KAKAS BANTU PERHITUNGAN NILAI KOPLING MENGGUNAKAN METRIK *COGNITIVE WEIGHTED COUPLING BETWEEN OBJECT* (CWCBO)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Muhammad Ubaidillah
125150201111072

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

PENGESAHAN

KAKAS BANTU PERHITUNGAN NILAI KOPLING MENGGUNAKAN METRIK *COGNITIVE WEIGHTED COUPLING BETWEEN OBJECT* (CWCBO)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Muhammad Ubaidillah
NIM: 125150201111072

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 2015

Muhammad Ubaidillah

NIM: 125150201111072

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas berkah, rahmat serta hidayah-Nya sehingga penulis dapat menyelesaikan laporan Skripsi dengan judul "Kakas Bantu Perhitungan NIlia Kopling Menggunakan Mertik *Cognitive Weighted Coupling Between Object* (CWCBO)" dengan baik dan tepat waktu. Keberhasilan tersebut tak lepas dari bantuan dan dukungan dari berbagai pihak baik secara moril maupun materil. Oleh karena itu dalam kesempatan ini penulis ingin mengucapkan terima kasih kepada:

1. Bapak Fajar Pradana, S.ST, M.Eng selaku dosen pembimbing 1.
2. Bapak Bayu Priyambadha, S.Kom, M.Kom selaku dosen pembimbing 2.
3. Bapak Adam Hendra Brata, S.Kom, M.t, M.Sc yang telah banyak memberikan saran-saran selama proses pengerjaan skripsi.
4. Seluruh Dosen FILKOM UB yang telah memberikan ilmu serta bimbingan kepada penulis selama masa perkuliahan berlangsung.
5. Bapak Achmad Rosyim dan Ibu Uswatun Hasanah, orang tua penulis dan seluruh keluarga atas dukungan dan do'a yang telah diberikan.
6. Mega Tricitanya selaku pemberi ide judul skripsi yaitu mengenai pemberian paper dengan judul "*Coupling Complexity Metric: A Cognitive Approach*".
7. Moh Arsyad Mubarak yang telah membantu dalam proses pembuatan program, terutama untuk mengajari penggunaan *library* Spoon.
8. Rayan Suryadikara yang telah membantu menerjemahkan (*translate*) abstrak kedalam bahasa inggris.
9. Sahabat serta teman-teman penulis beserta seluruh Civitas Akademika FILKOM UB yang telah memberikan berbagai macam dukungan, bantuan dan motivasi kepada penulis sehingga dapat menyelesaikan skripsi ini.

Penulis menyadari bahwa dalam penyusunan laporan ini masih banyak kekurangan baik format laporan maupun isinya. Oleh karena itu, kami mengharapkan saran dan kritik yang membangun dari para pembaca guna perbaikan laporan selanjutnya. Semoga laporan ini dapat memberikan manfaat bagi semua pihak, Aamiin.

Malang, 23 Agustus 2016

Muhammad Ubaidillah
ubaidillah.m26@gmail.com

ABSTRAK

Konsep *Object Oriented Programming* (OOP) merupakan pemrograman yang dibangun dengan berpusat pada beberapa objek. Dengan konsep OOP dapat dilakukan pengukuran kualitas perangkat lunak dengan melalui kemungkinan hubungan antar beberapa objek atau kopling. Perangkat lunak yang baik adalah perangkat lunak yang memiliki desain yang baik, salah satu ciri-cirinya adalah memiliki nilai kopling yang rendah. Nilai kopling yang tinggi dapat menyebabkan desain perangkat lunak yang semakin kompleks dan buruk, sehingga akan mengakibatkan sulit untuk dipahami, terutama ketika dilakukan *maintenance*. Untuk meningkatkan kualitas perangkat lunak perlu mengontrol nilai kopling agar dapat meminimalkan kompleksitas perangkat lunak. Namun perhitungan nilai kopling secara manual pada jumlah perangkat lunak yang banyak akan membutuhkan waktu dan sumber daya yang besar. Oleh karena itu dibuatlah kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO). Metrik CWCBO merupakan metrik kopling yang didasarkan pada bobot pemahaman untuk menghitung perbedaan jenis kopling oleh berbagai peneliti. Kakas bantu tersebut dibangun dengan bahasa pemrograman utama Java, *library* Spoon untuk menganalisis *source code*, dan *library* JFreeChart untuk menampilkan grafik. Dari hasil pengujian akurasi sistem pada 5 program inputan berbahasa Java didapatkan akurasi sebesar 100%. Pengujian akurasi dilakukan dengan membandingkan perhitungan secara manual dan dengan sistem.

Kata kunci: OOP, Kopling, CWCBO, *Source Code*

ABSTRACT

Object Oriented Programming (OOP) is a programming concept that was built with a focus on some objects. OOP concepts are able to measuring the quality of software through a possible connection between multiple objects or coupling. A good software is a software that has a good design, one of its characteristics is having a low value of coupling. High coupling value could cause an increasingly complex and bad software design, and so will become to difficult to understand, especially when maintenance carried out. To improve the software's quality, it's necessary to control coupling's value in order to minimize the complexity of the software. However, the manually calculation of coupling's value on the sizeable amount of software will take an ample time and resources. Therefore, a tool to calculate coupling's value were made using Cognitive Weighted Coupling Between Object (CWCBO) metric. CWCBO metric is a coupling metric that based on the understanding weight to count the different types of coupling by various researchers. A tool was built with primary component from Java programming language, Spoon library to analyze source code and JFreeChart library to display charts. From testing results the system's accuracy on five Java program inputs obtained a 100% of accuracy. Testing is done by comparing the accuracy of the calculations manually and from system.

Keyword: *OOP, Coupling, CWCBO, Source Code*

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xii
DAFTAR PERSAMAAN xiii
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 3
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan masalah 4
1.6 Sistematika penulisan 4
1.7 Rencana dan Jadwal Penelitian 5
BAB 2 LANDASAN KEPUSTAKAAN 6
2.1 Kajian pustaka 6
2.2 Dasar Teori 7
2.2.1 Metrik Kopling 7
2.2.2 Source Code 14
2.2.3 Spoon 15
2.2.4 JFreeChart 17
2.2.5 Konsep Dasar Rekayasa Perangkat Lunak 17
2.2.6 Model Pengembangan *Waterfall* 18
2.2.7 Konsep Dasar Berorientasi Objek 19
2.2.8 *Unified Modelling Language* (UML) 20
2.2.9 Pengujian Perangkat Lunak 25
BAB 3 METODOLOGI 28
3.1 Studi literatur 29

3.2 Rekayasa kebutuhan 29
3.3 Perancangan sistem 30
3.4 Implementasi sistem 30
3.5 Pengujian dan Analisis 30
3.6 Pengambilan Kesimpulan 31
BAB 4 REKAYASA KEBUTUHAN 32
4.1 Proses Rekayasa Kebutuhan 32
4.1.1 Gambaran Umum Sistem 32
4.1.2 Identifikasi Aktor 33
4.1.3 Spesifikasi Kebutuhan 34
4.2 Model Kebutuhan 36
4.2.1 Pemodelan *Use Case Diagram* 36
4.2.2 Skenario *Use Case* 36
BAB 5 PERANCANGAN DAN IMPLEMENTASI 39
5.1 Perancangan 39
5.1.1 Perancangan Arsitektur 39
5.1.2 Perancangan *Sequence Diagram* 40
5.1.3 Perancangan *Class Diagram* 43
5.1.4 Perancangan Alur Parser *Source Code* 44
5.1.5 Perancangan Komponen 46
5.1.6 Perancangan Antarmuka 52
5.2 Implementasi 53
5.2.1 Spesifikasi Sistem 53
5.2.2 Batasan Implementasi 54
5.2.3 Implementasi Antarmuka 54
BAB 6 PENGUJIAN 58
6.1 Pengujian 58
6.1.1 Pengujian Unit 58
6.1.2 Pengujian Integrasi 62
6.1.3 Pengujian Validasi 64
6.1.4 Pengujian Akurasi 67
6.2 Analisis 70

6.2.1 Analisis Hasil Pengujian Unit 70
6.2.2 Analisis Hasil Pengujian Integrasi 70
6.2.3 Analisis Hasil Pengujian Validasi 70
6.2.4 Analisis Hasil Pengujian Akurasi 71
6.3 Pembahasan Hasil 71
BAB 7 PENUTUP 73
7.1 Kesimpulan 73
7.2 Saran 73
DAFTAR PUSTAKA 74

DAFTAR TABEL

Tabel 1.1 Jadwal penelitian 5
Tabel 2.1 Contoh Kode Program dengan Beberapa Tipe Kopling 9
Tabel 2.2 Kategori Waktu Pemahaman 13
Tabel 2.3 Faktor Bobot pada Setiap Kopling 14
Tabel 2.4 Komponen *Use Case Diagram* 21
Tabel 2.5 Komponen *Sequence Diagram* 22
Tabel 4.1 Identifikasi Aktor 33
Tabel 4.2 Daftar Kebutuhan Fungsional 34
Tabel 4.3 Daftar Kebutuhan Non Fungsional 35
Tabel 4.4 Pemetaan kebutuhan fungsional dengan *use case* 36
Tabel 4.5 Skenario *Use Case* Cari Berkas 37
Tabel 4.6 Skenario *Use Case* Hitung Kopling 37
Tabel 4.7 Skenario *Use Case* Tampilkan Grafik 38
Tabel 5.1 Pemetaan kebutuhan fungsional dengan *use case* 40
Tabel 5.2 Atribut *Class* Controller 46
Tabel 5.3 Algoritma *method* parseFile() 47
Tabel 5.4 Algoritma *method* setDataChart() 47
Tabel 5.5 Atribut Class Spoon_metamodel 48
Tabel 5.6 Algoritma *method* checkMethodParent() 49
Tabel 5.7 Algoritma *method* searchData() 49
Tabel 5.8 Algoritma *method* searchInvocationMethod() 49
Tabel 5.9 Algoritma *method* checkStatement() 50
Tabel 5.10 Atribut Class ResultCoupling 51
Tabel 5.11 Algoritma *method* countCBO() 51
Tabel 5.12 Algoritma *method* countCWCBO() 52
Tabel 5.13 Spesifikasi Perangkat Keras 53
Tabel 5.14 Spesifikasi Perangkat Lunak 53
Tabel 6.1 Algoritma checkArrayList 58
Tabel 6.2 Kasus Uji *Method* checkArrayList 60
Tabel 6.3 Algoritma *Method* readFile 60

Tabel 6.4 Kasus Uji *Method* readFile 61
Tabel 6.5 Pengujian Integrasi Antar Modul 62
Tabel 6.6 Uji Cari Berkas 64
Tabel 6.7 Uji Cari Berkas Alternatif 1 64
Tabel 6.8 Uji Hitung Kopling 65
Tabel 6.9 Uji Hitung Kopling Alternatif 1 65
Tabel 6.10 Uji Tampilkan Grafik 66
Tabel 6.11 Uji Tampilkan Grafik Alternatif 1 66
Tabel 6.12 Daftar Kode Program Inputan 67
Tabel 6.13 Perbandingan Nilai Perhitungan Secara Manual dan Otomatis 69
Tabel 6.14 Hasil pengujian fungsional 70

DAFTAR GAMBAR

Gambar 2.1 Rata-rata Waktu Pemahaman Setiap Kode Program 12
Gambar 2.2 Gambaran Umum Spoon 15
Gambar 2.3 Bagian Struktural dari Spoon Java 5 metamodel 16
Gambar 2.4 Bagian kode dari Spoon Java 5 Metamodel 16
Gambar 2.5 Model Pengembangan Sistem *Waterfall* 18
Gambar 2.6 Contoh *use case diagram* 22
Gambar 2.7 Contoh *sequence diagram* 24
Gambar 2.8 Contoh *class diagram* 25
Gambar 2.9 Contoh notasi *flow graph* 27
Gambar 3.1 Alur metodologi penelitian 28
Gambar 4.1 Diagram rekayasa kebutuhan 32
Gambar 4.2 *Use Case Diagram* 36
Gambar 5.1 Arsitektur Sistem Perhitungan Nilai Kopling 39
Gambar 5.2 *Sequence Diagram* Cari Berkas 40
Gambar 5.3 *Sequence Diagram* Hitung Kopling 41
Gambar 5.4 *Sequence Diagram* Tampilkan Grafik 42
Gambar 5.5 *Class Diagram* Perhitungan Nilai Kopling 43
Gambar 5.6 Perancangan Antarmuka 52
Gambar 5.7 Antarmuka Halaman Utama 54
Gambar 5.8 Antarmuka Hasil Memasukkan Berkas FIle 55
Gambar 5.9 Antarmuka hasil perhitungan nilai kopling 56
Gambar 5.10 Antarmuka Hasil Tampilkan Grafik 57
Gambar 6.1 *Flow Graph Method* checkArrayList 59
Gambar 6.2 *Flow Graph Method* readFile 61
Gambar 6.3 Hasil Pengujian Integrasi Modul readFile dan showFileName 63
Gambar 6.4 Grafik Perbandingan Nilai Kopling CBO dan CWCBO 68

DAFTAR PERSAMAAN

Persamaan 2-1 13

BAB 1 PENDAHULUAN

1.1 Latar belakang

Rekayasa Perangkat Lunak (RPL) merupakan suatu cabang ilmu profesi yang membahas tentang teknik-teknik dalam mengembangkan perangkat lunak, mulai dari perencanaan, pembuatan, pengujian hingga pemeliharaan. Rekayasa perangkat lunak merupakan tugas yang rumit dan kompleks (Aloysius & Arockiam, 2012). Struktur perangkat lunak yang kompleks bisa berdampak pada kualitas perangkat lunak yang buruk, karena semakin sulit untuk dipahami, tidak hanya sulit dalam proses pengembangannya melainkan juga pada proses *maintenance*. Sehingga banyak para pengembang berupaya menciptakan teknik-teknik untuk mempermudah dalam proses pengembangan dan *maintenance*. Salah satu upayanya adalah menerapkan model-model pengembangan perangkat lunak, seperti pendekatan berorientasi objek dan pendekatan terstruktur. Selain itu, para pengembang juga berupaya dalam meningkatkan kualitas perangkat lunak. Salah satunya menciptakan metrik perangkat lunak yang bisa menjadi suatu solusi dalam memprediksi kualitas sistem, dan menunjuk ke area masalah yang dapat diatasi sebelum sistem diterbitkan (Aloysius & Arockiam, 2012). Metrik mencoba untuk mengukur aspek tertentu dari sistem perangkat lunak (Aloysius & Arockiam, 2012).

Dalam praktek pembangunan perangkat lunak terutama pada tahap implementasi banyak para pengembang menggunakan pemrograman berbasis objek. Bahkan beberapa tahun terakhir, sebagian besar industri perangkat lunak mengadopsi bahasa pemrograman berbasis objek seperti java, C++ dan php (Dallal & Briand, 2010; Misra & Akman, 2008). Sehingga lebih dari dua dekade terakhir, penggunaan teknik berbasis objek lebih dominan dibandingkan dengan penggunaan teknik terstruktur untuk mengembangkan perangkat lunak (Misra & Akman, 2008).

Perangkat lunak yang dibangun dengan menggunakan konsep *Object Oriented Programing* (OOP) akan dilakukan pemrograman yang berpusat pada beberapa objek. Pendekatan Berorientasi objek dibangun berdasarkan *class, subclass* dan objek yang berisi unsur-unsur seperti *attribute*, *method* dan pesan (Misra & Akman, 2008). Unsur-unsur tersebut diidentifikasikan dalam deklarasi kelas dan bertanggung jawab pada kompleksitas kelas (Misra & Akman, 2008).

Penggunaan teknik berorientasi objek menjadi langkah penting dalam mengukur kualitas suatu perangkat lunak dengan mengendalikan konstruksi berorientasi objek (Aloysius & Arockiam, 2012). Dengan pemanfaatan konsep OOP dapat dilakukan pengukuran kualitas suatu perangkat lunak berdasarkan kemungkinan hubungan antar *atibute* dan *method* yang terdapat pada masing-masing *class* (Lestari, 2015). Proses ini dapat dilakukan sebagai alat dalam memutuskan kualitas suatu perangkat lunak pada tahap implementasi berdasarkan nilai kopling antar objek. Hal ini yang mendasari dilakukannya perhitungan nilai kopling berdasarkan relasi antar objek pada perangkat lunak.

Perangkat lunak yang baik adalah perangkat lunak yang memiliki desain yang baik. Salah satu ciri-ciri desain yang baik adalah memiliki nilai kopling yang rendah. Nilai kopling mempunyai dampak yang negatif pada kualitas perangkat lunak (Yadav & Khan, 2009 dalam Aloysius & Arockiam, 2012). Semakin tinggi tingkat kopling pada suatu perangkat lunak, maka semakin buruk kualitas perangkat lunak tersebut. Apalagi kopling kelas yang tinggi tidak begitu diharapkan karena dianggap sebagai desain yang buruk dan dapat menyebabkan kesulitan dalam memahami kelas (Stevens, et al., 1974). Perangkat lunak yang sulit dipahami juga menyebabkan kesulitan *developer* untuk melakukan *maintenance* pada sistem. Namun, jika Perangkat lunak tersebut memiliki desain yang baik, maka akan lebih mudah untuk dipahami. Selain itu perangkat lunak dengan desain yang baik juga akan mudah dilakukan pengelolaan. Untuk meminimalkan kompleksitas dari suatu perangkat lunak diperlukan kontrol terhadap kopling antar objek karena kopling sangat berkaitan erat degan kualitas perangkat lunak (Aloysius & Arockiam, 2012). Sehingga untuk meningkatkan kualitas perangkat lunak harus mempertimbangkan nilai kopling untuk meminimalkan kompleksitas perangkat lunak (Yadav & Khan, 2009 dalam Aloysius & Arockiam, 2012).

Berdasarkan paparan sebelumnya, nilai kopling dapat menentukan tingkat kualitas dari desain perangkat lunak, maka diperlukan suatu perhitungan nilai kopling dengan harapan dapat menentukan kualitas perangkat lunak berdasarkan relasi-relasi antar objek. Selain itu, karena nilai kopling juga dapat berpengaruh pada tingkat kesulitan program untuk dipahami, maka juga diperlukan suatu perhitungan nilai kopling yang dapat mendefinisikan tingkat kesulitan dalam memahami hubungan antar kelas pada suatu program. Oleh karena itu, dipilihlah perhitungan kualitas desain berorientasi objek menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO). Metrik tersebut dipilih karena menurut Aloysius dan Arockiam (2012), belum pernah ada metrik kopling yang didasarkan pada bobot pemahaman untuk menghitung perbedaan jenis kopling dari berbagai peneliti, sehingga metrik CWCBO yang dihitung berdasarkan bobot pemahaman diharapkan dapat mendefinisikan kopling pada berbagai tingkatan. Dan metrik tersebut diharapkan dapat mempermudah para developer untuk membangun perangkat lunak yang lebih mudah untuk dipahami, terutama ketika dilakukan *maintenance*.

Adanya perhitungan tersebut dirasa masih kurang efektif dan efisien jika dilakukan perhitungan secara manual dengan jumlah perangkat lunak yang banyak. Karena selain membutuhkan waktu yang lama, proses tersebut juga membutuhkan sumber daya yang besar. Apalagi dalam menentukan nilai kopling tersebut dibutuhkan sumberdaya manusia yang benar-benar paham dengan model perhitungannya, sedangkan untuk mendapatkan sumberdaya tersebut tidaklah mudah.

Oleh karena itu, berdasarkan permasalahan yang telah dipaparkan, dibutuhkan suatu program yang dapat melakukan otomatisasi perhitungan nilai kopling, sehingga diharapkan dapat membantu para pengembang dalam

menentukan tingkat kualitas perangkat lunak secara efektif dan efisien. Perhitungan kualitas desain *Object-Oriented* ini menggunakan metrik kopling *Cognitive Weighted Coupling Between Object* (CWCBO) dan dibangun dengan menggunakan bahasa pemrograman Java. Dengan adanya sistem perhitungan kualitas ini nanti diharapkan dapat menjadi alternatif bagi *user* dalam mengembangkan perangkat lunak yang lebih baik. Terutama *source code* yang telah dibuat *user* akan menjadi lebih baik dan mudah untuk dipahami.

1.2 Rumusan masalah

Berdasarkan paparan latar belakang tersebut, permasalahan yang akan dikaji dalam skripsi ini, antara lain:

1. Bagaimana membangun kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object (CWCBO)?*
2. Bagaimana menguji keakuratan sistem kakas bantu perhitungan kopling pada sebuah program?

1.3 Tujuan

Tujuan yang ingin dicapai dari penelitian ini adalah sebagai berikut :

1. Mengetahui kualitas sebuah *source code* berdasarkan tingkat kesulitan untuk dipahami.
2. Dapat menghitung atau mengukur secara otomatis nilai kopling *source code* perangkat lunak di tahap implementasi dengan menggunakan *source code* dari beberapa *class*.
3. Menerapkan metode *Cognitive Weighted Coupling Between Object* (CWCBO) ke dalam sebuah sistem untuk mengetahui kualitas *source code* perangkat lunak menggunakan bahasa pemrograman Java.

1.4 Manfaat

Penelitian ini diharapkan dapat memberikan manfaat antara lain sebagai berikut :

1. Membantu *user requirement* dalam menerapkan pemrograman berorientasi objek agar mengetahui indikator awal kualitas atau kejadian kesalahan pada tahap implementasi.
2. Membantu penelitian-penilitan selanjutnya yang dapat meningkatkan tignkat akurasi dalam pengembangan di bidang ilmu Rekayasa Perangkat Lunak (RPL).
3. Memberikan referensi untuk perkembangan dan memajukan tingkat akurasi dalam pengembangan di bidang ilmu Rekayasa Perangkat Lunak (RPL).

1.5 Batasan masalah

Berdasarkan rumusan masalah yang telah dipaparkan, agar pembahasan lebih terfokus maka diperlukan pembatasan masalah sebagai berikut :

1. Masukan kode program berupa *source code* dengan bahasa Java, dan terdapat lebih dari satu *class* yang saling berkaitan atau mempunyai desain berbasis objek.
2. Metrik kualitas yang digunakan pada penelitian ini adalah metode *Cognitive Weighted Coupling between Object* (CWCBO).
3. Bahasa pemrograman yang digunakan dalam pengembangan sistem ini adalah bahasa pemrograman Java.

1.6 Sistematika penulisan

Sistematika penulisan skripsi ini adalah sebagai berikut :

BAB I PENDAHULUAN

Bab ini berisi latar belakang dibuatnya sistem, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan dalam membangun Kakas Bantu Perhitungan Nilai Kopling Menggunakan Metrik *Cognitive Weighted Coupling Between Object (CWCBO)* dan rencana serta jadwal penelitian.

BAB II LANDASAN KEPUSTAKAAN

Bab ini membahas mengenai konsep dasar dan teori-teori yang mendasari pembuatan sistem kakas bantu perhitungan kopling dengan menggunakan metrik CWCBO.

