PEMBANGUNAN KAKAS BANTU PERHITUNGAN NILAI *TESTABILITY* BERDASARKAN RANCANGAN PERANGKAT LUNAK

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Agita Novianingtyas
NIM: 165150201111180

POGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2020

PENGESAHAN

PEMBANGUNAN KAKAS BANTU PERHITUNGAN NILAI *TESTABILITY* BERDASARKAN RANCANGAN PERANGKAT LUNAK

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Agita Novianingtyas
NIM: 165150201111180

Skripsi ini telah diuji dan dinyatakan lulus pada
21 Juli 2020
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

Fajar Pradana, S.ST., M.Eng.
NIP: 19871121 201504 1 004

Dosen Pembimbing II

Djoko Pramono, S.T., M.Kom.
NIP: 19780108 200501 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

Achmad Basuki, S.T., M.MG., Ph.D.
NIP: 19741118 200312 1 002

PERNYATAAN ORISINILITAS

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 referensi.

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, 21 Juli 2020

Agita Novianingtyas

NIM: 165150201111180

PRAKATA

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufiq serta hidayahNya dan senantiasa memberikan kelancaran sehingga laporan skripsi yang berjudul "Pembangunan Kakas Bantu Perhitungan Nilai Testability Berdasarkan Rancangan Perangkat Lunak" ini dapat terselesaikan.

Selesai dan berhasilnya skripsi ini juga atas bantuan dan dukungan beberapa pihak kepada penulis. Oleh karena itu penulis ingin menyampaikan rasa hormat dan terimakasih kepada:

1. Bapak dan Ibu orang tua penulis serta kedua Vivin dan Amung selaku kakak penulis yang selalu memberikan dukungan, semangat, nasihat, doa, dan kasih sayang dalam mendidik penulis.
2. Bapak Fajar Pradana, S.ST., M.Eng. selaku pembimbing skripsi yang telah sabar dan ikhlas membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini
3. Bapak Djoko Pramono, S.T, M.Kom selaku pembimbing skripsi yang telah sabar dan ikhlas membimbing dan mengarahkan penulisan skripsi dan selalu memberi nasihat kepada penulis selama menempuh masa studi
4. Bapak Achmad Basuki, S.T., M.MG., Ph.D. selaku Ketua Jurusan Teknik Informatika, dan Bapak Aditya Bhawiyuga, S.Kom., M.Sc. selaku Ketua Program Studi Teknik Informatika,
5. Dhamai, Giga, Aul selaku sahabat baik penulis yang selalu mendampingi dan menyemangati penulis.
6. Semua pihak yang tidak dapat penulis sebutkan satu-persatu baik yang terlibat secara langsung ataupun tidak langsung.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga kritik dan saran yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat memberikan manfaat bagi semu pihak yang menggunakannya.

Malang, 2 Januari 2020

Penulis

Email:
agitanovianingtyas@gmail.com

ABSTRAK

Agita Novianingtyas, Pembangunan Kakas Bantu Perhitungan Nilai *Testability* Berdasarkan Rancangan Perangkat Lunak
Pembimbing: Fajar Pradana, S.ST., M.Eng. dan Djoko Pramono, S.T, M.Kom.

Mengukur *testability* pada fase awal desain perangkat lunak adalah hal yang penting untuk dilakukan. testability adalah salah satu faktor penting dari kualitas perangkat lunak yang berarti bahwa seberapa tinggi tingkat kemudahan suatu sistem dapat diuji. Mengukur *testability* pada fase awal desain dapat membantu perancang untuk memperbaiki rancangannya sebelum mulai ke tahap pengkodean. Berdasarkan penelitian dari Huda, Arya, dan Khan pada tahun 2015, yaitu *"Metric Based Testability Estimation Model for Object Oriented Design: Quality Perspective"*, mengukur testability menggunakan 2 faktor, yaitu reusability dan effectiveness dengan metrik *cohesion, coupling, inheritance* dan *encapsulation*. Penelitian ini bertujuan untuk membangun kakas bantu perhitungan nilai testability perangkat lunak yang nantinya akan diukur secara otomatis dengan masukan *xml class diagram* berstuktur *simple*. Kakas Bantu perhitungan nilai testability telah diuji semua kebutuhan yang didefinisikan dan mengahasilkan valid. dan untuk pengukuran kebutuhan efisiensi, sistem memiliki efisiensi sebesar 100% yang berarti dapat melakukan perhitungan lebih cepat dibandingkan perhitungan manual.

