakan konfirmasi dengan persyaratan yang telah ditentukan sebelumnya.

3. Akurasi

Perhitungan nilai akurasi menggunakan rumus sebagai berikut (Suswanto, 2016):

$$\text{Akurasi} = \frac{\sum data\ uji\ benar}{\sum data\ uji} \times 100\%$$

2.2.14 System Development Life Cycle

System Development Life Cycle (SDLC) adalah serangkaian proses pemahaman bagaimana sebuah sistem informasi dapat mendukung kebutuhan bisnis (*bussiness needs),* desain sistem, pengembangan sistem dan pada akhirnya sistem tersebut sampai ke pengguna. SDLC terdengar mudah namun dalam praktiknya pengembangan sistem berdasarkan SDLC tergolong sulit. Pada tahun 2004, sebuah survey yang dilakukan oleh *Standish Group* menemukan fakta bahwa hanya 28% proyek sistem informasi dapat berhasil dikembangkan saat itu dan 72% proyek dibatalkan sebelum selesai. Banyak proyek sistem informasi pada akhirnya sampai ke pengguna dalam waktu yang telat, memakan biaya melebihi rencana awal dan hanya sedikit fitur yang dapat direalisasikan (Dennis,dkk. 2006).

Kunci utama dalam metode SDLC adalah seorang analis sistem yang menganalisis situasi bisnis atau domain permasalahan, mengidentifikasi kesempatan perbaikan pada bisnis proses yang ada dan merancang sistem informasi untuk mengaplikasikan semua kebutuhan pengguna (Dennis,dkk. 2006).

Siklus pengembangan perangkat lunak menggunakan metode-metode pengembangan perangkat lunak yang sering dipakai oleh pengembang sebelumnya. Metode yang digunakan biasanya berupa *best practice* yang sudah teruji dengan baik. Software yang dikembangkan dengan SDLC dapat menghasilkan software berkualitas tinggi baik dari segi efisiensi waktu maupun biaya. Dengan menggunakan metode SDCL, *customer* dan pengembang dapat membuat sebuah kesepakatan yang baik demi terwujudnya *software* yang sesuai dengan keinginan *customer* (Rosa,2013). Secara umum, SDLC memiliki tahapan-tahapan sebagai berikut (Mishra, 2013):

1. Analisis

Pada tahap ini dilakukan analisa kebutuhan sistem dan mengembangkan kebutuhan pengguna. Salah satu cara menggali kebutuhan sistem adalah dengan melakukan wawancara kepada *customer*. Dokumentasi kebutuhan sangat diperlukan dalam tahap ini agar memudahkan dalam pelacakan kebutuhan untuk fase-fase berikutnya.

2. Perancangan

Mengubah kebutuhan sistem yang telah didapatkan pada tahap analisis menjadi sebuah rancangan sistem yang akan dibangun pada tahap implementasi.

3. Implementasi

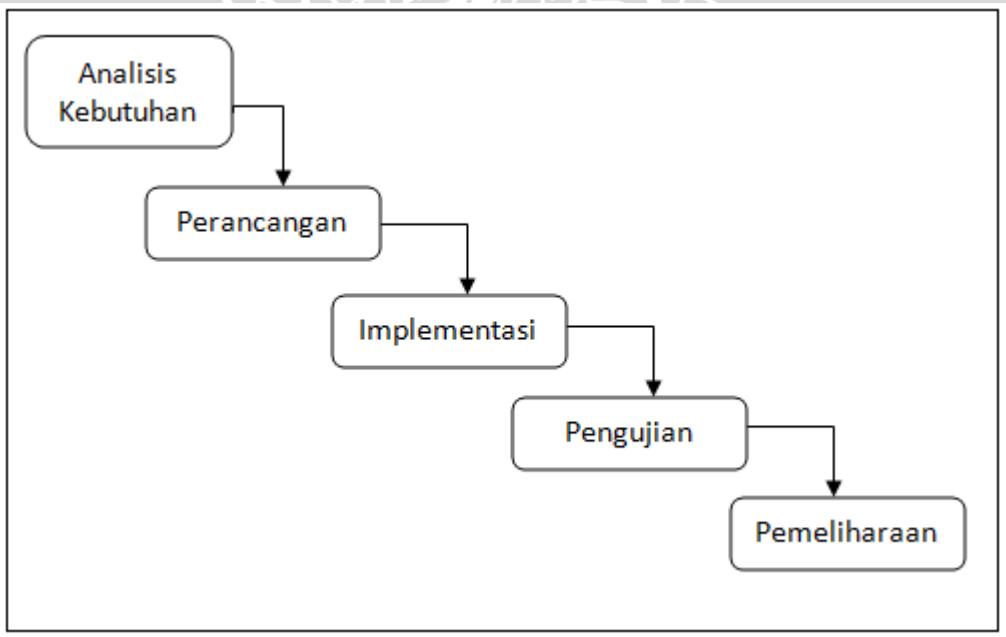
Tahap ini merupakan tahap penyusunan sistem berdasarkan perancangan yang telah dibuat sebelumnya. Sistem harus berdasarkan kebutuhan dan tujuan *customer* dan pengguna.

4. Pemeliharaan

Tahapan perbaikan dan pemeliharaan software dari kesalahan sistem yang muncul kapan saja.

2.2.15 Model SDLC Waterfall

Model SDLC waterfall atau biasa disebut dengan model air terjun merupakan salah satu model SDLC klasik yang memegang konsep pendekatan siklus hidup perangkat lunak secara berurutan atau sekuensial mulai dari analisis, perancangan, implementasi, pengujian dan pemeliharaan.



Gambar 2.4 SDLC Model Waterfall

Sumber: (Sommerville, 2010)

Keterangan:

1. Analisis Kebutuhan

   Pengembang melakukan analisis permasalahan atau *domain* serta menggali kebutuhan sistem salah satunya dengan cara melakukan wawancara secara langsung dengan *customer*. Tujuannya adalah untuk mengetahui perangkat lunak seperti apa yang diinginkan

customer, menjadi dasar bagi perancangan perangkat lunak serta menjadi referensi dalam melakukan validasi kebutuhan.

2. Perancangan

   Mengubah kebutuhan menjadi dalam bentuk perancangan sistem yang akan dibangun berupa arsitektur sistem. Tujuannya adalah memberikan gambaran sistem sesuai dengan kebutuhan yang sudah didapatkan sebelumnya.

3. Implementasi

   Mengimplementasikan rancangan sistem sebelumnya dalam bentuk kode pemrograman.

4. Pengujian

   Pengembang melakukan pengujian terhadap sistem yang telah diimplementasikan apakah sudah sesuai dengan kebutuhan dan sudah sesuai dengan perancangan atau belum. Pengujian perangkat lunak menggunakan metode tertentu diantaranya *white box* dan *black box.* Pada tahap ini pula dilakukan penggabungan modul-modul yang telah dibuat dan dilakukan pengujian unit, integrasi dan sistem.

5. Pemeliharaan

   Tahap terakhir yakni tahap pemeliharaan perangkat lunak yang telah dibangun, diuji dan diinstalasikan. Pada tahap ini dilakukan pemeliharaan dan perbaikan terhadap kesalahan-kesalahan sistem yang bisa terjadi kapan saja.

Model waterfall memiliki beberapa kelebihan antara lain mudah digunakan, semua kebutuhan dapat dijelaskan secara utuh pada awal pengerjaan projek, mudah menentukan biaya serta waktu dan dapat digunakan dalam sistem berskala besar. Di satu sisi, model waterfall juga memiliki kekurangan yaitu alur pengembangan dilakukan secara sekuensial. Tahapan yang sekuensial dapat menghambat tahapan selanjutnya jika terjadi keterlambatan dan adanya hambatan pada tahap sebelumnya.

2.2.16 Konsep Dasar Kopling

Kopling adalah sebuah pengukuran yang berdasarkan dari kekuatan hubungan dari komponen satu ke komponen lainnya (Dennis,dkk. 2006). Komponen dalam konsep *object oriented* disebut *class diagram*. Class diagram yang nantinya bisa dijadikan objek dikatakan memiliki hubungan antar komponen jika memiliki kriteria sebagai berikut:

1. Ada sebuah *message* yang dilewati oleh objek maka objek ini dikatakan ada keterkaitan atau hubungan (Dennis,dkk. 2006).
2. Jika *class-class* tersebut mendeklarasikan dan menggunakan *method* atau atribut dari *class* lain (Dennis,dkk. 2006).
3. Jika class-class tersebut memiliki relasi inheritance yang secara signifikan kuat antara superclass dengan subclassnya. Inheritance

adalah hubungan atau relasi antara dua *class* atau lebih dimana kedua class berbagi perilaku yang sama (Dennis,dkk. 2006).

Kopling merupakan salah satu parameter kualitas perancangan perangkat lunak untuk memudahkan *developer* perangkat lunak dalam melakukan *maintenance* perangkat lunak. Perancangan perangkat lunak yang baik adalah yang memiliki kopling serendah mungkin. Hal ini dikarenakan jika kopling rendah maka pada saat perubahan tidak banyak komponen lain ikut berubah (Poshyvanyk, 2006).

2.2.17 Konsep Dasar *Conceptual Coupling Metrics*

Conceptual coupling adalah sebuah dimensi pengukuran kopling yang baru untuk melengkapi pengukuran kopling yang sudah ada sebelumnya. *Conceptual coupling* menghitung tingkat kopling antar komponen berdasarkan informasi semantik yang dibagi antar elemen pada *source code* program. *Conceptual coupling* juga digunakan untuk memperbanyak ragam pengukuran kopling (Poshyvanyk, 2006).

2.2.18 *Conceptual Similarity between Methods (CSM)*

Parameter CSM menghitung nilai kemiripan antara *method* satu dengan *method* lainnya maupun *method* itu sendiri. Pengukurannya berdasarkan nilai vektor yang dimiliki masing-masing *method.* Nilai CSM sama dengan nilai *cosine similarity* yang dihasilkan pada perhitungan *Latent Semantic Indexing (LSI).* Perhitungan CSM dirumuskan pada rumus (1.1) sebagai berikut (Poshyvanyk, 2006):

$$\text{CSM}\ (m_k, m_j) = \frac{vm_k^T vm_j}{|vm_k|_2 \ x \ |vm_j|_2} \qquad (1.1)$$

Keterangan rumus (1.1):

CSM = *Conceptual Similarity between Methods*
m_k = *method* k
m_j = *method* j
vm_k^T = vektor matriks *method* k yang di transpose
vm_j = vektor matriks *method* j
$|vm_k|_2$ = panjang vektor *method* k
$|vm_j|_2$ = panjang vektor *method* j

2.2.19 *Conceptual Similarity between a Methods and a Class (CSMC)*

Parameter CSMC menghitung nilai kemiripan antara method dengan class. Perhitungan CSMC dirumuskan pada rumus (1.2) sebagai berikut (Poshyvanyk, 2006):

$$CSMC\ (m_k, c_j) = \frac{\sum_{q=1}^{t} CSM^1(m_j, m_{jq})}{t} \quad (1.2)$$

Keterangan rumus (1.2):

CSMC = Conceptual Similarity between a Methods and a Class

m_k = *method* yang menjadi pembanding.

c_j = *class* yang menjadi pembanding

t = jumlah *method* yang dimiliki c_j

$\sum_{q=1}^{t}$ = penjumlahan antara nilai CSM m_k terhadap masing-masing *method* yang dimiliki oleh c_j

$CSM^1(m_j, m_{jq})$ = nilai CSM m_j terhadap masing-masing *method* yang dimiliki *class* c_j

2.2.20 *Conceptual Similarity between Two Classes (CSBC)*

Parameter CSBC menghitung nilai kemiripan antara dua *class*. Perhitungan CSBC dirumuskan pada rumus (1.3) sebagai berikut (Poshyvanyk, 2006):

$$CSBC\ (c_k, c_j) = \frac{\sum_{l=1}^{r} CSMC\ (m_{kl}, c_j)}{r} \quad (1.3)$$

Keterangan rumus (1.3):

CSBC = *Conceptual Similarity between Two Classes*

c_k = *class* yang dibandingkan

c_j = *class* yang dibandingkan

$\sum_{l=1}^{r}$ = penjumlahan nilai CSMC

r = jumlah *method* yang dimiliki oleh c_k

2.2.21 Conceptual Coupling of a Class (CoCC)

Parameter CoCC menghitung nilai *conceptual coupling*. Perhitungan CoCC dirumuskan pada rumus (1.4) sebagai berikut (Poshyvanyk, 2006):

$$CoCC\ (c) = \frac{\sum_{i=1}^{n} CSBC\ (c, d_i)}{n-1} \quad (1.4)$$

Keterangan rumus (1.4):

COCC = *Conceptual Coupling of a Class*

$\sum_{i=1}^{n} CSBC(c, d_i)$ = penjumlahan nilai CSBC perbandingan antara *class* yang satu dengan *class* lainnya.

n = banyak *class*

2.2.22 Konsep Dasar *Information Retrieval*

Information retrieval (Sistem Temu Kembali Informasi) adalah pencarian materi atau dokumen dari yang bentuk informasinya tak

terstruktur (biasanya berupa teks) untuk memenuhi kebutuhan informasi dari dalam koleksi besar. Penyimpanan data berupa informasi ini akan di representasikan untuk memenuhi kebutuhan informasi pengguna. Proses *information retrieval* lebih ditekankan pada informasi bukan data (Ghosh, 2015). Contoh aplikasi yang menggunakan *information retrieval* adalah mesin pencari *Google*.

Information retrieval digunakan sebagai metode bantuan pencarian informasi. Pencarian informasi dilakukan dengan cara mencari informasi yang juga memuat *query*. Query adalah kalimat atau kata yang menjadi acuan untuk pencarian informasi (Bunyamin, 2015). Informasi yang relevan dengan *query* akan ditampilkan sebagai hasil cari informasi. Kategori informasi yang relevan tersebut terdiri dari dua syarat sebagai berikut :

1. Memuat kata atau kalimat yang **sama** dengan *query,* atau
2. Memuat kata atau kalimat yang **bermakna sama** dengan *query*

Makna kata sama dalam hal ini dilihat dari dua pandangan, yaitu sinonim dan polisemi. Sinonim adalah istilah untuk kata yang memiliki makna serupa atau sama. Sedangkan polisemi adalah istilah bagi kata yang sama namun memiliki makna berbeda. Salah satu contoh kata yang memiliki polisemi adalah "membajak". Membajak dalam kalimat "membajak sawah" berarti mengolah tanah di sawah agar dapat segera ditanami tumbuhan. Berbeda halnya dengan kalimat "membajak pesawat" dimana pada kalimat kata "membajak" berarti mengambil alih (Bunyamin, 2015).

Query bersifat tidak terstruktur karena *query* merupakan kalimat yang diekspresikan oleh pengguna. Dokumen adalah salah satu contoh informasi yang tidak terstruktur (Bunyamin, 2015).

2.2.23 *Latent Semantic Indexing (LSI)*

Salah satu contoh metode pencarian informasi dari segi kecocokan makna antara *query* dengan informasi adalah metode *Latent Semantic Indexing (LSI).* Metode LSI bekerja dengan cara mencari dan menemukan informasi dari sebuah dokumen tidak hanya memperhatikan kesamaan kata namun kesamaan makna kata per kata (Bunyamin, 2015).

Langkah-langkah perhitungan *Latent Semantic Indexing (LSI)* secara umum diawali dengan menghitung transpose matriks, menghitung determinan dari matriks yang sudah di transpose, mencari nilai *eigen value,* menghitung nilai *singular value,* menghitung eigen vektor, normalisasi *eigen vektor,* membuat matriks V, membuat matriks U, mereduksi dimensi SVD dengan mengurangi ukuran dimensi sebanyak *k* (dalam penelitian ini *k* = 5), menghitung *cosine similarity* (tingkat kemiripan antar dokumen), memasukkan *query* oleh pengguna, query dicocokkan dengan dokumen *method* yang ada dan langkah terakhir adalah perankingan kemiripan antara *query* dengan dokumen

berdasarkan nilai *cosine similarity*. Perangkingan diurutkan mulai dari nilai *cosine similarity* tertinggi hingga terendah (Bunyamin, 2015).

2.2.24 *Java Matrix Library*

Java Matrix Library atau biasa yang disingkat dengan nama *Jama Library*. *Jama Library* merupakan sebuah *library* yang berisi sekumpulan klas-klas untuk melakukan operasi matriks seperti membuat struktur matriks dan mendekomposisikan matriks, contohnya *Singular Value Decomposition (SVD)* (Bunyamin, 2005).

2.2.25 *Spoon Library*

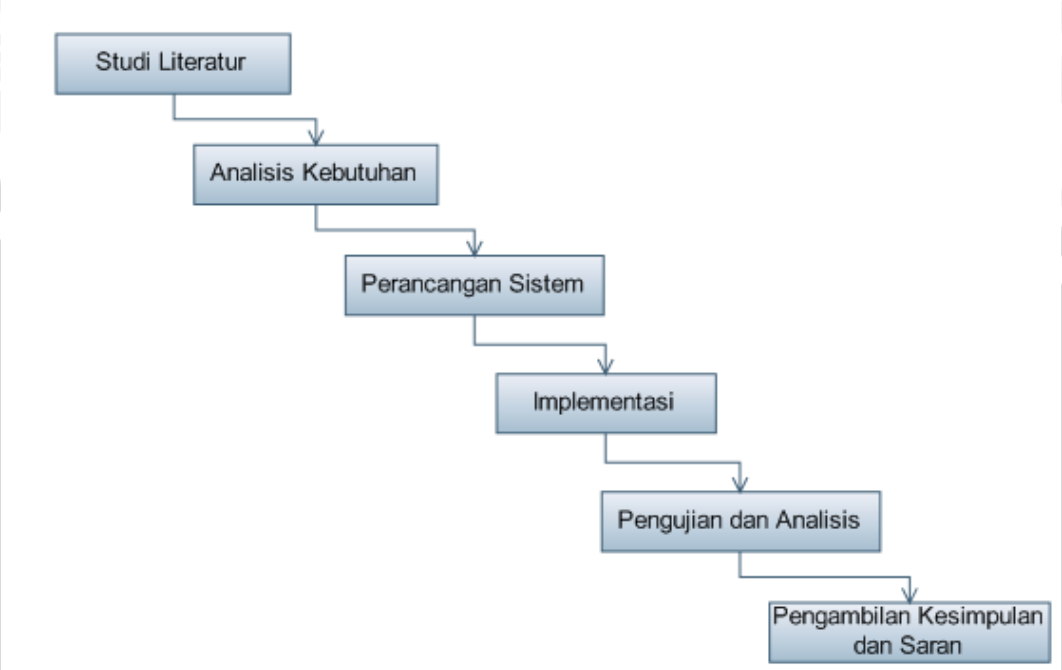
Spoon library merupakan salah satu *library* yang tersebar dan tersedia bebas bagi semua orang. *Library* ini bersifat *open-source* yang berarti *library* ini dapat dipelajari dengan bebas dan kode sumbernya dapat dikembangkan oleh semua orang. *Spoon library* dapat membantu para pengembang untuk menganalisis kode sumber Java. *Library* ini mampu membangun sebuah *metamodel* Java dimana setiap elemen dalam kode sumber seperti *class, method,* atribut, *statement, expression*, dan lain sebagainya dapat diakses dengan baik untuk dibaca maupun dimodifikasi oleh pengguna (Pawlak,dkk. 2006).