BAB III METODOLOGI

Bab ini menguraikan tentang metode dan langkah yang digunakan dalam penelitian yang terdiri dari studi literatur, rekayasa kebutuhan, perancangan dan implementasi sistem perangkat lunak, pengujian dan analisis, serta pengambilan kesimpulan.

BAB IV REKAYASA KEBUTUHAN

Bab ini menguraikan tentang analisis kebutuhan sistem kakas bantu perhitungan kopling dengan menggunakan metrik *Cognitive Weighted Coupling between Object* (CWCBO) yang mencakup gambaran umum sistem, spesifikasi, validasi dan verifikasi, dan manajemen kebutuhan serta pemodelan hasil rekayasa kebutuhan.

BAB V PERANCANGAN DAN IMPLEMENTASI

Bab ini menguraikan tentang perancangan dan implementasi pembangunan kakas bantu perhitungan nilai kopling dengan menggunakan metrik CWCBO berdasarkan kebutuhan sistem.

BAB VI PENGUJIAN DAN ANALISIS

Bab ini menguraikan tentang proses dan hasil pengujian terhadap sistem yang telah diimplementasikan.

BAB VII PENUTUP

Bab terakhir ini menguraikan tentang kesimpulan yang diperoleh berdasarkan hasil penelitian dan memberikan saran untuk pengembangan sistem selanjutnya agar menjadi lebih baik.

1.7 Rencana dan Jadwal Penelitian

Penelitian direncanakan akan dilaksanakan selama enam bulan yang dimulai dari bulan Februari 2016 dan akan selesai pada bulan Juli 2016. Rincian rencana jadwal penelitian dicantumkan dalam Tabel 1.1 berikut,

Tabel 1.1 Jadwal penelitian

No	Kegiatan	Bulan 2015-2016																							
		Februari				Maret				April				Mei				Juni				Juli			
1.	Telaah literatur																								
2.	Penyusunan Proposal																								
3.	Penelitian																								
4.	Rekayasa kebutuhan																								
5.	Perancangan sistem																								
6.	Implementasi sistem																								
7.	Pengujian sistem																								
8.	Penyusunan Laporan																								

BAB 2 LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan terdiri dari kajian pustaka dan dasar teori. Kajian pustaka membahas penelitian terdahulu dari jurnal A. Aloysius dan L. Arockiam yang menjadi acuan pengembangan skripsi. Penelitian terdahulu dengan judul jurnal "*Coupling Complexity Metric: A Cognitive Approach*" menjelaskan bahwa semakin tinggi nilai kopling pada perangkat lunak dengan pemrograman berorientasi objek memiliki kualitas yang semakin buruk. Penelitian ini membuat program atau sistem untuk mengotomatisasi nilai kualitas dari pemrograman berorientasi objek pada *source code* dengan menggunakan metrik kopling. Dasar teori membahas teori yang dijadikan landasan pembangunan sistem untuk menilai atau mengukur kualitas suatu program berdasarkan kopling antar objek. Dasar teori yang digunakan meliputi *source code*, metrik kopling, konsep dasar rekayasa perangkat lunak, konsep dasar berorientasi objek, dan *Unified Modelling Language* (UML).

2.1 Kajian pustaka

Kajian pustaka pada penelitian ini membahas penelitian sebelumnya dengan ditulis oleh A. Aloysius dan L. Arockiam dengan judul jurnal "*Coupling Complexity Metric: A Cognitive Approach*". Penelitian tersebut membahas tentang rekayasa perangkat lunak dengan tujuan untuk mengembangkan teknik dan alat untuk meningkatkan kualitas perangkat lunak yang stabil dan mudah dalam pemeliharaannya. Untuk menilai kualitas suatu perangkat lunak selama proses pengembangan dilakukan perhitungan nilai kopling secara otomatis dengan adanya alat bantu. Dari jurnal yang diusulkan Aloysius (2012) menjelaskan bahwa telah banyak metrik kopling untuk mengukur kopling antar objek atau kelas, namun tidak ada metrik kopling dengan bobot pemahaman untuk mengukur perbedaan jenis kopling oleh berbagai peneliti. Jadi Aloysius harapkan metrik kopling *Cognitive Weighted Coupling Between Object* (CWCBO) yang didasarkan pada bobot pemahaman dapat mendefinisikan kopling pada berbagai tingkatan.

Metrik CWCBO berisi berbagai macam kopling, yaitu *Control Coupling* (CC), *Global Data Coupling* (GDC), *Internal Data Coupling* (IDC), *Data Coupling* (DC) dan *Lexical Content Coupling* (LCC), dimana jenis-jenis kopling tersebut Aloysius tinjau dari usulan Edward Berard (Edward, 1993 dalam Aloysius & Arockiam, 2012). Bobot pemahaman pada jenis kopling tersebut dikalibrasikan melelui percobaan psikologis terhadap mahasiswa sarjana dan magister (Aloysius & Arockiam, 2012). Dengan adanya metrik tersebut diharapkan dapat mencerminkan kopleksitas dari sistem *Object-Oriented* (OO) secara nyata. Sehingga, hasil dari pengukuran akan mampu menunjukkan sebuah objek mempunyai keterkaitan yang sangat erat pada method di kelas lain (Aloysius & Arockiam, 2012). Hal ini dapat menjadi sebuah alternatif yang lebih dalam mengukur kualitas suatu sistem berdasarkan perhitungan kopling antar objek secara lebih nyata. Prosedur dalam penelitian ini meggunakan perhitungan manual tanpa menggunakan *tools* yang mendukung pada proses kalkulasinya.

2.2 Dasar Teori

Pada bagian ini akan dibahas menegenai teori-teori yang berkaitan dengan penelitian ini. Teori yang berkaitan diantaranya adalah metrik kopling, *source code*, Spoon, JFreeChrat, konsep dasar rekayasa perangkat lunak, konsep dasar berorientasi objek, *Unified Modelling Language* (UML) dan Pengujian perangkat lunak.

2.2.1 Metrik Kopling

Metrik kopling menunjukkan hubungan dan interaksi elemen-elemen antar objek pada suatu perangkat lunak. Semakin tinggi nilai kopling, maka hubungan antar objek atau modul pada suatu perangkat lunak semakin kuat dan semakin kompleks. Nilai kopling yang tinggi dapat berakibat pada semakin buruknya kualitas desain perangkat lunak, karena semakin banyak pertukaran pesan antar objek (Coad & Yourdon, 1991 dalam Misra & Akman, 2008).

Stevens dkk (Stevens, et al., 1974 dalam Aloysius & Arockiam, 2012), mengusulkan konsep kopling ke dalam desain struktur. Ia mendefinisikan bahwa kopling menjadi ukuran kekuatan asosiasi yang didirikan oleh koneksi dari satu modul ke yang lainnya. Dengan demikian dapat disimpulkan bahwa kopling kelas yang tinggi tidak diinginkan karena dianggap sebagai desain yang buruk dan dapat menyebabkan kesulitan memahami kelas.

2.2.1.1 *Coupling Between Object* (CBO)

Coupling between Object (CBO) merupakan metrik perhitungan nilai kopling yang pertama kali diperkenalkan oleh Chidamber dan Kamerer (CK). CK mendefinisikan bahwa CBO untuk kelas adalah perhitungan jumlah dari kelas-kelas yang saling berpasangan (Chidamber & Kemerer, 1994). CBO berkaitan dengan pandangan bahwa sebuah objek berpasangan dengan objek yang lainnya jika salah satu dari mereka bertindak pada sisi yang lainnya, yaitu *method* satu menggunakan *method* atau *instans variable* yang lain (Chidamber & Kemerer, 1994).

Definisi tersebut flexibel dalam tiga cara yaitu (Aloysius & Arockiam, 2012):

a. Yang mana arah dari kelas yang terhubung dengan lainnya.
b. Bagaimana kelas sebenarnya digabungkan ke yang lainnya.
c. Nilai untuk memberikan hubungan kopling untuk membedakan kekuatannya dari kopling lain.

Aloysius dan Arockiam (2012) berpendapat bahwa, metrik CBO hanya menghitung kopling luar dari kelas lain dengan cara menggabungkan dua kelas. Dan nilai yang diberikan untuk kopling ditetepkan dengan nilai 1. Sehingga mereka, mengusulkan penelitian dengan berbagai nilai-nilai lain juga. Seperti yang diusulkan oleh Edward Berard (Edward, 1993 dalam Aloysius & Arockiam, 2012), ia mengusulkan berbagai macam kopling yang didefinisikan sebagai berikut:

1. *Control Coupling* (CC)

   Suatu variabel yang melewati penanda kontrol antar modul, sehingga satu modul mengontrol pengurutan langkah-langkah dari modul lain. *Control Coupling* terjadi antar modul dimana ketika data yang lewat dapat mempengaruhi logika internal pada salah satu modul, misalnya penanda atau *switch* (Borysowich, 2007).

   Dalam sistem berorientasi objek, *Control Coupling* terjadi ketika sebuah objek menerima pesan dari objek lain dan objek penerima merespon pesan secara berbeda berdasarkan informasi kontrol yang terkandung dalam pesan (Borysowich, 2007). Contoh *Control Coupling* pada pemrograman berorientasi objek adalah *method* dimana didalamnya terdapat pengkodisian seperti mengembalikan kode kesalahan dengan nilai nol bila tidak ada kesalahan yang ditemui penanganan pesan dari suatu modul percabangan.

   Control Coupling dapat dihitung berdasarkan jumlah modul yang berisi pengkondisian sebagai pengontrol pada suatu percabangan. Kondisi ini biasanya bersifat *true* atau *false*. Bernilai *true* apabila kondisi utama dalam suatu percabangan atau perulangan menggunakan simbol "!=" atau symbol pertidaksamaan yang lainnya. Sedangkan *false* terjadi apabila, beberapa penggunaan parameter kondisi utama dalam percabangan menggunakan simbol "=", "<", ">" atau simbol persamaan yang lainnya. Dengan syarat parameter kondisi tersebut menggunakan salah satu *Global Data Coupling* yang ada.

2. *Global Data Coupling* (GDC)

   Dua atau lebih modul yang menggunakan struktur data global yang sama. Kondisi ini terjadi ketika pada objek/kelas *parent* terdapat suatu variabel global dimana variabel tersebut juga digunakan pada kelas atau objek yang lain. Jumlah GDC dapat diketahui dengan menghitung jumlah variabel global.

3. *Internal Data Coupling* (IDC)

   Suatu modul yang secara langsung memodifikasi data lokal dari modul yang lain. Pada sistem berorientasi objek, *Internal Data Coupling* dapat terjadi ketika suatu objek memanggil suatu method yang sama pada objek yang berbeda, dimana *method* yang dipanggil memiliki modifikasi yang berbeda terhadap variabel global yang sama di setiap objek. Jumlah IDC dapat diketahui dengan menghitung jumlah variabel global yang dipanggil oleh suatu objek, diamana variabel tersebut telah mengalami modifikasi oleh dua atau lebih objek lain yang berbeda.

4. Data Coupling (DC)

   Output dari satu modul merupakan suatu inputan pada modul yang lainnya menggunakan daftar parameter untuk melewatkan *item* antar *routine*. Data kopling pada sistem berorientasi objek dapat terjadi ketika suatu variabel dari suatu objek dilewatkan sebagai parameter pembanding pada beberapa kondisi percabangan dalam objek yang lain. Jumlah *Data Coupling* dapat

diketahui dengan menghitung jumlah variabel global yang dijadikan sebagai parameter dalam suatu kondisi percabangan dibeberapa objek yang lain.

5. *Lexical Content Coupling* (LCC)

   Beberapa atau semua isi dari suatu modul yang termasuk dalam isi yang lainnya. *Lexical Content Coupling* pada sistem berorientasi objek dapat terjadi ketika dua atau lebih objek melakukan modifikasi yang berbeda pada variabel global yang sama. Jumlah LCC dapat diketahui dengan menghitung jumlah variabel global yang sama dimana variabel tersebut mengalami modifikasi pada dua atau lebih objek yang berbeda

Contoh dari CC, GDC, IDC, DC dan LCC dapat dilihat dalam sebuah kode program pada Tabel 2.1.

Tabel 2.1 Contoh Kode Program dengan Beberapa Tipe Kopling

No	Kode Program
1	import java.io.*;
2	
3	class bank {
4	
5	DataInputStream in = new DataInputStream(System.in);
6	int accno, amtde; //GLOBAL DATA COUPLING
7	String name, acctype;
8	
9	void getdata() throws IOException {
10	System.out.println("Enter the Account No:");
11	accno = Integer.parseInt(in.readLine());
12	System.out.println("Enter the Account Type:");
13	acctype = in.readLine();
14	System.out.println("Enter the Customer Name:");
15	name = in.readLine();
16	System.out.println("Enter the Initial Deposit:");
17	amtde = Integer.parseInt(in.readLine());
18	}
19	
20	void display() {
21	System.out.println("********************");
22	System.out.println(" Account No:" + accno);
23	System.out.println("Account Type:" + acctype);
24	System.out.println("Customer Name:" + name);
25	System.out.println("Initial Deposit:" + amtde);
26	}
27	}
28	

Tabel 2.1 Contoh Kode Program dengan Beberapa Tipe Kopling (lanjutan)

No	Kode Program
29	class withdraw extends bank {
30	
31	int wd;
32	
33	void getin() throws IOException {
34	System.out.println("Enter the amount to be withdrawn:");
35	wd = Integer.parseInt(in.readLine());
36	}
37	
38	void show() {
39	if (wd <= amtde) //DATA COUPLING
40	{
41	amtde -= wd;
42	System.out.println("Balance after withdrawal:" + amtde);
43	System.out.println(" ******************");
44	} else //CONTROL COUPLING
45	{
46	System.out.println("You cannot withdraw this amount");
47	System.out.println(" **********************");
48	}
49	}
50	}
51	
52	class deposit extends bank {
53	
54	int dt;
55	
56	void get() throws IOException {
57	System.out.println("Enter the amount to be deposited:");
58	dt = Integer.parseInt(in.readLine());
59	}
60	
61	void print() {
62	amtde += dt; //LEXICAL CONTENT COUPLING
63	System.out.println("Balance after deposit:" + amtde);
64	System.out.println("********************");
65	}
66	}
67	
68	class bankacc {

Tabel 2.1 Contoh Kode Program dengan Beberapa Tipe Kopling (lanjutan)

No	Kode Program
69	
70	public static void main(String args[]) throws IOException {
71	DataInputStream in = new DataInputStream(System.in);
72	System.out.println("Enter the choice:");
73	int op = Integer.parseInt(in.readLine());
74	switch (op) {
75	case 1:
76	withdraw w = new withdraw();
77	w.getdata();
78	w.getin();
79	w.display();
80	w.show();
81	break;
82	case 2:
83	deposit d = new deposit();
84	d.getdata(); // INTERNAL DATA COUPLING
85	d.get();
86	d.display();
87	d.print();
88	break;
89	default:
90	System.out.println("Enter the choice 1 or 2");
91	break;
92	}
93	}
94	}

Sumber: (Aloysius & Arockiam, 2012)

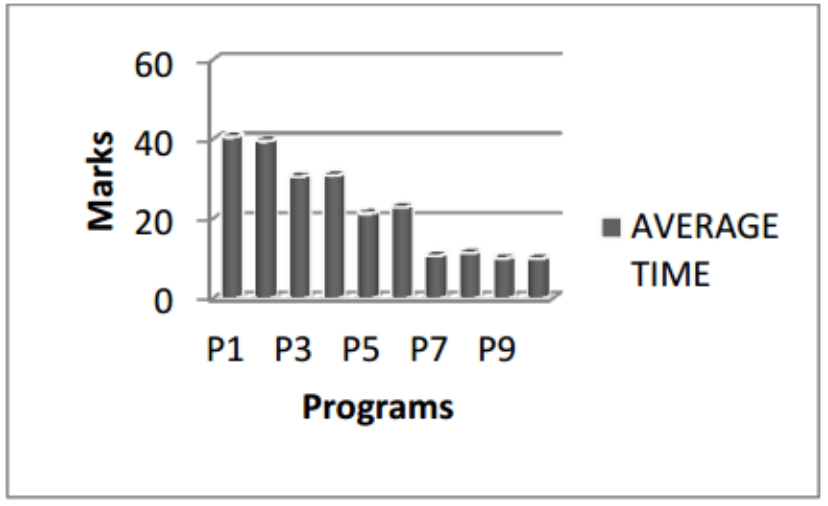
Dari contoh kode program di atas dapat diketahui bahwa jumlah CC, GDC, IDC, DC dan LCC masing-masing berjumlah satu. Dengan rincian sebagai berikut,

1. GDC terdapat pada baris ke-6, yaitu pada variabel amtde yang juga digunakan oleh beberapa kelas turunannya.
2. DC terdapat pada baris ke-39, yaitu variabel amtde yang merupakan variabel global digunakan sebagai parameter dalam pengkondisian percabangan pada kelas turunan.
3. CC terdapat pada baris ke-44, yaitu sebagai pengontrol suatu kondisi percabangan dimana kondisi-kondisi lain menggunakan variabel global yakni amtde sebagai parameternya.
4. LCC terdapat pada baris ke-41 dan ke-62, yaitu terdapat modifikasi dari variabel amtde oleh dua kelas turunan yang berbeda.

5. IDC terdapat pada baris ke-77 dan ke-84, yaitu pemanggilan *method* dari kelas parent oleh kelas utama, dimana pada *method* tersebut terdapat modfikasi variabel amtde oleh dua kelas turunan yang berbeda.

2.2.1.2 Kalibrasi Penilaian Bobot

Dalam menentukan bobot pemahaman pada berbagai tipe kopling yang diusulkan oleh Edward Berard (1993), Aloysius (2012) menyelenggarakan sebuah test pemahaman kepada kelompok mahasiswa. Test tersebut di tujukan untuk mencari waktu yang dibutuhkan dalam memahami kompleksitas dari program berorientasi objek dengan memperhatikan perbedaan macam kopling. Test pemahaman melibatkan 30 mahasiswa yang memiliki kemampuan cukup dalam menganalisis pemrograman berorientasi objek dan bahasa java serta mendapatkan nilai 65% lebih pada setiap ujian semester. Perhitungan waktu dicatat setelah mahasiswa selesai memahami setiap program. Setiap kasus atau kategori disediakan dua program berorientasi objek yang berbeda, jadi total keseluruhan terdapat sepuluh macam kode program. Waktu yang diperoleh mahasiswa pada setiap kode program dirata-rata berdasarkan tiap-tiap kategori. Hasil perhitungan waktu rata-rata yang diperoleh dari pemahaman mahasiswa dapat dilihat pada Gambar 2.1 dan Tabel 2.1.



Gambar 2.1 Rata-rata Waktu Pemahaman Setiap Kode Program

Sumber: (Aloysius & Arockiam, 2012)

Tabel 2.2 Kategori Waktu Pemahaman

Programs	Rata-rata Waktu Pemahaman	Kategori	Rata-rata Waktu Pemahaman
1	40.7	LCC	40.18333
2	39.66667		
3	30.76667	DC	30.88333
4	31		
5	21.43333	IDC	22.21667
6	23		
7	10.8	GDC	11.13333
8	11.46667		
9	10.16667	CC	10.11667
10	10.06667		

Sumber: (Aloysius & Arockiam, 2012)

Berdasarkan Tabel 2.2 dapat disimpullkan bahwa waktu pemahaman untuk kategori LCC paling besar dibandingkan kategori-kategori yang lainnya. Kemudian dibawahnya secara berurutan mulai dari yang terbesar yaitu DC, IDC, GDC dan yang paling kecil CC.

2.2.1.3 *Cognitive Coupling between Object* (CWCBO)

Metrik *Cognitive Weighted Coupling Between Object* (CWCBO) yang diusulkan oleh Aloysius adalah metrik yang didasarkan pada kompleksitas teori dari berbagai macam kopling yang berbeda, yaitu *data coupling*, *control coupling*, *global coupling* dan *interface coupling* (Aloysius & Arockiam, 2012). Metrik ini merupakan perpaduan antara metrik yang diusulkan oleh Edward Berard dan penggunaan teori bobot berdasarkan waktu pemahaman pada setiap kategori dalam penelitian yang dilakukan oleh Aloysius. Metrik CWCBO dapat dihitung menggunakan persamaan (1) sebagai berikut.

$$cwcbo = ((CC * WFCC) + (GDC * WFGDC) + (IDC * WFIDC) + (DC * WFDC) + (LCC * WFLCC)$$

Persamaan 2-1

Keterangan:

CC : Jumlah dari modul yang berisi *Control Coupling*.

GDC : Jumlah *Global Data Coupling*

IDC : Jumlah *Internal Data Coupling*

DC : Jumlah *Data Coupling*

LCC : Jumlah *Lexical Content Coupling*

Penilaian faktor bobot pada setiap jenis kopling didasarkan pada hasil kalibrasi yang dilakukan Aloysius, yaitu pengukuran rata-rata waktu pemahaman mahasiswa pada setiap kode program dengan kategori-kategori yang telah ditentukan. Pemberian nilai bobot pada setiap jenis kopling dapat dilihat pada Tabel 2.3 berikut ini:

Tabel 2.3 Faktor Bobot pada Setiap Kopling

No	Faktor Bobot	Bobot	Keterangan
1	WFCC	1	Faktor bobot *Control Coupling*
2	WFGDC	1	Faktor bobot *Global Data Coupling*
3	WFIDC	2	Faktor bobot *Internal Data Coupling*
4	WFDC	3	Faktor bobot *Data Coupling*
5	WFLCC	4	Faktor bobot *Lexical Data Coupling*

Sumber: (Aloysius & Arockiam, 2012)

Dari studi kasus pada point 2.2.1 dapat dilakukan perhitungan pada persamaan metrik CBO dan CWCBO, yaitu

$$\begin{aligned} cbo &= CC + GDC + IDC + DC + LCC \\ &= 1 + 1 + 1 + 1 + 1 \\ &= 5 \end{aligned}$$

$$\begin{aligned} cwcbo &= ((CC * WFCC) + (GDC * WFGDC) + (IDC * WFIDC) \\ &\quad + (DC * WFDC) + (LCC * WFLCC) \\ &= ((1 * 1) + (1 * 1) + (1 * 2) + (1 * 3) + (1 * 4) \\ &= ((1) + (1) + (2) + (3) + (4) \\ &= ((1 * 1) + (1 * 1) + (1 * 2) + (1 * 3) + (1 * 4) \\ &= 11 \end{aligned}$$

Dari perhitungan kedua metrik kopling tersebut didapatkan hasil bahwa dengan menggunakan persamaan metrik CBO diperoleh nilai kopling sebesar 5, sedangkan dengan menggunakan persamaan metrik CWCBO yang telah ditambahkan dengan bobot pemahaman diperoleh nilai kopling sebesar 11. Dengan hasil tersebut dapat disimpulkan bahwa penggunaan metrik CWCBO lebih presisi dan relatif lebih tinggi dibandingkan metrik CBO. Hal ini tidak lain karena adanya perkalian dengan bobot pemahaman di setiap kategori.