Kata kunci: *testability, perancangan desain, perangkat lunak, effectiveness, reusability, metrik*

ABSTRACT

Agita Novianingtyas, *Tools of Testability Measurement Based on Software Design*
Supervisors: Fajar Pradana, S.ST., M.Eng. and Djoko Pramono, S.T, M.Kom.

Measuring testability in the early phases of software design is an important thing to do. Testability is one important factor of software quality which means that the level of ease of a system can be tested. Measuring testability in the early design phase can help the designer to improve their design before starting to the coding phase. Based on research from Huda, Arya, and Khan in 2015, "Metric Based Testability Estimation Model for Object Oriented Design: Quality Perspective", measuring testability using 2 factors, reusability and effectiveness that using metric cohesion, coupling, inheritance and encapsulation. This research aims to build a tool for calculating the value of software testability which will be measured automatically by inputting a simple structured xml class diagram. Tools help calculate the value of testability has been tested all needs are defined and produce valid. And for the measurement of efficiency requirements, the system has an efficiency of 100% which means it can perform calculations faster than manual calculations.

Keywords: *testability, design phase, software, effectiveness, reusability, metric*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Rekayasa perangkat lunak adalah disiplin teknik yang berkaitan dengan semua aspek produksi perangkat lunak termasuk atribut penting seperti *maintainability, dependability, security, efficiency,* dan *acceptability* (Sommerville, 2011). Rekayasa perangkat lunak mencakup proses, metode, dan alat yang memungkinkan sistem berbasis komputer kompleks yang dapat dibangun dan dikembangkan secara tepat waktu dengan kualitas yang tinggi. Proses pengembangan perangkat lunak menggabungkan lima proses, yaitu, analisis kebutuhan, perancangan sistem, pemodelan, konstruksi, dan *deployment*. Hal tersebut penting karena memungkinkan pembangunan sistem yang berkelanjutan kemudian dan juga untuk mempertahankan kualitas perangkat lunak yang dikembangkan. Kualitas perangkat lunak diberi singkatan FURPS atau *functionality, usability, reliability, performance,* dan *supportability*. Salah satu atribut kualitas perangkat lunak adalah *testability* (Pressman, 2010).

Testability merupakan satu dari banyak faktor penting kualitas perangkat lunak. *Testability* menunjukan kemampuan perangkat lunak menguji suatu program untuk memastikan bahwa program tersebut telah melakukan fungsi yang dimaksudkan (Pressman, 2010). Memperkirakan nilai *testability* pada tahap desain adalah kriteria penting bagi perancang untuk membuat desain lebih dapat diuji. Maka dari itu perlu dilakukan identifikasi faktor-faktor testabilitas untuk menetapkan nilai *testability.* Faktor-faktor yang mempengaruhi testabilitas yaitu *effectiveness* dan *reusability*. Ketiga faktor tersebut berkorelasi dengan bantuan langkah-langkah statistik (Huda, dkk, 2015). *Testability* memiliki kriteria yang membantu pengembang dan perancang perancang untuk memperbaiki rancangannya sebelum mulai ke tahap pengkodean perangkat lunak.

Pengukuran nilai *testability* pada rancangan perangkat lunak sebelumnya telah diteliti dan dilakukan oleh Huda dan diterbitkan pada *paper* yang berjudul *"Metric Based Testability Estimation Model of Object Oriented Design : Quality Perspective"*. Perhitungan nilai *testability me*nggunakan metrik-metrik desain berorientasi objek yaitu CAM untuk *cohesion*, MFA untuk *inheritance,* DCC untuk *coupling* serta DAM untuk *encapsulation*. Penelitian ini menghasilkan validasi model yang kemudian dikembangkan untuk mengukur *testability* dan mempunyai nilai *acceptance* yang tinggi yaitu 99% (Huda, dkk, 2015).

Kualitas perangkat lunak yang baik dapat dinilai dari desain yang baik. Adapun ciri desain perangkat lunak yang baik yaitu memiliki kopling rendah dan memiliki kohesi yang tinggi (Ubaidillah, Pradana dan Priyambadha, 2017), hal ini juga dibahas karena jika nilai kopling tinggi, maka desain menjadi buruk karena sulitnya developer dalam *maintanance* dan memahami kelas (A. Yadav, R. A. Khan, 2009). Sedangkan untuk nilai *cohesion* harus tinggi karena jika nilai *cohesion* rendah maka sistem akan sulit dipertahankan karena akan menjadi sensitif terhadap perubahan. Kemudian, fungis *reusability* pada rancangan perangkat

lunak tidak didapatkan karena memiliki banyak fungsi yang berbeda dan seringkali tidak terkait (Martin, Robert C, dkk, 2003).