2.2.26 Aplikasi *Latent Semantic Analysis (LSA) Boulder*

Aplikasi *Latent Semantic Analysis (LSA) Boulder* adalah sebuah aplikasi yang dibangun oleh sekelompok grup riset di Universitas Colorado. Aplikasi ini dibuat untuk menghitung kesamaan kata maupun makna kata. Pencocokkan dapat dilakukan antar kata, kata dengan dokumen maupun antar dokumen satu sama lain. Pada aplikasi ini metode pencocokan yang digunakan adalah *Latent Semantic Analysis (LSA)* (Landauer,dkk. 1998). Aplikasi *online* ini tersedia pada *http://lsa.colorado.edu*

BAB 3 METODOLOGI PENELITIAN

Pada bab ini dijelaskan metode pengembangan perangkat lunak yang digunakan dalam penelitian ini dan sekaligus langkah-langkah yang akan dilakukan dalam pengerjaan tugas akhir yaitu Studi Literatur, Analisis Kebutuhan, Perancangan Desain dan Sistem, Implementasi, Pengujian Perangkat Lunak dan Analisis. Kesimpulan dan saran disertakan sebagai catatan atas aplikasi dan kemungkinan arah pengembangan aplikasi selanjutnya. Gambar 3.1 menunjukkan tahapan penilitian secara umum.



Gambar 3.1 Tahapan Penelitian

Metode Pengembangan

Agar penelitian yang dilakukan lebih terarah maka akan digunakan suatu tahapan penelitian. Berikut rincian penjelasan tahapan penelitian seperti yang tergambar pada gambar 3.1:

3.1 Studi Literatur

Pada tahap ini adalah melakukan studi literatur untuk memahami teori atau konsep yang akan dilakukan sebelum dilakukannya perancangan dan implementasi sistem. Studi literatur ini telah dijelaskan pada bab 2 yang menjelaskan kajian pustaka dan dasar teori tentang konsep dasar Rekayasa Perangkat Lunak (RPL), konsep dasar pendekatan berorientasi objek, konsep dasar *Object Oriented System Analysis and Design (OOAD)*, konsep dasar *Object Oriented Programming (OOP)*, konsep kebutuhan dalam tahapan pengembangan perangkat lunak, konsep dasar pemodelan, konsep dasar diagram *Unified Modelling Languange* (UML), konsep *use case diagram,* konsep *use case*

scenario, konsep *activity diagram,* konsep *state machine diagram,* konsep *sequence diagram,* konsep *class diagram,* konsep pengujian perangkat lunak seperti pengujian *white box* dan *black box*, konsep pengujian akurasi hasil, konsep dasar *System Development Life Cycle* (SDLC), konsep dasar SDLC *waterfall,* konsep dasar *class*, konsep dasar *method*, konsep dasar *library spoon*, konsep dasar *Latent Semantic Indexing (LSI)*, konsep dasar *library JAMA Matrix Package*, konsep dasar kopling, konsep dasar *conceptual coupling* dalam perangkat lunak, konsep dasar *information retrieval,* dan konsep dasar *coupling metrics* seperti *Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Methods and a Class (CSMC), Conceptual Similarity between Two Classes (CSBC), Conceptual Coupling of a Class (CoCC).*

3.2 Analisis Kebutuhan

Analisis kebutuhan merupakan suatu tahapan yang diawali dengan menjabarkan gambaran umum sistem yang dibangun. Fase analisis kebutuhan merupakan fase terpenting dalam tahapan pengembangan sistem agar perangkat lunak dapat memenuhi keinginan pengguna dan keberhasilan sistem.

Konsep analisis kebutuhan pada penelitian ini ialah konsep OOAD dimana pada tahap ini kebutuhan sistem berupa kode program. Kode program inilah yang akan di parsing dan dipetakan per elemen menggunakan *spoon library.*

Secara garis besar, spesifikasi kebutuhan dalam penelitian ini adalah sebagai berikut :

1) Spesifikasi kebutuhan perangkat keras (*hardware)*
   - ✓ Sebuah komputer PC
2) Spesifikasi kebutuhan perangkat lunak (*software)*
   - ✓ *Spoon library* untuk melakukan parsing kode program dan pemetaan kode program per elemen-elemen di dalamnya. *Library* ini berfungsi untuk mengambil *method-method* yang ada di dalam *class*.
   - ✓ *Jama Matrix Package library* untuk membangun matriks dalam perhitungan *Latent Semantic Indexing (LSI)*. Matrik ini berguna sebagai tempat untuk mengolah informasi semantik dari method-method tersebut sehingga didapatkan nilai kemiripan antar *class* dan *method* tersebut.
   - ✓ Netbeans IDE 8.0 untuk mengimplementasikan bahasa pemrograman java dan menjalankan sistem kakas bantu pada penelitian ini.
   - ✓ *Java Development Kit (JDE) 8* sebagai perangkat lunak untuk melakukan proses kompilasi dari kode java ke bytecode yang dapat dimengerti dan dapat dijalankan oleh *Java Runtime Environment (JRE).*
   - ✓ *Java Runtime Environment (JRE)* untuk menjalankan aplikasi yang dibangun dengan menggunakan bahasa pemrograman *Java.*

3) Spesifikasi kebutuhan data untuk penelitian

- ✓ Kumpulan data mentah berupa kode program yang diimplementasikan menggunakan pendekatan berorientasi objek dan berbahasa pemrograman java.
- ✓ *Data corpus*. *Data corpus* pada penelitian ini berupa sekumpulan data method beserta *body method* namun komentar tidak termasuk di dalamnya.
- ✓ Data yang digunakan dalam penelitian ini adalah berupa data dari tugas akhir salah satu mata kuliah. Data tersebut terdiri dari 3 *class* dengan 24 *method*.

3.3 Perancangan Sistem

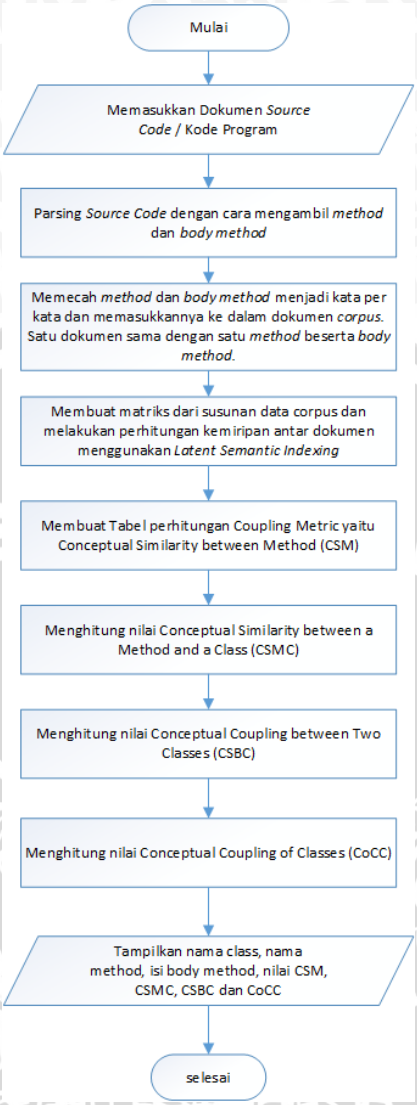
Tahap perancangan dilakukan setelah tahap analisis kebutuhan. Selanjutnya, tahap perancangan dibangun menggunakan konsep *Object Oriented Analysis and Design (OOAD)* menggunakan pemodelan *Unified Modelling Languange (UML).* Dalam perancangan ini dijelaskan bagaimana dokumen kode program menjadi masukkan bagi sistem dan bagaimana kode program di parsing di petakan *method-method* yang dimiliki masing-masing *class* dengan bantuan *spoon library*.

Kakas bantu perhitungan nilai kopling ini mengolah data kode program dengan mengambil nama *class* dan *method-method* yang dimiliki seluruh *class.* Method-method tersebut dikumpulkan di sebuah data *corpus*. Data *corpus* berisi dokumen hasil *parsing* kode program. Hasil parsing tersebut berupa dokumen dimana per satu dokumen berisi satu *method* beserta *body method*. Hasil parsing tersebut bukan lagi satu bagian method utuh seperti yang ada di kode program pada umumnya, melainkan sudah menjadi *term-term* atau kata per kata yang sudah dipisah. Dokumen per *method* ini disebut dengan data training. Data training berfungsi sebagai data pembanding jika ada data testing yang dimasukkan oleh pengguna. Kata per kata pada dokumen ini berguna untuk memudahkan pembacaan dokumen pada saat perhitungan kemiripan antar dokumen. Setiap kata beserta judul dokumen (dalam hal ini kata berarti kata per kata di body method dan judul dokumen berarti nama method yang bersangkutan) dimasukkan ke dalam sebuah matriks. Matriks ini berguna sebagai masukkan untuk memulai perhitungan *Latent Semantic Indexing (LSI).*

Langkah selanjutnya, perhitungan *Latent Semantic Indexing (LSI)* dimulai. Langkah-langkah perhitungan *Latent Semantic Indexing (LSI)* secara umum diawali dengan menghitung transpose matriks, menghitung determinan dari matriks yang sudah di transpose, mencari nilai *eigen value,* menghitung nilai *singular value,* menghitung eigen vektor, normalisasi *eigen vektor,* membuat matriks V, membuat matriks U, mereduksi dimensi SVD dengan mengurangi ukuran dimensi sebanyak *k* (dalam penelitian ini *k* = 5), menghitung *cosine similarity* (tingkat kemiripan antar dokumen), memasukkan *query* oleh

pengguna, query dicocokkan dengan dokumen *method* yang ada dan langkah terakhir adalah perankingan kemiripan antara *query* dengan dokumen berdasarkan nilai *cosine similarity*. Perangkingan diurutkan mulai dari nilai cosine similarity tertinggi hingga terendah.

Langkah selanjutnya masuk ke perhitungan nilai kopling menggunakan *coupling metrics Conceptual Coupling of a Class (CoCC).* Proses perhitungan *Conceptual Coupling of a Class (CoCC)* dimulai dari menghitung nilai *Conceptual Similarity between Methods (CSM)* berdasarkan *cosine similarity query* terhadap dokumen yang sudah didapatkan pada perhitungan LSI sebelumnya. Nilai *Conceptual Similarity between Methods (CSM)* ditampilkan dalam bentuk tabel seperti yang tertera pada tabel 4.1. Kemudian, dari nilai *Conceptual Similarity between Methods (CSM)* yang sudah diketahui langkah selanjutnya ialah menghitung nilai *Conceptual Similarity between a Methods and a Class (CSMC), Conceptual Similarity between Two Classes (CSBC)* dan *Conceptual Coupling of a Class (CoCC).* Hasil akhir program ini menghasilkan nilai *Conceptual Coupling of a Class (CoCC)*. Berikut gambaran proses keseluruhan pembuatan sistem pada penelitian ini yang dijelaskan pada gambar 3.2:



Gambar 3.2 Diagram Alur Sistem

Pada diagram diatas dijelaskan bahwa alur pembuatan sistem dimulai dari memasukkan *source code* atau kode program, parsing kode program, perhitungan kemiripan dokumen menggunakan *Latent Semantic Indexing (LSI)* dan diakhiri dengan perhitungan *coupling metric* menggunakan empat parameter yaitu *Conceptual Similarity between Method (CSM), Conceptual Similarity between Method and a Class (CSMC), Conceptual Coupling between Two Classes (CSBC) dan Conceptual Coupling of Classes (CoCC).*

3.4 Implementasi

Pada tahap ini, seluruh analisis dan perancangan sistem akan diimplementasikan. Implementasi aplikasi diawali dengan penjabaran spesifikasi lingkungan perancangan aplikasi. Selanjutnya mengimpelementasikan hasil rancangan *class diagram* ke bahasa pemrograman java. Tahap implementasi

sistem merupakan tahap dilakukannya pembuatan sistem yang dibuat sesuai dengan perancangan sistem yang telah dibuat sebelumnya. Proses pembuatan sistem pada tahap ini, antara lain:

1. Membuat sistem "Kakas bantu perhitungan nilai kopling menggunakan coupling metric dan *Latent Semantic Indexing (LSI)*."
2. Memasukkan Dokumen Source Code / Kode Program ke dalam sistem kakas bantu perhitungan nilai kopling.
3. Melakukan parsing terhadap kode program yang dimasukkan ke dalam sistem menggunakan java dan *spoon library*.
4. Mengimplementasikan *Latent Semantic Indexing* menggunakan *java* dan *Jama Matrix library*.
5. Mengimplementasikan *coupling metric* menggunakan *java*.
6. Pada tahap akhir dilakukan proses pengujian aplikasi dari sistem menggunakan metode *white box, black box* dan pengujian akurasi nilai perhitungan.

3.5 Pengujian dan Analisis

Pengujian aplikasi dilakukan untuk mengetahui apakah kinerja sistem atau *tools* telah sesuai dengan spesifikasi kebutuhan. Pengujian akan dilakukan dengan menggunakan metode *black-box testing, white-box testing* dan pengujian akurasi nilai perhitungan. Pengujian dengan metode *black-box testing* dilakukan menggunakan pengujian validasi. Pengujian dengan metode *white-box testing* dilakukan menggunakan pengujian didasarkan pada membuat *test cases* dengan melihat *source code* untuk mencari adanya kesalahan pada program. Penelitian ini juga melakukan pengujian terhadap akurasi perhitungan metrik. Akurasi perhitungan metrik dilakukan dengan cara membandingkan selisih perhitungan pada sistem dengan perhitungan pada sebuah aplikasi pembanding perhitungan LSI yaitu *lsa.colorado.edu*. Sistem perhitungan LSI pada *lsa.colorado.edu* menggunakan cara seperti memasukkan dokumen yang ingin dicocokkan secara manual, memilih topik yang ingin digunakan dan memilih kategori pencocokkan apakah antar kata, kata dengan dokumen atau antar dokumen. Aplikasi ini akan secara otomatis menampilkan matriks hasil perhitungan pencocokkan data yang dimasukkan. Sedikit berbeda dengan sistem pada aplikasi pembanding, sistem kakas bantu pada penelitian ini menggunakan cara perhitungan nilai kecocokan menggunakan *Latent Semantic Indexing* (LSI) dengan cara memasukkan dokumen kode program dan sistem kakas bantu akan otomatis menghitung mulai dari *parsing method*, mencocokkan dokumen hingga menghitung nilai kopling. Hal yang dibandingkan dalam menguji kedua sistem ini adalah pada perhitungan LSI di aplikasi pembanding yang dibandingkan dengan nilai *Conceptual Similarity between Method* (CSM) pada sistem kakas bantu. Nilai *Conceptual Similarity between Method* (CSM) ini juga nilai LSI pada sistem kakas bantu. Pengujian dilakukan dengan cara menangkap nilai *Latent Semantic Indexing (LSI)* dan nilai *Conceptual Similarity between Method* (CSM) lalu dibandingkan dan dicari selisih antar kedua nilai dari sistem yang berbeda

tersebut. Setelah mengetahui selisihnya, lalu nilai toleransi diberikan untuk mengetahui akurasi sistem. Nilai toleransi tersebut merupakan nilai yang dapat dan perlu diteliti lebih lanjut.

3.6 Pengambilan Kesimpulan dan Saran

Setelah tahap pengujian, dilakukan analisis terhadap hasil pengujian aplikasi untuk mendapatkan kesimpulan dari aplikasi yang telah dibuat. Pada tahap ini dilakukan pengambilan kesimpulan serta saran untuk mengevaluasi kembali kesalahan pada penelitian ini. Saran yang diperoleh diharapkan mampu menyempurnakan hasil dari penulisan dan dapat menjadi ide-ide serta gagasan dalam penelitian selanjutnya.

BAB 4 ANALISIS KEBUTUHAN

Pada bab ini dijelaskan tentang analisis kebutuhan dari sistem yang akan dibuat. Tahap analisis menjelaskan kebutuhan sistem yang akan dibangun. Tahap analisis ini memiliki pengaruh yang besar terhadap keberhasilan sistem agar sesuai dengan keinginan pengguna. Metode analisis yang digunakan *Object-Oriented Analysis (OOA)* dengan berdasarkan model *Unified Model Languange (UML*) sebagai kebutuhan sistem. Kebutuhan sistem pada penelitian ini adalah kode program karena kode program ini mampu menentukan tingkat kopling suatu perancangan perangkat lunak pada tahap implementasi.

Analisis dibutuhkan untuk memenuhi kebutuhan pengguna. Proses analisis kebutuhan diawali dengan identifikasi aktor, analisis kebutuhan fungsional, pemodelan *use case* diagram, skenario *use case* dan analisis kebutuhan non-fungsional.

4.1 Gambaran Umum Sistem

Gambaran umum sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* dibagi menjadi dua bagian yaitu deskripsi umum sistem dan lingkungan sistem.

4.1.1 Deskripsi Umum Sistem

Sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* ini adalah sistem yang dapat membantu pengembang perangkat lunak khususnya analis sistem untuk melihat tingkat kopling pada rancangan perangkat lunaknya. Data input pada sistem ini ialah kode program. Input dapat berupa maksimal empat dokumen kode program yang berisi maksimal tiga puluh *method*. Sistem ini akan membantu pengembang menghitung nilai kopling secara otomatis melalui kode program. Sehingga pengembang perangkat lunak dapat mengetahui apakah rancangan class diagram yang telah dibuat sebelumnya memiliki nilai kopling seberapa besar.

Alur kerja sistem diawali dengan melakukan parsing kode program yang telah dimasukkan pengguna. Data yang diambil pada kode program yakni berupa nama *class*, nama *method* dan *body method*. *Body method* tersebut dipecah menjadi kata per kata dan dimasukkan ke dalam data *corpus* agar nantinya mempermudah saat menghitung kemiripan dokumen. Dokumen disini adalah dokumen yang berisi satu *method* beserta *body method*. Lalu, sistem akan membuat sebuah matriks dari data corpus dan dokumen kode program yang dimasukkan pengguna. Matriks ini berguna untuk mempermudah perhitungan kemiripan antar dokumen menggunakan metode *Latent Semantic Indexing (LSI)*. Langkah terakhir ialah menghitung nilai *coupling metrics*. Output dari sistem ini adalah menampilkan nama *class*, nama *method*, kata per kata dari *body method*, nilai *Conceptual Similarity*

between Method (CSM), nilai Conceptual Similarity between a Method and a Class (CSMC), nilai Conceptual Coupling between Two Classes (CSBC) dan nilai *Conceptual Coupling of Classes (CoCC)*.

4.1.2 Lingkungan Sistem

Sistem perhitungan nilai kopling ini membutuhkan lingkungan untuk berjalan. Lingkungan yang dibutuhkan sistem ini berbasis java sehingga memerlukan *Java Runtime Environment (JRE)* dan *Netbeans IDE* versi 8.0 untuk menjalankannya. Sistem ini juga membutuhkan *Java Development Kit* (JDK) untuk melakukan proses kompilasi. *Java Development Kit* (JDK) yang digunakan minimal versi 8.0.

4.2 Analisis Kebutuhan

Analisis kebutuhan perangkat lunak pada penelitian ini terdiri dari identifikasi aktor, analisis kebutuhan fungsional, pemodelan *use case* diagram, skenario *use case* dan analisis kebutuhan non-fungsional.

4.2.1 Identifikasi Aktor

Tabel 4.1 berikut menjelaskan aktor mana saja yang terlibat dalam interaksi sistem beserta deskripsinya.

Tabel 4.1 Identifikasi Aktor

Aktor	Deskripsi
Pengguna	Pengguna adalah orang yang dapat menggunakan seluruh fitur dalam sistem tanpa terkecuali.