2.2.2 Source Code

Source code merupakan suatu kumpulan deklarasi bahasa pemrograman komputer seperti Java, C++, php dan lainnya (Primadhana, 2015). *Source code* dapat memungkinkan *programmer* untuk berkomunikasi atau memberikan instruksi pada perangkat keras komputer dengan perintah-perintah yang telah terdefinisi. Perintah tersebut biasanya ditulis dengan kumpulan dari huruf, angka dan simbol-simbol khusus (Primadhana, 2015). *Source code* bersifat statis tekstual, dapat dibaca oleh manusia dan merupakan deskripsi penuh dari

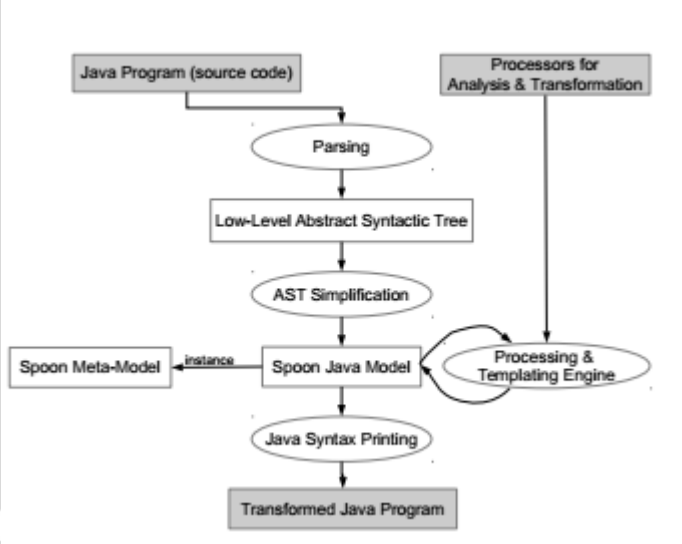
program komputer yang dapat dikompilasi secara otomatis ke dalam bentuk yang dapat dieksekusi (Binkley, 2007).

2.2.3 Spoon

SPOON merupakan sebuah *library* yang dapat digunakan untuk menganalisis dan merubah kode program dengan bahasa Java. Spoon memungkinkan pengembang untuk menulis dengan cakupan besar dalam menganalisis *domain* yang spesifik dan merubahnya dengan mudah sesuai kebutuhan (Pawlak, et al., 2015). Fitur-fitur utama dari Spoon adalah sebagai berikut (Pawlak, et al., 2015):

1. Sebuah Java metamodel untuk merepresentasikan Java *abstract syntax tree* (AST) yang secara mudah dapat dipahami dan dimanipulasi.
2. Sebuah *first-class intercession programming interface* (*intercession* API) untuk memodifikasi dan membangkitkan *source code* Java.
3. Penggunaan *generic typing* untuk pengecekkan statis pada proses analisis dan perubahan.
4. Integrasi yang halus dan asli serta pemrosesan dari anotasi Java.
5. Murni pengecekkan statis *template* dari mesin Java.
6. Mesin pengecekan secara statis pada *template* Java murni.

Secara umum Spoon dapat dilihat dalam bentuk bagan pada Gambar 2.2 berikut.

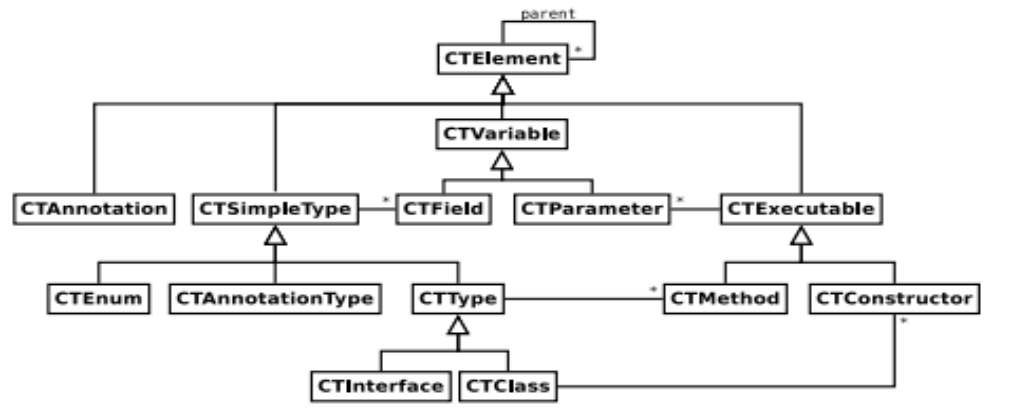


Gambar 2.2 Gambaran Umum Spoon

Sumber: (Pawlak, et al., 2015)

Spoon metamodel dapat dipecah menjadi tiga bagian. Ketiga bagian tersebut adalah sebagai berikut:

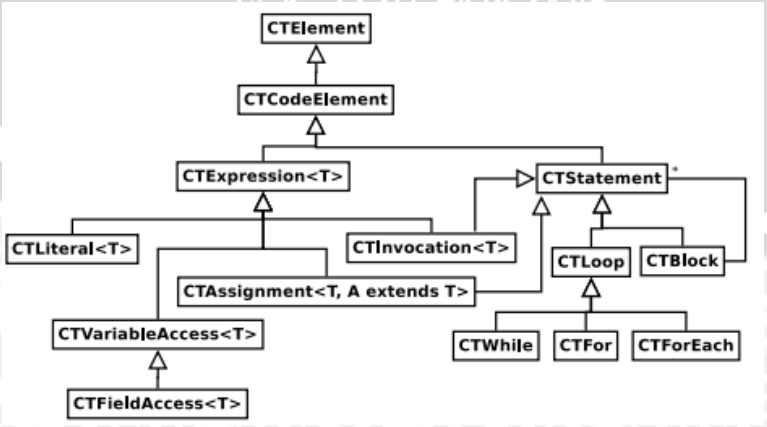
Bagian struktural (Gambar 2.3) mengandung deklarasi dari elemen program, seperti *interface, class, method, variable, annotation,* dan *enum declaration.* Pada metamodel bagian struktural, semua *element* turunan dari CtElement yang mendeklarasikan sebuah elemen induk menandakan relasi yang terkandung dalam file sumber. Sebagai contoh, induk dari *method node* adalah *class node*. Metamodel bagian struktural ini diperlihatkan pada Gambar 2.3 berikut.



Gambar 2.3 Bagian Struktural dari Spoon Java 5 metamodel

Sumber: (Pawlak, et al., 2015)

Bagian kode (Gambar 2.4) mengandung *executable java code,* seperti yang ditemukan dalam *body method*. Terdapat dua tipe dari kode elemen. Pertama, *statement* (CtStatement) adalah *untype top-level instructions* yang dapat digunakan secara langsung dalam sebuah blok kode. Kedua, *expressions* (CtExpression) digunakan dalam *statement*. Sebagai contoh sebuah CtLoop (sebuah *statement*) menunjuk pada sebuah CtExpression yang mana menunjukkan kondisi *boolean*. Metamodel bagian struktural ini diperlihatkan pada Gambar 2.4 berikut.



Gambar 2.4 Bagian kode dari Spoon Java 5 Metamodel

Sumber: (Pawlak, et al., 2015)

Bagian referensi memodelkan referensi ke elemen program (untuk instansiasi sebuah referensi ke sebuah tipe). Metamodel bagian referensi mengekspresikan elemen referensi program yang tidak perlu dimasukkan dalam metamodel (hal itu mungkin terdapat pada *library* pihak ketiga). Sebagai contoh, *expression node* mengembalikan sebuah String yang terikat pada sebuah referensi tipe dalam String dan tidak di-*compile* dalam model dari String.java selama *source code* dari String merupakan (biasanya) bukan bagian dari kode aplikasi yang dianalisis.

2.2.4 JFreeChart

JFreeChart merupakan sebuah *library* gratis yang digunakan oleh para *developer* untuk menampilkan diagram pada aplikasinya. JFreeChart pertama kali dibuat oleh David Gilbert pada Februari 2000. Hingga saat ini JFreeChart menjadi *library* diagram untuk Java yang paling banyak digunakan oleh para *developer*. Hal itu dapat dilihat pada situs resmi JFreeChart, bahwa lebih dari 2,2 juta pengguna yang telah mengunduhnya.

JFreeChart memiliki beberapa fitur, yaitu (Gilbert, David, 2005):

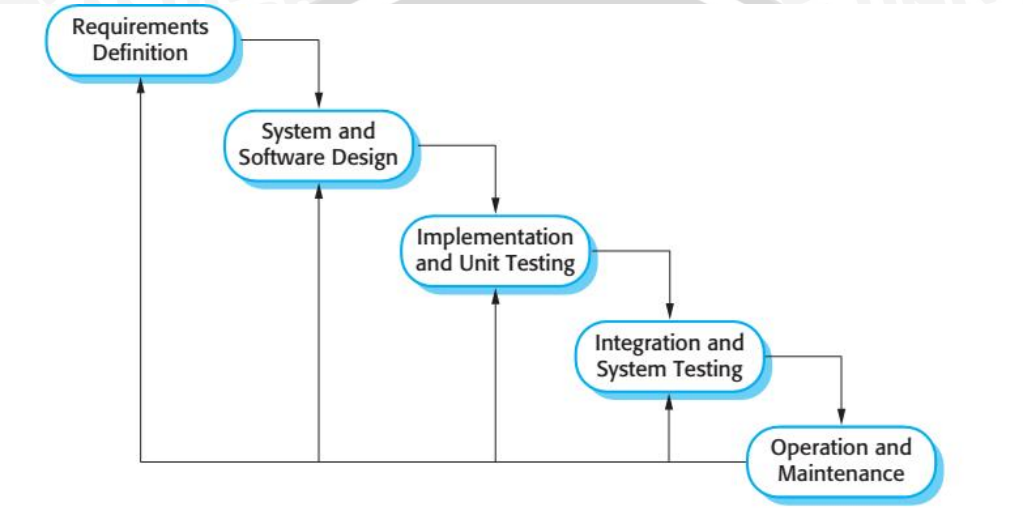
1. API konsisten dan terdokumentasi dengan baik, mendukung berbagai jenis grafik.
2. Desain yang fleksibel dan mudah untuk dikembangkan, serta dapat bekerja pada dua sisi, yaitu aplikasi pada sisi server dan sisi klien.
3. Dukungan untuk berbagai jenis *output*, termasuk Swing dan JavaFX components, file gambar (termasuk PNG dan JPEG), dan vektor format file grafis (termasuk PDF, EPS dan SVG);
4. JFreeChart adalah *open source* atau, lebih khusus, perangkat lunak gratis. Hal ini didistribusikan di bawah GNU *Lesser General Public License* (LGPL), yang memungkinkan digunakan dalam aplikasi berpemilik.

2.2.5 Konsep Dasar Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak merupakan suatu cabang ilmu profesi yang mempelajari tentang teknik dalam mengembangkan perangkat lunak, mulai dari perencanaan, pembuatan, pengujian hingga pemeliharaan. Rekayasa perangkat lunak didefinisikan sebagai pembentukan dan penggunaan prinsip-prinsip rekayasa untuk mendapat software yang handal, dikembangkan dengan tepat waktu dan bekerja secara optimal pada mesin yang nyata (Gaur, et al., 2014). Rekayasa Perangkat lunak saat ini lebih dipusatkan pada prosesnya, dengan penekanan pada aktifitas dan teknik-teknik yang dilakukannya (Nori & Swaminathan, 2006). Tujuan dari proses RPL sendiri adalah untuk mengoptimalkan parameter-parameter kinerja dalam prosesnya, seperti prediksi tingkat kecacatan, biaya dan waktu (Nori & Swaminathan, 2006).

2.2.6 Model Pengembangan *Waterfall*

Model pengembangan *waterfall* merupakan model pengembangan perangkat lunak yang terbit pertama dan paling dikenal dari beberapa model pengembangan yang lainnya (Munassar & Govardhan, 2010). Model ini banyak digunakan dalam pengembangan proyek-proyek pemerintahan dan perusahaan-perusahaan besar (Munassar & Govardhan, 2010). Model *waterfall* menjadi model dasar bagi model-model pengembangan yang lainnya. Bagan dari tahapan pemodelan waterfall dapat dilihat pada Gambar 2.5 berikut ini.



Gambar 2.5 Model Pengembangan Sistem *Waterfall*

Sumber: (Sommerville, 2011)

Menurut Sommerville (2011) tahapan-tahapan dari model *waterfall* adalah sebagai berikut :

1. Analisis Kebutuhan
   Analisis kebutuhan merupakan tahapan awal untuk menganalisis kebutuhan-kebutuhan yang harus ada pada sistem yang akan dikembangkan. Rekayasa kebutuhan dibuat dalam bentuk yang dapat dimengerti oleh staff pengembang. Tahapan analisis kebutuhan terdiri dari tahap elisitasi, spesifikasi, validasi dan verifikasi serta manajemen kebutuhan.
2. Perancangan sistem dan perangkat lunak
   Perancangan sistem dan perangkat lunak merupakan tahapan dimana pengembang membuat perancangan atau desain dari sistem secara keseluruhan, baik desain tampilan maupun desain arsitektur algoritma yang dapat menggambarkan alur sistem secara detil. Perancangan terdiri dari perancangan arsitektur, perancangan data, perancangan komponen dan perancangan antarmuka.
3. Implementasi dan pengujian unit
   Tahap implementasi, yaitu tahapan dimana desain perangkat lunak direalisasikan sebagai serangkaian program atau unit program. Pengujian unit melibatkan memverifikasi bahwa setiap unit memenuhi spesifikasinya.

4. Pengujian integrasi dan sistem
   Setiap unit program diintegrasikan dan diuji sebagai sistem yang lengkap untuk memastikan bahwa persyaratan perangkat lunak telah dipenuhi. Setelah pengujian, sistem perangkat lunak yang dikirimkan ke pelanggan.
5. Operasi dan pemeliharaan

Biasanya ini adalah fase terpanjang dalam siklus hidup. Sistem ini dipasang dan dimasukkan ke dalam penggunaan praktis. Pemeliharaan melibatkan koreksi kesalahan yang tidak ditemukan di awal tahap siklus hidup, meningkatkan pelaksanaan unit sistem dan meningkatkan pelayanan sistem sebagai requirement baru yang baru ditemukan.

Tahapan-tahapan tersebut bersifat runtut dan saling berkaitan, sehingga jika langkah ke-1 belum dikerjakan, maka langkah ke-2 tidak dapat dikerjakan. Langkah ke-2 hanya dapat dilakukan ketika langkah ke-1 telah selesai dilakukan, begitu juga dengan langkah-langkah selanjutnya. Namun, model *waterfalll* memiliki kelebihan pada prosesnya yang mudah dipahami dan strukturnya jelas. Model *waterfall* dapat digunakan pada proses pengembangan perangkat lunak ketika kebutuhan *user* telah dipahami dan jelas serta kemungkinan adanya perubahan kebutuhan *user* kecil.

2.2.7 Konsep Dasar Berorientasi Objek

Object-Oriented Programming (OOP) adalah suatu paradigma pemrograman yang menganut pada pendekatan berorientasi objek. Penggunaan OOP memiliki tujuan untuk mempermudah mengembangkan program berdasarkan model-model yang terdapat di kehidupan sehari-hari. Pendekatan Berorientasi objek dibangun berdasarkan *class, subclass* dan objek yang berisi unsur-unsur seperti *attribute*, *method* dan pesan (Misra & Akman, 2008). Unsur-unsur tersebut diidentifikasikan dalam deklarasi kelas dan bertanggung jawab pada kompleksitas kelas (Misra & Akman, 2008).

Menurut Rogers (Rogers, 1991), Terdapat 4 karakteristik yang dibutuhkan dalam pendekatan berorientasi objek, yaitu sebagai berikut:

1. Identitas

   Identitas berarti data dan prosedur yang beroperasi padanya dienkapsulasi secara terpisah. Dua objek bisa dibedakan dan diidentifikasi bahkan jika keduanya memiliki nilai atribut yang identik. Keduanya dapat dibedakan dengan lokasi dalam memori dan nama referensinya.

2. Klasifikasi

   Klasifikasi berarti objek dengan struktur data dan perilaku yang yang sama dikelompokkan ke dalam suatu kelas. Kelas merupakan abstrkasi yang menggambarkan sifat penting dari suatu aplikasi dan mengabaikan yang lainnya. Suatu kelas mendeskripsikan kumpulan objek individu yang tak terbatas. Setiap objek disebut juga sebagai *instance* dari kelas.

3. *Polimorfisme*

   Polimorfisme berarti operasi yang sama dapat berperilaku berbeda pada pada kelas yang berbeda. Operasi adalah suatu tindakan atau transformasi yang dilakukan atau ditujukan oleh objek. Sebuah implementasi khusus dari operasi dalam kelas tertentu disebut *method*.

4. *Inheritance* (Pewarisan)

   Inheritance adalah berbagi atribut dan operai antar kelas berdasarkan hubungan hirarki. Sebuah kelas dapat didefinisikan secara luas dan disempurnakan oleh *subclass* yang lebih halus. *Subclass* mewarisi semua sifat dari kelas induknya dan memungkinkan unutk menambahkan sifat sendiri yang unik.

2.2.8 *Unified Modelling Language* (UML)

Unified Modelling Language adalah bahasa pemodelan berupa diagram dan teks-teks pendukung pada perangkat lunak yang dibangun berdasarkan pendekatan berbasis objek (Primadhana, 2015). UML juga bisa disebut sebagai *tools* dalam mebuat pemodelan berorientasi objek. UML digunakan untuk menggambarkan, menspesifikasikan, merancang dan sebagai dokumentasi dari perangkat lunak yang dibangun (Primadhana, 2015). Pemodelan dilakukan pada fase perancangan. Membuat pemodelan sangat penting, karena tanpa pemodelan, seorang pengembang tidak dapat memahami sistem yang kompleks secara menyeluruh.

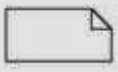
UML dikategorikan menjadi tiga macam (Rosa & Shalahuddin, 2013):

1. *Structure diagram* adalah kumpulan diagram statis dari sistem.
2. *Behavior diagram* adalah kumpulan diagram perilaku atau sifat serta perubahan dari sistem.
3. *Interaction diagram* adalah kumpulan diagram interaksi yang menjadi penghubung antara sistem dengan subsistem maupun dengan lingkungan luar.

2.2.8.1 *Use Case Diagram*

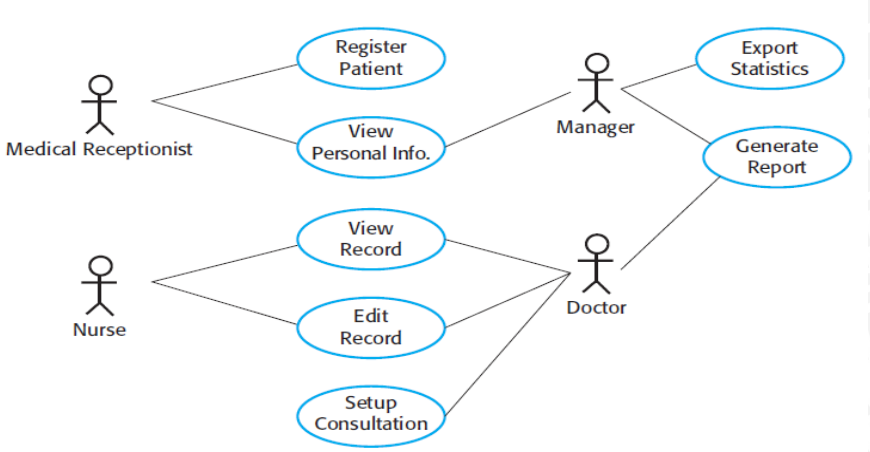
Use case diagram merupakan suatu diagram yang digunakan untuk menggambarkan konteks dari sistem yang akan dibangun dan fungsi yang dihasilkan dari sistem tersebut (Booch, 2007). Diagram ini berfungsi untuk menggambarkan kumpulan perilaku atau fitur yang dapat dilakukan oleh satu atau lebih aktor pada sistem yang dibangun. Setiap *use case* menggambarkan bagaimana pengguna memicu suatu event yang harus segera direspon oleh sistem. Penamaan *use case diagram* harus jelas, sederhana dan mudah dipahami. Komponen-komponen yang ada pada *use case diagram* dapat dilihat pada Tabel 2.4 (Rosa & Shalahuddin, 2014):

Tabel 2.4 Komponen *Use Case Diagram*

Nama	Simbol	Deskripsi
Actor		Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
Dependency		Hubungan dimana suatu elemen *independent* akan mempengaruhi elemen yang tidak *independent* yang bergantung padanya.
Generalization		Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
Include		Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.
Extend		Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.
Association		Apa yang menghubungkan antara objek satu dengan objek lainnya.
System		Menspesifikasikan paket yang menampilkan sistem secara terbatas.
Use Case		Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor
Collaboration		Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan prilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi).
Note		Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi

Sumber: (Rosa & Shalahuddin, 2014)

Use case diagram selain menggambarkan proses dalam bentuk diagram, juga disertai dengan tabel yang disebut dengan skenario *use case.* Table tersebut menjelaskan rangkaian isi setiap *use case.* Dalam tabel tersebut juga dijelaskan aktor mana yang menggunakan *use case* dan *use case* mana yang memasukkan *use case* lain. Pada Gambar 2.6 dapat dilihat contoh *use case diagram*:



Gambar 2.6 Contoh *use case diagram*

Sumber: (Sommerville, 2011)

Pada *use case diagram* tersebut terdapat empat aktor yaitu *Medical Receptionist*, N*urse, Export Statistic* dan *Doctor*. Setiap aktor mempunyai fungsi masing-masing atau memiliki beberapa fungsi yang sama dengan aktor yang lain seperti *Doctor* dan *Nurse* sama-sama memiliki fungsi *View Record* dan *Edit Record*.