Perhitungan nilai *testability* yang dilakukan secara manual memerlukan waktu dan usaha yang tidak sedikit, terlebih jika perangkat lunak yang diukur cukup kompleks. Perhitungan nilai *testability* secara manual juga memperbesar peluang terjadinya kesalahan pada perhitungan metrik. Berdasarkan hal tersebut, diperlukan sistem sebagai alat bantu bagi perhitungan manual. Dengan adanya sistem, waktu yang digunakan untuk mengukur kualitas perangkat lunak menjadi efisien karena otomatis dan dapat digunakan secara sering tanpa membuang waktu yang berlebihan (P. Tomas, M.J. Escalona, M. Mejias, 2013). Menurut Tomas, dkk, manfaat lain yang didapat dengan adanya sistem pengukuran kualitas perangkat lunak yaitu memperkecil peluang terjadinya kesalahan perhitungan metrik dan akurasi yang dicapai lebih besar pada nilainya.

Menurut uraian diatas, dalam penulisan ini diusulkan pengembangan "Pembangunan Kakas Bantu Perhitungan Nilai *Testability* Berdasarkan Rancangan Perangkat Lunak". Sistem kakas bantu ini dibuat agar pengembang dapat memperhitungkan nilai *testability* pada tahap perancangan perangkat lunak yaitu dari *class diagram* secara otomatis. Perhitungan nilai *testability* pada penulisan ini menggunakan teknik regresi berganda menggunakan metrik desain berorientasi objek yaitu *cohesion, coupling, inheritance* dan *encapsulation* diperoleh dari paper (Huda, dkk, 2015) dengan judul *"Metric Based Testability Estimation Model of Object Oriented Design : Quality Perspective"* sebagai rujukan saat mencari kebutuhan sistem. Dengan adanya hasil perhitungan dari kakas bantu ini, tim pengembang dapat terbantu karena selain menghemat waktu dan juga perbaikan rancangan dapat dilakukan sebelum masuk ke tahap pengkodean perangkat lunak.

1.2 Rumusan Masalah

1. Bagaimana hasil perancangan dan implementasi aplikasi Pembangunan Kakas Bantu Perhitungan Nilai *Testability*?
2. Bagaimana efisiensi waktu aplikasi kakas bantu perhitungan nilai *testability?*

1.3 Tujuan

1. Membangun aplikasi Pembangunan Kakas Bantu Perhitungan Nilai *Testability* Berdasarkan Rancangan Perangkat Lunak pada fase awal pengembangan.
2. Mengoptimalkan efisiensi dalam melakukan pengukuran kualitas *testability* sebuah perangkat lunak.

1.4 Manfaat

Manfaat yang diharapkan dari penelitian ini adalah aplikasi kakas bantu perhitungan nilai *testability* dapat membantu pada saat pengujian dengan membuat desain perangkat lunak lebih dapat teruji untuk beberapa kuantitas dengan hasil yang dapat meningkatkan kualitas perangkat lunak tanpa usaha lebih.

1.5 Batasan Masalah

1. Pengembangan sistem kakas bantu menggunakan bahasa pemrograman Java.
2. Penelitian ini hanya mengukur nilai *testability* dan tidak menentukan batasan untuk nilai yang buruk maupun baik.
3. Masukan dari desain metrik rancangan perangkat lunak berorientasi objek yang diukur berupa class diagram.
4. Masukan yang digunakan adalah format XML yang didapatkan dari Visual Paradigm.
5. Perhitungan hanya digunakan pada *class diagram* berorientasi objek.

1.6 Sistematika Pembahasan

Bab 1 PENDAHULUAN

Pada bab pendahuluan ini akan menerangkan tentang latar belakang dari pembangunan aplikasi, rumusan masalah, tujuan, manfaat, batasan masalah, serta sistematika pembahasan.

Bab 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan ini akan menerangkan teori yang akan digunakan serta riset penelitian yang telah dilakukan untuk mendasari penulisan dan penelitian pembangunan aplikasi kakas bantu perhitungan nilai *testability*.