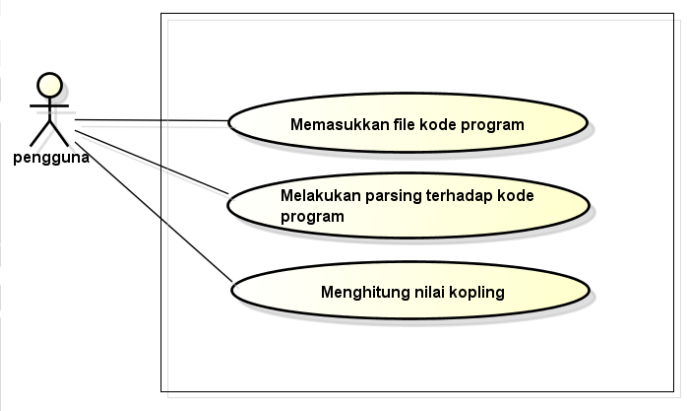
4.2.2 Analisis Kebutuhan Fungsional

Kebutuhan fungsional adalah fitur yang harus ada dalam sistem yang memenuhi bisnis proses dan dapat diterima serta memenuhi kebutuhan pengguna. Kebutuhan fungsional dalam sistem "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*" adalah sebagai berikut:

1) Pengguna dapat memasukkan file kode program ke dalam sistem "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*" **[SKPL_KBPK_001].**
2) Pengguna dapat menekan tombol *run project* pada *text editor Netbeans*. Projek yang dijalankan dalam *Netbeans* tersebut adalah sistem kakas bantu perhitungan nilai kopling. Hal ini berguna untuk mengambil informasi nama *class*, nama *method* dan *body method*. Lalu, *body method* di pisahkan kata per kata **[SKPL_KBPK_002].**
3) Pengguna dapat secara otomatis menghitung nilai kopling setelah menjalankan *use case* kedua. Sistem akan secara otomatis menghitung kopling untuk mendapatkan nilai COCC **[SKPL_KBPK_003].**

4.2.3 Pemodelan Use Case Diagram

Use case diagram adalah interaksi atau dialog antara sistem dan aktor yang menyatakan unit fungsi/layanan yang disediakan sistem ke pengguna. Use case merupakan gambaran fungsionalitas dari suatu sistem, sehingga pengguna sistem dapat memahami dengan mudah mengenai kegunaan sistem yang akan dibangun.



Gambar 4.1 *Use Case Diagram*

Penjelasan:

Use case diagram yang pertama antara aktor pengguna yang dapat memasukkan file ke sistem "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*". Kedua, pengguna dapat melakukan parsing terhadap kode program yang dimasukkan dan secara otomatis *use case* ketiga juga dijalankan yaitu pengguna dapat menghitung nilai kopling. *Text editor Netbeans IDE 8.0* diperlukan untuk dapat menjalankan sistem kakas bantu perhitungan nilai *coupling* tersebut dan menjalankan ketiga *use case* pada gambar 4.1.

4.2.4 Use case scenario

Fungsi *use case scenario* ini adalah sebagai penjelas *use case diagram* secara lebih detail.

4.2.4.1 Use case scenario memasukkan file kode program

Tabel 4.2 merupakan use case scenario memasukkan file kode program yang menggambarkan alur dari use case memasukkan file kode program secara lebih detail.

Tabel 4.2 *Use Case Scenario* Memasukkan File Kode Program

Nomor *Use Case*	SKPL_KBPK_001
Nama	Memasukkan file kode program
Tujuan	Pengguna dapat memasukkan file ke dalam sistem.
Deskripsi	Use case ini menjelaskan bahwa sistem membutuhkan masukkan file kode program untuk menjadi acuan perhitungan nilai kopling.
Aktor	Pengguna
Kondisi Awal	*Text editor Netbeans IDE* 8.0 dan sistem perhitungan *coupling* "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*" sudah dibuka.
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
Pengguna meletakkan *kursor* pada *folder* data dan klik kanan pada *folder* data tersebut. Lalu, pengguna membuat *class* baru dengan cara memilih *sub menu new Java class.*	Sistem menampilkan *window new java class.*
Pengguna mengisi nama	Sistem menampilkan halaman kode untuk

klas, lokasi penyimpanan klas tersebut, lokasi folder penyimpanan klas, jika sudah selesai pengguna dapat menekan *button finish.*	menulis kode program.
Pengguna memindahkan isi kode program *Java* yang ingin dihitung nilai koplingnya ke dalam halaman kode program pada *class* yang baru dibuat.	Sistem menampilkan kode program yang ingin dihitung nilai koplingnya.
Alur Alternatif	
Alternatif 1: Jika file yang dimasukkan oleh pengguna tidak valid (file bukan bertipe java).	
Aksi Aktor	**Reaksi Sistem**
	Sistem menampilkan pesan kesalahan kepada pengguna bahwa *"java.lang.NullPointerException"*. Hal ini karena *spoon library* tidak mampu melakukan analisis *class* selain bahasa pemrograman Java.
Kondisi Akhir	File kode program berhasil dimasukkan ke dalam sistem.

4.2.4.2 Use case scenario melakukan parsing terhadap kode program.

Tabel 4.3 merupakan *use case scenario* melakukan *parsing* terhadap kode program yang menggambarkan alur proses *parsing* kode program secara detail.

Tabel 4.3 Use Case Scenario Melakukan *Parsing* terhadap Kode Program

Nomor *Use Case*	SKPL_KBPK_002
Nama	Melakukan *parsing* terhadap kode program
Tujuan	Pengguna dapat melakukan *parsing* terhadap kode program.
Deskripsi	Use case ini menjelaskan tentang bagaimana pengguna dapat melakukan *parsing* terhadap kode program dengan menekan tombol *run project* pada *Netbeans IDE* dan sistem akan melakukan proses *parsing* kode program tersebut.
Aktor	Pengguna
Kondisi Awal	*Text editor Netbeans IDE* 8.0 dan sistem perhitungan *coupling* "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*" sudah dibuka serta file kode program telah dimasukkan ke dalam sistem perhitungan *coupling*.
Skenario Utama	

Aksi Aktor	**Reaksi Sistem**
Pengguna menekan tombol *run project* yang disediakan oleh *Netbeans* pada sistem kakas bantu perhitungan nilai kopling untuk melakukan parsing terhadap *file* kode program yang telah dimasukkan ke dalam sistem.	Sistem melakukan parsing kode program berdasarkan file kode program yang telah dimasukkan ke dalam sistem dengan cara mengambil nama *class,* nama *method* dan *body method* yang dimiliki oleh *class* dalam kode program tersebut. Lalu, sistem akan otomatis memecah *body method* menjadi kata per kata dan memasukkannya ke dalam data *corpus*.
Alur Alternatif	
Aksi Aktor	**Reaksi Sistem**
Kondisi Akhir	Nama *class,* nama *method* dan *body method* berhasil didapatkan. Data *corpus* berhasil dibangun.

4.2.4.3 Use case scenario menghitung nilai kopling

Tabel 4.4 merupakan *use case scenario* menghitung nilai kopling dimana pada *use case* ini terdapat perhitungan nilai masing-masing *coupling metric. Scenario use case* ketiga pada tabel 4.4 secara otomatis dijalankan setelah *scenario use case* kedua dijalankan.

Tabel 4.4 *Use Case Scenario* Menghitung Nilai Kopling

Nomor *use case*	SKPL_KBPK_003
Nama	Menghitung nilai kopling.
Tujuan	Pengguna dapat menghitung nilai kopling.
Deskripsi	*Use case* ini menjelaskan bahwa sistem menghitung nilai kopling menggunakan empat parameter yaitu CSM, CSMC, CSBC dan CoCC.
Aktor	Pengguna
Kondisi Awal	*Scenario use case* kedua pada tabel 4.3 telah dijalankan yang berarti *method* dan *kata-kata dalam body method* telah di *parsing* dan dibobotkan menggunakan metode *Latent Semantic Indexing (LSI).*
Skenario Utama	
Aksi Aktor	**Reaksi Sistem**
	Sistem melakukan perhitungan nilai tiap-tiap *coupling metric* yakni nilai CSM, CSMC, CSBC dan CoCC.
Alur Alternatif	
Aksi Aktor	**Reaksi Sistem**
Kondisi Akhir	Nilai kopling berhasil ditampilkan oleh sistem.

Tidak ada aksi yang dilakukan oleh aktor dalam *scenario use case* pada tabel 4.4 diatas. Hal itu dikarenakan *scenario use case* ketiga pada tabel 4.4 diatas secara otomatis dijalankan setelah *scenario use case* kedua pada tabel 4.3 dijalankan. Hal ini berarti ketika pengguna menjalankan sistem kakas bantu perhitungan nilai kopling ini dengan cara menekan tombol *run project* pada *text editor Netbeans* maka *scenario use case* kedua dan ketiga secara otomatis dijalankan.

4.2.5 Daftar Kebutuhan Non Fungsional

Kebutuhan Non Fungsional sistem merupakan deskripsi dari fitur-fitur, karakteristik dan batasan-batasan yang lain yang mendefinisikan sistem.Berikut

adalah kebutuhan non fungsional yang dimiliki dalam sistem "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics*":

Tabel 4.5 Kebutuhan Non-Fungsional

Nomor SRS_NF	Parameter	Deskripsi Kebutuhan
SRS_NF001	Tingkat kemiripan	Sistem harus memiliki tingkat kemiripan hasil perhitungan nilai *Conceptual Coupling between Method (CSM)* yang dibandingkan dengan dengan hasil perhitungan di *lsa.colorado.edu* minimal tingkat kemiripan tersebut sebesar 50%.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

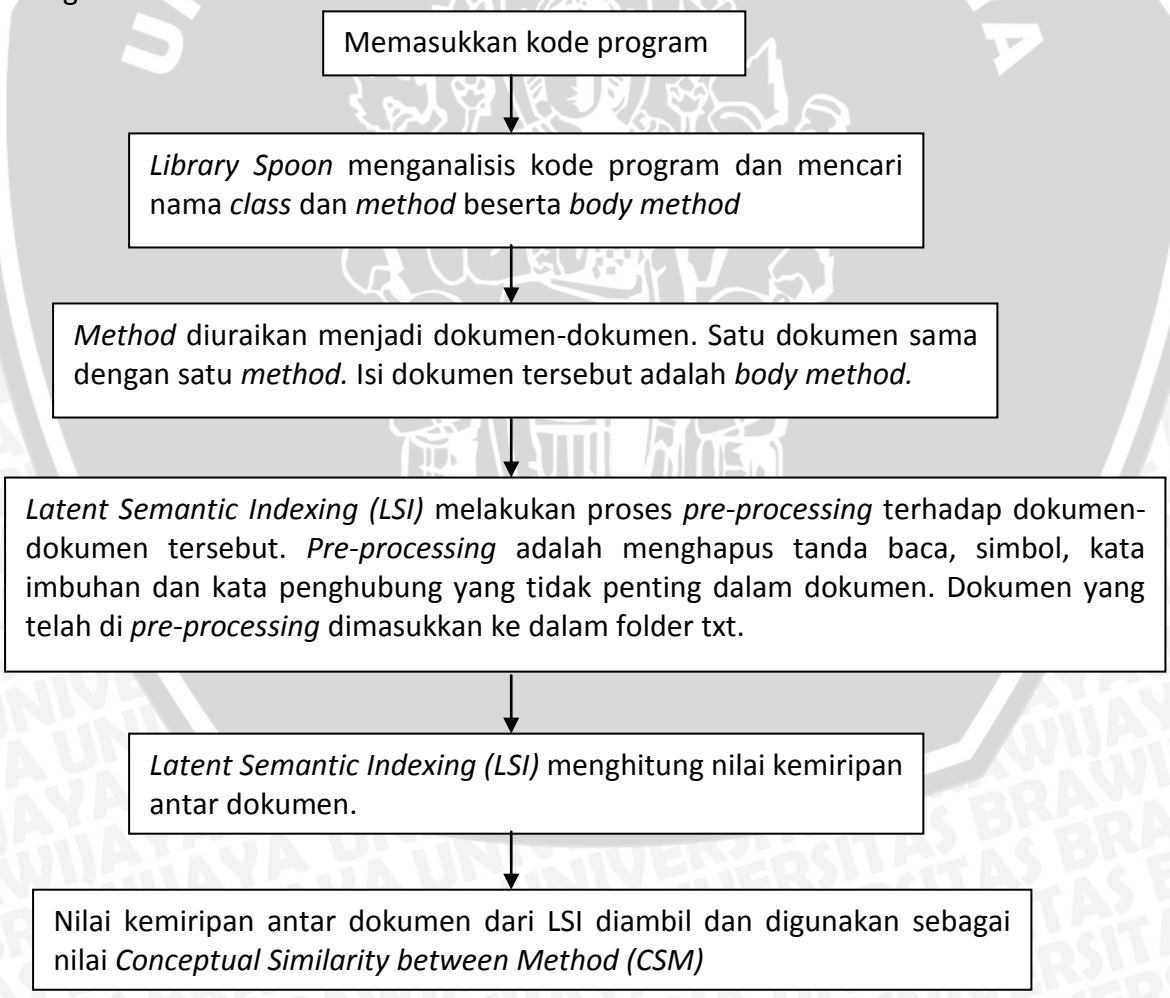
Bab ini membahas tentang tahapan perancangan dan implementasi dalam pembuatan sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* dimana perancangan sistem dilakukan berdasarkan analisis kebutuhan sebelumnya sedangkan proses pembuatan sistem berdasarkan dari analisis kebutuhan dan perancangan sistem yang telah dilakukan sebelumnya.

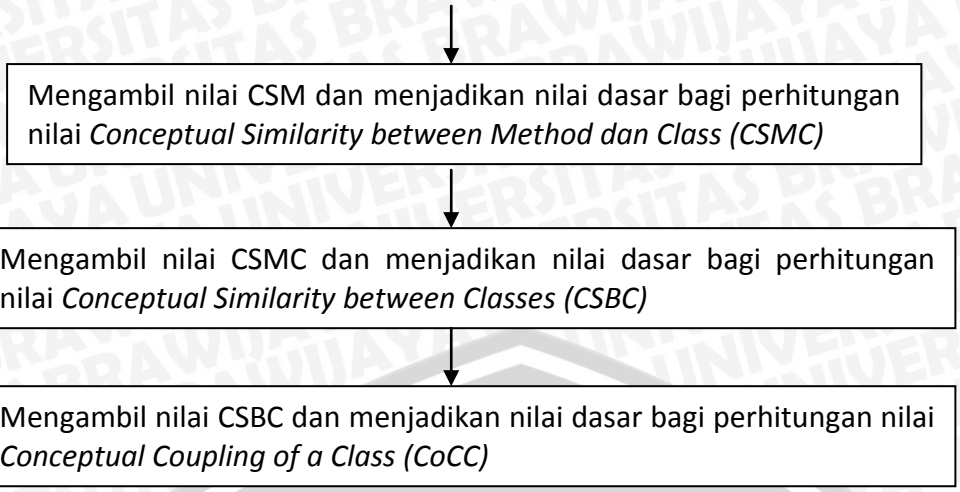
5.1 Perancangan

Tahapan perancangan perangkat lunak dalam penelitian ini terdiri dari perancangan arsitektur, perancangan komponen dan perancangan *user interface.*

5.1.1 Perancangan Detil Sistem

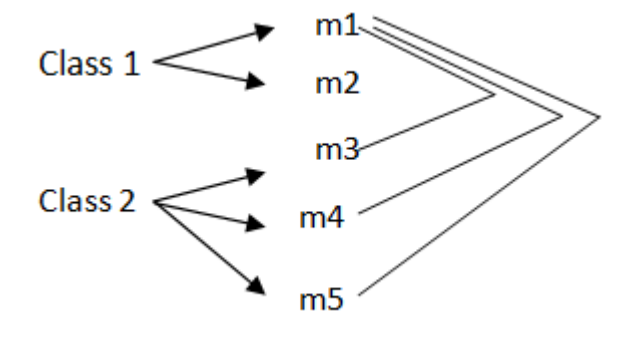
Berikut perancangan sistem secara detil yang akan digambarkan pada gambar 5.7:





Gambar 5.7 Perancangan Detil Sistem

Perhitungan kopling dalam sistem kakas bantu ini ilustrasinya pada gambar 5.8 sebagai berikut:



Gambar 5.8 Ilustrasi Perhitungan Kopling

Parameter yang dihitung pertama kali adalah menghitung nilai kemiripan antar *method* atau yang disebut dengan *Conceptual Similarity between Methods (CSM).* Seperti ilustrasi yang digambarkan pada gambar 5.8 dimana untuk menghitung CSM maka *method* 1 (m1) pada *class 1* dibandingkan dengan *method 3* (m3) pada *class 2*. Lalu, *method* 1 (m1) dibandingkan dengan *method 4* (m4). Lalu, *method 1* (m1) dibandingkan dengan *method 5* (m5). Begitu pula untuk *method 2* (m2) dibandingkan dengan *method 3* (m3) dan seterusnya.

Parameter kedua ialah menghitung nilai kemiripan antar *method* dengan *class* atau yang disebut dengan C*onceptual Similarity between a Methods and a Class (CSMC).* Dimana nilai kemiripan antar *method* yang didapat dari nilai CSM ditotalkan lalu dibagi dengan jumlah *method* pada *class 2.* Untuk menghitung nilai kemiripan antara *method 1* dengan *class 2* maka nilai CSM antara *method 1* dengan masing-masing *method* yang ada pada *class 2* dijumlahkan dan dibagi dengan total *method* yang dimiliki oleh *class 2* yaitu sebanyak 3 *method.*

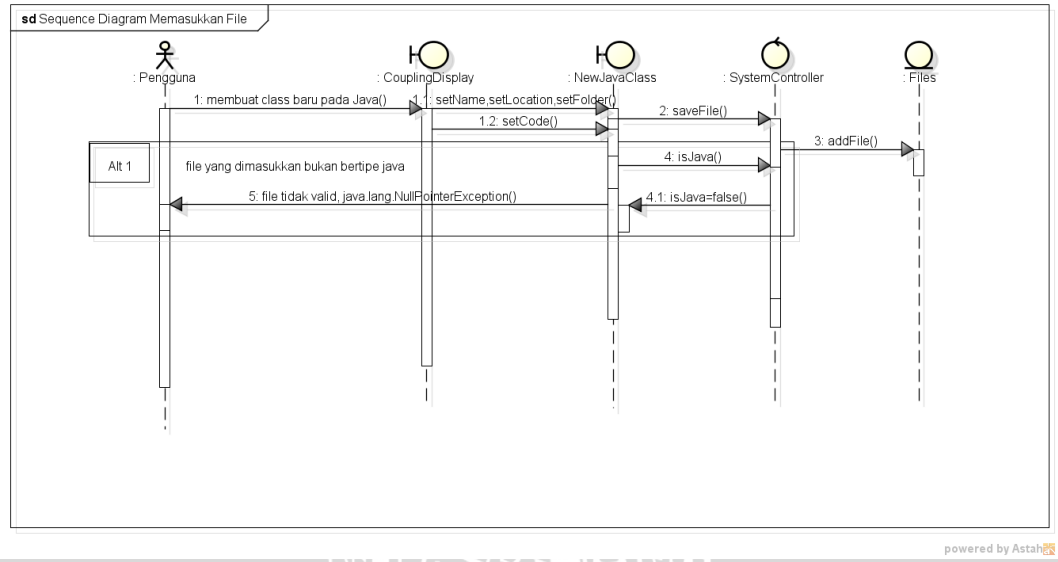
Parameter ketiga ialah menghitung nilai kemiripan antara dua *class* atau yang disebut dengan C*onceptual Similarity between Two Classes (CSBC).* Untuk menghitung nilai CSBC dibutuhkan nilai CSMC pada parameter sebelumnya. Dimana cara perhitugan dari CSBC adalah nilai CSMC antara *method 1* dengan *class 2* ditotalkan lalu dibagi dengan jumlah *method* yang dimiliki oleh *class 1.*

Parameter keempat dan terakhir ialah menghitung nilai kopling yang dimiliki masing-masing *class* atau yang disebut dengan *Conceptual Coupling of a Class (CoCC).* Misalnya, untuk menghitung nilai kopling yang dimiliki oleh *class 1* maka total nilai CSBC antara *class 1* terhadap *class 2* dijumlahkan lalu dibagi dengan total *class* pada sistem tersebut dikurangi 1.