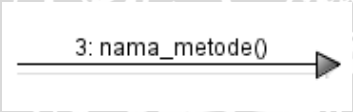
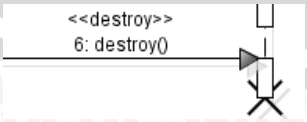
2.2.8.2 *Sequence Diagram*

Sequence diagram adalah suatu diagram interaksi yang menggambarkan interaksi aliran data antar beberapa behavior (Sefihara, 2016). Keuntungan dari penggunaan diagram ini adalah dapat dengan mudah membaca pesan yang lewat. Pembuatan *sequence diagram* didasarkan pada *use case scenario,* jadi masing-masing *use case* memiliki satu *sequence diagram.* Dalam proses pembuatan *sequence diagram,* diawali dengan menentukan objek-objek yang terlibat dan *method-method* terkait yang terdapat dalam *class* yang nantinya diintansiasikan menjadi sebuah objek (Rosa & Shalahuddin, 2013). *Sequence diagram* menunjukkan sejumlah *instance class* dan pesan yang melewati objek-objek tersebut dalam sebuah *use case*. Komponen-komponen *sequence diagram* dapat dilihat pada Tabel 2.5 berikut (Rosa & Shalahuddin, 2013):

Tabel 2.5 Komponen *Sequence Diagram*

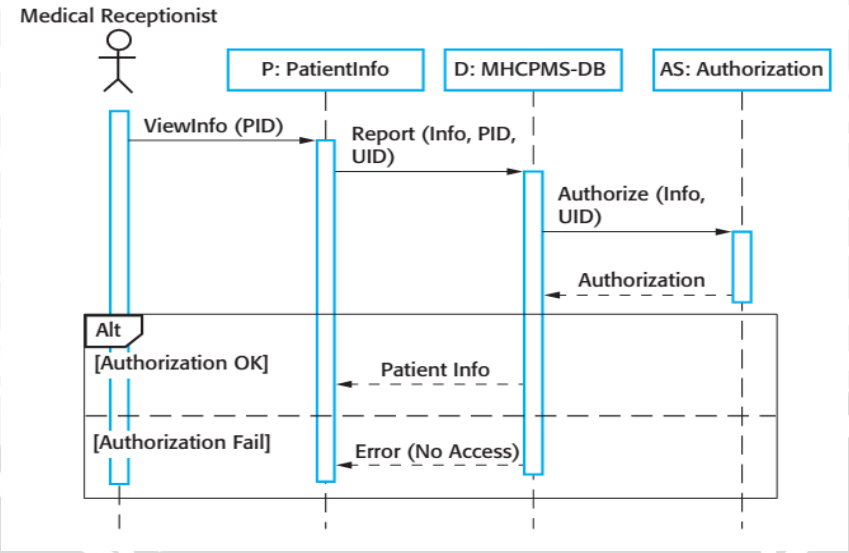
Nama Komponen	Simbol	Deskripsi
Actor		*Actor* dapat berupa orang atau sistem lain atau proses yang yang berada di luar sistem. Penamaan *actor* menggunakan kata benda.
Garis Hidup/*lifeline*		simbol garis putus-putus ini merepresestasikan waktu hidup sebuah objek

Tabel 2.5 Komponen *Sequence Diagram* (lanjutan)

Nama Komponen	Simbol	Deskripsi
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* lain yg terdapat pada objek lain atau dirinya sendiri. Anak panah menunjukkan *method* yang dipanggil. *Method* yang dipanggil harus ada dan sesuai dengan *class diagram*.
Pesan/*message* tipe *return*		Simbol ini menandakan bahwa *method* tersebut telah dijalankan oleh suatu objek, dan *method* tersebut mengembalikan nilai. Tanda panah mengarah pada objek yang menerima kembalian nilai.
Pesan/*message* tipe *destroy*		Simbol ini menandakan bahwa suatu objek menghentikan *lifeline* objek lain. Tanda panah mengarah kepada objek yang diakhiri. Simbol ini dapat digunakan ketika ada *message create* yang harus diakhiri.

Sumber: (Rosa & Shalahuddin, 2013)

Contoh s*equence diagram* dapat dilihat pada Gambar 2.7 berikut ini:



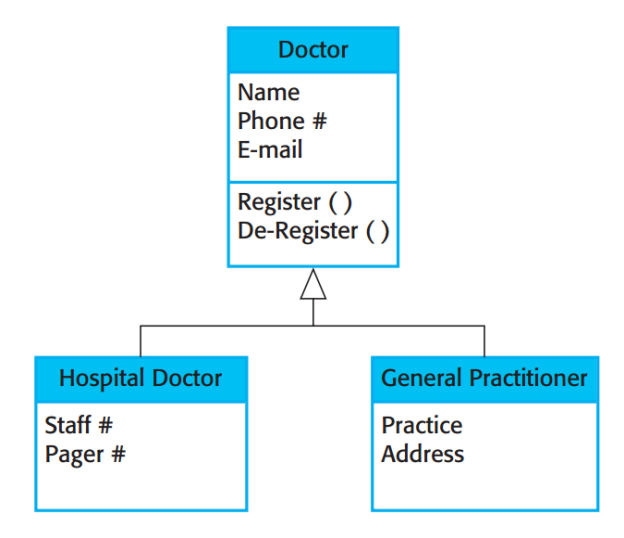
Gambar 2.7 Contoh *sequence diagram*

Sumber: (Sommerville, 2011)

Pada diagram tersebut aktor Medical Receptionist memicu *method* ViewInfo dalam instan P dari objek PatientInfo dengan membawa parameter pengenal (PID). P merupakan objek antarmuka pengguna dengan menampilkan informasi pasien. Kemudian P memamnggil *database* untuk mengambil informasi yang diperlukan. Sebelum memberikan informasi objek D mengotorisasi pada objek AS dengan menggunakan parameter UID dan Info. Pada tahap selanjutnya terdapat dua alternatif yaitu dimana ketika autorisasi berhasil maka D mengembalikan info pasien atau jika otorisasi gagal mala D akan mengembalikan *Error* (tidak bisa mengakses).

2.2.8.3 *Class Diagram*

Class diagram merupakan bagian yang paling penting dalm analisis dan perancangan berorientasi obyek (Booch, 2007). *Class diagram* merepresentasikan jenis-jenis objek dalam sebuah sistem dan berbagai hubungan statis yang terdapat diantara objek-objek tersebut. *Class diagram* digambarkan dalam bentuk persegi panjang dan terbagi menjadi 3 bagian, yaitu bagian atas berisi nama *class*, bagian tengah berisi nama atribut atau variabel dan bagian bawah berisi berisi operasi atau *method-method* dari objek tersebut. Contoh dari *class diagram* dapat dilihat pada Gambar 2.8 berikut ini:



Gambar 2.8 Contoh *class diagram*

Sumber: (Sommerville, 2011)

Dalam contoh *class diagram* pada Gambar 2.8 tersebut terdiri dari kelas Hospital Doctor, general Practitioner yang generalisasi (*extends)* terhadap kelas Doctor. Generalisasi berarti *inherite* atau bermakna sebagai anak kelas (*is-a)* terhadap kelas orang tua yaitu kelas Doctor.

2.2.9 Pengujian Perangkat Lunak

Pengujian adalah salah satu kegiatan penting dalam rekayasa perangkat lunak (Bertolino, 2007). Dalam istilah sederhana, pengujian dilakukan dengan mengamati eksekusi dari sistem untuk divalidasi apakah sistem telah berjalan sebagaimana mestinya dan mengidentifikasi potensi kerusakan (Bertolino, 2007). Adapun tujuan pengujian perangkat lunak menurut Rouf adalah sebagai berikut (Rouf, 2012):

1. Menjalankan perangkat lunak untuk dievaluasi dan dicari kesalahannya.
2. Pengujian yang baik harus menggunakan kasus uji dengan tingkat peluang kesalahan terbesar yang belum diketahui.
3. Pengujian yang baik mampu menemukan kesalahan yang belum diketahui pada program.
4. Pengujian dapat dikatakan berhasil jika seorang penguji mampu menemukan kesalahan yang banyak pada perangkat lunak. Dari kesalahan-kesalahan tersebut kemudian pengembang melakukan evaluasi ulang dan memperbaikinya. Hal inilah yang mendasari bahwa pengujian perangkat lunak tidak cukup hanya untuk memastikan ketika program dijalankan tidak terjadi kesalahan, sedangkan banyak komponen-komponen di luar atau di dalm sistem yang mungkin memiliiki kesalahan.

Secara umum, pengujian perangkat lunak mempunyai empat tahapan, diantaranya adalah sebagai berikut (Dennis, 2006):

1. ***Unit Testing***

   Unit Testing merupakan pengujian yang fokus pada unit-unit tertentu, seperti pengujian terhadap modul-modul yang merepresentasikan fungsi spesifik pada perangkat lunak. Pengujian ini berfungsi untuk memastikan bahwa modul-modul pada perangkat lunak berfungsi sesusai dengan spesifikasi perangkat lunak. Pengujian ini dilakukan setelah dilakukan tahapan implementasi yang menghasilkan kode program. Kode program tersebut yang akan diuji dengan beberapa kasus uji untuk memastikan tidak terdapat kesalahan pada kode program yang mengimplementasikan modul-modul.

2. ***Integration Testing***

   Integration testing adalah proses pengujian untuk memastikan hubungan-hubungan antar modul dalam program dapat bekerja bersama-sama tanpa adanya kesalahan.

3. ***System Testing***

   System testing adalah proses pengujian perangkat lunak secara keseluruhan dan untuk memastikan bahwa perangkat lunak dapat berkerja bersama-sama tanpa terjadi kesalahan.

4. ***Acceptance Testing***

   Acceptance testing proses pengujian yang bertujuan untuk memastikan bahwa perangkat lunak mampu memenuhi kebutuhan bisnis yang telah ditentukan sebelumnya dan mampu diterima dengan baik oleh pengguna.

2.2.9.2 *White Box Testing*

White box testing merupakan salah satu metode kasus uji yang menggunakan struktur kontrol dari perancangan prosedural untuk menghasilkan kasus-kasus uji dalam algoritma (Pressman, 2010). Dengan menggunakan metode ujicoba *whitebox*, para pengembang *software* dapat menghasilkan kasus-kasus uji diantaranya (Pressman, 2010):

1. Menjamin bahwa seluruh jalur independen dalam modul telah dilakukan minimal satu kali
2. Melakukan semua keputusan logika baik dari sisi benar maupun salah
3. Melakukan semua perulangan sesuai batasan yang ada dan dalam batasan operasionalnya
4. Menguji struktur data internal untuk memastikan validitas data.

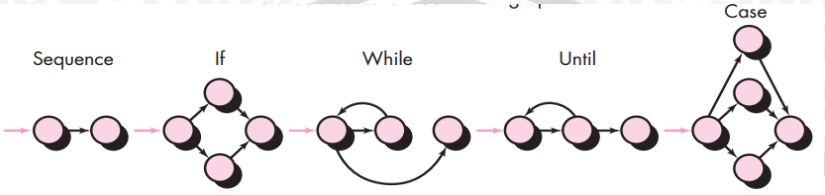
White box testing juga sering disebut sebagai *unit testing*. Pengujian ini berfokus pada upaya verifikasi pada unit terkecil dari desain perangkat lunak, komponen perangkat lunak atau modul (Pressman, 2010). Salah satu metode ujicoba unit adalah ujicoba berbasis alur (*Basis Path Testing*). Ujicoba berbasis alur merupakan teknik ujicoba *whitebox* pertama yang diusulkan oleh Tom McCabe (Pressman, 2010). Adapaun langkah-langkah dalam melakukan pengujian *basis path* adalah sebagai berikut (Pressman, 2010):

1. Mendefinisikan *node* pada kode program

   Hal ini dilakukan dengan menentukan kode program yang akan berdiri sebagai *node-node* yang akan digambarkan pada *flow graph*.

2. Membuat notasi *flow graph*

   Notasi *flow graph* mendefinisikan kontrol alur dari program. Contoh notasi *flow graph* dapat dilihat pada Gambar 2.9 berikut,



Gambar 2.9 Contoh notasi *flow graph*

Sumber: (Pressman, 2010)

3. Mendefinisikan *independent path* program

   Jalur independen adalah setiap jalan melalui program yang memperkenalkan setidaknya satu set pernyataan baru. Contoh dari *independent path* seperti, *Path* 1: 1-2-4-6-7; *Path* 2: 1-2-3-4-5-6-7.

4. Menentukan *cyclomatic complexity* V(G)

   Cyclomatic complexity dapat dihitung menggunakan tiga cara yaitu, (1) jumlah dari *region* dari *flow graph*, (2) V(G) = E-N+2 dimana E adalah *edge* atau garis dan N adalah *node* atau simpul, (3) V(G) = P+1 dimana P adalah *predicate node* pada *flow graph*.

5. Membuat *test case*

   Test case adalah merupakan tindakan yang dilakukan dalam kasus uji sesuai dengan jalur yang didefinisikan.

2.2.9.3 *Black Box Testing*

Pengujian *black box* adalah jenis pengujian yang mengabaikan komponen atau mekanisme internal sistem dan hanya berfokus pada keluaran yang dihasilkan dalam menanggapi input yang dipilih dan dieksekusi (Bhasin, et al., 2014). Pengujian *black box* bertujuan untuk memastikan bahwa perangkat lunak telah memenuhi semua kebutuhan fungsional yang telah didefinisikan sebelumnya. Salah satu jenis pengujian *black box* adalah pengujian validasi. Pengujian ini dilakukan pada tahap akhir pengujian perangkat lunak. Pengujian validasi fokus pada sudut pandang pengguna dan *output* dari sistem apakah telah sesuai dengan kebutuhan pengguna atau tidak. Pengujian dikatakan berhasil jika pengguna dapat mengenali semua fungsi yang terdapat pada perangkat lunak. Validasi dapat dilakukan dengan cara menyediakan informasi dengan persyaratan yang telah ditentukan sebelumnya.

BAB 3 METODOLOGI

Metodologi penelitian menjelaskan metode pengembangan perangkat lunak yang digunakan dalam penelitian ini dan sekaligus langkah-langkah yang akan dilakukan dalam pengerjaan tugas akhir. Metode pengembangan yang digunakan adalah metode pengembangan *waterfall,* karena kebutuhan *user* terhadap sistem telah dipahami dan jelas, serta kemungkinan adanya perubahan kebutuhan *user* kecil. Tahapan metodologi penelitian terdiri dari Studi Literatur, Rekayasa Kebutuhan, Perancangan, Implementasi, Pengujian dan analisis, pengambilan kesimpulan dan saran serta penulisan laporan. Gambar 3.1 menunjukkan tahapan penelitian secara umum:

Gambar 3.1 Alur metodologi penelitian

3.1 Studi literatur

Dalam sebuah penelitian, studi literatur digunakan sebagai sumber acuan dalam penulisan skripsi dan pengembangan sistem agar dapat mempelajari dan memahami secara mendalam terkait teori-teori dasar keilmuan yang akan menjadi objek penelitian yang dilakukan. Teori-teori mengenai metode sistematika pembangunan kakas bantu perhitungan kopling menggunakan metrik *Cognitive Weight Coupling Between Object* (CWCBO) menjadi dasar penelitian yang diperoleh dari buku, jurnal, ebook, penelitian sebelumnya, situs internet serta literatur lain yang berkaitan. Teori pustaka yang berkaitan dengan penelitian skripsi ini meliputi:

1. Metrik Kopling
2. *Source Code*
3. Spoon
4. JfreeChart
5. Konsep Dasar Berorientasi Objek
6. Model Pengembangan *Waterfall*
7. Konsep Dasar Rekayasa Perangkat Lunak
8. *Unified Modelling Language* (UML)
9. Pegujian Perangkat Lunak

Setelah melakukan tahap studi literatur kemudian dilakukan tahap analisis kebutuhan yang diperlukan oleh sistem.

3.2 Rekayasa kebutuhan

Rekayasa kebutuhan dilakukan untuk mengetahui kebutuhan apa saja yang diperlukan dalam pembangunan kakas bantu perhitungan kopling menggunakan metrik *Cognitive Weight Coupling Between Object* (CWCBO). Metode analisis yang digunakan dalam penelitian ini adalah *object-oriented analysis* dan pemodelan dilakukan menggunakan *Unified Modelling Language* (UML). Pada tahap Rekayasa kebutuhan perangkat lunak dibagi menjadi 2 tahapan yaitu, proses rekayasa kebutuhan dan model kebutuhan. Rekayasa kebutuhan meliputi gambaran umum sistem, identifikasi aktor dan spesifikasi kebutuhan. Model kebutuhan meliputi pemodelan *use case diagram* dan skenario *use case.*

Dalam melakukan rekayasa kebutuhan akan didapatkan kebutuhan fungsional dan kebutuhan non-fungsional. Salah satu contoh kebutuhan fungsional adalah sistem dapat menghitung nilai kopling dari souce code yang diinputkan. Berdasarkan kebutuhan fungsional yang telah dibuat, lalu dibuat diagram *use case* untuk mendeskripsikan *requirements* (kebutuhan) perangkat lunak dari sudut pandang pengguna kemudian dijelaskan lebih rinci dengan skenario *use case*. Sedangkan parameter yang akan digunakan pada kebutuhan non-fungsional adalah *accuration*.

3.3 Perancangan sistem

Perancangan sistem dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan. Perancangan sistem berdasarkan *object-oriented analysis* dan *object-oriented design* yaitu menggunakan pemodelan *Unified Modelling Language* (UML). Pemodelan UML terdiri dari beberapa diagram yaitu *use case diagram*, *class diagram* dan *sequence diagram*.

Perancangan perangkat lunak terdiri dari beberapa tahapan diantaranya adalah perancangan arsitektur, perancangan *class diagram*, perancangan *sequence diagram* dan perancangan antarmuka. Perancangan arsitektur digunakan untuk mengetahui gambaran kinerja sistem secara keseluruhan. Perancangan *class diagram* digunakan untuk mengidentifikasi kelas-kelas yang berada dalam sistem. Perancangan *sequence diagram* digunakan untuk menjelaskan interaksi antar objek yang disusun dalam urutan waktu. Perancangan antarmuka digunakan sebagai media interaksi antara pengguna dan sistem

3.4 Implementasi sistem

Implementasi sistem merupakan proses penerapan pembuatan aplikasi berdasarkan pada proses analisis dan perancangan yang telah dilakukan pada tahap sebelumnya. Implementasi perangkat lunak dilakukan dengan menggunakan bahasa pemrograman berorientasi objek yaitu menggunakan bahasa pemrograman Java dengan *software* Netbeans IDE 8.0. Implementasi sistem ini meliputi:

1. Membuat sistem "Kakas Bantu Perhitungan Nilai Kopling Menggunakan Metrik *Cognitive Weighted Coupling Between Object* (CWCBO)".
2. Mengimplementasikan metrik *Cognitive Weighted Coupling Between Object* (CWCBO) dengan menggunakan bahasa Java.
3. Mengimplementasikan sistem agar dapat menerima masukan berupa *source code* yang akan dilakukan perhitungan kopling.
4. Mengimplementasikan antarmuka sistem.

3.5 Pengujian dan Analisis

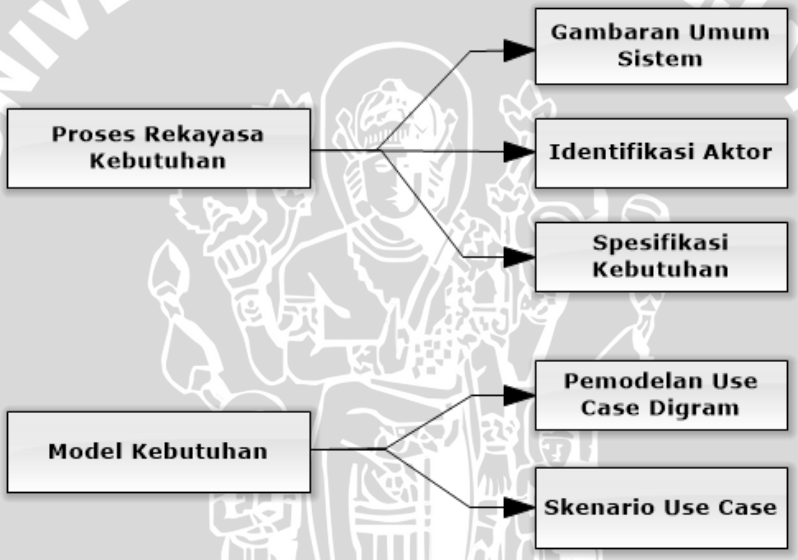
Strategi pengujian perangkat lunak yang digunakan yaitu pengujian unit, pengujian integrasi, pengujian validasi, dan pengujian akurasi. Metode pengujian yang digunakan adalah *white-box testing* dan *black-box testing*. Pada tahap pengujian unit digunakan metode *white-box testing* dengan teknik *basis path*. Sementara untuk pengujian integrasi, akurasi dan validasi digunakan metode *black-box testing*. Kemudian dilakukan analisis dari hasil pengujian perangkat lunak sehingga dapat diperoleh kesimpulan dari pembuatan perangkat lunak yang telah dilakukan.

3.6 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah dilakukan proses pengujian dan analisis sistem sehingga dapat diketahui efektifitas kinerja dari sistem. Kesimpulan dibuat berdasarkan hasil pengujian dan analisis sehingga diperoleh inti dari keseluruhan proses penelitian. Pengambilan kesimpulan bertujuan untuk menjawab permasalahan yang ada dalam rumusan masalah. Saran digunakan untuk memberi masukan untuk menjadi bahan pertimbangan untuk pengembangan sistem selanjutnya. Tahap terakhir yaitu penulisan laporan yang dapat membantu dalam pengembangan sistem selanjutnya.

BAB 4 REKAYASA KEBUTUHAN

Bab ini membahas tentang proses rekayasa kebutuhan yang digunakan dalam penelitian pembangunan kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO). Rekayasa kebutuhan perangkat lunak dibuat untuk menggambarkan kebutuhan-kebutuhan yang harus disediakan oleh sistem agar dapat memenuhi kebutuhan pengguna. Dalam rekayasa kebutuhan terbagi menjadi proses rekayasa kebutuhan dan model kebutuhan. Proses rekayasa kebutuhan terdiri dari gambaran umum sistem, identifikasi aktor dan spesifikasi kebutuhan. Pemodelan kebutuhan terdiri dari pemodelan *usecase* diagram dan skenario *use case*. Tahap-tahap rekayasa kebutuhan kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO) dapat dilihat pada gambar 4.1 berikut,



Gambar 4.1 Diagram rekayasa kebutuhan

4.1 Proses Rekayasa Kebutuhan

Proses rekayasa kebutuhan merupakan tahap untuk mengumpulkan kebutuhan-kebutuhan sistem dalam membangun kakas bantu perhitungan kopling. Tahap-tahap dalam proses ini adalah gambaran umum sistem, identifikasi aktor, spesifikasi kebutuhan.

4.1.1 Gambaran Umum Sistem

Gambaran umum sistem perhitungan nilai kopling menggunakan metrik CWCBO terdiri dari dua bagian, yaitu deskripsi umum sistem dan lingkungan sistem.

1. Deskripsi Umum Sistem

Sistem yang dibangun dalam penelitian ini adalah sebuah perangkat lunak dengan judul "Kakas Bantu Perhitungan Nilai Kopling Menggunakan Metrik *Cognitive Weighted Coupling Between Object* (CWCBO)". Tujuan utama dari sistem ini adalah untuk mengetahui masukan, keluaran dan efek berupa perhitungan kualitas sebuah *source code* berdasarkan nilai kopling antar objek. Dimana nilai kopling tersebut juga didasarkan pada metrik kopling dengan penilaian bobot berupa tingkat pemahaman terhadap berbagai jenis kopling. Pengguna diharapkan dapat memperoleh informasi masukan, keluaran dan efek serta mengetahui nilai kopling dari *source code* tersebut. Pada tahap awal sistem mendapat masukan berupa sejumlah *source code* dari pengguna, kemudian sistem akan melakukan perhitungan kopling dengan metrik kopling CWCBO dan menghasilkan keluaran berupa nilai kopling. Selain itu sistem ini juga dapat menampilkan data berupa grafik perbandingan nilai kopling dari sejumlah kode program yang telah dihitung berdasarkan persamaan metrik CBO dan persamaan metrik CWCBO.