5.1.2 Perancangan arsitektur

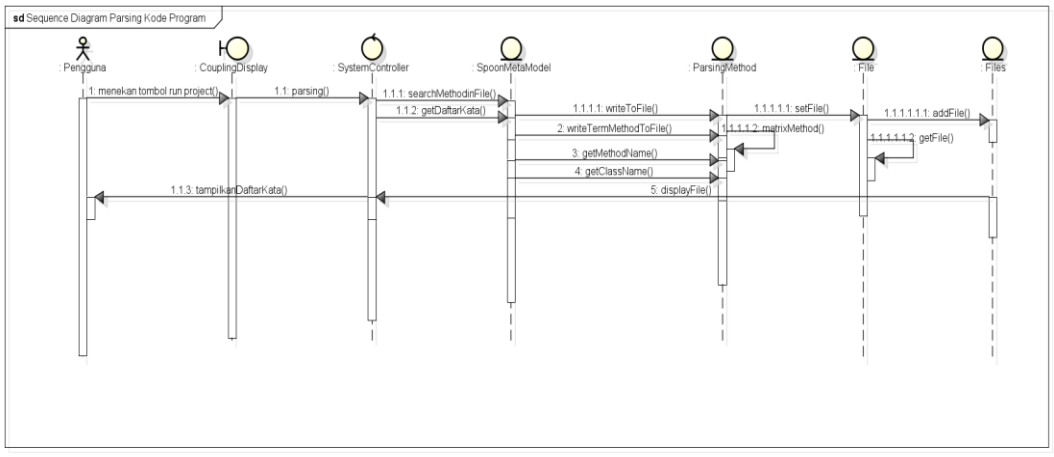
Perancangan arsitektur adalah tahap pertama pada perancangan perangkat lunak. Dari perancangan arsitektur tersebut diperoleh *output* berupa *class diagram* yang berdasarkan dari identifikasi objek menggunakan *sequence diagram.*

5.1.1.1 Pemodelan Sequence Diagram Memasukkan File Kode Program



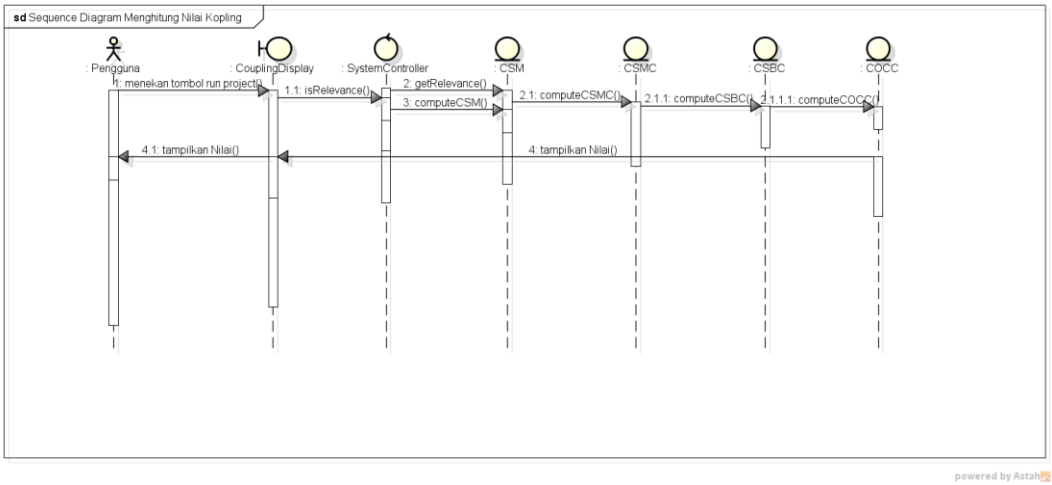
Gambar 5.1 Sequence Diagram Memasukkan *File* Kode Program

5.1.2.1 Pemodelan Sequence Diagram Melakukan Parsing terhadap Kode Program.



Gambar 5.2 Sequence Diagram Melakukan *Parsing* terhadap Kode Program.

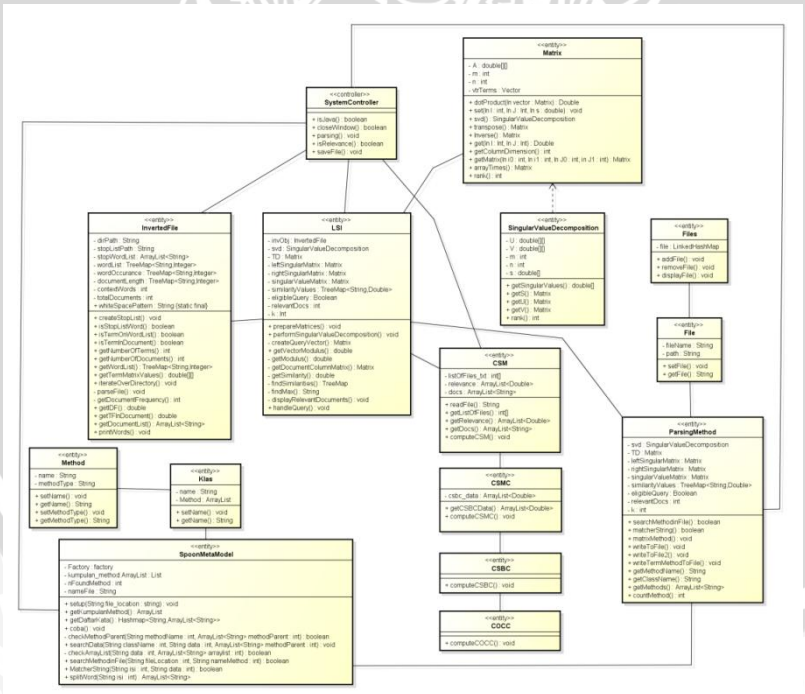
5.1.2.2 Pemodelan Sequence Diagram Menghitung Nilai Kopling



Gambar 5.3 Sequence Diagram Menghitung Nilai Kopling

5.1.2.3 Pemodelan Class Diagram

Class diagram perancangan kakas bantu perhitungan nilai kopling ditunjukkan pada Gambar 5.4 atau untuk lebih jelasnya *class diagram* dapat dilihat pada lampiran 2.



Gambar 5.4 *Class Diagram* Kakas Bantu Perhitungan Nilai Kopling

5.1.3 Perancangan Komponen

Perancangan komponen bertujuan sebagai sarana penjelas *atribut* dan algoritma *method* dalam sebuah *class* yang telah dimodelkan sebelumnya dalam *class diagram*. Dalam penelitian ini, perancangan komponen diambil dari tiga *sample class*, yaitu *class SpoonMetaModel, class LSI*, *class ParsingMethod, class CSM, class CSMC, class CSBC* dan *class CoCC.*

5.1.3.1 Class SpoonMetaModel

Atribut yang digunakan dalam class *SpoonMetaModel* sebanyak empat atribut seperti yang tertera pada tabel 5.1.

Tabel 5.1 Atribut *Class SpoonMetaModel*

No	Nama Atribut	Tipe Data	Modifier
1	factory	Factory	public
2	listOfMethods	ArrayList<String>	public
3	nFoundMethod	int	private
4	nameFile	String	public

Algoritma *Method getMethodFromClasses*

```
PROCEDURE getPackage
    getListofClass using CtType
     getSimpleName.toString()
     listOfWords=new HashMap<>();
     Object[] listMethod=get.getMethod().toArray();
FOR j<listMethod.length ; j++
     CtMethod getMethod=(CtMethod) listMethod[j]
     print getMethod.toString();
     ArrayList<String> words=splitWord(getMethod.toString());
FOR k<words.size ; k++
     string=words.get(k);
     print kata;
     listOfWords.put(getMethod.getSimpleName(), words);
     listOfMethods.add(getMethod.getSimpleName());
```

Algoritma Method searchData

```
PROCEDURE searchData(String className, String data, ArrayList<String>    methodParent)
     CtType selectClass=getType(className);
     getMethods().toArray();
```

```
FOR i<listMethod.length ; i++
    CtMethod get=listMethod[i];
```

5.1.3.2 Class ParsingMethod

Terdapat sembilan atribut yang dideklarasikan dalam class *ParsingMethod,* antara lain seperti tertera pada tabel 5.2

Tabel 5.2 Atribut *Class ParsingMethod*

No	Nama Atribut	Tipe Data	Modifier
1	svd	SingularValueDecomposition	Public
2	TD	Matrix	Public
3	leftSingularMatrix	Matrix	Public
4	rightSingularMatrix	Matrix	Public
5	singularValueMatrix	Matrix	Public
6	eligibleQuery	Boolean	Public
7	k	Int	Public
8	relevantDocs	Int	Public
9	similarityValues	TreeMap<String, Double>	Public

Algoritma *Method searchMethodinFile*

```
FUNCTION searchMethodinFile(String fileLocation, String nameMethod)
TRY
FOR readAllLines(path.get(fileLocation))
IF matcherString(line, nameMethod)
        return true;
CATCH IOException ex
        ParsingMethod.class.getName();
        return false;
```

Algoritma Method matcherString

```
FUNCTION matcherString(String isi, String data)
    tokens=isi.split("[\\s']");
FOR s:tokens
    temp="";
    list=s.toCharArray();
FOR i<list.length ; i++
    c=list[i];
    code=(int) c;
```

```
IF code between a until z
   and code  between A until Z
      and code between 0 until 9
THEN temp +=c;
TRY patern.compile(data)
        myPattern.matcher(temp)
IF matcher.find() THEN
        return =true;
CATCH except data temp;
ELSE
        return false;
```

Algoritma *Method writeToFile2*

```
PROCEDURE writeToFile2(String filename, ArrayList<String> method, HashMap<String,
        ArrayList<String>> listOfWords)
        listOfWords.keySet
        listOfWords.values().toString()
        listOfWords.forEach((key,value))->Paths.get("txt\\" +filename+"__"+key+".txt")
TRY files.write()
CATCH Exception
        ParsingMethod.class.getName()
```

Algoritma *Method writeTermMethodToFile*

```
PROCEDURE writeTermMethodToFile(ArrayList<String> words, ArrayList<String> listOfMethods,
        ArrayList<String> filename)
FOR i<filename.size(); i++
        List<String> lines_words = new ArrayList<>();
        get Path;
        write File;
FOR j<words.size(); j++
IF searchMethodinFile
        Add words to lines_words;
ENDIF
ENDFOR
        Write file;
ENDFOR
END
```

Algoritma Method getMethods

```
FUNCTION getMethods(File dir, String c)
                foundFiles=dir.listFiles;
        FUNCTION accept(File dir, String name)
                        return name;
```

```
        END
methods=new ArrayList;
FOR file is found
        add methods;
        get method name;
ENDFOR
        return methods;
END
```

Algoritma Method countMethod

```
FUNCTION countMethod(File dir, String c)
                foundFiles=dir.listFiles;
        FUNCTION accept(File dir, String name)
                        return name;
END
        i=0;
FOR file is found
        i++;
ENDFOR
        return i;
END
```

5.1.3.3 Class LSI

Terdapat sepuluh atribut yang digunakan dalam class LSI yaitu seperti yang tertera pada tabel 5.3.

Tabel 5.3 Atribut Class LSI

No	Nama Atribut	Tipe Data	Modifier
1	svd	SingularValueDecomposition	Public
2	TD	Matrix	Public
3	leftSingularMatrix	Matrix	Public
4	rightSingularMatrix	Matrix	Public
5	singularValueMatrix	Matrix	Public
6	eligibleQuery	Boolean	Public
7	k	Int	Public
8	relevantDocs	Int	Public
9	similarityValues	TreeMap<String, Double>	Public
10	invObj	InvertedFile	Public

5.1.3.4 Class CSM

Terdapat tiga atribut yang digunakan dalam class CSM yaitu seperti yang tertera pada tabel 5.4.

Tabel 5.4 Atribut Class CSM

No	Nama Atribut	Tipe Data	Modifier
1	ListOfFiles_txt	Array File	Public
2	relevance	ArrayList<Double>	Public
3	docs	ArrayList<String>	Public

Algoritma Method computeCSM

```
PROCEDURE computeCSM(LSI lsiObj, File folder_temp)
        ArrayList<String> docs_temp = new ArrayList<>();
        listOfFiles_txt=folder_temp.listFiles();
FOR i<listOfFiles_txt; i++
        print listOfFiles_txt[i].getName;
        query=readFile(listOfFiles_txt[i].getAbsolutePath();
        lsiObj.handleQuery(query);
FOR j<lsi.OBJ.getDocs().size(); j++
        d1= listOfFiles_txt[i].getName();
        d2=lsiObj.getDocs().get(j);
        s=d1+"<>"+d2;
        docs.add(s);
ENDFOR
ENDFOR
        add all relevance;
        add all docs_temp;
END
```

5.1.3.5 Class CSMC

Terdapat satu atribut yang digunakan dalam class CSMC yaitu seperti yang tertera pada tabel 5.5.

Tabel 5.5 Atribut Class CSMC

No	Nama Atribut	Tipe Data	Modifier
1	csbc_data	ArrayList<Double>	Public

Algoritma Method computeCSMC

```
PROCEDURE computeCSMC(ParsingMethod cpl, CSM csm, File folder_temp,
        ArrayList<Double> relevance, ArrayList<String> docs)
        Int counter=0;
        Double csbc=0;
        Csbc_data=new ArrayList<>();
        File[] listOfFiles_txt=csm.getListOfFiles();
FOR int i=0; i<listOfFiles_txt.length;i++
        String m = cpl.getMethodName(listOfFiles_txt[i].getName());
       String c = cpl.getClassName(listOfFiles_txt[i].getName());
       int jmlMethod = cpl.countMethod(folder_temp, c);
FOR int j = 0; j < listOfFiles_txt.length; j++
        int temp = 0;
      String c2 = cpl.getClassName(listOfFiles_txt[j].getName());
IF !c.equals(c2)
        THEN System.out.println("m:" + m + " c: " + c2);
        getClassName(listOfFiles_txt[j].getName()));
      double tot = 0;
     ArrayList<String> methods = cpl.getMethods(folder_temp, c2);
FOR int k = 0; k < methods.size(); k++
        String m2 = cpl.getMethodName(methods.get(k));
        System.out.println(relevance.get(docs.indexOf(listOfFiles_txt[i].getName() +          " <> " +
methods.get(k))) + " " + "m:" + m + " m2: " + m2);
       tot += relevance.get(docs.indexOf(listOfFiles_txt[i].getName() + " <> " +     methods.get(k)));
```

5.1.3.6 Class CSBC

Berikut implementasi dari class CSBC.

Algoritma Method computeCSBC

```
public void computeCSBC(ParsingMethod cpl, File[] listOfFiles,ArrayList<Double> csbc_data) {
      System.out.println("\n\n\n==========\tCSBC\t==========");
      for (int i = 0; i < listOfFiles.length; i++) {
        String c = cpl.getClassName(listOfFiles[i].getName());
        for (int j = 0; j < listOfFiles.length; j++) {
          String c2 = cpl.getClassName(listOfFiles[j].getName());
          if (!c.equals(c2)) {
            System.out.println("CSBC (" + c2 + "," + c + ") = " + csbc_data.get(i));
          }
        }

      }
   }
```

5.1.3.7 Class COCC

Berikut implementasi dari *class* COCC.

Algoritma Method computeCOCC

```
public void computeCOCC(ParsingMethod cpl, File[] listOfFiles,ArrayList<Double> csbc_data) {
  System.out.println("\n\n\n==========\tCOCC\t==========");
  for (int i = 0; i < listOfFiles.length; i++) {
    String c = cpl.getClassName(listOfFiles[i].getName());
    System.out.println("COCC (" + c + ") = " + csbc_data.get(i) / (listOfFiles.length - 1));
  }
}
```

5.2 Implementasi

Tahapan implementasi dilaksanakan setelah melakukan analisis kebutuhan dan perancangan sistem. Proses implementasi tersebut harus berdasarkan analisis kebutuhan dan perancangan sistem yang telah dilakukan sebelumnya. Hal ini berguna agar sistem yang dibangun sesuai dengan kebutuhan pengguna. Pada tahapan implementasi ini dimulai dari penjelasan tentang spesifikasi lingkungan pengembangan sistem, batasan-batasan implementasi, implementasi kode program dan diakhiri dengan implementasi antarmuka sistem.

5.2.1 Spesifikasi Lingkungan Pengembangan Sistem

Sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* ini dibangun dan dikembangkan dalam lingkungan implementasi yang terdiri dari perangkat keras dan perangkat lunak. Berikut merupakan spesifikasi lingkungan pengembangan sistem pada penelitian ini :

5.2.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan dalam pengembangan sistem kakas bantu perhitungan nilai kopling ini menggunakan sebuah komputer dengan spesifikasi perangkat keras seperti yang tertera pada tabel 5.6.

Tabel 5.6 Spesifikasi Perangkat Keras Pengembangan Sistem

Nama Komponen	Spesifikasi
System Model	Asus X455L
Processor	Intel Core i3 4030U 1,9 Ghz
Memory	2 GB RAM
Harddisk	500 GB

5.2.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam pengembangan sistem kakas bantu perhitungan nilai kopling pada penelitian ini akan dijelaskan pada tabel 5.7.

Tabel 5.7 Spesifikasi Perangkat Lunak Pengembangan Sistem

Nama Komponen	Spesifikasi
Operating System	*Windows 8.1 Single Languange 64-bit*
Programming Languange	*Java*
Text Editor	*Netbeans IDE 8.0* dengan *JDK 8*

5.2.1.3 Batasan Implementasi

Berikut adalah batasan implementasi sistem kakas bantu perhitungan nilai kopling :

1. Sistem dapat menerima dokumen yang berisi *source code*.
2. Sistem hanya dapat menghitung nilai kopling berdasarkan nama *method* dan *body method*.
3. Sistem tidak memasukkan komentar pada perhitungan nilai kopling.

5.3 Implementasi Kode Program

Implementasi kode program merupakan implementasi dari perancangan perangkat lunak menggunakan bahasa pemrograman tertentu. Dalam penelitian ini, implementasi dilakukan menggunakan bahasa java.