2. Lingkungan Sistem

Sistem perhitungan nilai kopling ini dibangun menggunakan bahasa Java, sehingga membutuhkan *Java Runtime Environment* (JRE) untuk berjalan. Sistem ini dapat berjalan di seluruh sistem operasi yang sudah terpasang JRE minimal versi 1.8. Dalam sistem terdapat *library* Spoon yang digunakan dalam memparser *source code* untuk mendapatkan nilai dari setiap jenis kopling yang menjadi parameter. Dalam sistem juga terdapat *library* JFreeChart untuk menampilkan grafik perbandingan nilai CBO dan CWCBO.

4.1.2 Identifikasi Aktor

Identifikasi aktor merupakan tahap untuk melakukan identifikasi terhadap aktor-aktor yang dapat melakukan interaksi kepada sistem. Tahap identifikasi aktor dilakukan dengan mendefinisikan aktor-aktor pengguna sistem beserta deskripsinya. Tabel 4.1 menunjukkan aktor yang terdapat dalam sistem ini beserta deskripsinya.

Tabel 4.1 Identifikasi Aktor

Aktor	Deskripsi Aktor
Pengguna	Pengguna merupakan satu-satunya aktor yang menggunakan sistem, dan dapat menjalankan semua fungsi yang ada pada sistem.

4.1.3 Spesifikasi Kebutuhan

Spesifikasi kebutuhan adalah proses untuk menuliskan daftar kebutuhan yang harus terdapat pada sistem kakas bantu perhitungan kopling. Sasaran pada tahap ini adalah mendefinisikan sejumlah kebutuhan yang harus dapat dilakukan oleh sistem. Spesifikasi kebutuhan perangkat lunak pada penelitian ini terbagi menjadi dua bagian yaitu, spesifikasi kebutuhan fungsional dan spesifikasi kebutuhan non fungsional.

4.1.3.1 Spesifikasi Kebutuhan Fungsional

Spesifikasi kebutuhan fungsional menghasilkan daftar kebutuhan fungsionalitas sistem atau layanan-layanan yang dapat dilakukan oleh sistem. Kebutuhan fungsional dari sistem dibuat untuk pengguna sistem. Tabel 4.2 menunjukkan spesifikasi daftar kebutuhan fungsional dengan menggunakan format nomor *Software Requirement Specification* (SRS_F).

Tabel 4.2 Daftar Kebutuhan Fungsional

Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi	Prioritas
SRS_F001	Menginputkan *source code*	Sistem harus menyediakan fasilitas bagi pengguna untuk memasukkan sejumlah *source code*. Sistem harus dapat menampilkan *pop-up* pencarian berkas dan dapat menampilkan daftar dari *source code* yang telah diinputkan.	Tinggi
SRS_F002	Menghitung nilai kopling	Sistem harus menyediakan fasilitas bagi pengguna untuk dapat menghitung dan menampilkan nilai kopling berdasarkan persamaan metrik kopling *CBO* dan *CWCBO.*	Tinggi
SRS_F003	Menampilkan grafik perbandingan nilai kopling	Sistem harus menyediakan fasilitas untuk menampilkan grafik perbandingan nilai kopling dari sejumlah masukan *source code* berdasarkan persamaan metrik kopling CBO dan *CWCBO*	Sedang

Berikut ini adalah penjelasan dari setiap kebutuhan fungsional yang dapat dilakukan oleh aktor pada sistem.

1. **[SRS_F001]** Menginputkan *source code*

   Pengguna dapat memasukkan sejumlah berkas *source code* ke dalam sistem dengan ekstensi .java. Kemudian beberapa file *source code* tersebut akan diolah untuk dilakukan perhitungan nilai kopling menggunakan metrik *Coupling Between Object* (CBO) dan *Cognitive Weighted Coupling Between Object* (CWCBO).

2. **[SRS_F002]** Menghitung nilai kopling

   Pengguna dapat menekan tombol Hitung Kopling untuk mengidentifikasi dan menghitung jumlah masing-masing tipe kopling berdasarkan metrik CBO dari *source code* yang telah dimasukkan. Kemudian dilakukan perhitungan nilai kopling berdasarkan persamaan metrik kopling *Coupling Between Object* (CBO) dan *Cognitive Weighted Coupling Between Object* (CWCBO).

3. **[SRSF_N003]** Menampilkan grafik perbandingan nilai kopling

   Pengguna dapat menekan tombol Tampilkan Grafik untuk menampilkan grafik perbandingan nilai kopling berdasarkan metrik kopling CBO dan CWCBO.

4.1.3.2 Spesifikasi Kebutuhan Non Fungsional

Kebutuhan non fungsional berisi daftar kebutuhan yang dapat dijadikan acuan dalam menentukan kualitas secara keseluruhan dari suatu sistem. Kebutuhan non fungsional menetukan batasan pada produk yang sedang dikembangkan serta dapat menentukan batasan-batasan eksternal yang harus dipenuhi oleh produk tersebut. Spesifikasi daftar kebutuhan non fungsional sistem diperlihatkan pada Tabel 4.3 dengan menggunakan format nomor *Software Requirement Specification* (SRS_NF).

Tabel 4.3 Daftar Kebutuhan Non Fungsional

Kode Kebutuhan	Jenis Kebutuhuhan	Deskripsi
SRS_NF001	*Akurasi*	Sistem harus dapat melakukan perhitungan nilai kopling antar objek dalam *source code* program yang diuji sesuai dengan perhitungan manual minimal 90%.

4.2 Model Kebutuhan

Setelah proses rekayasa kebutuhan maka perlu dimodelkan untuk memudahkan pembacaan kebutuhan dalam tahap perancangan. Proses pemodelan kebutuhan ini menggunakan salah satu diagram dari *Unified Modelling Language (*UML) yaitu *use case diagram.*

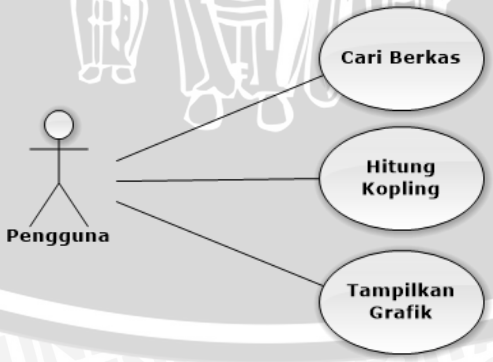
4.2.1 Pemodelan *Use Case Diagram*

Use case diagram digunakan untuk mengetahui fungsi-fungsi yang dapat dilakukan oleh sistem, serta dapat melihat aktor yang berhak menggunakan fungsi-fungsi tersebut. Aktor memiliki 3 fitur (*use case*) yang dapat dilakukan pengguna kepada sistem. *Use Case Diagram* dapat dilihat pada Gambar 4.4 berikut ini.

Tabel 4.4 Pemetaan kebutuhan fungsional dengan *use case*

Nomor SRS	Kebutuhan	Kode *Use Case*	Nama Use Case	Aktor
SRS_F001	Menginputkan *source code*	**UC_01**	Cari Berkas	Pengguna
SRS_F002	Menghitung nilai kopling	**UC_02**	Hitung Kopling	Pengguna
SRS_F003	Menampilkan grafik perbandingan nilai kopling	**UC_03**	Tampilkan Grafik	Pengguna

Dari pemetaan kebutuhan fungsional dan *use case* pada Tabel 4.4 tersebut akan teridentifikasi aktor dalam kebutuhan fungsional yang telah didefinisikan diawal. Sistem ini memiliki beberapa kegiatan utama. Penggambaran kegiatan tersebut dapat dilihat dalam Gambar 4.2 diagram *use case* berikut,



Gambar 4.2 *Use Case Diagram*

4.2.2 Skenario *Use Case*

Skenario *use case* digunakan untuk menjelaskan secara detail setiap kebutuhan fungsional yang terdapat pada *use case diagram*.

1. **[UC_01] Cari Berkas**

Skenario *use case* cari berkas merupakan penjelasan lebih rinci dari *use case* cari berkas. Tabel 4.5 menujukkan skenario *use case* untuk mencari berkas.

Tabel 4.5 Skenario *Use Case* Cari Berkas

Objective	Pengguna dapat mencari informasi berkas yang dicari.
Actors	Pengguna
Pre-Condition	Aplikasi "Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Cognitive Weighted Coupling Between Object*" sudah berjalan (*running)*.
Main Flow	1. Pengguna menekan tombol Cari Berkas untuk mencari file yang akan dimasukkan ke dalam sistem dan untuk melakukan perhitungan nilai kopling. 2. Sistem menampilkan *window browse file*. 3. Pengguna memilih file dengan format .java 4. Pengguna menekan tombol Open. 5. Sistem mengambil file yang dipilih dan menampilkan halaman aplikasi yang berisi form perhitungan kopling.
Alternative flows	Jika pengguna menekan tombol cancel maka file tidak akan dimasukkan ke sistem.
Post-condition	File berhasil dimasukkan ke dalam sistem.

2. **[UC_02] Skenario *Use Case* Hitung Kopling**

Skenario *use case* Hitung Kopling merupakan penjelasan lebih rinci dari *use case* Hitung Kopling. Tabel 4.6 menujukkan skenario *use case* untuk Hitung Kopling.

Tabel 4.6 Skenario *Use Case* Hitung Kopling

Objective	Pengguna dapat melakukan melakukan perhitungan nilai kopling berdasarkan metrik CBO dan CWCBO
Actors	Pengguna
Pre-Condition	File telah dimasukkan ke dalam sistem.
Main Flow	1. Pengguna menekan tombol Hitung kopling untuk melakukan perhitungan nilai kopling. 2. Sistem melakukan *parser source code* untuk mengidentifikasi jumlah dari masing-masing tipe kopilng (CC, GDC, IDC, DC dan LCC) berdasarkan metrik CBO. 3. Sistem melakukan perhitungan dan menampilkan nilai kopling berdasarkan metrik CBO dan CWCBO.
Alternative flows	Sistem menampilkan pesan "File kosong, masukkan file yang akan di uji!"
Post-condition	Nilai kopling berhasil ditampilkan oleh sistem

3. [UC_03] Tampilkan Grafik

Skenario *use case* Tampilkan Grafik merupakan penjelasan lebih rinci dari *use case* Tampilkan Grafik. Tabel 4.7 menujukkan skenario *use case* untuk Tampilkan Grafik.

Tabel 4.7 Skenario *Use Case* Tampilkan Grafik

Objective	Pengguna dapat menampilkan grafik perbandingan nilai kopling CBO dan CWCBO
Actors	Pengguna
Pre-Condition	Nilai kopling berhasil ditampilkan oleh sistem
Main Flow	1. Pengguna menekan tombol Tampilkan Grafik untuk menampilkan grafik perbandingan nilai kopling berdasarkan metrik CBO dan CWCBO. 2. Sistem menampilkan grafik perbandingan nilai kopling berdasarkan metrik CBO dan CWCBO.
Alternative flows	Sistem menampilkan pesan "File kosong, masukkan file yang akan di uji!"
Post-condition	Grafik perbandingan nilai kopling CBO dan CWCBO berhasil ditampilkan.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

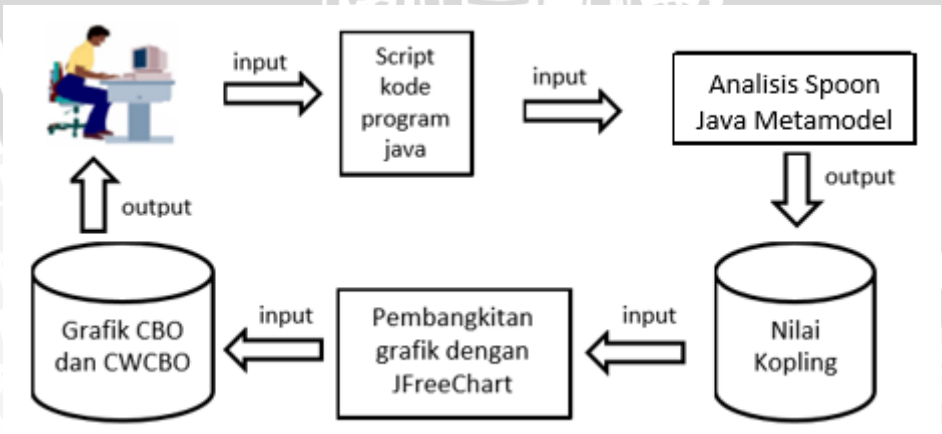
Bab ini menguraikan tentang perancangan dan implementasi sistem yang digunakan dalam penelitian pembangunan kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO). Pada bab ini terdiri dari dua sub bab, yaitu Perancangan dan Implementasi. Pada Perancangan terdiri dari Perancangan arsitektur, perancangan *sequence diagram*, perancangan *class diagram*, perancangan alur *parser source code,* perancangan komponen dan perancangan antarmuka. Sedangkan pada Implementasi terdiri dari Spesifikasi sistem, batasan implementasi dan Implementasi antarmuka.

5.1 Perancangan

Pada tahap perancangan dalam penelitian ini terdapat lima jenis perancangan yang terdiri dari perancangan arsitektur, perancangan sequence diagram, perancangan class diagram, perancangan alur *parser source code*, perancangan komponen, dan perancangan antarmuka.

5.1.1 Perancangan Arsitektur

Perancangan arsitektur merupakan tahap pertama dari perancangan perangkat lunak. Perancangan arsitektur berhubungan dengan cara merancang struktur keseluruhan sistem sehingga sistem dapat teroganisir (Sommerville, 2011). Perancangan arsitektur menggunakan notasi-notasi untuk menggambarkan langkah-langkah penelitan yang digunakan dalam pembangunan kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO). Arsitektur perangkat lunak dapat dilihat pada gambar 5.1 berikut:



Gambar 5.1 Arsitektur Sistem Perhitungan Nilai Kopling

5.1.2 Perancangan *Sequence Diagram*

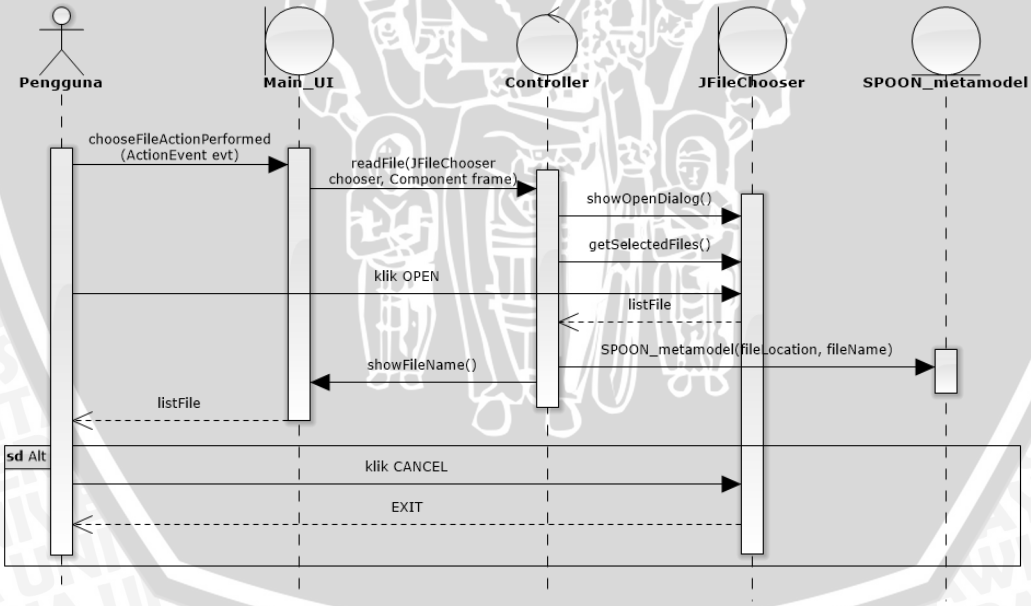
Diagram *sequence* digunakan untuk menelusuri eksekusi skenario dalam konteks yang sama sebagai diagram komunikasi yang menunjukkan bagaimana aluran data saling berinteraksi dalam beberapa behavior. Diagram *sequence* dibuat berdasarkan diagram *use case* sistem. Pemetaan *use case* dan *sequence* dapat dilihat pada Tabel 5.1 berikut,

Tabel 5.1 Pemetaan kebutuhan fungsional dengan *use case*

Kode Use Case	Nama Use Case	Kode *Sequence*	Nama Sequence	Aktor
UC_01	Cari Berkas	**SC_01**	Cari Berkas	Pengguna
UC_02	Hitung Kopling	**SC_02**	Hitung Kopling	Pengguna
UC_03	Tampilkan Grafik	**SC_03**	Tampilkan Grafik	Pengguna

1. [SC_01] Cari Berkas

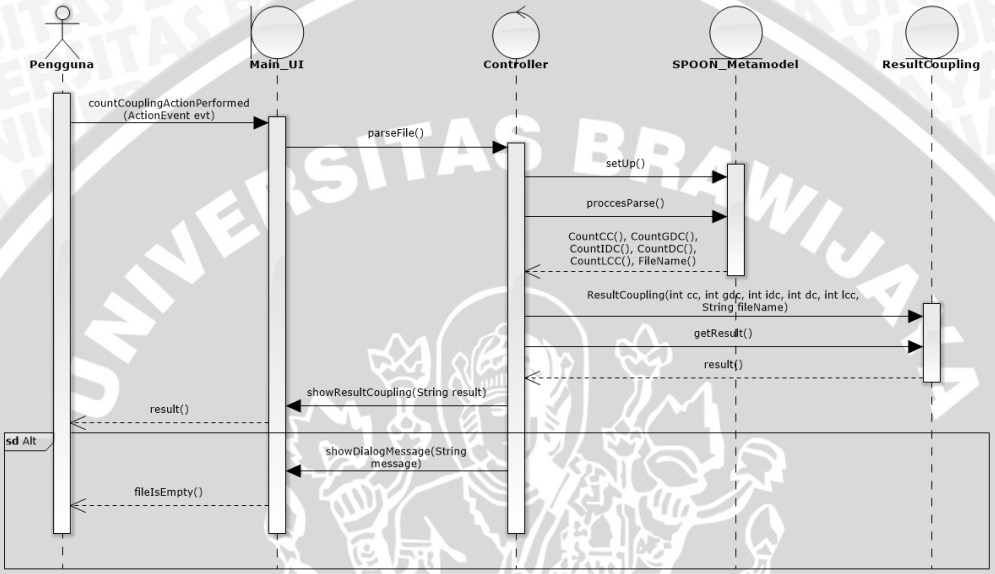
Sequence diagram cari berkas menjelaskan interaksi *user* dengan sistem dalam memasukkan file yang berupa *source code* dengan bahasa Java ke dalam sistem. Objek-objek yang berinteraksi pada *sequence diagram* ini adalah *Main_UI, Controller, SPOON_metamodel* dan *JFileChooser. Sequence diagram* dari cari berkas diperlihatkan pada gambar 5.2 berikut.



Gambar 5.2 *Sequence Diagram* Cari Berkas

2. [SC_02] Hitung Kopling

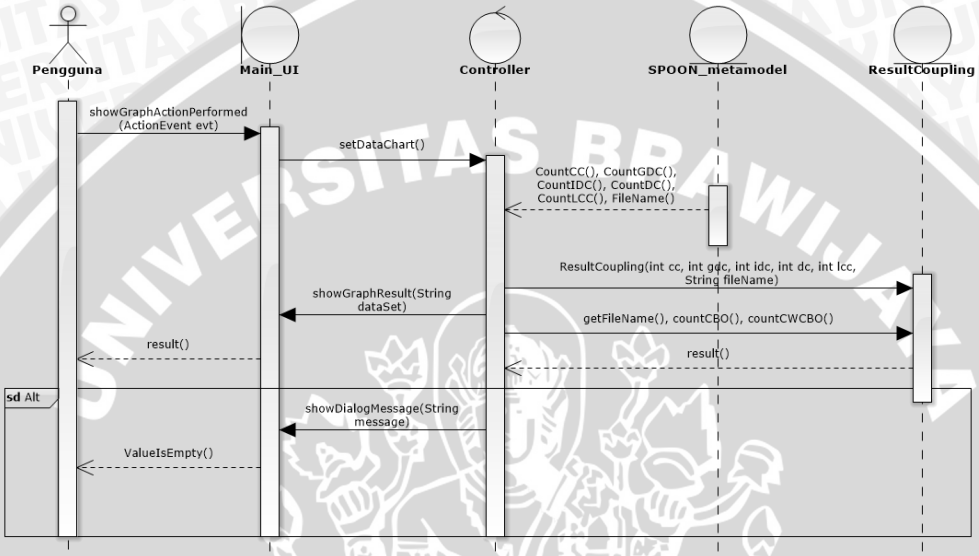
Sequence diagram hitung kopling menjelaskan interaksi *user* dengan sistem dalam menganalisis dan menghitung nilai kopling terhadap *source code* dengan bahasa java yang telah diinputkan sebelumnya. Hasil dari operasi ini adalah berupa tampilan nilai kopling CBO dan CWCBO. Objek-objek yang berinteraksi pada *sequence diagram* ini adalah *Main_UI, Controller, SPOON_metamodel* dan *ResultKopling*. *Sequence diagram* dari hitung kopling diperlihatkan pada gambar 5.3 berikut.



Gambar 5.3 *Sequence Diagram* Hitung Kopling

3. [SC_03] Tampilkan Grafik

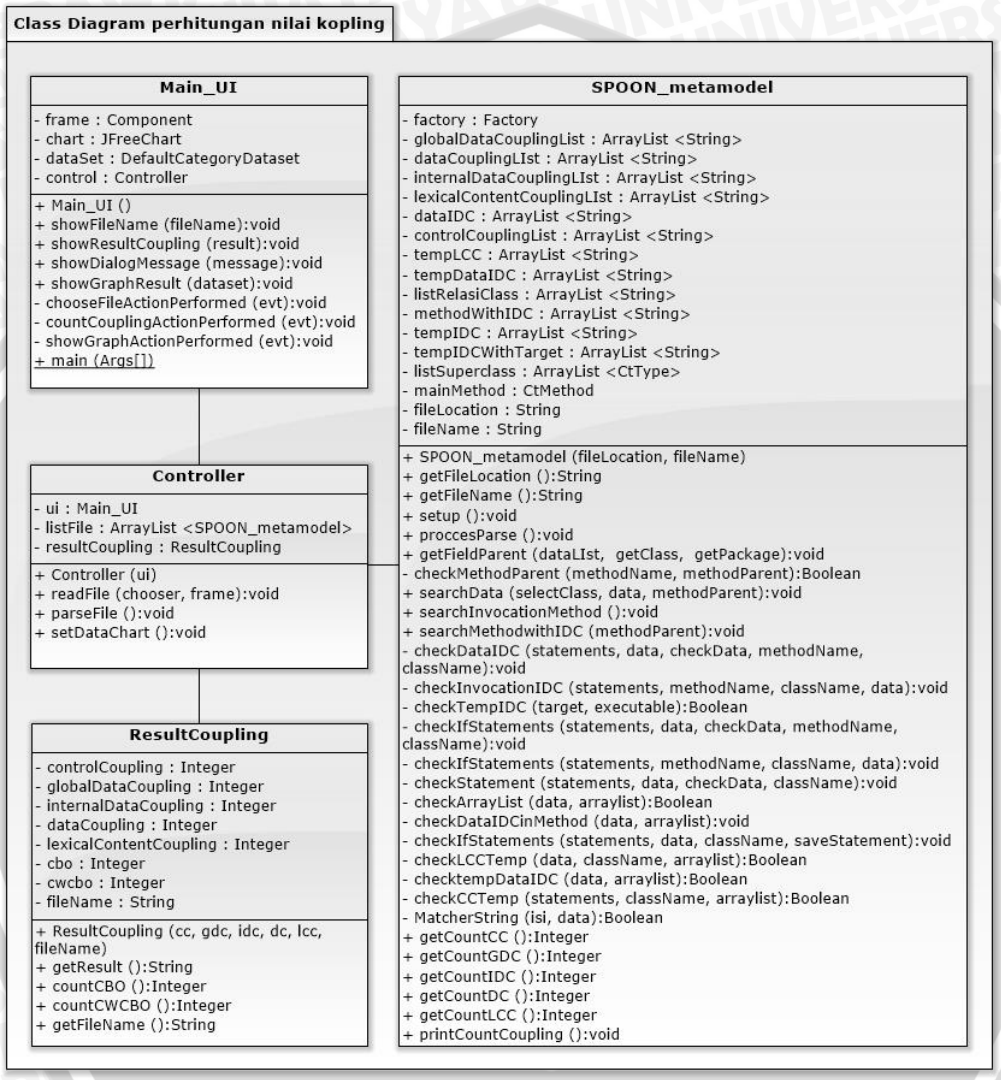
Sequence diagram Tampilkan grafik menjelaskan interaksi *user* dengan sistem dalam membangkitkan grafik berdasarkan nilai kopling CBO dan CWCBO yang telah didapatkan dari proses Hitung kopling sebelumnya. Hasil dari operasi ini adalah berupa tampilan grafik perbandingan nilai kopling CBO dan CWCBO. Objek-objek yang berinteraksi pada *sequence diagram* ini adalah *Main_UI, Controller, SPOON_metamodel* dan *ResultKopling*. *Sequence diagram* dari Tampilkan Grafik diperlihatkan pada gambar 5.4 berikut.