5.3.1 Implementasi Parsing

Implementasi parsing dilakukan hanya jika pengguna telah memasukkan dokumen *source code* yang ingin dihitung nilai koplingnya. Sistem akan melakukan proses parsing ketika pengguna menekan tombol *run project* yang disediakan oleh *Netbeans* dimana *project* yang dijalankan adalah sistem kakas bantu perhitungan nilai kopling. Sistem melakukan proses parsing berdasarkan dokumen *source code* yang telah dimasukkan oleh pengguna. Implementasi proses parsing adalah sebagai berikut pada kode 5.1**.**

Kode 5.1 Kode parsing *source code*

```
public class ParsingMethod {
.
.
public boolean searchMethodinFile(String fileLocation, String
nameMethod) {
        try {
            for (String line :
Files.readAllLines(Paths.get(fileLocation))) {
                if (MatcherString(line, nameMethod)) {
                    return true;
                }
            }
        } catch (IOException ex) {
```

```
Logger.getLogger(ParsingMethod.class.getName()).log(Level.SEVERE,
null, ex);
        }
return false;
    }

public boolean MatcherString(String isi, String data) {
        String[] tokens = isi.split("[\\s']");
        for (String s : tokens) {

            String temp = "";
            char[] list = s.toCharArray();
            for (int i = 0; i < list.length; i++) {
                char c = list[i];
                int code = (int) c;
                if ((code >= 97 && code < 123) || (code >= 65 &&
code < 91) || (code >= 48 && code < 58)) {
                    temp += c;
                }
            }
            try {
                Pattern myPattern = Pattern.compile(data);
                Matcher matcher = myPattern.matcher(temp);
                if (matcher.find()) {
                    return true;
                }
            } catch (Exception e) {
                System.out.println("except:" + data + ", temp:" +
temp);
            }
        }
        return false;
    }
public   void   writeToFile2(String   filename,   ArrayList<String>
method, HashMap<String, ArrayList<String>> listOfWords) {
        System.out.println("=========");
        System.out.println(listOfWords.keySet());
        System.out.println(listOfWords.values().toString());
        System.out.println("=========");
        listOfWords.forEach((key, value) -> {
            Path file = Paths.get("txt\\" + filename + "___" + key
+ ".txt");
            try {
                Files.write(file, value, Charset.forName("UTF-8"),
StandardOpenOption.CREATE, StandardOpenOption.APPEND);
            } catch (IOException ex) {

Logger.getLogger(ParsingMethod.class.getName()).log(Level.SEVERE,
```

```
null, ex);
            }
        });
    }

public    void    writeTermMethodToFile(ArrayList<String>    words,
ArrayList<String>   listOfMethods,   ArrayList<String>   filename)
throws IOException {
        for (int i = 0; i < filename.size(); i++) {
            List<String> lines_words = new ArrayList<>();
            Path  file  =  Paths.get("txt\\"  +  filename.get(i)  +
".txt");
        Files.write(file,lines_words,Charset.forName("UTF-8"),
StandardOpenOption.TRUNCATE_EXISTING);
            for (int j = 0; j < words.size(); j++) {
                if       (searchMethodinFile("src\\data\\"       +
filename.get(i), words.get(j))) {
                    lines_words.add(words.get(j));
                }
            }
            Files.write(file,lines_words,Charset.forName("UTF-
8"),StandardOpenOption.CREATE, StandardOpenOption.APPEND);
        }
    }

public String getMethodName(String s) {
        String  method  =  s.substring(s.lastIndexOf("___")  +  3,
s.length() - 4);
        return method;
    }

    public String getClassName(String s) {
        String segments[] = s.split("___");
        String className = segments[0];
        return className;
    }

    public ArrayList<String> getMethods(File dir, String c) {
      File[] foundFiles = dir.listFiles(new FilenameFilter() {
            public boolean accept(File dir, String name) {
                return name.startsWith(c);
            }
        });
        ArrayList<String> methods = new ArrayList<>();
        for (File file : foundFiles) {
            methods.add(file.getName());
        }
        return methods;
```

```
    }

  public int countMethod(File dir, String c) {
      File[] foundFiles = dir.listFiles(new FilenameFilter() {
          public boolean accept(File dir, String name) {
              return name.startsWith(c);
          }
      });
      int i = 0;
      for (File file : foundFiles) {
          i++;
      }
      return i;
    }
}
```

Kode diatas menjelaskan proses utama parsing dokumen berdasarkan source code. Proses parsing disini bertujuan untuk mengambil *method* beserta *body method* dari kumpulan *source code* yang dimasukkan ke dalam sistem.

5.3.2 Implementasi Perhitungan Pembobotan Menggunakan Metode Latent Semantic Indexing (LSI)

Implementasi perhitungan pembobotan secara otomatis dieksekusi setelah pengguna melakukan *parsing* terhadap kode program atau dengan kata lain pengguna telah menekan tombol *run project* yang disediakan oleh *Netbeans* dimana *project* yang dijalankan adalah sistem kakas bantu perhitungan nilai kopling. Perhitungan pembobotan dilakukan berdasarkan sekumpulan *method* yang telah berhasil didapatkan dari hasil parsing sebelumnya. Metode perhitungan pembobotan dilakukan berdasarkan penerapan dari analisis kebutuhan pada bab sebelumnya. Implementasi perhitungan pembobotan akan ditunjukkan pada kode 5.2 berikut.

Kode 5.2 Kode perhitungan pembobotan

```
package coupling;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.TreeMap;

import Jama.Matrix;
import Jama.SingularValueDecomposition;

public class LSI {

    InvertedFile invObj;
    SingularValueDecomposition svd;
```

```
    Matrix TD, leftSingularMatrix, rightSingularMatrix,
singularValueMatrix;
    TreeMap<String, Double> similarityValues;
    boolean eligibleQuery = false;
    int relevantDocs = 100;
    int k;

    public LSI(String direktoryPath, String stoplistPath, int
kValue) throws IOException { //membuat object dari class
InvertedFile
        invObj = new InvertedFile(direktoryPath,
stoplistPath);
        invObj.createStopList();
        invObj.iterateOverDirectory(); //mengiterasi semua
dokumen di direktori, dokumen mana yang berasal dari corpus
        k = kValue;
    }

    public LSI(String direktoryPath, String stoplistPath)
throws IOException {
        invObj = new InvertedFile(direktoryPath,
stoplistPath);
        invObj.createStopList();
        invObj.iterateOverDirectory();
        k = -1;
    }

    public void createTermDocumentMatrix() {
        TD = new Matrix(invObj.getTermMatrixValues());
    }

    private void prepareMatrices() {
        for (int i = 0; i <
leftSingularMatrix.getRowDimension(); i++) {
            for (int j = k; j <
leftSingularMatrix.getColumnDimension(); j++) {
                leftSingularMatrix.set(i, j, 0);
            }
        }

        for (int i = k; i <
singularValueMatrix.getRowDimension(); i++) {
            for (int j = k; j <
singularValueMatrix.getColumnDimension(); j++) {
                singularValueMatrix.set(i, j, 0);
            }
        }

        for (int i = 0; i <
rightSingularMatrix.getRowDimension(); i++) {
            for (int j = k; j <
rightSingularMatrix.getColumnDimension(); j++) {
                rightSingularMatrix.set(i, j, 0);
            }
        }
    }

    void performSingularValueDecomposition() {
        svd = new SingularValueDecomposition(TD);
```

```
        leftSingularMatrix = svd.getU();
        singularValueMatrix = svd.getS();
        rightSingularMatrix = svd.getV();

        if (k == -1) {
            k = leftSingularMatrix.getColumnDimension();
        }

        if (k >= 0 && k <
leftSingularMatrix.getColumnDimension()) {
            prepareMatrices();
        }
    }

    private Matrix createQueryVector(String query) {
        List<String> words =
Arrays.asList(query.split(InvertedFile.whiteSpacePattern));
        TreeMap<String, Integer> wordList =
invObj.getWordList();
        Matrix queryVector = new Matrix(wordList.size(), 1);
        int termCounter;

        for (termCounter = 0; termCounter < words.size();
termCounter++) {
            words.set(termCounter,
words.get(termCounter).toLowerCase());
        }

        termCounter = 0;
        for (String term : wordList.keySet()) {
            if (words.contains(term)) {
                queryVector.set(termCounter, 0,
queryVector.get(termCounter, 0) + 1);
                eligibleQuery = true;
            }
            termCounter++;
        }

        return queryVector;
    }

    public double getVectorModulus(Matrix matrix) {
        double product = 0;

        for (int i = 0; i < matrix.getRowDimension(); i++) {
            product += matrix.get(i, 0) * matrix.get(i, 0);
        }

        product = Math.sqrt(product);
        return product;
    }

    private double getModulus(Matrix matrix) {
        return
(getVectorModulus(leftSingularMatrix.transpose().times(matrix
)));
    }

    private Matrix getDocumentColumnMatrix(int columnNumber)
```

```
{
        Matrix columnMatrix = new
Matrix(rightSingularMatrix.getColumnDimension(), 1);
        Matrix resultMatrix;
        for (int i = 0; i < columnMatrix.getRowDimension();
i++) {
            columnMatrix.set(i, 0,
rightSingularMatrix.get(columnNumber, i));
        }

        resultMatrix =
leftSingularMatrix.times(singularValueMatrix);
        resultMatrix = resultMatrix.times(columnMatrix);

        return resultMatrix;
    }

    private double getSimilarity(Matrix queryMatrix, Matrix
documentMatrix) {
        Matrix productMatrix;
        double denominator, similarity;

        productMatrix =
(queryMatrix.transpose()).times(documentMatrix);
        denominator = getModulus(queryMatrix) *
getModulus(documentMatrix);

        similarity = productMatrix.det() / denominator;
        return similarity;
    }

    private TreeMap<String, Double> findSimilarities(Matrix
queryVector) {
        ArrayList<String> documents =
invObj.getDocumentList();
        similarityValues = new TreeMap<String, Double>();
        Matrix documentMatrix;
        int columnNumber = 0;

        for (String document : documents) {
            documentMatrix =
getDocumentColumnMatrix(columnNumber++);
            similarityValues.put(document,
getSimilarity(queryVector, documentMatrix));
        }

        return similarityValues;
    }


    private String findMax(TreeMap<String, Double>
similarityValues) {
        double max = Double.NEGATIVE_INFINITY;
        String maxDocName = null;
        for (String document : similarityValues.keySet()) {
            if (similarityValues.get(document) != Double.NaN
&& similarityValues.get(document) > max) {
                maxDocName = document;
                max = similarityValues.get(document);
```

```
            }
        }

        return maxDocName;
    }

    private void displayRelevantDocuments(TreeMap<String,
Double> similarityValues, int docCount) throws IOException {
        docs = new ArrayList<>();
        if (docCount > TD.getColumnDimension()) {
            docCount = TD.getColumnDimension();
        }

        System.out.println();
        for (int i = 0; i < docCount; i++) {
            String documentName = findMax(similarityValues);
            if (documentName != null) {
            System.out.println(i + 1 + " ("
+(Math.round(similarityValues.get(documentName)*100)/100D)+
") " + documentName);

relevance.add((Math.round(similarityValues.get(documentName)*
100)/100D));
                docs.add(documentName);
                similarityValues.remove(documentName);
            } else {
                System.out.println("\nHanya " + i + "dokumen
yang relevan ada disana\n");
                return;
            }
        }
    }
    ArrayList<Double> relevance = new ArrayList<>();
    ArrayList<String> docs = new ArrayList<>();
    public ArrayList<Double> getRelevance(){
        return relevance;
    }
    public ArrayList<String> getDocs(){
        return docs;
    }

    public void handleQuery(String query) throws IOException
{
        Matrix queryVector = createQueryVector(query);

        if (eligibleQuery == true) {
            TreeMap<String, Double> documentSimilarityValues
= findSimilarities(queryVector);

displayRelevantDocuments(documentSimilarityValues,
relevantDocs);
            eligibleQuery = false;
        } else {
            System.out.println("Query yang Anda masukkan
tidak terdapat dalam susunan kata-kata pada data corpus atau
tidak ditemukan dalam dokumen apapun");
        }
    }
}
```

Kode diatas menjelaskan proses perhitungan pembobotan method menggunakan metode Latent Semantic Indexing (LSI). Perhitungan pembobotan method dilakukan dengan cara membandingkan keberadaan method di masing-masing *class*. Hasil matriks yang tercipta pada proses perhitungan ini adalah 0 sampai dengan 1. Method tersebut dianalisis, apakah digunakan atau dipanggil oleh *class* lain sehingga method tersebut juga ada di *class* lain atau method tidak dipanggil oleh *class* lain. Jika nilai matriks bernilai mendekati 1 maka berarti *method* tersebut memiliki tingkat kemiripan cukup kuat dengan *method* lain. Kemiripan tersebut dihitung berdasarkan kemiripan kata per kata yang terdapat dalam masing-masing *method*. Selain itu, jika nilai matriks mendekati 0 maka method tersebut memiliki nilai kemiripan yang rendah dengan *method* lain.

5.3.3 Implementasi Perhitungan Nilai Kopling Menggunakan Parameter *Conceptual Coupling of a Class (CoCC).*

Implementasi perhitungan nilai kopling pada penelitian ini menggunakan parameter perhitungan kopling *Conceptual Coupling of a Class (CoCC).* Dimana untuk mendapatkan hasil dari *CoCC* terlebih dahulu sistem harus menghitung nilai dari *Conceptual Similarity between Methods (CSM), Conceptual Similarity between a Method and a Class (CSMC) dan Conceptual Similarity between two classes (CSBC).* Berikut adalah implementasi perhitungan keempat parameter tersebut :

5.3.3.1 Implementasi Perhitungan Conceptual Similarity between Methods (CSM)

Implementasi perhitungan Conceptual Similarity between Methods (CSM) didapatkan dengan cara menghitung perbandingan antar method. Nilai perbandingan antar method ini menggunakan vektor matriks yang didapatkan dari hasil perhitungan pembobotan method menggunakan metode *Latent Semantic Indexing (LSI)*. CSM membandingkan kemiripan antara dua *method*. Berikut kode perhitungan dari perhitungan parameter Conceptual Similarity between Methods (CSM):

Kode 5.4 Kode perhitungan parameter CSM

```
package coupling;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;

public class CSM {
    File[] listOfFiles_txt;
```

```
    ArrayList<Double> relevance;
    ArrayList<String> docs;

    public CSM(LSI lsiObj, File folder_temp) throws
IOException {
        computeCSM(lsiObj, folder_temp);
    }

    public String readFile(String path)
            throws IOException {
        byte[] encoded =
Files.readAllBytes(Paths.get(path));
        return new String(encoded,
StandardCharsets.UTF_8);
    }
    public File[] getListOfFiles(){
        return listOfFiles_txt;
    }
    public ArrayList<Double> getRelevance(){
        return relevance;
    }
    public ArrayList<String> getDocs(){
        return docs;
    }
    public void computeCSM(LSI lsiObj, File
folder_temp) throws IOException {
        relevance = new ArrayList<>();
        docs = new ArrayList<>();
        ArrayList<String> docs_temp = new
ArrayList<>();


System.out.println("\n\n\n==========\tCSM\t==========="
);
        listOfFiles_txt = folder_temp.listFiles();
        for (int i = 0; i < listOfFiles_txt.length;
i++) {

System.out.println(listOfFiles_txt[i].getName());
            String query =
readFile(listOfFiles_txt[i].getAbsolutePath());
            lsiObj.handleQuery(query);
            System.out.println("");
            for (int j = 0; j <
lsiObj.getDocs().size(); j++) {
                String d1 =
listOfFiles_txt[i].getName();
                String d2 = lsiObj.getDocs().get(j);
                String s = d1 + " <> " + d2;
```

```
                docs.add(s);
            }
        }
        relevance.addAll(lsiObj.getRelevance());
        docs_temp.addAll(lsiObj.getDocs());
    }
}
```

5.3.3.2 Implementasi Perhitungan Conceptual Similarity between a Method and a Class (CSMC)

Implementasi perhitungan Conceptual Similarity between a Method and a Class (CSMC) berdasarkan nilai dari CSM pada tahap sebelumnya.

Kode 5.5 Kode perhitungan parameter CSMC

```
package coupling;

import java.io.File;
import java.util.ArrayList;

public class CSMC {

    ArrayList<Double> csbc_data;

      public CSMC(ParsingMethod cpl, CSM csm, File
      folder_temp, ArrayList<Double> relevance,
      ArrayList<String> docs) {
      computeCSMC(cpl, csm, folder_temp, relevance,
docs);
    }

    public ArrayList<Double> getCSBCData() {
        return csbc_data;
    }

    public void computeCSMC(ParsingMethod cpl, CSM
csm, File folder_temp, ArrayList<Double> relevance,
ArrayList<String> docs) {

System.out.println("\n\n\n==========\tCSMC\t==========
");
        int counter = 0;
        double csbc = 0;
        csbc_data = new ArrayList<>();
        File[] listOfFiles_txt = csm.getListOfFiles();
        for (int i = 0; i < listOfFiles_txt.length;
i++) {
      String m =
cpl.getMethodName(listOfFiles_txt[i].getName());
            String c =
cpl.getClassName(listOfFiles_txt[i].getName());
            int jmlMethod =
cpl.countMethod(folder_temp, c);
   for (int j = 0; j <  listOfFiles_txt.length; j++) {
                int temp = 0;
```

```
                String c2 =
cpl.getClassName(listOfFiles_txt[j].getName());
                    if (!c.equals(c2)) {
System.out.println("m:" + m + " c: " + c2);
                    double tot = 0;
      ArrayList<String> methods =
                    cpl.getMethods(folder_temp, c2);
      for (int k = 0; k < methods.size(); k++) {
                        String m2 =
cpl.getMethodName(methods.get(k));

System.out.println(relevance.get(docs.indexOf(listOfFi
les_txt[i].getName() + " <> " + methods.get(k))) + " "
+ "m:" + m + " m2: " + m2);
 tot +=
relevance.get(docs.indexOf(listOfFiles_txt[i].getName(
) + " <> " + methods.get(k)));
                    }
                    double csmc = (Math.round(tot /
methods.size() * 100) / 100D);
                    System.out.println("CSMC = " +
csmc);
                    System.out.println("");
                    csbc += csmc;
                    temp++;
                    counter++;
                }
                if (counter == jmlMethod) {
                    csbc_data.add(csbc / jmlMethod);
                    counter = 0;
                    csbc = 0;
                }
                if (temp == jmlMethod) {
                    break;
                }
              }
          }
    }
}
```

5.2.3.3.3 Implementasi Perhitungan *Conceptual Coupling between Two Classes (CSBC)*

Implementasi perhitungan *Conceptual Coupling between Two Classes (CSBC)* berdasarkan nilai dari CSMC pada tahap sebelumnya.

Kode 5.6 Kode perhitungan parameter CSBC

```
Package coupling;

import java.io.File;
import java.util.ArrayList;

public class CSBC {

    public CSBC(ParsingMethod cpl, File[]
listOfFiles,ArrayList<Double> csbc_data) {
        computeCSBC(cpl, listOfFiles,csbc_data);
    }
```

```
    public void computeCSBC(ParsingMethod cpl, File[]
listOfFiles,ArrayList<Double> csbc_data) {

System.out.println("\n\n\n==========\tCSBC\t=========="
);
        for (int i = 0; i < listOfFiles.length; i++) {
            String c =
cpl.getClassName(listOfFiles[i].getName());
            for (int j = 0; j < listOfFiles.length;
j++) {
                String c2 =
cpl.getClassName(listOfFiles[j].getName());
                if (!c.equals(c2)) {
                    System.out.println("CSBC (" + c2 +
"," + c + ") = " + csbc_data.get(i));
                }
            }

        }
    }
}
```

5.2.3.3.4 Implementasi Perhitungan *Conceptual Coupling of Classes (CoCC)*

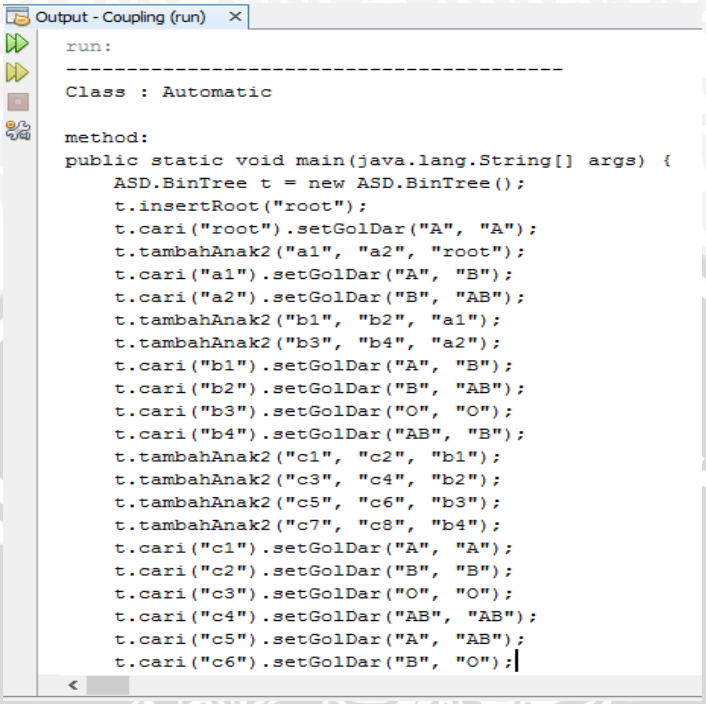
Implementasi perhitungan *Conceptual Coupling of Classes (CoCC)* berdasarkan nilai dari CSBC pada tahap sebelumnya.