Gambar 5.4 *Sequence Diagram* Tampilkan Grafik

5.1.3 Perancangan *Class Diagram*

Diagram kelas menggambarkan sistem dan relasi-relasi yang ada didalamnya. *Class diagram* yang digunakan sebagai perancangan kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO) dapat dilihat pada gambar 5.5 berikut.



Gambar 5.5 *Class Diagram* Perhitungan Nilai Kopling

5.1.4 Perancangan Alur Parser *Source Code*

Perancangan alur parser *source code* merupakan perancangan yang menjelaskan tentang alur bagaimana menganalisis sebuah inputan yang berupa *source code* atau kode program. Perancangan ini bertujuan untuk mendapatkan nilai dari lima tipe kopling berdasarkan metrik CBO. Kelima tipe kopling tersebut yaitu, *Control Coupling*, *Global Data Coupling*, *Internal Data Coupling, Data Coupling* dan *Lexical Content Coupling*. Penulisan rancangan alur parser *source code* ini diurutkan berdasarkan tipe kopling yang pertama hingga terkahir dianalisis, karena dalam proses analisisnya, setiap tipe kopling saling berkaitan. Urutan-urutan perancangannya adalah Perancangan *Global Data Coupling*, Perancangan *Data Coupling*, Perancangan *Control Coupling*, Perancangan *Lexical Content Coupling* dan Perancangan *Internal Data Coupling*.

5.1.4.1 Perancangan *Global Data Coupling*

Alur dalam mendapatkan nilai *global data coupling* adalah sebagai berikut:

1. Mencari semua *class* pada file Java yang diinputkan.
2. Mencari *parentclass* beserta variabelnya.
3. Mencari *method* dari *class* yang merupakan *class* turunannya.
4. Menelusuri setiap *method-method* berdasarkan *statement-statement*. Jika terdapat variabel yang sama dengan variabel yang ada pada *parentclass* maka variabel tersebut merupakan *global data coupling*.

5.1.4.2 Perancangan *Data Coupling*

Alur dalam mendapatkan nilai *data coupling* adalah sebagai berikut:

1. Mencari semua *class* pada file Java yang diinputkan.
2. Mencari *parentclass* beserta variabelnya.
3. Mencari *method* dari *class* yang merupakan *class* turunannya.
4. Menelusuri setiap *method-method* berdasarkan *statement-statement*.
5. Mencari *statement* yang merupakan percabangan *if else*.
6. Menelusuri setiap parameter kondisi yang ada. Jika parameter kondisinya terdapat variabel yang sama dengan *global data coupling* yang dicari sebelumnya, maka variabel tersebut termasuk ke dalam *data coupling.*

5.1.4.3 Perancangan *Control Coupling*

Alur dalam mendapatkan nilai *control coupling* adalah sebagai berikut:

1. Mencari semua *class* pada file Java yang diinputkan.
2. Mencari *parentclass* beserta variabelnya.
3. Mencari *method* dari *class* yang merupakan *class* turunannya.
4. Menelusuri setiap *method* berdasarkan *statement-statement*.
5. Mencari *statement else* (tidak terdapat parameter kondisi) pada percabangan *if else*. Jika parameter kondisi pada *statement if else* sebelumnya terdapat *data coupling* yang dicari sebelumnya, maka *statement else* tersebut merupakan *control coupling.*

5.1.4.4 Perancangan *Lexical Content Coupling*

Alur dalam mendapatkan nilai *lexical content coupling* adalah sebagai berikut:

1. Mencari semua *class* pada file Java yang diinputkan.
2. Mencari *parentclass* beserta variabelnya.
3. Mencari *method* dari *class* yang merupakan *class* turunannya.
4. Menelusuri setiap *method-method* berdasarkan *statement-statement*. Jika terdapat variabel yang sama dengan *global data coupling* namun memiliki *class* yang berbeda, maka variabel tersebut merupakan *lexical content coupling*

5.1.4.5 Perancangan *Internal Data Coupling*

Alur dalam mendapatkan nilai *internal data coupling* adalah sebagai berikut:

1. Mencari semua *class* pada file Java yang diinputkan.
2. Mencari *parentclass* beserta variabelnya.
3. Mencari *method* dari *class* yang merupakan *class* turunannya.
4. Menelusuri setiap *method* berdasarkan *statement-statement*.
5. Mencari *statement* yang merupakan *statement* operasi. Jika *leftassigned* (variabel yang berada di sebelah kiri simbol operator, seperti “+=”) sama dengan *global data coupling*, maka variabel tersebut disimpan sementara

pada suatu variabel penyimpanan, diasumsikan variabel penyimpanannya tempIDC.

6. Menelusuri setiap *method* berdasarkan *statement-statement* pada class yang lainnya. Mencari *statement* yang merupakan *statement* operasi. Jika *leftassigend* sama dengan tempIDC, variabel tersebut disimpan sementara pada suatu variabel penyimpanan, diasumsikan variabel penyimpanannya dataIDC.
7. Menelusuri *parentclass*. Jika ada *method* pada *parentclass* yang berisi *dataIDC*, dimana *dataIDC* tersebut merupakan *leftassigned* dalam *statement* persamaan, maka method tersebut disimpan dalam *variable* methodWithIDC.
8. Mencari *class* yang mengandung *main method*.
9. Menelusuri *main method.* Jika pada *main method* terdapat lebih dari satu *instance* objek dari *parentclass,* dan setiap objek memanggil methodWithIDC, maka variabel dalam dataIDC merupakan internal data coupling.

5.1.5 Perancangan Komponen

Perancangan komponen merupakan kegiatan menguraikan setiap komponen untuk menjelaskan bagaimana setiap komponen dalam sistem berjalan (Sommerville, 2011). Perancangan komponen menjelaskan secara detail dari atribut-atribut dan algoritma-algoritma yang akan diimplementasikan pada kelas-kelas yang yang telah dimodelkan pada *class diagram.* Algoritma yang dituliskan merupakan *pseudocode* yang menjembatani antara bahasa manusia dan kode program. Dalam perancangan komponen pada sistem ini memakai tiga *sample class,* yaitu *class Controller, class SPOON_metamodel* dan *class ResultCoupling*.

5.1.5.1 *Class* Controller

Pada *class* Controller terdapat tiga atribut, antara lain ditunjukkan pada tabel 5.2 berikut ini.

Tabel 5.2 Atribut *Class* Controller

Nama Atribut	Tipe Data	Modifier
ui	Main_UI	Private
listFile	ArrayList<SPOON_metamodel>	Private
resultCoupling	ResultCoupling	Private

Tabel 5.3 Algoritma *method* parseFile()

Algoritma *method* parseFile()
parseFile() variabel hasil = kosong Melakukan perulangan sebanyak isi listFile Menampikan nama dan lokasi file Membuat objek baru dari *class* SPOON_metamodel dengan nama a Mengeksekusi jika tidak terdapat error pada kode program inputan Menjalankan fungsi setup() dari objek a Menjalankan fungsi ProccesParse() dari objek a Menjalankan fungsi printCountCOupling() dari objek a Membuat objek baru dari class ResultCoupling dengan nama resultCoupling Mendeklarasikan objek resultCoupling dengan memanggil constract pada class ResultCoupling(jumlah CC, jumlah GDC, jumlah IDC, jumlah DC, jumlah LCC, nama file) Menambahkan nilai variable hasil dengan memanggil fungsi getResult() dari objek resultCOupling Mengeksekusi jika terdapat error pada kode program inputan Variable pesan = "Program yang anda masukkan error"; Memanggil pesan dari objek ui untuk menampilkan pesan Menampilkan isi dari variabel hasil Jika listFile = kosong Variable pesan = "File kosong, masukkan file yang akan di uji!" Memanggil fungsi dari class Main_UI untuk menampilkan isi pesan; Memanggil fungsi dari ui untuk menampilkan nilai dari variabel hasil

Tabel 5.4 Algoritma *method* setDataChart()

Algoritma *method* setDataChart()
setDataChart() membuat objek baru dari class DefaultCategoryDataset dengan nama dataSet membuat objek baru dari class SPOON_metamodel dengan nama a; variabel value = 0 melakukan perulangan sejumlah isi listFile objek a sama dengan isi dari setiap listFIle mengeksekusi jika terdapat error pada kode program inputan menambahkan nilai pada variabel value dengan memanggil fungsi getCountGDC() dari objek a Mendeklarasikan objek resultCoupling dengan memanggil constract pada class ResultCoupling(jumlah CC, jumlah GDC, jumlah IDC, jumlah DC, jumlah LCC, nama file) Variabel file = memanggil fungsi getFileName() dari objek resultCoupling Menambahkan data pada objek dataset dengan parameter nilai CBO, “CBO” dan nama file Menambahkan data pada objek dataset dengan parameter nilai CWCBO, “CWCBO” dan nama file Mengeksekusi jika tidak error pada kode program inputan Variabel pesan = "Program yang anda masukkan error"

Tabel 5.4 Algoritma *method* setDataChart() (lanjutan)

Algoritma *method* setDataChart()
Memanggil fungsi dari objek ui untuk menampilkan isi pesan Menampilkan isi dari variabel value Jika nilai value kurang atau sama dengan nol Variabel pesan = "Hasil kosong, lakukan perhitungan kopling terlebih dahulu!" Memanggil fungsi dari objek ui untuk menampilkan isi pesan Jika nilai value lebih besar dari 0 Memanggil fungsi dari objek ui untuk menampilkan isi dari variabel value dalam bentuk diagram

5.1.5.2 *Class* SPOON_metamodel

Pada *class* SPOON_metamodel terdapat 17 atribut, antara lain ditunjukkan pada tabel 5.5 berikut ini.

Tabel 5.5 Atribut Class Spoon_metamodel

Nama Atribut	Tipe Data	Modifier
factory	Factory	Private
globalDataCouplingList	ArrayList<String>	Private
dataCouplingList	ArrayList<String>	Private
lexicalContentCouplingList	ArrayList<String>	Private
dataIDC	ArrayList<String>	Private
internalDataCouplingList	ArrayList<String>	Private
tempLCC	ArrayList<String[]>	Private
controlCouplingList	ArrayList<String[]>	Private
tempDataIDC	ArrayList<String[]>	Private
listRelasiClass	ArrayList<String[]>	Private
methodWithIDCList	ArrayList<String[]>	Private
tempIDC	ArrayList<String[]>	Private
tempIDCwithTarget	ArrayList<String[]>	Private
listSuperclass	ArrayList<CtType>	Private
mainMethod	CtMethod	Private
fileLocation	String	Private
fileName	String	Private

Tabel 5.6 Algoritma *method* checkMethodParent()

Algoritma *method* checkMethodParent()
checkMethodParent(methodName, methodParent) { mengulang sejumlah methodParent variabel get = methodParent terpilih jika methodName = get return true return false

Tabel 5.7 Algoritma *method* searchData()

Algoritma *method* searchData()
searchData(selectClass, data, methodParent) variabel className = parameter selectClass, yaitu class terpilih dari inputan kode program variabel listMethod = daftar method pada setiap class yang terpilih mengulang sejumlah method pada class terpilih variabel get = setiap method pada class terpilih jika method yang terpilih bukan merupakan method pada parentclass variabel listParameters = daftar parameter pada method terpilih variabel checkData = parameter data, yaitu variabel pada parentclass mengulang sejumlah parameter pada method terpilih variabel parameter = setiap parameter pada method terpilih jika nama parameter = isi data variabel checkData = nama package + nama class + ".this." + isi data break memanggil fungsi checkStatement(statement dari method terpilih, data, checkData, className)

Tabel 5.8 Algoritma *method* searchInvocationMethod()

Algoritma *method searchInvocationMethod()*
searchInvocationMethod() variabel packageList = daftar package dari inputan kode program mengulang sejumlah package dari inputan kode program variabel getPackage = setiap package yang terpilih; variabel a = daftar class pada setiap package terpilih mengulang sejumlah class pada package terpilih variabel get = setiap class terpilih variabel listMethod = daftar method dari class terpilih mengulang sejumlah method dari class terpilih variabel getMethod = setiap method terpilih jika method = method main mengulang sejumlah methodWithIDCList variabel get1 = methodWithIDCList yang terpilih memanggil fungsi checkInvocationIDC(statement method terpilih, nama method terpilih, nama class terpilih, get1);

Tabel 5.9 Algoritma *method* checkStatement()

Algoritma *method* checkStatement()

private void checkStatement(statements, data, checkData, className) {
mengulang sejumlah statements dari inputan parameter
variabel get1 = setiap statements inputan parameter
jika pada statements terpilih terdapat checkData
jika isi parameter data belum ada pada variabel globalDataCouplingList
menambahkan isi data pada globalDataCouplingList
variabel save = data dan className
menambahkan data dan className pada variabel tempLCC
 	
jika isi data ada pada variabel global globalDataCouplingList dan isi data belum ada pada variabel lexicalContentCoupling
jika fungsi checkLCCTemp(data, className, tempLCC) bernilai true
menambahkan isi data pada variabel lexicalContentCouplingList
jika jenis statements merupakan percabangan if else
memanggil fungsi checkIfStatements(statements If else, data, checkData, className, statements terpilih);
jika jenis statements merupakan perulangan while
variabel condition = parameter kondisi perulangan while
jika pada parameter kondisi perulangan terdapat isi dari variabel checkData
jika isi data terilih belum ada pada variabel dataCouplingLIst
menambahkan data terpilih ke variabel dataCouplingLIst
variabel bodyStatements = statements dari isi body perulangan while
memanggil fungsi checkStatement(statements body, data, checkData, className)
jika jenis statements merupakan perulangan for
variabel condition = parameter kondisi perulangan for
jika pada parameter kondisi perulangan terdapat isi dari variabel checkData
jika isi data terilih belum ada pada variabel dataCouplingLIst
menambahkan data terpilih ke variabel dataCouplingLIst
variabel bodyStatements = statements dari isi body perulangan while
memanggil fungsi checkStatement(statements body, data, checkData, className)
jika jenis statements merupakan percabangan switch
variabel condition = parameter kondisi percabangan switch
jika pada parameter kondisi percabangan terdapat isi dari variabel checkData
jika isi data terpilih belum ada pada variabel dataCouplingLIst
menambahkan data terpilih ke variabel dataCouplingLIst
variabel cases = daftar case dari percabangan switch case
mengulang sejumlah case
variabel get = case terpilih
variabel save = statement terpilih, className
jika fungsi checkCCTemp(statements, className, controlCouplingList) bernilai false dan fungsi checkArrayList(data, dataCouplingList) bernilai benar
menambahkan data terpilih pada variabel controlCouplingList
memanggil fungsi checkStatement(statement dalam case, data, checkData, className);

Tabel 5.9 Algoritma *method* checkStatement() (lanjutan)

Algoritma *method* checkStatement()
jika jenis statements merupakan operator perhitungan variabel leftAssigned = daftar variabel yang berada di sebelah kiri tanda "=" variabel save = leftAssigned dan className jika checkData terpilih sama dengan leftAssigned terpilih jika data terpilih terdapat pada variabel globalDataCouplingList dan data terpilih belum ada pada variabel dataIDC serta data terpilih ada variabel tempDataIDC menambahkan data terpilih pada variabel dataIDC jika isi dari save tidak terdapat pada variabel tempDataIDC menambahkan isi save pada variabel tempDataIDC

5.1.5.3 *Class* ResultCoupling

Pada *class* ResultCoupling terdapat 8 atribut, antara lain ditunjukkan pada tabel 5.10 berikut ini.

Tabel 5.10 Atribut Class ResultCoupling

Nama Atribut	**Tipe Data**	**Modifier**
controlCoupling	int	Private
globalDataCoupling	Int	Private
internalDataCoupling	Int	Private
dataCoupling	Int	Private
lexicalContentCoupling	Int	Private
cbo	Int	Private
cwcbo	Int	Private
fileName	String	Private

Tabel 5.11 Algoritma *method* countCBO()

Algoritma *method* countCBO()
countCBO() cbo = (jumlah Control Coupling) + (jumlah Global Data Coupling) + (jumlah Interna Data Coupling) + (jumlah Data Coupling) + (jumlah Lexical Data Coupling) return cbo

Tabel 5.12 Algoritma *method* countCWCBO()

Algoritma *method* countCWCBO()
countCWCBO() { cwcbo = (jumlah Control Coupling x 1) + (jumlah Global Data Coupling x 1) + (jumlah Interna Data Coupling x 2) + (jumlah Data Coupling x 3) + (jumlah Lexical Data Coupling x 4) return cwcbo

5.1.6 Perancangan Antarmuka

Pada perancangan antarmuka menghasilkan rancangan tampilan antarmuka sistem untuk berkomunikasi dengan *user.* Perancangan antarmuka kakas bantu perhitungan nilai kopling menggunakan metrik *Cognitive Weighted Coupling Between Object* (CWCBO) dapat diperlihatkan pada Gambar 5.6 berikut.

Gambar 5.6 Perancangan Antarmuka

Keterangan Gambar 5.5:

1. Tombol “Cari Berkas”, tombol untuk mencari file sumber kode.
2. Tombol “Hitung Kopling”, tombol untuk menghitung nilai kopling.
3. Tombol “Tampilkan Grafik”, tombol untuk menampilkan grafik nilai kopling.
4. *Text Area*, tempat untuk menampilkan daftar nama file yang diinputkan.
5. *Text Area*, tempat menampilkan hasil dari perhitungan nilai kopling.

5.2 Implementasi

Pada tahap implementasi perangkat lunak terdapat tiga bagian yaitu, spesifikasi pengembangan sistem, batasan implementasi dan implementasi antarmuka.

5.2.1 Spesifikasi Sistem

Spsefikasi pengembangan yang terdapat pada sistem kakas bantu perhitungan nilai kopling dengan metrik *Cognitive Weighted Coupling Between Object* (CWCBO) dibagi menjadi dua lingkungan implementasi yaitu, spesifikasi perangkat keras dan spesifikasi perangkat lunak. Berikut ini penjelasan spesifikasi perangkat keras dan spesifikasi perangkat lunak pengembangan sistem.

5.2.1.1 Spesifikasi Perangkat Keras

Pada tabel 5.13 berikut ini, merupakan spesifikasi sistem perangkat keras yang digunakan dalam pengembangan sistem kakas bantu perhitungan nilai kopling dengn metrik *Cognitive Weighted Coupling Between Object (CWCBO)*.

Tabel 5.13 Spesifikasi Perangkat Keras

Nama Komponen	**Spesifikasi**
System Model	ASUS A43S
Processor	Intel Core i5
Hard Disk Drive	640GB
Memory RAM	4GB DDR3
Display	14” HD (1920 x 1080)

5.2.1.2 Spesifikasi Perangkat Lunak

Spesifikasi sistem perangkat lunak yang digunakan dalam pengembangan sistem kakas bantu perhitungan nilai kopling dengn metrik *Cognitive Weighted Coupling Between Object* (CWCBO) dapat dilihat pada tabel 5.14 berikut ini.

Tabel 5.14 Spesifikasi Perangkat Lunak

Nama Komponen	**Spesifikasi**
Operating System	Windows 10 Pro 64-bit
Programming Language	Java
Text Editor/ IDE	Netbeans 8.1.0 dengan JDK 1.8

5.2.2 Batasan Implementasi

Batasan-batasan yang terdapat pada implementasi sistem kakas bantu perhitungan nilai kopling ini adalah sebagai berikut:

1. Data yang dimasukkan kedalam sistem untuk dibangkitkan kasus ujinya merupakan file berekstensi Java.
2. Program yang menjadi inputan harus dijadikan dalam satu file Java.
3. Sistem hanya dapat melakukan perhitungan pada source code yang *object oriented* dan terdapat struktur hirarki.
4. Perhitungan nilai kopling didasarkan pada metrik CBO dan CWCBO, dimana pada masing metrik terdapat lima jenis tipe kopling, yaitu *Control Coupling, Global Data Coupling, Internal Data Coupling, Data Coupling* dan *Lexical Content Coupling.*

5.2.3 Implementasi Antarmuka

Implementasi antarmuka dilakukan berdasarkan pada hasil perancangan antarmuka yang telah dibuat sebelumnya. Sistem yang dibuat terdapat dua halaman antarmuka, yaitu halaman utama untuk menampilkan daftar file inputan beserta hasil perhitungan nilai koplingnya dan halaman grafik untuk menampilkan grafik perbandingan nilai kopling CBO dan CWCBO.

5.2.3.1 Antarmuka Halaman Utama

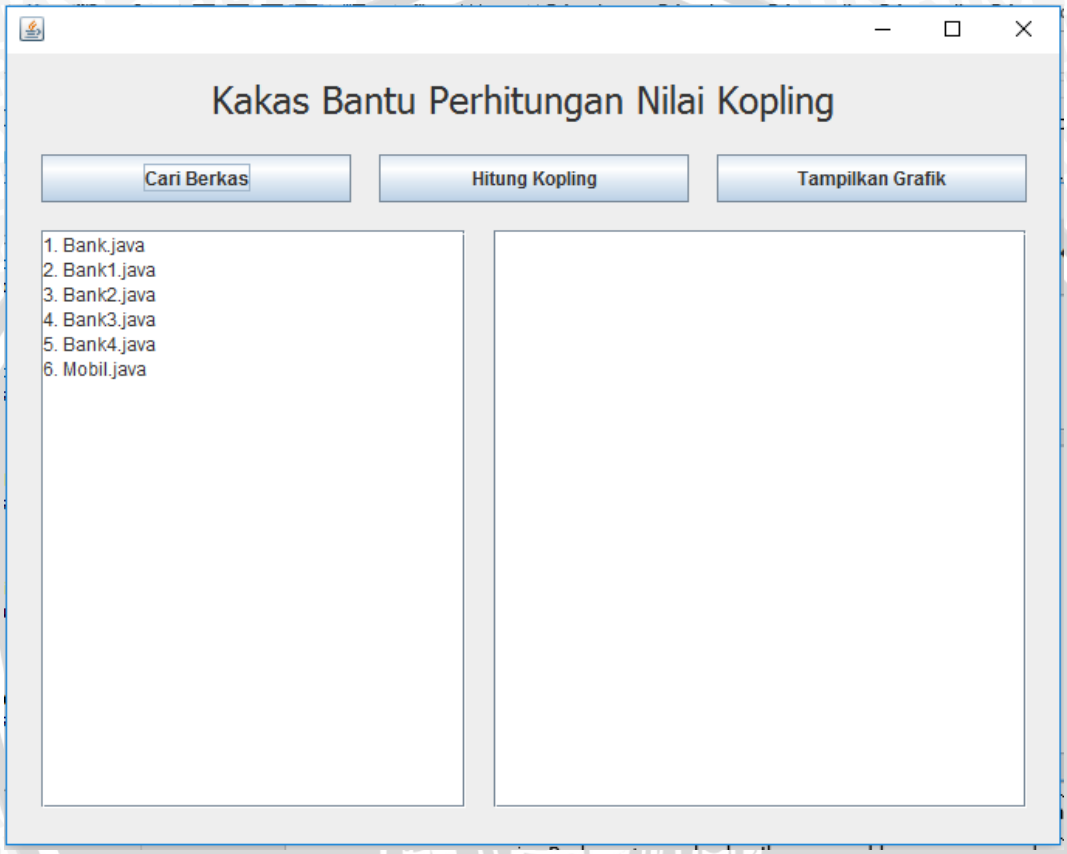
Implementasi antarmuka halaman utama diperlihatkan pada Gambar 5.7 berikut ini.

Gambar 5.7 Antarmuka Halaman Utama

Pada halaman utama sistem terdapat tiga tombol dan dua *textfield* yang digunakan *user* untuk menjalankan fungsi sistem antara lain: tombol "Cari Berkas", tombol "Hitung Kopling", tombol "Tampilkan Grafik", *textfield* daftar file inputan dan hasil perhitungan kopling.

5.2.3.2 Antarmuka Hasil Memasukkan Berkas File

Implementasi antarmuka hasil memasukkan *source code* diperlihatkan pada Gambar 5.8 berikut ini.



Gambar 5.8 Antarmuka Hasil Memasukkan Berkas FIle

Pada antarmuka hasil memasukkan Berkas File ditampilkan daftar dari file *source code* yang terdapat pada *text area* sebelah kiri.

5.2.3.3 Antarmuka Hasil Perhitungan Kopling

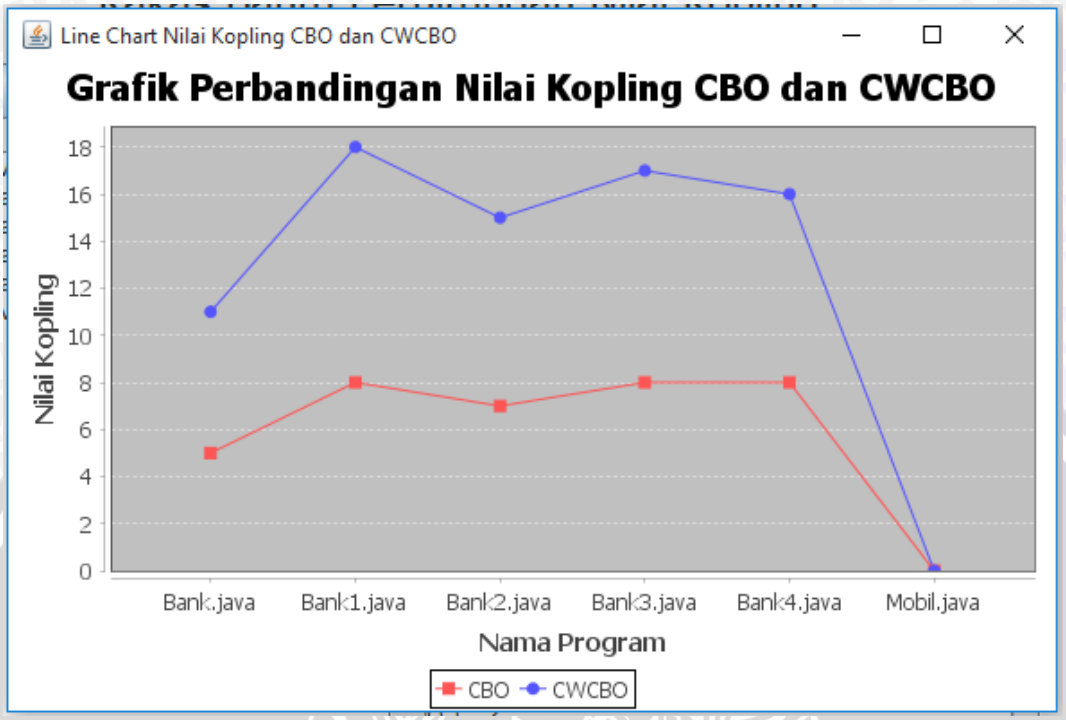
Implementasi antarmuka hasil perhitungan nilai kopling diperlihatkan pada Gambar 5.9 berikut ini.

Gambar 5.9 Antarmuka hasil perhitungan nilai kopling

Hasil perhitungan nilai kopling dari file *source code* ditunjukkan pada *text area* sebelah kanan. Pada hasil pembangkitan ditampilkan nama file, nilai tiap tipe kopling yang menjadi parameter serta nilai kopling CBO dan CWCBO.

5.2.3.4 Antarmuka Hasil Tampilkan Grafik

Implementasi antarmuka hasil tampilkan grafik diperlihatkan pada Gambar 5.10 berikut ini.



Gambar 5.10 Antarmuka Hasil Tampilkan Grafik

Hasil dari perhitungan kopling CBO dan CBO ditampilkan pada tampilan berupa grafik baris. Pada grafik dapat dilihat bahwa nilai kopling CBO dan CWCBO ditampilkan berdasarkan setiap nama kode program yang diinputkan.

BAB 6 PENGUJIAN

Bab ini menjelaskan tentang pengujian dan analisis hasil pengujian terhadap sistem yang telah dibuat. Proses pengujian dimulai dari pengujian kode program yang dilakukan pada pengujian unit, kemudian dilanjutkan pengujian arsitektur yang dilakukan pada pengujian integrasi, kemudian dilanjutkan dengan pengujian fungsionalitas sistem yang dilakukan pada pengujian validasi, dan yang terakhir adalah pengujian keakuratan sistem dalam melakukan perhitungan kopling yang dilakukan pada pengujian akurasi. Sedangkan proses analisis bertujuan untuk mendapatkan kesimpulan dari semua hasil pengujian yang telah dilakukan.

6.1 Pengujian

Proses pengujian terdiri dari pengujian unit, pengujian integrasi, pengujian validasi dan pengujian akurasi. Pengujian unit digunakan untuk memastikan bahwa algoritma yang dibuat pada setiap unit dalam sistem sudah benar. Pengujian integrasi digunakan untuk memastikan bahwa hubungan antar modul-modul dalam sistem sudah berjalan dengan benar. Pengujian validasi digunakan untuk memastikan bahwa sistem telah mencakup semua kebutuhan yang telah didefinisikan sebelumnya. Pengujian akurasi digunakan untuk memastikan bahwa sistem dapat menghitung nilai kopling secara akurat sesuai batasan minimal akurasi yang telah ditentukan.

6.1.1 Pengujian Unit

Pengujian unit merupakan proses pengujian komponen yang berfokus untuk memverifikasi unit terkecil pada perancangan perangkat lunak (Pressman, 2010). Pengujian unit yang dilakukan meliputi: pengujian unit Spoon_metamodel dan pengujian unit Controller.

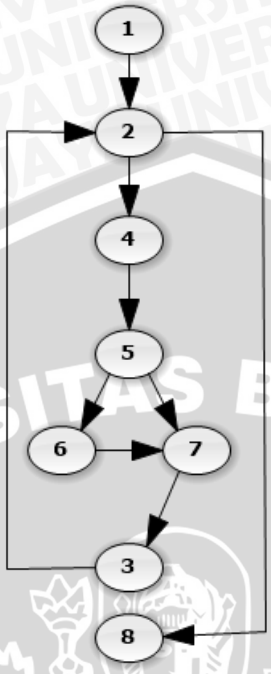
6.1.1.1 Pengujian Unit SPON_metamodel

Berikut ini adalah pengujian unit pada *class* SPOON_metamodel *method* checkArrayList. Algoritma checkArrayList dijelaskan secara detail pada tabel 6.1.

Tabel 6.1 Algoritma checkArrayList

```
boolean checkArrayList(Object data, ArrayList arraylist) {
    (1)for (int i = 0; (2)i < arraylist.size();(3)i++) {
        (4)Object get = arraylist.get(i);
        (5)if (get.equals(data)) {
            (6)return true;
        (7)}
    (8)}
    (8)return false;
}
```

Berdasarkan algoritma *checkArrayList* pada tabel 6.1, maka dihasilkan *flow graph* yang diperlihatkan pada Gambar 6.1 berikut.



Gambar 6.1 *Flow Graph Method* checkArrayList

Berdasarkan *flow graph method checkArrayList* pada Gambar 6.1, maka dapat dihitung nilai cyclometic complexity V(G) sebagai berikut.

V(G) = R = 3

V(G) = E – N + 2

= 9 - 8 + 2

= 3

V(G) = P + 1

= 2 + 1

= 3

Sehingga, dapat dihasilkan tiga jalur independen berikut ini.

Jalur 1 : 1 – 2 – 8

Jalur 2 : 1 – 2 – 4 – 5 – 7 – 3 – 2 – 8

Jalur 3 : 1 – 2 – 4 – 5 – 6 – 7 – 3 – 2 – 8

Berdasarkan jalur independen yang dihasilkan, maka dapat dihasilkan kasus ujinya yang ditunjukkan pada Tabel 6.2.

Tabel 6.2 Kasus Uji *Method* checkArrayList

Jalur	Kasus Uji	Prosedur Uji	*Excpected Result*	*Test Result*
1	arrayList kosong	arrayList.size() = 0	Method mengembalikan nilai “false”	Method mengembalikan nilai “false”
2	arrayList tidak kosong, dan hasilnya false	Inputan parameter arrayList = amtde; data = accno;	Method mengembalikan nilai “false”	Method mengembalikan nilai “false”
3	arrayList tidak kosong dan hasilnya true	Inputan parameter arrayLIst = amtde; data = amtde;	Method mengembalikan nilai “true”	Method mengembalikan nilai “true”

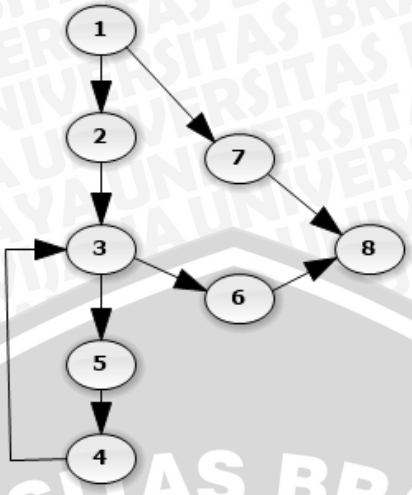
6.1.1.2 Pengujian Unit *Controller*

Berikut ini adalah pengujian unit pada *class Controller method* readFile. Algoritma readFile dijelaskan secara detail pada tabel 6.3.

Tabel 6.3 Algoritma *Method* readFile

```
public void readFile(JFileChooser chooser, Component frame) {
    (1)String showFileName = "";
    (1)chooser.setMultiSelectionEnabled(true);
    (1)int konten = chooser.showOpenDialog(null);
    (1)if (konten == JFileChooser.APPROVE_OPTION){
        (2)File[] files = chooser.getSelectedFiles();
        (2)for (int i = 0; (3)i < files.length; (4)i++) {
            (5)File file = files[i];
            (5)showFileName += (i + 1 + ". " + file.getName() + "\n");
            (5)listFile.add(new SPOON_metamodel(file.getAbsolutePath(),
file.getName()));
        (6)}
        (6)ui.showFileName(showFileName);
    (7)}else {
        (7)System.out.println("cancel");
    (8)}
}
```

Berdasarkan algoritma readFile pada tabel 6.3, maka dihasilkan *flow graph* yang diperlihatkan pada Gambar 6.2 berikut.



Gambar 6.2 *Flow Graph Method* readFile

Berdasarkan *flow graph method* readFile pada Gambar 6.2, maka dapat dihitung nilai *cyclometic complexity* V(G) sebagai berikut.

V(G) = R = 3

V(G) = E – N + 2

= 9 - 8 + 2

= 3

V(G) = P + 1

= 2 + 1

= 3

Sehingga, dapat dihasilkan tiga jalur independen berikut ini.

Jalur 1 : 1 – 7 – 8

Jalur 2 : 1 – 2 – 3 –6 – 8

Jalur 3 : 1 – 2 – 3 – 5 – 4 – 3 – 6 – 8

Berdasarkan jalur independen yang dihasilkan, maka dapat dihasilkan kasus ujinya yang ditunjukkan pada Tabel 6.4.

Tabel 6.4 Kasus Uji *Method* readFile

Jalur	Kasus Uji	Prosedur Uji	*Excpected Result*	*Test Result*
1	Cancel	Menekan tombol "Cancel" ketika ditampilkan jendela *browse file*	Jendela *browse file* keluar dan tidak ada file yang diinputkan pada sistem	Jendela *browse file* keluar dan tidak ada file yang diinputkan pada sistem

Tabel 6.4 Kasus Uji *Method* readFile (lanjutan)

Jalur	Kasus Uji	Prosedur Uji	*Excpected Result*	*Test Result*
2	Tidak ada file yang diinputkan	Tekan tombol "Open" pada jendela *browse file* tanpa memilih file	Tidak terjadi apa-apa. Dan jendela *browse file* tidak keluar	Tidak terjadi apa-apa. Dan jendela *browse file* tidak keluar
3	Terdapat sejumlah file inputan	1. Memilih sejumlah file Java 2. Tekan tombol "Open"	Sistem menyimpan file yang dipilih, dan ditampilkan daftar namanya	Sistem menyimpan file yang dipilih, dan ditampilkan daftar namanya

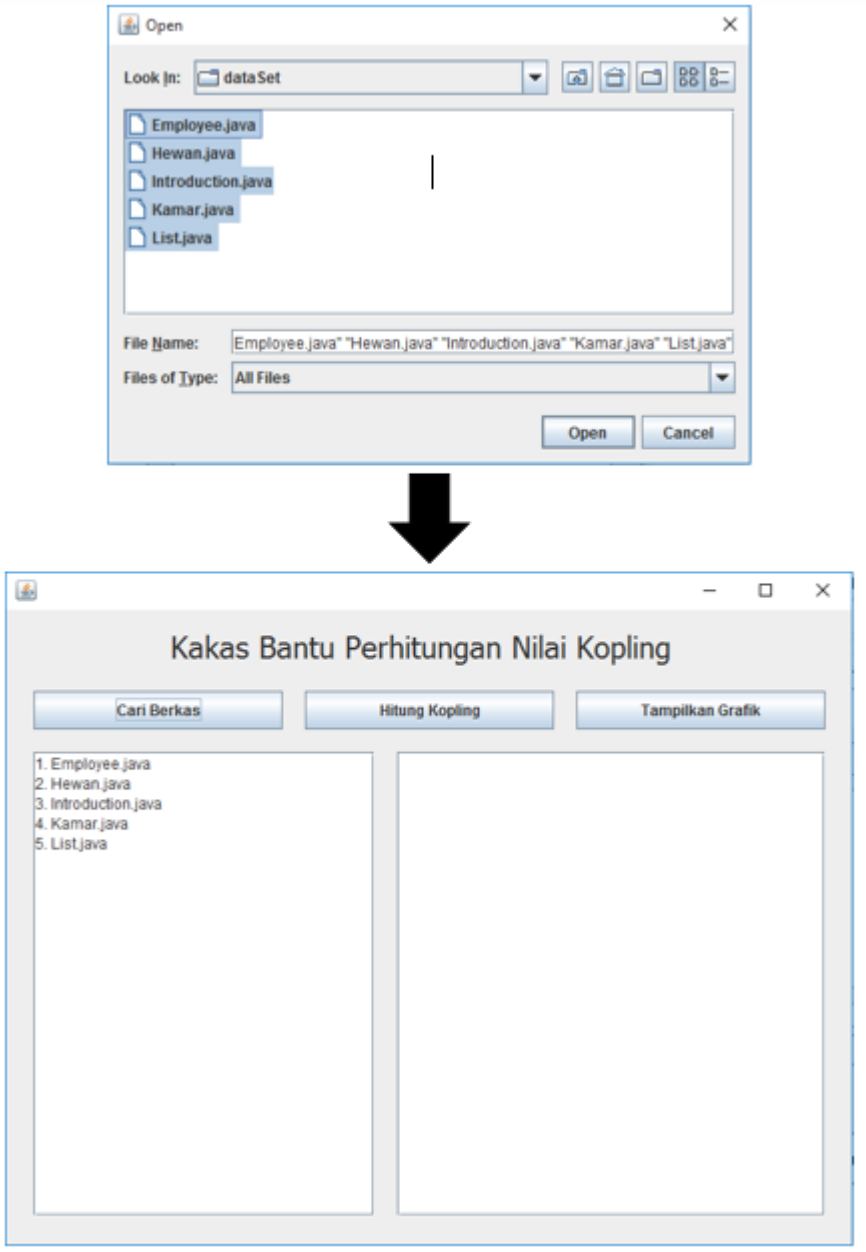
6.1.2 Pengujian Integrasi

Pengujian integrasi adalah teknik yang sistematis untuk membangun arsitektur perangkat lunak dengan melakukan tes untuk mengungkap kesalahan yang terkait dengan tampilan program. Pada pengujian integrasi ini dilakukan dengan menguji relasi antara dua modul, yaitu modul readFile() pada *class* Controller dan modul showFileName() pada *class* Main_UI. Kedua modul tersebut memilifi fungsi untuk memasukkan dan menampilkan berkas ke dalam sistem. Modul showFileName() merupakan modul yang akan dipanggil oleh modul readFile. Untuk pengujian integrasi antara kedua modul dapat dilihat pada Tabel 6.5 sebagai berikut:

Tabel 6.5 Pengujian Integrasi Antar Modul

Nama Kasus Uji	Pengujian Integrasi Antar Modul
Objek Uji	Modul showFileName() dan readFile()
Tujuan Pengujian	Memastikan bahwa kedua modul tersebut telah terintegrasi dengan benar
Data Masukan	File kode berekstensi java: Employee.java, Hewan.java, Introduction.java, Kamar.java, List.java
Prosedur Uji	1. User menekan tombol "Cari Berkas" 2. User memilih beberapa file seperti pada daftar masukan 3. User menekan tombol "*open*"
Hasil yang Diharapkan	Sistem dapat memasukkan file yang dipilih dan menampilkan daftar filenya.
Hasil yang Diperoleh	Sistem dapat memasukkan file yang dipilih dan menampilkan daftar filenya.
Hasil Pengujian	Valid

Dari hasil pengujian pada Tabel 6.5 integrasi antara modul showFileName() dan readFile() dapat tervalidasi. Untuk tampilan *interface* hasil uji integrasi dari kedua modul tersebut dapat dilihat pada Gambar 6.3 berikut,



Gambar 6.3 Hasil Pengujian Integrasi Modul readFile dan showFileName

6.1.3 Pengujian Validasi

Pengujian validasi merupakan proses pengujian yang dilakukan untuk menguji validasi kebutuhan fungsional dan non fungsional sistem dengan metode *black box testing*. Penjelasan berikut ini merupakan pengujian validasi berdasarkan *use case scenario* yang telah dibuat.

6.1.3.1 Cari Berkas

Berikut ini merupakan pengujian validasi cari berkas yang dijelaskan pada tabel 6.6.

Tabel 6.6 Uji Cari Berkas

Nama Kasus Uji	Uji Cari Berkas
Objek Uji	Kebutuhan Fungsional SRS_F001
Tujuan Pengujian	Untuk memastikan sistem mampu menjalankan kebutuhan fungsional cari berkas.
Data Masukan	File kode berekstensi Java
Prosedur Uji	1. User menekan tombol "Cari Berkas" 2. User memilih beberapa file kode sumber .java 3. User menekan tombol "*open*"
Hasil yang Diharapkan	Sistem dapat menerima beberapa inputan yang berupa file berektensi java.
Hasil Pengujian	valid

Berikut ini merupakan pengujian validasi cari berkas alternatif 1 yang dijelaskan pada tabel 6.7.

Tabel 6.7 Uji Cari Berkas Alternatif 1

Nama Kasus Uji	Uji Cari Berkas Alternatif 1
Tujian Pengujian	Untuk memastikan sistem mampu menangani kondisi ketika pengguna batal memasukkan file
Objek Uji	Kebutuhan Fungsional SRS_F001
Data Masukan	-
Prosedur Uji	1. User menekan tombol "Cari Berkas" 2. User memilih file kode sumber .java 3. User menekan tombol "Cancel"
Hasil yang Diharapkan	Sistem tidak memasukkan file apapun ketika user menekan tombol "Cancel"
Hasil Pengujian	valid

6.1.3.2 Hitung Kopling

Berikut ini merupakan pengujian validasi hitung kopling yang dijelaskan pada tabel 6.8.

Tabel 6.8 Uji Hitung Kopling

Nama Kasus Uji	Uji Hitung Kopling
Tujian Pengujian	Untuk memastikan sistem mampu menjalankan kebutuhan fungsional Hitung Kopling.
Objek Uji	Kebutuhan Fungsional SRS_F002
Data Masukan	File berektensi Java
Prosedur Uji	1. User menekan tombol "Hitung Kopling"
Hasil yang Diharapkan	Sistem menampilkan hasil parse dan perhitungan kopling dari file *source code* yang diinputkan.
Hasil Pengujian	Valid

Berikut ini merupakan pengujian validasi Hitung Kopling alternatif 1 yang dijelaskan pada tabel 6.9.

Tabel 6.9 Uji Hitung Kopling Alternatif 1

Nama Kasus Uji	Uji Hitung Kopling Alternatif 1
Tujian Pengujian	Untuk memastikan sistem mampu menangani kondisi ketika pengguna melakukan perhitungan kopling tanpa memasukkan file inputan terlebih dahulu.
Objek Uji	Kebutuhan Fungsional SRS_F002
Data Masukan	-
Prosedur Uji	1. User menekan tombol "Hitung Kopling"
Hasil yang Diharapkan	Sistem menampilkan dialog message "File kosong, masukkan file yang akan diuji!"
Hasil Pengujian	Valid

6.1.3.3 Tampilkan Grafik

Berikut ini merupakan pengujian validasi Tampikan Grafik yang dijelaskan pada tabel 6.10.