Kode 5.7 Kode perhitungan parameter CoCC

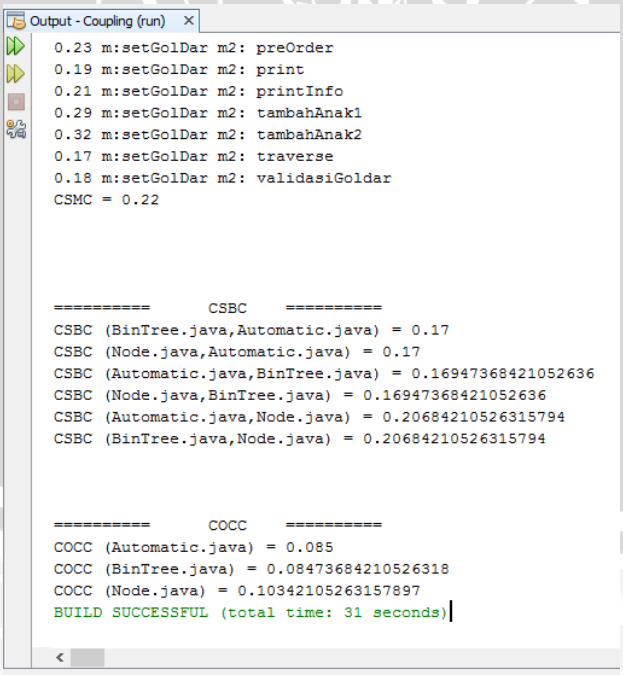
```
package coupling;

import java.io.File;
import java.util.ArrayList;

public class COCC {

    public COCC(ParsingMethod cpl, File[]
listOfFiles,ArrayList<Double> csbc_data) {
        computeCOCC(cpl, listOfFiles, csbc_data);
    }

    public void computeCOCC(ParsingMethod cpl, File[]
listOfFiles,ArrayList<Double> csbc_data) {

System.out.println("\n\n\n==========\tCOCC\t=========="
);
        for (int i = 0; i < listOfFiles.length; i++) {
            String c =
cpl.getClassName(listOfFiles[i].getName());
            System.out.println("COCC (" + c + ") = " +
csbc_data.get(i) / (listOfFiles.length - 1));
        }
    }
}
```

5.4 Tampilan Implementasi Program

Berikut tampilan implementasi program kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics.*



Gambar 5.5 Tampilan Program Kakas Bantu



Gambar 5.6 Tampilan Program Kakas Bantu (lanjutan)

BAB 6 PENGUJIAN

Bab ini membahas tentang tahapan pengujian terhadap sistem kakas bantu perhitungan nilai kopling yang telah dibangun. Tahapan pengujian yang dilakukan terdiri dari pengujian *white box,* pengujian *black box* dan pengujian akurasi perhitungan. Pengujian *white box* dilakukan dilakukan dengan cara pengujian unit. Pengujian unit tersebut diuji dengan teknik *basis path testing*. Pengujian black box dilakukan dengan cara menguji validasi terhadap kasus uji tertentu.

6.1 Pengujian *White Box*

Pengujian *white box* yang diterapkan pada pengujian ini ialah pengujian unit dengan teknik *basis path testing*. Tujuan pengujian ini adalah untuk menganalisa kebutuhan fungsionalitas utama pada sistem dengan menampilkan algoritma dalam bentuk sebuah *flow graph. Flow graph* dibentuk untuk mengetahui *cyclomatic complecity. Cyclomatic complecity* ditentukan dari beberapa proses yang terpilih untuk diuji dengan teknik *basis path testing*. Pengujian unit yang dilakukan meliputi pengujian ParsingMethod, LSI dan pengujian unit CSM. Pengujian unit pada tiga class tersebut dilakukan dengan cara menguji secara mandiri salah satu method pada masing-masing *class* yaitu method searchMethodinFile pada *class ParsingMethod, method displayRelevantDocuments* pada *class LSI* dan method *computeCSM* pada *class CSM*.

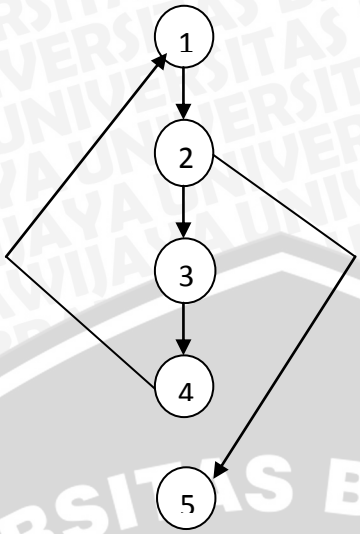
6.1.1 Pengujian Unit ParsingMethod.

Berikut ini merupakan pengujian unit *ParsingMethod* dengan menguji salah satu *method* pada *class* ini yaitu *method searchMethodinFile*. Algoritma *method searchMethodinFile* ditunjukkan pada Tabel 6.1.

Tabel 6.1 Algoritma *Method* searchMethodinFile

```
FUNCTION searchMethodinFile(String fileLocation, String nameMethod)
TRY                                                              } 1
FOR readAllLines(path.get(fileLocation))
IF matcherString(line, nameMethod)                               } 2
        return true;                                             } 3
CATCH IOException ex
        ParsingMethod.class.getName();                           } 4
  return false;                                                  } 5
```

Berdasarkan algoritma pada Tabel 6.1, maka diperoleh sebuah *flow graph* yang tertera pada Gambar 6.1 berikut ini.



Gambar 6.1 Flow Graph Method searchMethodinFile

Langkah selanjutnya adalah menghitung jumlah *Cyclomatic Complexity* dengan persamaan berikut ini.

V(G) = E – N + 2 persamaan (6.1)

Dimana keterangan untuk persamaan 6.1 sebagai berikut :

V(G) : *cyclomatic complexity* untuk *flow graph* G

E : Jumlah *Edge* (panah)

N : Jumlah Node (lingkaran)

Nilai *cyclomatic complexity* untuk *flow graph* pada gambar 6.1 adalah sebagai berikut.

V(G) = E – N + 2

V(G) = 5 – 5 + 2

V(G) = 2

Sehingga, dari nilai *cyclomatic complexity* tersebut diperoleh dua jalur independen yaitu:

Jalur 1 **:** 1 – 2 – 5

Jalur 2 **:** 1 – 2 – 3 – 4 – 1 – 2 - 5

Berdasarkan jalur independen yang telah diketahui, maka dapat didefinisikan kasus uji seperti yang dipaparkan dalam Tabel 6.2.

Tabel 6.2 Kasus Uji Method searchMethodinFile

Jalur	Kasus Uji	Prosedur Uji	Expected Result	Test Result
1.	Lokasi method dan nama method ditemukan.	Meletakkan file sesuai dengan *path* *location* yang telah ditentukan dan memberikan nama method yang terdiri dari huruf, angka dan simbol.	Mengembalikan nilai true.	Mengembalikan nilai true.
2.	Lokasi method dan lokasi tidak ditemukan.	Mengubah lokasi penyimpanan dan mengubah nama method menjadi hanya terdiri dari simbol dan angka (tidak termasuk huruf).	Mengembalikan nilai false.	Mengembalikan nilai false.

6.1.2 Pengujian Unit ComputeCSM

Berikut ini pengujian unit CSM. Dalam pengujian ini, *method* yang diuji adalah *method computeCSM*. Algoritma *method computeCSM* tertera pada Tabel 6.3 sebagai berikut.

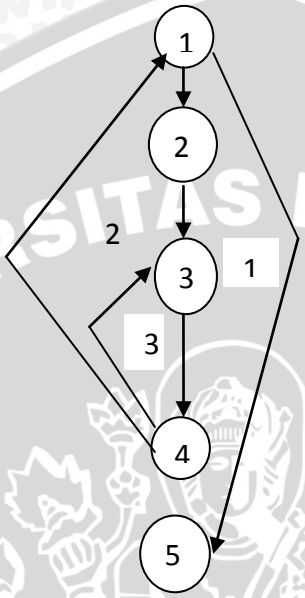
Tabel 6.3 Algoritma Method computeCSM

```
PROCEDURE computeCSM(LSI lsiObj, File folder_temp)
    ArrayList<String> docs_temp = new ArrayList<>();
    listOfFiles_txt=folder_temp.listFiles();
FOR i<listOfFiles_txt; i++                          } 1
    print listOfFiles_txt[i].getName;
    query=readFile(listOfFiles_txt[i].getAbsolutePath();   } 2
    lsiObj.handleQuery(query);
FOR j<lsi.OBJ.getDocs().size(); j++                 } 3
    d1= listOfFiles_txt[i].getName();
    d2=lsiObj.getDocs().get(j);                     } 4
    s=d1+"<>"+d2;
    docs.add(s);
ENDFOR
```

```
ENDFOR
    add all relevance;
    add all docs_temp;
END
```

5

Berdasarkan algoritma pada Tabel 6.3 maka dapat diperoleh flow graph yang akan digambarkan pada Gambar 6.3 sebagai berikut.



Gambar 6.2 Flow Graph Method computeCSM

Nilai *cyclomatic complexity* untuk *flow graph* pada gambar 6.2 adalah sebagai berikut.

V(G) = E – N + 2

V(G) = 6 – 5 + 2

V(G) = 3

Maka, nilai *cyclomatic complexity* tersebut menghasilkan tiga jalur independen yaitu :

Jalur 1 : 1 – 5

Jalur 2 : 1 – 2 – 3 – 4 – 1 – 5

Jalur 3 : 1 – 2 – 3 – 4 – 3 – 4 – 1 – 5

Berdasarkan jalur independen yang telah diketahui, maka dapat didefinisikan kasus uji seperti yang dipaparkan dalam Tabel 6.3.

Tabel 6.3 Kasus Uji Method computeCSM

Jalur	Kasus Uji	Prosedur Uji	Expected Result	Test Result
1.	Daftar file yang ingin dianalisis ada tidak ada.	Tidak memasukkan file apapun ke dalam sistem	Mengeluarkan pesan error karena tidak ada file yang dianalisis.	Mengeluarkan pesan error.
2.	Sistem menganalisis kode program sebanyak file yang ada dan jika file ditemukan maka file tersebut dicocokkan dengan *query. Query* disini berarti dokumen tersebut maupun dokumen lainnya.	File dimasukkan ke dalam sistem sebanyak n file.	Sistem menganalisis kode program sebanyak n file dan dicocokkan dengan *query*. Jika lebih dari n file maka analisis kode program dihentikan dan cetak nilai relevansi/kemiripan.	Sistem mampu menganalisis sebanyak n file dan mencocokkan ya dengan *query.* Jika lebih dari n file maka analisis kode program dihentikan dan cetak nilai relevansi/kemiripan.
3.	Sistem mengecek file beserta isi file dan jika ada file yang dimasukkan maka file bertambah.	File dimasukkan ke dalam sistem sebanyak n file dan file yang dimasukkan ditambah.	Sistem menganalisis kode program sebanyak n file dan jika ada file baru dimasukkan maka file ditambahkan ke dalam matriks LSI. Selanjutnya, balik lagi mengecek file. Saat pengecekan sudah lebih dari n file dan tidak ada file tambahan maka sistem berhenti	Sistem berhasil menganalisis file sebanyak n file dan menambahkan file jika ada file tambahan. Sistem berhasil berhenti menganalisis file jika pengecekan melebihi n file dan tidak ada file yang ditambahkan.

			menganalisis file.	

6.2 Pengujian Integrasi

Pengujian integrasi dilakukan untuk mengetahui dan menguji relasi antar *class.Class* yang diuji meliputi dua *class* yaitu *class LSI* dan *class CSM.* Kedua *class* ini memiliki relasi karena *class CSM* menggunakan hasil keluaran dari *LSI* berupa nilai kemiripan antar dokumen. Kedua *class* ini berelasi melalui sebuah *method* bernama *method computeCSM*.

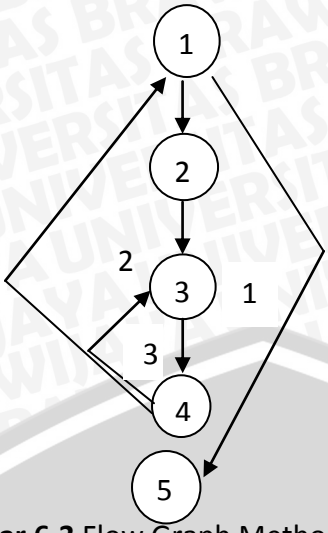
6.2.1 Pengujian Integrasi computeCSM

Berikut ini merupakan pengujian integrasi menghitung nilai CSM melalui sebuah *method* bernama *computeCSM.* Algoritma pada *method computeCSM* ditunjukkan pada tabel 6.4.

Tabel 6.4 Algoritma Method computeCSM

```
PROCEDURE computeCSM(LSI lsiObj, File folder_temp)
    ArrayList<String> docs_temp = new ArrayList<>();
    listOfFiles_txt=folder_temp.listFiles();
FOR i<listOfFiles_txt; i++                                  } 1
    print listOfFiles_txt[i].getName;
    query=readFile(listOfFiles_txt[i].getAbsolutePath();    } 2
    lsiObj.handleQuery(query);
FOR j<lsi.OBJ.getDocs().size(); j++                         } 3
    d1= listOfFiles_txt[i].getName();
    d2=lsiObj.getDocs().get(j);                             } 4
    s=d1+"<>"+d2;
    docs.add(s);
ENDFOR
ENDFOR
    add all relevance;                                      } 5
    add all docs_temp;
END
```

Berdasarkan algoritma pada Tabel 6.4 maka dapat diperoleh flow graph yang akan digambarkan pada gambar 6.3 sebagai berikut.



Gambar 6.3 Flow Graph Method computeCSM

Nilai *cyclomatic complexity* untuk *flow graph* pada gambar 6.3 adalah sebagai berikut.

V(G) = E – N + 2

V(G) = 6 – 5 + 2

V(G) = 3

Maka, nilai *cyclomatic complexity* tersebut menghasilkan tiga jalur independen yaitu :

Jalur 1 **:** 1 – 5

Jalur 2 **:** 1 – 2 – 3 – 4 – 1 – 5

Jalur 3 **:** 1 – 2 – 3 – 4 – 3 – 4 – 1 – 5

Berdasarkan jalur independen yang telah diketahui, maka dapat didefinisikan kasus uji seperti yang dipaparkan dalam Tabel 6.5.

Tabel 6.5 Kasus Uji Method computeCSM

Jalur	Kasus Uji	Prosedur Uji	Expected Result	Test Result
1.	Daftar file yang ingin dianalisis ada tidak ada.	Tidak memasukkan file apapun ke dalam sistem	Mengeluarkan pesan error karena tidak ada file yang dianalisis.	Mengeluarkan pesan error.
2.	Sistem menganalisis kode program sebanyak file yang ada dan jika file ditemukan maka file tersebut	File dimasukkan ke dalam sistem sebanyak n file.	Sistem menganalisis kode program sebanyak n file dan dicocokkan dengan *query*. Jika lebih dari n file maka analisis kode program	Sistem mampu menganalisis sebanyak n file dan mencocokkan ya dengan *query.* Jika lebih dari n

	dicocokkan dengan *query. Query* disini berarti dokumen tersebut maupun dokumen lainnya.		dihentikan dan cetak nilai relevansi/kemiripan.	file maka analisis kode program dihentikan dan cetak nilai relevansi/kemiripan.
3.	Sistem mengecek file beserta isi file dan jika ada file yang dimasukkan maka file bertambah.	File dimasukkan ke dalam sistem sebanyak n file dan file yang dimasukkan ditambah.	Sistem menganalisis kode program sebanyak n file dan jika ada file baru dimasukkan maka file ditambahkan ke dalam matriks LSI. Selanjutnya, balik lagi mengecek file. Saat pengecekan sudah lebih dari n file dan tidak ada file tambahan maka sistem berhenti menganalisis file.	Sistem berhasil menganalisis file sebanyak n file dan menambahkan file jika ada file tambahan. Sistem berhasil berhenti menganalisis file jika pengecekan melebihi n file dan tidak ada file yang ditambahkan.

6.3 Pengujian Black Box

Metode pengujian yang dilakukan pada pengujian black box adalah menggunakan metode pengujian validasi. Pengujian validasi berdasarkan kepada skenario use case yang telah dibuat pada tahap analisis kebutuhan.

6.3.1 Memasukkan File Kode Program

Berikut ini adalah pengujian validasi memasukkan file kode program yang dijelaskan pada tabel 6.6 dan memasukkan file kode program alur alternatif 1 yang ditunjukkan pada tabel 6.7.

Tabel 6.6 Memasukkan File Kode Program

Nama Kasus Uji	Memasukkan File Kode Program (TC_001)
Objek Uji	Kebutuhan Fungsional SKPL_KBPK_001
Tujuan Pengujian	Untuk memastikan sistem mampu memenuhi kebutuhan fungsional melakukan masukkan file kode program
Data Masukan	*File* kode program bertipe *Java.*
Prosedur Uji	1. Jalankan sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* menggunakan *Netbeans IDE.* 2. Membuat *class* baru pada *folder data* dalam sistem kakas bantu. Bahasa pemrograman yang digunakan bahasa *Java*. 3. *Class* tersebut disimpan pada folder data pada sistem kakas bantu.
Hasil yang diharapkan	Sistem menyimpan *class* tersebut ke dalam sistem yaitu pada folder data agar file tersebut menjadi data untuk dihitung dan dianalisis.

Tabel 6.7 Memasukkan File Kode Program Alur Alternatif 1

Nama Kasus Uji	Memasukkan File Kode Program Alternatif 1 (TC_002)
Objek Uji	Kebutuhan Fungsional SKPL_KBPK_001
Tujuan Pengujian	Untuk memastikan sistem mampu menangani kondisi masukan file yang tidak bertipe *Java.*
Data Masukan	*File* kode program bukan bertipe *Java.*
Prosedur Uji	1. Jalankan sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* menggunakan *Netbeans IDE.* 2. Membuat *class* baru pada *folder data* dalam sistem kakas bantu. Bahasa pemrograman yang digunakan bahasa *Java*. 3. *Class* tersebut disimpan pada folder data pada sistem kakas bantu.
Hasil yang diharapkan	Sistem menampilkan pesan kesalahan *"java.lang.NullPointerException".*

6.3.2 Melakukan *Parsing* terhadap Kode Program

Berikut ini adalah pengujian validasi melakukan *parsing* terhadap kode program yang dijelaskan pada tabel 6.8.

Tabel 6.8 Melakukan P*arsing* terhadap Kode Program

Nama Kasus Uji	Melakukan P*arsing* terhadap Kode Program (TC_003)
Objek Uji	Kebutuhan Fungsional SKPL_KBPK_002
Tujuan Pengujian	Untuk memastikan sistem mampu memenuhi kebutuhan fungsional melakukan *parsing* terhadap kode program.
Data Masukan	*File* kode program bertipe *Java.*
Prosedur Uji	1. Jalankan sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* menggunakan *Netbeans IDE.* 2. Menekan tombol *run project* pada *Netbeans text editor. Project* yang dijalankan adalah sistem kakas bantu perhitungan nilai kopling.
Hasil yang diharapkan	Sistem mampu melakukan *parsing* terhadap file kode program yang dimasukkan.

6.3.3 Menghitung Nilai Kopling

Berikut ini adalah pengujian validasi menghitung nilai kopling yang dijelaskan pada tabel 6.9.

Tabel 6.9 Menghitung Nilai Kopling

Nama Kasus Uji	Menghitung Nilai Kopling (TC_004)
Objek Uji	Kebutuhan Fungsional SKPL_KBPK_003
Tujuan Pengujian	Untuk memastikan sistem mampu menghitung nilai kopling.
Data Masukan	*File* kode program bertipe *Java.*
Prosedur Uji	1. Jalankan sistem kakas bantu perhitungan nilai kopling menggunakan *conceptual coupling metrics* menggunakan *Netbeans IDE.* 2. Menekan tombol *run project* pada *Netbeans text editor. Project* yang dijalankan adalah sistem kakas bantu perhitungan nilai kopling.
Hasil yang diharapkan	Sistem mampu menghitung dan menampilkan nilai kopling.