Tabel 6.10 Uji Tampilkan Grafik

Nama Kasus Uji	Uji Tampilkan Grafik
Tujian Pengujian	Untuk memastikan sistem mampu menjalankan kebutuhan fungsional Tampilkan Grafik
Objek Uji	Kebutuhan Fungsional SRS_F003
Data Masukan	Nilai kopling CBO dan CWCBO
Prosedur Uji	1. User menekan tombol “Tampilkan Grafik”
Hasil yang Diharapkan	Sistem menampilkan grafik perbandingan nilai kopling *CBO* dan *CWCBO* pada *window* baru
Hasil Pengujian	valid

Berikut ini merupakan pengujian validasi Tampilkan Grafik alternatif 1 yang dijelaskan pada tabel 6.11.

Tabel 6.11 Uji Tampilkan Grafik Alternatif 1

Nama Kasus Uji	Uji Tampilkan Grafik Alternatif 1
Tujian Pengujian	Untuk memastikan sistem mampu menangani kondisi ketika *user* menampilkan grafik tanpa menghitung kopling terlebih dahulu, atau nilai kopling sama dengan nol.
Objek Uji	Kebutuhan Fungsional SRS_F003
Data Masukan	-
Prosedur Uji	1. User menekan tombol “Tampilkan Grafik”
Hasil yang Diharapkan	Sistem menampilkan dialog message “File kosong, masukkan file yang akan diuji!”
Hasil Pengujian	valid

6.1.4 Pengujian Akurasi

Pengujian akurasi merupakan proses pengujian yang membandingkan hasil perhitungan nilai kopling secara manual dan hasil perhitungan nilai kopling secara otomatis menggunakan sistem. Akurasi minimal kesesuian hasil perhitungannya adalah 90%. Dalam proses pengujian akurasi menggunakan 5 program yang akan dijadikan sebagai inputan sistem. Setiap program tersimpan dalam satu file berekstensi Java. Setiap file terdiri dari beberapa kelas yang saling berkaitan (berbasis objek). Program-program tersebut diambil dari sampel beberapa projek mata kuliah. Daftar kelima program inputan beserta hasil perhitungan kopling secara manual dapat dilihat secara lebih detail pada Tabel 6.12 sebagai berikut.

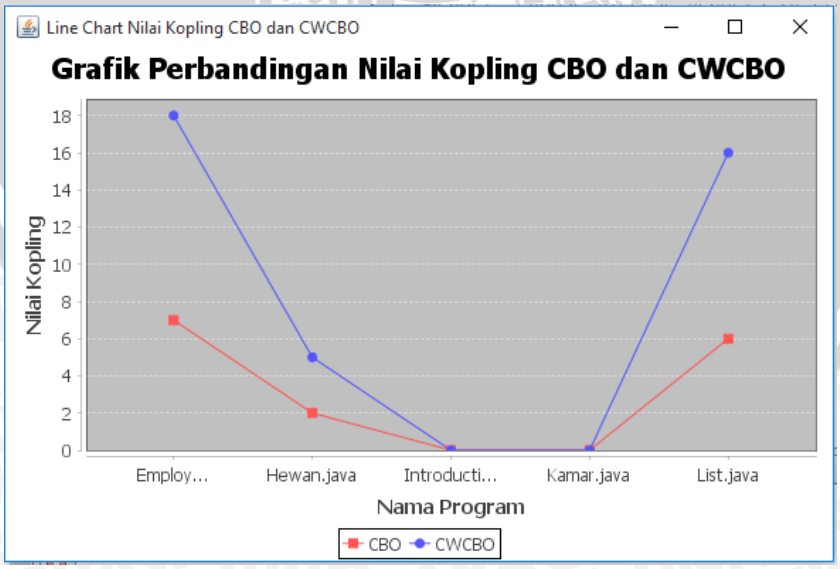
Tabel 6.12 Daftar Kode Program Inputan

Nama Program	Deskripsi	Analisis	CBO	CWCBO
Employee .java	Merupakan program sistem informasi sederhana untuk menghitung gaji pegawai berdasarkan lama bekerja dan jenis jabatan.	Tedapat 4 *class* dan mempunyai struktur *inheritance* (1 *parent class*, 2 class turunan dan 1 main class), terdapat 3 variabel global pada *parent class* yang berpotensi menjadi kopling pada class lainnya.	7	18
Hewan .java	Merupakan program untuk mendeskripsikan karakteristik beberapa hewan, seperti suara, habitat dan geraknya.	Terdapat 5 *class*, dimana 1 *class* yg jadi *parent class* merupakan class abstrak, 3 *class* lainnya merupakan *class* turunan dan 1 *class* lainnya merupakan *main class*. Pada *class parent* terdapat 1 variabel global yang berpotensi menjadi kopling pada class lainnya, 1 *construct* dan 4 *method abstract*.	2	5
Introduction .java	Merupakan program untuk pemesanan paket tur ke beberapa Negara seperti Jepang dan Prancis. Pada program ini juga dicantumkan biaya, jadwal tour dan jumlah tiket yang dapat dipesan pada setiap objek tujuan.	Tedapat 4 *class* dan mempunyai struktur *inheritance* (1 *parent class*, 2 *class* turunan dan 1 *main class*), terdapat 3 variabel global 1 pada *parent class* yang berpotensi menjadi kopling pada class lainnya. Terdapat 1 *interface* yang di-*implement* oleh kedua *class* turunan	0	0

Tabel 6.12 Daftar Kode Program Inputan (lanjutan)

Nama Program	Deskripsi	Analisis	CBO	CWCBO
Kamar .java	Program reservasi untuk kamar rawat inap pada rumah sakit. Pengguna dapat memasukkan nama, jenis penyakit dan alamat. Dan pengguna juga dapat memilih jenis kamar yang selanjutnya akan dicantumkan rincian biaya tiap hari,	Tedapat 4 *class* dan mempunyai struktur *inheritance* (1 *parent class*, 2 *class* turunan dan 1 *main class*), terdapat 3 variabel global 1 pada *parent class* yang berpotensi menjadi kopling pada *class* lainnya. Terdapat 1 *interface* yang di-*implement* oleh kedua *class* turunan	0	0
List.java	Program untuk mengurutkan nilai menggunakan *Sortable Inverted List*.	Terdapat 3 *class* yang mempunyai struktur hirarki bertingkat, yaitu 1 *class* menjadi *class parent*, 1 *class* menjadi turunan dari *class parent*, dan *class* yang lain menjadi turunan dari *class* turunan sebelumnya. Terdapat 2 variabel global yang berpotensi menjadi kopling pada *class* lainnya.	6	16

Hasil perhitungan nilai kopling menggunakan sistem pada lima kode program yang menjadi inputannya, dapat dilihat pada Gambar 6.4 sebagai berikut.



Gambar 6.4 Grafik Perbandingan Nilai Kopling CBO dan CWCBO

Dari grafik perbandingan nilai kopling tersebut menunjukkan bahwa metrik kopling CWCBO mempunyai nilai yang relatif lebih tinggi dibandingkan dengan metrik kopling CBO, hal ini didasari karena metrik CWCBO telah ditambahkan bobot pemahaman pada setiap jenis kopling yang menjadi parameternya. Setiap jenis kopling yang menjadi parameter metrik tersebut mempunyai bobot yang berbeda sesuai dengan tingkat kesulitan untuk dipahami. Sehingga 2 program yang berbeda dengan nilai CBO yang sama memungkinkan untuk menghasilkan nilai CWCBO yang berbeda, jika jumlah kopling pada masing-masing parameter berbeda.

Seperti contoh, Program1 memiliki nilai CBO = 5 dengan rincian sebagai berikut, CC = 1; GDC = 1; IDC = 1; DC = 1 dan LCC = 1. Sedangkan Porgram2 memiliki nilai CBO yang sama yaitu 5, dengan rincian sebagai berikut, CC = 0; GDC = 2; IDC = 1; DC = 0 dan LCC 2. Maka nilai CWCBO kedua program yaitu

Program1:

$$\begin{aligned} cwcbo &= ((CC * WFCC) + (GDC * WFGDC) + (IDC * WFIDC) \\ &\quad + (DC * WFDC) + (LCC * WFLCC) \\ &= ((1 * 1) + (1 * 1) + (1 * 2) + (1 * 3) + (1 * 4)) \\ &= (1) + (1) + (2) + (3) + (4) \\ &= 11 \end{aligned}$$

Program2:

$$\begin{aligned} cwcbo &= ((CC * WFCC) + (GDC * WFGDC) + (IDC * WFIDC) \\ &\quad + (DC * WFDC) + (LCC * WFLCC) \\ &= ((0 * 1) + (2 * 1) + (1 * 2) + (0 * 3) + (2 * 4)) \\ &= (0) + (2) + (2) + (0) + (8) \\ &= 12 \end{aligned}$$

Setelah dilakukan perhitungan kopling secara manual pada Tabel 11 dan secara otomatis menggunakan system pada Gambar 6.1, maka dapat diperoleh hasil perbandingan nilainya yang dapat dilihat pada Tabel 13 sebagai berikut.

Tabel 6.13 Perbandingan Nilai Perhitungan Secara Manual dan Otomatis

Nama Program	Secara Manual		Dengan Sistem		Hasil
	CBO	CWCBO	CBO	CWCBO	
Employee.java	7	18	7	18	Valid
Hewan.java	2	5	2	5	Valid
Introduction.java	0	0	0	0	Valid
Kamar.java	0	0	0	0	Valid
List.java	6	16	6	16	Valid

Dari hasil perbandingan nilai perhitungan secara manual dan otomatis, maka dapat dilakukan perhitungan akurasi sebagai berikut.

$$Akurasi = \frac{\sum program\ valid}{\sum program\ yang\ diuji} \times 100\%$$

$$= \frac{5}{5} \times 100\%$$

$$= 100\%$$

6.2 Analisis

Analisis dilakukan untuk mendapatkan kesimpulan dari hasil pengujian pada sistem yang telah dilakukan, yaitu pengujian *white box* yang meliputi pengujian unit dan pengujian integrasi, pengujian *black box* atau pengujian fungsional dan pengujian akurasi. Proses analisis analisis terdiri dari analisis hasil pengujian unit, analisis hasil pengujian integritas, analisis hasil pengujian validasi dan analisis hasil pengujian akurasi.

6.2.1 Analisis Hasil Pengujian Unit

Dari kasus uji yang telah dilaksanakan sesuai dengan prosedur pengujian basis path yang telah disebutkan dalam sub pokok bahasan 6.1.1 pada algoritma *checkDataIDCinMethod* di kelas *SPOON_metamodel* dan algoritma *parseFile* di kelas *Controller* bahwa seluruh kasus uji memiliki hasil yang telah sesuai dengan yang diharapkan atau bernilai valid.

6.2.2 Analisis Hasil Pengujian Integrasi

Dari pengujian integrasi yang dilakukan terhadap dua modul yaitu, modul readFile() pada *class* Controller dan modul showFileName() pada *class* Main_UI memiliki hasil yang valid. Hal ini dapat disimpulkan bahwa pengujian arsitektur yang dilakukan telah berhasil dan dapat dilanjutkan ke pengujian validasi.

6.2.3 Analisis Hasil Pengujian Validasi

Dari kasus uji yang telah dilaksanakan sesuai dengan prosedur pengujian fungsional yang telah disebutkan dalam sub pokok bahasan 6.1.3 maka didapatkan hasil seperti yang ditunjukkan dalam tabel 6.14 berikut ini.

Tabel 6.14 Hasil pengujian fungsional

No.	Kasus Uji	Hasil yang Didapatkan	Status
1.	Uji Cari Berkas	Sistem dapat menampilkan *window* bagi *user* untuk memasukkan file inputan. Sistem dapat menerima beberapa inputan yang berupa file berektensi Java.	Valid
2.	Uji Cari Berkas Alternatif 1	Sistem dapat membatalkan inputan file ketika *user* menekan tombol Cancel	Valid

Tabel 6.14 Hasil pengujian fungsional (lanjutan)

No.	Kasus Uji	Hasil yang Didapatkan	Status
3.	Uji Hitung Kopling	Sistem dapat menampilkan hasil perhitungan kopling dari beberapa inputan kode program	Valid
4.	Uji Hitung Kopling ALternatif 1	Sistem dapat menampilkan pesan "File kosong, masukkan file yang akan diuji!" ketika *user* menekan tombol Hitung Kopling tanpa memasukkan file inputan terlebih dahulu	Valid
5.	Uji Tampilkan Grafik	Sistem dapat menampilkan grafik perbandingan nilai kopling CBO dan CWCBO sesuai perhitungan kopling sebelumnya.	Valid
6.	Uji Tampilkan Grafik alternative 1	Sistem dapat menampilkan pesan "File kosong, masukkan file yang akan diuji!" ketika *user* menekan tombol Tampikan Grafik tanpa memasukkan file inputan atau melakukan perhitungan kopling terlebih dahulu.	Valid

6.2.4 Analisis Hasil Pengujian Akurasi

Dari kasus uji yang telah dilaksanakan sesuai dengan prosedur pengujian akurasi yaitu, membandingkan hasil perhitungan secara menual dan dengan sistem pada lima program diperoleh hasil akurasi sebesar 100%. Hal ini dapat disimpulkan bahwa pengujian akurasi yang dilakukan telah berhasil karena nilai akurasi lebih besar dari pada nilai batas minimalnya yang telah didefinisikan pada kebutuhan non fungsional, yaitu 90%.

6.3 Pembahasan Hasil

Berdasarkan hasil perhitungan nilai kopling yang telah dilakukan pada lima program, seperti yang tertera pada Gambar 6.4, diketahui bahwa dua program menghasilkan nilai kopling CBO dan CWCBO = 0. Kedua program tersebut adalah Introduction.java dan Kamar.java. Hal tersebut dapat dikatakan bahwa kedua program dianggap mempunyai desain yang baik karena memiliki kopling rendah, sehingga mudah untuk dipahami. Sedangkan program dengan nama Employee.java memiliki nilai kopling tertinggi, yaitu CBO = 8 dan CWCBO = 18. Maka program Employee.java dapat dianggap mempunyai desain perangkat lunak yang buruk, sehingga akan lebih sulit untuk dipahami dibandingkan program Introduction.java dan Kamar.java.

Nilai kopling CWCBO relatif lebih tinggi dibandingkan dengan nilai kopling CBO. Hal ini dikarenakan pada perhitungan kopling CWCBO ditambahkan bobot pemahaman pada setiap kategorinya. Nilai bobot yang diberikan pada setiap kategori memiliki nilai yang berbeda sesuai dengan tingkat kesulitan ketegori untuk dipahami. Nilai bobot paling kecil adalah 1 dan paling besar adalah 4. Suatu program memungkinkan mempunyai nilai kopling CBO dan CWCBO yang sama, jika program tersebut hanya mengandung jenis-jenis kopling dengan

bobot pemahaman yang paling rendah, yaitu *Control Coupling* dan *Global Data Coupling*. Kedunya memiliki bobot pemahaman dengan nilai 1. Namun suatu program akan memiliki nilai kopling CBO dan CWCBO dengan selisih yang semakin jauh jika program tersebut banyak mengandung kategori kopling yang memiliki bobot pemahaman tinggi, seperti *Lexical Content Coupling* dengan bobot pemahaman = 4.

BAB 7 PENUTUP

7.1 Kesimpulan

Dalam penelitian pembangunan kakas bantu perhitungan nilai kopling menggunakan metrik Cognitive Weighted Coupling Between Object (CWCBO) yang dilakukan oleh peneliti, terdapat beberapa hal yang dapat disimpulkan yaitu:

1. Kakas bantu perhitungan nilai kopling ini mempunyai 3 fitur yaitu, cari berkas, hitung kopling dan tampilkan grafik. Sistem ini dibangun dengan menggunakan 4 class, yaitu Controller, *Main_UI, ResultCoupling* dan *SPOON_metamodel.* Sistem ini diimplementasikan menggunakan bahasa pemrograman utama java, *library* Spoon dan *library* JFreeChart. Cara kerja sistem, pertama sistem akan menerima inputan berupa source code program berbahasa java, kemudian sistem akan menganalisis source code dengan *library* Spoon untuk mendapatkan jumlah dari masing-masing tipe kopling yang menjadi parameter dari metrik CWCBO. Setelah itu sistem akan menampilkan grafik perbandingan nilai kopling CBO dan CWCBO dengan pemanfaatan *library* JFreeChart.
2. Pengujian akurasi pada sistem ini dilakukan dengan menguji tingkat akurasi antara perhitungan yang dilakukan secara manual dan perhitungan yang dilakukan dengan sistem. Dari pengujian pada lima program yang menjadi data ujinya menghasilkan nilai kopling CBO dan CWCBO sama semua, sehingga nilai akurasinya adalah 100%. Dengan rincian dua pogram memiliki nilai kopling 0 dan nilai kopling tertinggi adalah 18.

7.2 Saran

Kakasa bantu perhitungan nilai kopling ini masih belum sempurna, sehingga masih perlu dilakukan pengembangan, perbaikan dan penyempurnaan. Beberapa hal yang perlu dikembangkan yaitu:

1. Pada sistem kakas bantu ini perlu ditambahkan *threshold* atau batasan nilai untuk menentukan tingkat kualitas suatu nilai kopling antar objek. Sehingga dengan adanya *threshold* dapat menentukan apakah perangkat lunak dengan nilai kopling tertentu sudah baik atau tidak.
2. Pada kakas bantu ini juga perlu ditambahkan agar sistem dapat menghitung nilai kopling tidak hanya dari program yang memiliki struktur *inheritance*, melainkan segala jenis program berorientasi objek.

DAFTAR PUSTAKA

Aloysius, A. & Arockiam, L., 2012. Coupling Complexity Metric: A Cognitive Approach. *I.J. Information Technology and Computer Science,* pp. 29-35.

Bertolino, A., 2007. Software Testing Research: Achievements, Challenges, Dreams. *Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo".*

Bhasin, H., Khanna, E. & S., 2014. Black Box Testing based on Requirement Analysis and. *International Journal of Computer Applications,* 18 Februari.Volume 87.

Binkley, D., 2007. Source Code Analysis: A Road Map. *Loyola College, Baltimore MD, 21210-2699, USA.*

Booch, G., 2007. *Object-oriented Analysis and Design with Application.* s.l.:Addison-Wisley.

Borysowich, C., 2007. *Design Principles: Coupling (Data and otherwise).* [Online] Available at: http://it.toolbox.com/blogs/enterprise-solutions/design-principles-coupling-data-and-otherwise-16061
[Diakses 22 3 2016].

Chidamber, S. R. & Kemerer, C. F., 1994. A Metric Suite for Object-Oriented Design. *IEEE Trans. on Software Engineering,* pp. 476-493.

Coad, P. & Yourdon, E., 1991. Object Oriented Analysis. *Prentice-Hall, New Jersey,* Volume Edisi ke-2.

Dallal, J. A. & Briand, L. C., 2010. An object-oriented high-level design-based class cohesion metric. *Department of Information Science, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait,* p. 1346–1361.

Dennis, d., 2006. *System Analys & Design.* Third Edition penyunt. America: Jhon Wiley & Sons.

Edward, B. V., 1993. *Essays on object-oriented.* Vol. 1 penyunt. Prentice-Hall: Berard Software.

Gaur, G., Suri, B. & Singhal, S., 2014. Overview of Software Engineering Metrics for Procedural Paradigm. *USICT, G.G.s.I.P. University, Dwarka, New Delhi, India.*

Gilbert, David, 2005. *JFreeChart.* [Online] Available at: http://www.jfree.org/jfreechart/
[Diakses 22 Juli 2016].

Lestari, E., 2015. PERHITUNGAN KUALITAS DESAIN OBJECT-ORIENTED MENGGUNAKAN MATRIX SIMILARITY-BASED CLASS COHESION (SCC) PADA KELAS KOHESI BERBASIS WEB. *Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya.*

Misra, S. & Akman, K. I., 2008. Weighted Class Complexity: A Measure of Complexity for Object Oriented System. *Department of Computer Engineering, Atilim University, Ankara, Turkey,* pp. 1689-1708.

Munassar, N. M. A. & Govardhan, A., 2010. A Comparison Between Five Models Of Software Engineering. *IJCSI International Journal of Computer Science Issues,* Volume 7, p. 94.

Nori, K. V. & Swaminathan, N., 2006. A Product Engineering Approach to Software Development. *Business Systems & Cybernetics Centre, Tata Consultancy Services, Hyderabad, India.*

Pawlak, R. et al., 2015. Spoon: A Library for Implementing Analyses and Transformations of Java Source Code. *Software: Practice and Experience, Wiley,* p. 33.

Pressman, R., 2010. *Software Engineering A Practitioner's Aproach.* Sevent Edition penyunt. New York: The McGraw-Hill COmpanies, Inc..

Pressman, R. S., 2010. *Software engineering: a practitioner's approach.* 5th penyunt. New York: McGraw-Hill.

Primadhana, F., 2015. PEMBANGUNAN KAKAS BANTU PEMBANGKITAN PROGRAM DEPENDENCE GRAPH BERBASIS JAVA. *Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya.*

Rogers, R. V., 1991. UNDERSTATED IMPLICATIONS OF OBJECTED-ORIENTED SIMULATION AND MODELING. *Industrial Engineering and Management Systems Department, University of Central Florida.*

Rosa, A. S. & Shalahuddin, M., 2013. Rekayasa Perangkat Lunak. Dalam: *Rekayasa Perangkat Lunak: Terstruktur dan Berorientasi Objek.* Bandung: Informatika.

Rosa, S. A. & Shalahuddin, M., 2014. Rekayasa Perangkat Lunak. Dalam: Bandung: Informatika Bandung.

Rouf, A., 2012. *Pengujian Perangkat Lunak dengan Menggunakan Metode White Box dan Black Box.* [Online] Available at: http://ejournal.himsya.ac.id/index.php/HIMSYATECH/article/view/28 [Diakses 19 April 2016].

Sefihara, F. A., 2016. PEMBANGUNAN SISTEM INFORMASI REGISTER KOHORT BIDAN DESA DAN PEMETAAN KESEHATAN IBU DAN ANAK MENGGUNAKAN POLA PERANCANGAN FACTORY METHOD. *Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya.*

Sommerville, I., 2011. *Software engineering.* 9th penyunt. London: Addison-Wesley.

Stevens, W., Myers, G. & Constantine, L., 1974. Structured design. *IBM Systems Journal, ,* pp. 115-139.

Yadav, A. & Khan, R. A., 2009. Complexity: A Reliability Factor. *IEEE International Advance Computing Conference (IACC-2009), Patiala, India*, 6-7 Maret, pp. 2375-2378.

Yadav, A. & Khan, R. A., 2009. Measuring Design Complexity – An Inherited Method Perspective. *SIGSOFT Software Engineering Notes, 24 No.4,pp:,* pp. 1-5.