6.4 Analisis Hasil Pengujian Validasi

Analisis hasil pengujian validasi didapatkan dari hasil pengujian validasi. Analisis hasil pengujian validasi ini adalah hasil pengujian validasi kebutuhan fungsional.

Tabel 6.10 Hasil Pengujian Validasi Kebutuhan Fungsional

No	Kebutuhan	Kasus Uji	*Test Result*	Status Validitas
1.	SKPL_KBPK_001	TC_001	Sistem menyimpan *file* ke dalam sistem kakas bantu.	Valid
2.	SKPL_KBPK_001	TC_002	Sistem menampilkan pesan kesalahan *"java.lang.NullPointerException".*	Valid
3.	SKPL_KBPK_002	TC_003	Sistem mampu melakukan *parsing* terhadap kode program.	Valid
4.	SKPL_KBPK_003	TC_004	Sistem mampu menghitung nilai kopling.	Valid

6.5 Pengujian Akurasi Perhitungan

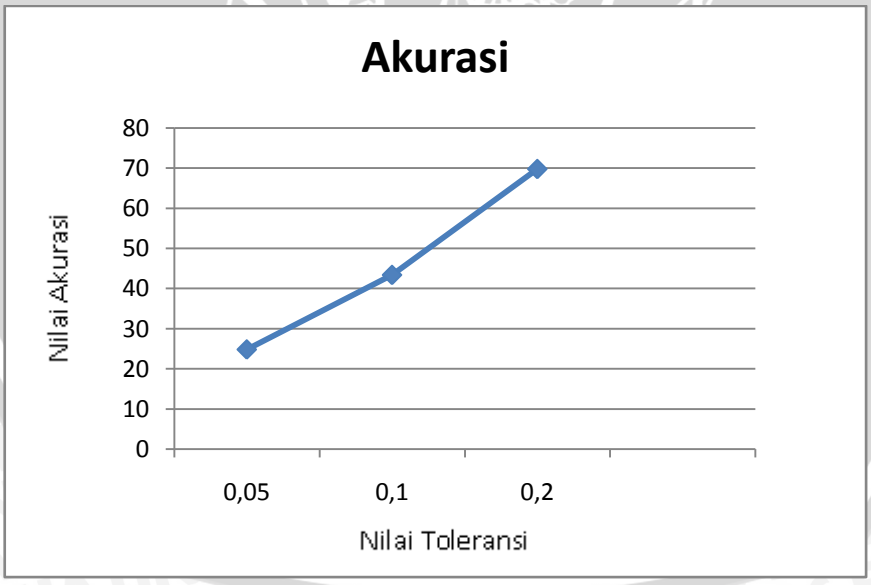
Konsep pengujian akurasi perhitungan pada penelitian ini adalah dengan cara melakukan analisis perbandingan antara hasil perhitungan implementasi program kakas bantu dan hasil perhitungan dari sebuah aplikasi pembanding perhitungan *Latent Semantic Analysis (LSI)* yaitu lsa.colorado.edu. Sistem perhitungan LSI pada *lsa.colorado.edu* menggunakan cara seperti memasukkan dokumen yang ingin dicocokkan secara manual, memilih topik yang ingin digunakan dan memilih kategori pencocokkan apakah antar kata, kata dengan dokumen atau antar dokumen. Aplikasi ini akan secara otomatis menampilkan matriks hasil perhitungan pencocokkan data yang dimasukkan. Sedikit berbeda dengan sistem pada aplikasi pembanding, sistem kakas bantu pada penelitian ini menggunakan cara perhitungan nilai kecocokan menggunakan *Latent Semantic Indexing* (LSI) dengan cara memasukkan dokumen kode program dan sistem kakas bantu akan otomatis menghitung mulai dari *parsing method*, mencocokkan dokumen hingga menghitung nilai kopling.

Skenario pengujian yang dilakukan adalah dengan cara menangkap nilai kemiripan antar dokumen yang dihitung menggunakan metode *Latent Semantic Indexing (LSI)* dimana nilai kemiripan dokumen dengan LSI ini juga menjadi nilai *Conceptual Similarity between Method* (CSM) lalu dibandingkan dan dicari selisih antar kedua nilai dari sistem yang berbeda tersebut. Setelah mengetahui selisihnya, lalu nilai toleransi diberikan untuk mengetahui akurasi sistem. Pemberian nilai toleransi tersebut hanya berupa sebuah percobaan saja. Nilai toleransi tersebut merupakan nilai yang dapat dan perlu diteliti lebih lanjut. Pemberian nilai toleransi ini dikarenakan nilai yang ditangkap dari kedua sistem

tersebut hanya sedikit yang sama. Pengujian dilakukan dengan cara mengambil sebanyak 576 dataset didapatkan dari 24 *method* dan 3 *class*. Dataset sebanyak 576 didapatkan dari matriks kemiripan antar dokumen dengan ordo matriks sebesar 24 kali 24 sehingga menghasilkan total data uji sebanyak 576 data uji.

6.5.1 Analisis Data Hasil Pengujian Akurasi

Berdasarkan selisih hasil perbandingan perhitungan implementasi program dan perhitungan aplikasi pembanding seperti yang tertera pada lampiran 1, didapatkan bahwa dari total data uji sebanyak 24 method dalam tiga *class* semuanya memberikan nilai akurasi berbeda. Percobaan dilakukan dengan cara memberikan tiga nilai toleransi yang berbeda yaitu 0.05, 0.1 dan 0.20. Pemberian nilai toleransi sebesar 0.05, 0.1 dan 0.20 dirasakan memiliki *range* yang cukup untuk dijadikan sebagai percobaan. Pemberian toleransi tersebut berguna untuk memberikan toleransi akurasi yang artinya adalah jika nilai selisih kurang dari nilai toleransi maka data tersebut masih bisa di toleransi keakuratan datanya. Pada saat memberikan ketiga nilai toleransi tersebut, hasilnya jika nilai toleransi sebesar 0.05 maka nilai akurasi sistem sebesar 24,83%. Jika nilai toleransi sebesar 0.1 maka nilai akurasi sistem sebesar 43,41%. Sedangkan, jika nilai toleransi sebesar 0.20 maka nilai akurasi yang diperoleh diatas 50% yaitu 69,78%. Grafik perbedaan nilai akurasi pada tiap-tiap nilai toleransi dapat dilihat pada Gambar 6.3.



Gambar 6.3 Grafik Perbedaan Nilai Akurasi pada Masing-Masing Nilai Toleransi

Nilai akurasi didapat dari rumus sebagai berikut.

$$\text{Akurasi} = \frac{jumla\ h\ data\ \ yang\ \ benar}{jumla\ h\ data\ \ keseluru\ han} \times 100\%$$

Keterangan:

Jumlah data valid = jumlah data yang hasil selisihnya kurang dari nilai toleransi.

Jumlah keseluruhan data = jumlah keseluruhan data uji yaitu 576 data. Jumlah ini didapatkan dari 24 method yang saling dibandingkan satu sama lain sehingga total data uji sebenarnya adalah 576 data uji.

Dari perhitungan akurasi seperti pada gambar 6.3 yang dilakukan dengan metode perhitungan perbedaan selisih antara perhitungan implementasi program dan perhitungan aplikasi pembanding maka dapat dilakukan analisis bahwa berdasarkan hasil perhitungan dengan cara mencari perbedaan selisih antara hasil perhitungan implementasi program dan perhitungan aplikasi bantuan terhadap 576 data uji dimana data uji ini didapat dari 24 method yang saling dibandingkan satu sama lain dan terdiri dari tiga *class.* Dari hasil analisis ini didapatkan bahwa terdapat perbedaan hasil akurasi. Hal ini disebabkan oleh perbedaan hasil perhitungan antara perhitungan pada saat implementasi program dengan hasil perhitungan pada aplikasi pembanding. Perbedaan ini disebabkan oleh kamus yang digunakan pada aplikasi pembanding dengan program berbeda. Pada saat analisis data atau pencarian informasi menggunakan metode *Latent Semantic Indexing (LSI)* kamus yang digunakan dapat berbeda-beda sesuai dengan ruang lingkup bahasan informasi yang ingin dicari. Pada aplikasi pembanding, pengguna dapat memilih topik bahasan informasi yang ingin dicocokkan dan tentu saja aplikasi akan memberikan kamus yang sesuai dengan topik yang dibutuhkan. Berbeda halnya dengan implementasi program pada penelitian ini yang hanya menggunakan satu kamus umum saja.

BAB 7 KESIMPULAN DAN SARAN

7.1 Kesimpulan

Kesimpulan yang diperoleh dalam penelitian Kakas Bantu Perhitungan Nilai Kopling Menggunakan Conceptual Coupling Metrics, antara lain:

1. Pengidentifikasian *class* yang saling berkaitan satu sama lain dapat diukur dengan pertama-tama melakukan parsing terhadap method-method yang dimiliki tiap-tiap class. Lalu, method-method tersebut diuraikan menjadi masing-masing dokumen agar memudahkan pada saat dicocokkan satu sama lain. Proses pencocokan dokumen yang berisi *body method* tersebut menggunakan sebuah metode *information retrieval* yaitu *Latent Semantic Indexing (LSI)*. Terakhir, perhitungan kopling dilakukan dengan menggunakan kopling metrik yaitu *conceptual coupling metric. Conceptual coupling metric* terdiri dari empat parameter yaitu *Conceptual Similarity between Method (CSM), nilai Conceptual Similarity between a Method and a Class (CSMC), nilai Conceptual Coupling between Two Classes (CSBC)* dan nilai *Conceptual Coupling of Classes (CoCC)*.
2. Cara menentukan *conceptual coupling metrics* berdasarkan pada method dalam sebuah kumpulan dokumen *source code* program adalah dengan cara mengambil nilai relevance tiap-tiap method dan nilai relevance tersebut dimasukkan ke dalam tabel *Conceptual Similarity between Method (CSM).* Nilai relevance dalam tabel CSM tersebut menjadi nilai CSM. Lalu, nilai CSM tersebut menjadi dasar bagi perhitungan parameter *Conceptual Similarity between a Method and a Class (CSMC).* Nilai CSMC menjadi dasar bagi perhitungan nilai *Conceptual Coupling between Two Classes (CSBC)* dan nilai CSBC menjadi dasar bagi perhitungan nilai *Conceptual Coupling of Classes (CoCC)*.
3. Tingkat akurasi dari sistem berbanding lurus dengan nilai toleransi yang diberikan. Pada saat memberikan ketiga nilai toleransi tersebut, hasilnya jika nilai toleransi sebesar 0.05 maka nilai akurasi sistem sebesar 24,83%. Jika nilai toleransi sebesar 0.1 maka nilai akurasi sistem sebesar 43,41%. Sedangkan, jika nilai toleransi sebesar 0.20 maka nilai akurasi yang diperoleh diatas 50% yaitu 69,78%. Semakin tinggi nilai toleransi maka semakin tinggi pula nilai akurasi sistem.

7.2 Saran

Saran untuk penelitian Kakas Bantu Perhitungan Nilai Kopling Menggunakan Conceptual Coupling Metrics, antara lain:

1. Sebaiknya komentar dalam *body method* dapat dianalisis juga agar tingkat kopling lebih sempurna.
2. Kategori tingkat kopling sebaiknya lebih jelas antara kopling rendah, sedang atau tinggi agar memudahkan pengembang dalam menentukan tingkat kopling dari implementasi programnya.
3. Dikembangkan secara lebih luas untuk mengukur tingkat kopling menggunakan bahasa pemrograman lain selain bahasa java.

DAFTAR PUSTAKA

Poshyvanyk, Denys and Andrian Marcus., 2006. *The Conceptual Coupling Metrics for Object-Oriented Systems. 22nd IEEE International Conceference on IEEE* [e-journal]. Tersedia melalui: IEEE <ieeexplore.ieee.org> [Diakses 10 Oktober 2015]

Ghosh, Joydeep Prof., 2015. *Information Retrieval Models Boolean Retrieval Model. Open Courseware. Virtual University of Pakistan.*

Scoot, D, Paul and Gui Gui., 2007. *Ranking reusability of software components using coupling metrics.* United of Kingdom. The Journal of System and Software. Tersedia melalui: Science Direct <www.sciencedirect.com> [Diakses 12 November 2015]

Rosa, A. S dan Shalahuddin, M., 2013. *Rekayasa Perangkat Lunak: Terstruktur dan Berorientasi Objek* [Buku Modul Ajar]. Bandung. Informatika.

Widodo, Prabowo Pudjo dan Herlawati., 2011. Menggunakan UML (Unified Modelling Languange). Informatika. Bandung.

Dennis, Wixom dan Roth., 2006. *System Analysis & Design: Third Edition.* America. John Wiley & Sons.

Chhabra, Jitender Kumar and Anshu Parashar., 2011. *Clustering Dynamic Class Coupling Data using K-Mean and Cosine Similarity Measure to Predict Class Reusability Pattern.* India. 5th IEEE International Conference.ISBN 81-87885-30-3. Tersedia melalui: IEEE <ieeexplore.ieee.org> [Diakses 12 November 2015]

Sarashadi,Kenneth Setiawan, Fajar Pradana dan Denny Sagita Rusdianto., 2015. Kakas Bantu Perhitungan Nilai Kohesi Menggunakan Metrik *Distance Design Design-Based Direct Class Cohesion.* Universitas Brawijaya. Malang.

Lestari,Elliya, Fajar Pradana dan Denny Sagita Rusdianto., 2015. Perhitungan Kualitas Desain *Object-Oriented* Menggunakan *Matrix Similarity-Based Class Cohesion (SCC)* Pada Kelas Kohesi Berbasis Web. Universitas Brawijaya. Malang.

Adi,Nugroho., 2010. *Rekayasa Perangkat Lunak Berorientasi Objek dengan Metode USDP*. Andi. Yogyakarta.

Fowler,Martin., 2004. *UML Distilled (Panduan Singkat Bahasa Pemodelan Berorientasi Objek) Edisi 3*. Andi. Yogyakarta.

Ardisasmita, S,M., 2000. Penerapan Extensible Markup Languange Pada Database Sistem Informasi Nuklir Internasional. Pusat Pengembangan Teknologi Informatika dan Komputasi. Batam.

Mishra A, and Dubey, D., 2013. *A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios*. Computer Science & Engineering. India.

Sommerville,Ian., 2010. *Software Engineering*. Eight Edition.

Shumate, Ken and Marilyn Keller., 1992. *Software Spesification and Design.* John Wiley & Sons. New York.

Rouf, Abdul., 2012. Pengujian Perangkat Lunak dengan Menggunakan Metode *White Box* dan *Black Box* [e-journal]. Tersedia melalui: himsya e-journal <ejournal.himsya.ac.id> [Diakses 19 April 2016]

Pressman, S Roger., 2010. *Software Engineering:Practitioner's Approach. Eight Edition.*

Melian, Lusi dan Hendi Hermawan., 2011. Aplikasi Mobile Piano, Gitar dan Drum Virtual Berbasis Android (Studi Kasus:Purwacaraka Music Studio). Universitas Komputer Indonesia. Bandung.

Zhang, Yong and Jian-lin Li., 2009. *Research and Improvement of Search Engine Based on Lucene.* Hangzhou. Zhejiang. China.

Bunyamin, Hendra., 2005. Tesis:*Information Retrieval System* dengan Metode *Latent Semantic Indexing*. Institut Teknologi Bandung. Bandung.

Pawlak, Renaud, Martin Monperrus, dkk., 2014. *Spoon:A Library for Implementing Analyses and Transformations of Java Souce Code. Software:Practice and Experience. Wiley*.

Landauer, T. K., Foltz, P.W., and Laham, D. 1998. *Introduction to Latent Semantic Analysis. Discourse Processes*, 25, 259-284. Colorado.

Suswanto, Deni., Wisnu Ananta Kususma dan Vektor Dewanto., 2016. Analisis Perbandingan Metode *Machine Learning* untuk Prediksi Khasiat Jamu. Makalah Seminar S1 Ilmu Komputer. FMIPA. IPB. Bogor.

LAMPIRAN 1 PENGUJIAN AKURASI

1. Hasil Perhitungan LSI atau Nilai CSM Aplikasi Pembanding Colorado

Hasil perhitungan LSI atau nilai CSM pada aplikasi pembanding Colorado

Document	main	cari	cariParent	cariParent Internal	cekAda	*getChild*	getOrtu
main	1,00	0,12	0,21	0,22	0,22	0,33	0,15
cari	0,12	1,00	0,91	0,22	0,91	0,30	0,88
cariParent	0,21	0,91	1,00	0,27	0,99	0,40	0,82
cariParentInternal	0,22	0,22	0,27	1,00	0,26	0,90	0,32
cekAda	0,22	0,91	0,99	0,26	1,00	0,41	0,82
getChild	0,33	0,30	0,40	0,90	0,41	1,00	0,40
getOrtu	0,15	0,88	0,82	0,32	0,82	0,40	1,00
inOrder	0,13	0,48	0,43	0,45	0,43	0,44	0,55
insertRoot	0,10	0,70	0,65	0,07	0,64	0,16	0,62
isEmpty	0,08	0,91	0,82	0,22	0,84	0,27	0,78
isiCari	0,04	0,51	0,48	0,06	0,48	0,14	0,58
pencarian	0,29	0,61	0,62	0,78	1	0,83	0,72
postOrder	0,13	0,48	0,43	0,45	0,43	0,44	0,55
preOrder	0,13	0,48	0,43	0,45	0,43	0,44	0,55
print	0,12	0,50	0,46	0,34	0,45	0,33	0,48
printInfo	0,09	0,80	0,73	0,06	0,74	0,18	0,68
tambahAnak1	0,07	0,46	0,41	0,56	0,41	0,64	0,51
tambahAnak2	0,07	0,52	0,47	0,48	0,46	0,58	0,56
traverse	0,45	0,28	0,35	0,23	0,35	0,34	0,22
validasiGoldar	0,40	0,41	0,39	0,21	0,39	0,21	0,48
cekDarah	0,76	0,04	0,12	0,27	0,12	0,36	0,08
cekDarahAuto	0,73	0,05	0,15	0,35	0,15	0,47	0,11
getKemungkinan	0,07	0,79	0,71	0,25	0,72	0,31	0,86
setGoldar	0,11	0,83	0,76	0,07	0,77	0,26	0,88

Hasil perhitungan LSI atau nilai CSM pada aplikasi pembanding Colorado (lanjutan)

Document	*inOrder*	insertRoot	isEmpty	isiCari	*pencarian*	*postOrder*	*preOrder*	*print*
main	0,13	0,10	0,08	0,04	0,29	0,13	0,13	0,12
cari	0,48	0,70	0,91	0,51	0,61	0,48	0,48	0,50
cariParent	0,43	0,65	0,82	0,48	0,62	0,43	0,43	0,46
cariParentInternal	0,45	0,07	0,22	0,06	0,78	0,45	0,45	0,34
cekAda	0,43	0,64	0,84	0,48	0,63	0,43	0,43	0,45
getChild	0,44	0,16	0,27	0,14	0,83	0,44	0,44	0,33
getOrtu	0,55	0,62	0,78	0,58	0,72	0,55	0,55	0,48
inOrder	1,00	0,32	0,50	0,35	0,62	1,00	1,00	0,88
insertRoot	0,32	1,00	0,61	0,84	0,38	0,32	0,32	0,36
isEmpty	0,50	0,61	1,00	0,54	0,55	0,50	0,50	0,52
isiCari	0,35	0,84	0,54	1,00	0,34	0,35	0,35	0,33
pencarian	0,62	0,38	0,55	0,34	1,00	0,62	0,62	0,50
postOrder	1,00	0,32	0,50	0,35	0,62	1,00	1,00	0,88
preOrder	1,00	0,32	0,50	0,35	0,62	1,00	1,00	0,88
print	0,88	0,36	0,52	0,33	0,50	0,88	0,88	1,00
printInfo	0,52	0,62	0,86	0,56	0,43	0,52	0,52	0,63
tambahAnak1	0,48	0,44	0,33	0,24	0,58	0,48	0,48	0,38
tambahAnak2	0,45	0,50	0,34	0,26	0,56	0,45	0,45	0,35
traverse	0,56	0,23	0,29	0,16	0,34	0,56	0,56	0,56
validasiGoldar	0,27	0,62	0,39	0,70	0,34	0,27	0,27	0,25
cekDarah	0,30	0,04	0,05	0,06	0,24	0,30	0,30	0,24
cekDarahAuto	0,03	0,05	0,07	0,08	0,31	0,03	0,03	0,02
getKemungkinan	0,56	0,52	0,85	0,60	0,61	0,56	0,56	0,47
setGoldar	0,42	0,64	0,63	0,54	0,55	0,42	0,42	0,35

Hasil perhitungan LSI atau nilai CSM pada aplikasi pembanding Colorado (lanjutan)

Document	printInfo	*tambahAnak1*	*tambahAnak2*	*traverse*	*validasiGoldar*	*cekDarah*	*cekDarahAuto*	getKemungkinan	setGoldar
main	0,09	0,07	0,07	0,45	0,40	0,76	0,73	0,07	0,11
cari	0,80	0,46	0,52	0,28	0,41	0,04	0,05	0,79	0,83
cariParent	0,73	0,41	0,47	0,35	0,39	0,12	0,15	0,71	0,76
cariParentInternal	0,06	0,56	0,48	0,23	0,21	0,27	0,35	0,25	0,07
cekAda	0,74	0,41	0,46	0,35	0,39	0,12	0,15	0,72	0,77
getChild	0,18	0,64	0,58	0,34	0,21	0,36	0,47	0,31	0,26
getOrtu	0,68	0,51	0,56	0,22	0,48	0,08	0,11	0,86	0,88
inOrder	0,52	0,48	0,45	0,56	0,27	0,30	0,03	0,56	0,42
insertRoot	0,62	0,44	0,50	0,23	0,62	0,04	0,05	0,52	0,64
isEmpty	0,86	0,33	0,34	0,29	0,39	0,05	0,07	0,85	0,63
isiCari	0,56	0,24	0,26	0,16	0,70	0,06	0,08	0,60	0,54
pencarian	0,43	0,58	0,56	0,34	0,34	0,24	0,31	0,61	0,55
postOrder	0,52	0,48	0,45	0,56	0,27	0,30	0,03	0,56	0,42
preOrder	0,52	0,48	0,45	0,56	0,27	0,30	0,03	0,56	0,42
print	0,63	0,38	0,35	0,56	0,25	0,24	0,02	0,47	0,35
printInfo	1,00	0,31	0,33	0,37	0,30	0,03	0,06	0,73	0,64
tambahAnak1	0,31	1,00	0,98	0,08	0,18	0,08	0,12	0,37	0,53
tambahAnak2	0,33	0,98	1,00	0,09	0,17	0,07	0,10	0,39	0,63
traverse	0,37	0,08	0,09	1,00	0,13	0,72	0,50	0,19	0,18
validasiGoldar	0,30	0,18	0,17	0,13	1,00	0,28	0,28	0,42	0,34

cekDarah	0,03	0,08	0,07	0,72	0,28	1,00	0,87	0,07	0,02
cekDarahAuto	0,06	0,12	0,10	0,50	0,28	0,87	1,00	0,09	0,03
getKemungkinan	0,73	0,37	0,39	0,19	0,42	0,07	0,09	1,00	0,70
setGoldar	0,64	0,53	0,63	0,18	0,34	0,02	0,03	0,70	1,00

2. Hasil Perhitungan LSI atau Nilai CSM pada Sistem Kakas Bantu Perhitungan Nilai Kopling beserta Selisih Antara Hasil Perhitungan LSI atau Nilai CSM pada Aplikasi Pembanding dibandingkan dengan Hasil Perhitungan LSI atau Nilai CSM pada Sistem Kakas Bantu Perhitungan Nilai Kopling.

Hasil perhitungan LSI atau nilai CSM pada sistem kakas bantu perhitungan nilai kopling dan selisih antara kedua sistem.

	main	selisih	cari	selisih	cariParent	selisih	cariParentInternal
main	0,99	0,01	0,15	-0,03	0,12	0,10	0,07
cari	0,15	-0,03	0,98	0,02	0,57	-0,35	0,18
cariParent	0,16	0,05	0,55	0,36	0,98	-0,71	0,45
cariParentInternal	0,11	0,11	0,22	0,00	0,47	0,53	0,99
cekAda	0,12	0,10	0,57	0,34	0,59	-0,33	0,16
getChild	0,25	0,08	0,36	-0,06	0,35	0,55	0,44
getOrtu	0,15	0,00	0,36	0,52	0,36	-0,04	0,28
inOrder	0,16	-0,03	0,32	0,16	0,33	0,12	0,38
insertRoot	0,17	-0,07	0,48	0,22	0,41	-0,34	0,27
isEmpty	0,10	-0,02	0,27	0,64	0,32	-0,10	0,17
isiCari	0,10	-0,06	0,35	0,16	0,28	-0,22	0,15
pencarian	0,17	0,12	0,55	0,06	0,48	0,30	0,45
postOrder	0,16	-0,03	0,32	0,16	0,33	0,12	0,38
preOrder	0,16	-0,03	0,32	0,16	0,33	0,12	0,38
print	0,20	-0,08	0,31	0,19	0,31	0,03	0,33
printInfo	0,15	-0,06	0,21	0,59	0,18	-0,12	0
tambahAnak1	0,25	-0,18	0,45	0,01	0,43	0,13	0,36
tambahAnak2	0,25	-0,18	0,46	0,06	0,4	0,08	0,29
traverse	0,16	0,29	0,17	0,11	0,15	0,08	0,04
validasiGoldar	0,23	0,17	0,23	0,18	0,2	0,01	0,15
cekDarah	0,35	0,41	0,18	-0,14	0,16	0,11	0,18
cekDarahAuto	0,27	0,46	0,13	-0,08	0,11	0,24	0,23
getKemungkinan	0,16	-0,09	0,3	0,49	0,26	-0,01	0,21
setGoldar	0,15	-0,04	0,26	0,57	0,23	-0,16	0

Hasil perhitungan LSI atau nilai CSM pada sistem kakas bantu perhitungan nilai kopling dan selisih antara kedua sistem (lanjutan).

	selisih	**cekAda**	selisih	***getChild***	selisih	**getOrtu**	selisih	***inOrder***
main	0,15	0,12	0,10	0,22	0,11	0,15	0,00	0,12
cari	0,04	0,64	0,27	0,36	-0,06	0,43	0,45	0,35
cariParent	-0,18	0,6	0,39	0,36	0,04	0,42	0,40	0,35
cariParentIntern al	0,01	0,15	0,11	0,36	0,54	0,33	-0,01	0,42
cekAda	0,10	0,98	0,02	0,4	0,01	0,27	0,55	0,2
getChild	0,46	0,38	0,03	0,98	0,02	0,32	0,08	0,4
getOrtu	0,04	0,25	0,57	0,27	0,13	0,95	0,05	0,46
inOrder	0,07	0,21	0,22	0,35	0,09	0,48	0,07	0,96
insertRoot	-0,20	0,32	0,32	0,21	-0,05	0,44	0,18	0,36
isEmpty	0,05	0,27	0,57	0,23	0,04	0,25	0,53	0,24
isiCari	-0,09	0,06	0,42	0,05	0,09	0,39	0,19	0,27
pencarian	0,33	0,53	0,10	0,4	0,43	0,6	0,12	0,6
postOrder	0,07	0,21	0,22	0,35	0,09	0,48	0,07	0,83
preOrder	0,07	0,21	0,22	0,35	0,09	0,48	0,07	0,83
print	0,01	0,23	0,22	0,35	-0,02	0,41	0,07	0,72
printInfo	0,06	0,21	0,53	0,08	0,10	0,18	0,50	0,3
tambahAnak1	0,20	0,22	0,19	0,56	0,08	0,42	0,09	0,46
tambahAnak2	0,19	0,24	0,22	0,46	0,12	0,4	0,16	0,41
traverse	0,19	0,22	0,13	0,21	0,13	0,14	0,08	0,3
validasiGoldar	0,06	0,23	0,16	0,28	-0,07	0,22	0,26	0,17
cekDarah	0,09	0,1	0,02	0,22	0,14	0,19	-0,11	0,23
cekDarahAuto	0,12	0,02	0,13	0,19	0,28	0,14	-0,03	0,14
getKemungkinan	0,04	0,26	0,46	0,22	0,09	0,28	0,58	0,27
setGoldar	0,07	0,26	0,51	0,23	0,03	0,3	0,58	0,22

Hasil perhitungan LSI atau nilai CSM pada sistem kakas bantu perhitungan nilai kopling dan selisih antara kedua sistem (lanjutan).

	selisih	insertRoot	selisih	isEmpty	selisih	isiCari	selisih	*pencarian*	selisih
main	0,01	0,14	-0,04	0,05	0,03	0,05	-0,01	0,16	0,13
cari	0,13	0,49	0,21	0,29	0,62	0,37	0,14	0,55	0,06
cariParent	0,08	0,41	0,24	0,32	0,50	0,3	0,18	0,5	0,12
cariParentInternal	0,03	0,28	-0,21	0,22	0,00	0,17	-0,11	0,49	0,29
cekAda	0,23	0,31	0,33	0,26	0,58	0,06	0,42	0,49	0,14
getChild	0,04	0,18	-0,02	0,29	-0,02	0,05	0,09	0,45	0,38
getOrtu	0,09	0,37	0,25	0,23	0,55	0,35	0,23	0,59	0,13
inOrder	0,04	0,33	-0,01	0,24	0,26	0,26	0,09	0,62	0,00
insertRoot	-0,04	0,97	0,03	0,18	0,43	0,5	0,34	0,37	0,01
isEmpty	0,26	0,19	0,42	0,99	0,01	0,09	0,45	0,21	0,34
isiCari	0,08	0,47	0,37	0,09	0,45	0,98	0,02	0,33	0,01
pencarian	0,02	0,33	0,05	0,22	0,33	0,31	0,03	0,97	0,03
postOrder	0,17	0,33	-0,01	0,24	0,26	0,26	0,09	0,62	0,00
preOrder	0,17	0,33	-0,01	0,24	0,26	0,26	0,09	0,62	0,00
print	0,16	0,33	0,03	0,27	0,25	0,21	0,12	0,52	-0,02
printInfo	0,22	0,33	0,29	0,31	0,55	0,24	0,32	0,15	0,28
tambahAnak1	0,02	0,41	0,03	0,13	0,20	0,39	-0,15	0,46	0,12
tambahAnak2	0,04	0,44	0,06	0,06	0,28	0,37	-0,11	0,42	0,14
traverse	0,26	0,21	0,02	0,12	0,17	0,07	0,09	0,15	0,19
validasiGoldar	0,10	0,28	0,34	0,25	0,14	0,18	0,52	0,19	0,15
cekDarah	0,07	0,21	-0,17	0,02	0,03	0,06	0,00	0,18	0,06
cekDarahAuto	-0,11	0,16	-0,11	0,03	0,04	0,19	-0,11	0,15	0,16
getKemungkinan	0,29	0,23	0,29	0,21	0,64	0,27	0,33	0,24	0,37
setGoldar	0,20	0,33	0,31	0,07	0,56	0,13	0,41	0,25	0,30

Hasil perhitungan LSI atau nilai CSM pada sistem kakas bantu perhitungan nilai kopling dan selisih antara kedua sistem (lanjutan).

	postOrder	selisih	*preOrder*	selisih	*print*	selisih	printInfo	selisih	*tambahAnak1*
main	0,12	0,01	0,12	0,01	0,15	-0,03	0,07	0,02	0,19
cari	0,35	0,13	0,35	0,13	0,36	0,14	0,21	0,59	0,42
cariParent	0,35	0,08	0,35	0,08	0,35	0,11	0,17	0,56	0,41
cariParentInternal	0,42	0,03	0,42	0,03	0,36	-0,02	0	0,06	0,36
cekAda	0,2	0,23	0,2	0,23	0,23	0,22	0,19	0,55	0,2
getChild	0,4	0,04	0,4	0,04	0,4	-0,07	0,06	0,12	0,62
getOrtu	0,46	0,09	0,46	0,09	0,39	0,09	0,15	0,53	0,36
inOrder	0,83	0,17	0,83	0,17	0,71	0,17	0,28	0,24	0,44
insertRoot	0,36	-0,04	0,36	-0,04	0,37	-0,01	0,3	0,32	0,4
isEmpty	0,24	0,26	0,24	0,26	0,29	0,23	0,28	0,58	0,15
isiCari	0,27	0,08	0,27	0,08	0,23	0,10	0,21	0,35	0,35
pencarian	0,6	0,02	0,6	0,02	0,51	-0,01	0,13	0,30	0,41
postOrder	0,96	0,04	0,83	0,17	0,71	0,17	0,28	0,24	0,44
preOrder	0,83	0,17	0,96	0,04	0,71	0,17	0,28	0,24	0,44
print	0,72	0,16	0,72	0,16	0,97	0,03	0,35	0,28	0,36
printInfo	0,3	0,22	0,3	0,22	0,36	0,27	0,99	0,01	0,17
tambahAnak1	0,46	0,02	0,46	0,02	0,39	-0,01	0,13	0,18	0,98
tambahAnak2	0,41	0,04	0,41	0,04	0,35	0,00	0,13	0,20	0,78
traverse	0,3	0,26	0,3	0,26	0,29	0,27	0,27	0,10	0,13
validasiGoldar	0,17	0,10	0,17	0,10	0,14	0,11	0,05	0,25	0,22
cekDarah	0,23	0,07	0,23	0,07	0,3	-0,06	0,05	-0,02	0,24
cekDarahAuto	0,14	-0,11	0,14	-0,11	0,12	-0,10	0,06	0,00	0,22
getKemungkinan	0,27	0,29	0,27	0,29	0,23	0,24	0,1	0,63	0,19
setGoldar	0,22	0,20	0,22	0,20	0,19	0,16	0,16	0,48	0,28

Hasil perhitungan LSI atau nilai CSM pada sistem kakas bantu perhitungan nilai kopling dan selisih antara kedua sistem (lanjutan).

	selisih	*tambahAnak2*	selisih	*traverse*	selisih	*validasiGoldar*	selisih	*cekDarah*	selisih
main	-0,12	0,2	-0,13	0,16	0,29	0,29	0,11	0,33	0,43
cari	0,04	0,43	0,09	0,18	0,10	0,29	0,12	0,2	-0,16
cariParent	0,00	0,36	0,11	0,15	0,20	0,23	0,16	0,17	-0,05
cariParentInternal	0,20	0,3	0,18	0,05	0,18	0,12	0,09	0,21	0,06
cekAda	0,21	0,2	0,26	0,22	0,13	0,25	0,14	0,1	0,02
getChild	0,02	0,49	0,09	0,23	0,11	0,28	-0,07	0,28	0,08
getOrtu	0,15	0,32	0,24	0,14	0,08	0,22	0,26	0,19	-0,11
inOrder	0,04	0,4	0,05	0,35	0,21	0,19	0,08	0,24	0,06
insertRoot	0,04	0,42	0,08	0,22	0,01	0,33	0,29	0,25	-0,21
isEmpty	0,18	0,07	0,27	0,13	0,16	0,27	0,12	0,05	0,00
isiCari	-0,11	0,36	-0,10	0,09	0,07	0,19	0,51	0,19	-0,13
pencarian	0,17	0,38	0,18	0,16	0,18	0,21	0,13	0,2	0,04
postOrder	0,04	0,4	0,05	0,35	0,21	0,19	0,08	0,24	0,06
preOrder	0,04	0,4	0,05	0,35	0,21	0,19	0,08	0,24	0,06
print	0,02	0,33	0,02	0,31	0,25	0,15	0,10	0,29	-0,05
printInfo	0,14	0,17	0,16	0,32	0,05	0,08	0,22	0,12	-0,09
tambahAnak1	0,02	0,83	0,15	0,14	-0,06	0,24	-0,06	0,31	-0,23
tambahAnak2	0,20	0,98	0,02	0,15	-0,06	0,21	-0,04	0,27	-0,20
traverse	-0,05	0,13	-0,04	0,99	0,01	0,1	0,03	0,37	0,35
validasiGoldar	-0,04	0,14	0,03	0,08	0,05	0,98	0,02	0,34	-0,06
cekDarah	-0,16	0,22	-0,15	0,38	0,34	0,37	-0,09	0,98	0,02
cekDarahAuto	-0,10	0,19	-0,09	0,3	0,20	0,33	-0,05	0,84	0,03
getKemungkinan	0,18	0,2	0,19	0,12	0,07	0,13	0,29	0,22	-0,15
setGoldar	0,25	0,29	0,34	0,17	0,01	0,23	0,11	0,26	-0,24

Hasil perhitungan LSI atau nilai CSM pada sistem kakas bantu perhitungan nilai kopling dan selisih antara kedua sistem (lanjutan).

	cekDarahAuto	selisih	**getKemungkinan**	selisih	**setGoldar**	selisih
main	0,29	0,44	0,1	-0,03	0,18	-0,07
cari	0,13	-0,08	0,31	0,48	0,29	0,54
cariParent	0,12	0,03	0,27	0,44	0,23	0,53
cariParentInternal	0,24	0,11	0,27	-0,02	0	0,07
cekAda	0,04	0,11	0,24	0,48	0,25	0,52
getChild	0,26	0,21	0,21	0,10	0,19	0,07
getOrtu	0,14	-0,03	0,25	0,61	0,28	0,60
inOrder	0,15	-0,12	0,26	0,30	0,23	0,19
insertRoot	0,2	-0,15	0,26	0,26	0,34	0,30
isEmpty	0,05	0,02	0,21	0,64	0,08	0,55
isiCari	0,21	-0,13	0,28	0,32	0,15	0,39
pencarian	0,17	0,14	0,24	0,37	0,25	0,30
postOrder	0,15	-0,12	0,26	0,30	0,23	0,19
preOrder	0,15	-0,12	0,26	0,30	0,23	0,19
print	0,12	-0,10	0,21	0,26	0,19	0,16
printInfo	0,14	-0,08	0,11	0,62	0,21	0,43
tambahAnak1	0,27	-0,15	0,21	0,16	0,29	0,24
tambahAnak2	0,22	-0,12	0,21	0,18	0,32	0,31
traverse	0,27	0,23	0,09	0,10	0,17	0,01
validasiGoldar	0,34	-0,06	0,13	0,29	0,18	0,16
cekDarah	0,79	0,08	0,22	-0,15	0,22	-0,20

getKemungkinan	0,24	-0,15	0,99	0,01	0,07	0,63
setGoldar	0,2	-0,17	0,06	0,64	0,98	0,02

Hasil perhitungan akurasi

Rumus Akurasi = $\frac{\text{jumlah data yang benar}}{\text{jumlah data keseluruhan}} \times 100$

Total Data = 576

Total data tidak valid toleransi 0,05 =	433	Total data valid toleransi 0,05 =	143
Total data tidak valid toleransi 0,10 =	326	Total data valid toleransi 0,10 =	250
Total data tidak valid toleransi 0,20 =	174	Total data valid toleransi 0,20 =	402

Akurasi jika nilai toleransi 0,05 =	24,8
Akurasi jika nilai toleransi 0,10 =	43,4
Akurasi jika nilai toleransi 0,20 =	69,8

LAMPIRAN 2 PEMODELAN *CLASS DIAGRAM*

Pemodelan *Class Diagram*