Bab 3 METODOLOGI PENELITIAN

Pada bab metodologi penelitian ini menerangkan tentang metodologi penelitian yang digunakan, yaitu: studi literatur, analisis kebutuhan, perancangan dan implementasi, pengujian, serta kesimpulan dan saran. Sedangkan untuk pengembangan sistem menggunakan *SDLC Model Waterfall.*

Bab 4 ANALISIS KEBUTUHAN

Pada bab analisis kebutuhan ini berisi identifikasi aktor, spesifikasi kebutuhan yaitu kebutuhan fungsional dan non-fungsional, serta lingkungan pengembangan sistem.

Bab 5 PERANCANGAN DAN IMPLEMENTASI SISTEM

Pada bab perancangan ini menerangkan rancangan dari sistem yang akan dibangun, yaitu perancangan arsitektur, perancangan algoritme, dan perancangan antarmuka pengguna. Pada bagian implementasi sistem terdiri dari spesifikasi sistem, lingkungan pengembangan, sampel kode program, dan hasil implementasi antarmuka.

Bab 6 PENGUJIAN SISTEM DAN PEMBAHASAN HASIL

Pada bab pengujian ini menerangkan proses pengujian dari sistem kakas bantu perhitungan nilai *testability* yang telah dikembangkan sehingga sistem dipastikan layak digunakan calon pengguna dan juga memaparkan pembahasan hasil dari pengujian berkas uji.

Bab 7 PENUTUP

Pada bab penutup ini menyampaikan kesimpulan penelitian ini dan juga saran yang dapat digunakan bagi penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Kajian pustaka ini mengacu pada beberapa penelitian yang sudah diselesaikan sebelumnya. Penelitian pertama dilakukan oleh Huda, Sharma Arya dan Hasan Khan yang berjudul "*Evaluating Effectiveness Factor Of Object Oriented Design: A Testability Perspective*" dengan menghitung efektivitas dari perangkat lunak menggunakan metric *encapsulation, cohesion, coupling* dan *inheritance*. Penelitian kedua dengan penulis yang sama dengan judul "*Quantifying Reusability of Object Oriented Design: A Testability Perspective*" (Huda, dkk, 2015) dengan menghitung reusabilitas perangkat lunak menggunakan *encapsulation, coupling* dan *inheritance* yang mana ketiga metrik tersebut sangat berpengaruh untuk perhitungan *testability* dari perangkat lunak.

Kemudian penelitian ketiga berjudul "Metric Based *Testability* Estimation Model for Object Oriented Design: Quality Perspective" yang ditulis oleh Huda, Sharma Arya dan Hasan Khan dan menjadi akhir dari penelitian karena penelitian ini digunakan untuk menghitung testabilitas perangkat lunak berdasarkan kualitas. Dari penelitian ini juga didapatkan perhitungan bahwa nilai *effectiveness* dan *reusability* mempengaruhi nilai *testability*. Pengukuran nilai *testability* dilakukan dengan menggunakan *class diagram*. Pengukuran *testability* menggunakan model yang berdasarkan hubungan antara *reusability* dan *effectiveness.* Perhitungan nilai *testability* menggunakan teknik regresi berganda dengan metrik-metrik properti desain berorientasi objek yaitu CAM untuk *cohesion*, MFA untuk *inheritance,* DCC untuk *coupling* serta DAM untuk *encapsulation*. Hasil yang didapatkan pada penelitian ini yaitu validasi model yang dikembangkan untuk mengukur *testability* memiliki nilai *acceptance* yang tinggi yaitu 95% (Huda, dkk, 2015).

Selain penelitian tentang perhitungan *testability,* dilakukan pengamatan juga pada penelitian sebelumnya yang berjudul "Kakas bantu Perhitungan Nilai Kopling Menggunakan Metrik Cognitive Weighted *Coupling* Between Object". Penelitian tersebut melakukan pengembangan sistem untuk membantu perhitungan kopling. Pengukuran kopling dilakukan dengan menggunakan *source code* program berbahasa java. Penelitian tersebut juga melakukan perbandingan nilai kopling menggunakan CBO dan CWCBO serta perhitungan manual dan juga otomatis menggunakan sistem. Pengujian akurasi dengan membandingkan hasil perhitungan secara menual dan dengan sistem pada lima program diperoleh hasil akurasi sebesar 100% yaitu lebih besar dari pada nilai batas minimalnya yaitu 90% (Ubaidillah, Pradana dan Priyambadha, 2017).

Penelitian yang akan dilakukan saat ini adalah membangun kakas bantu perhitungan *testability*. Meninjau dari penelitian sebelumnya, menggunakan bantuan sistem diharapkan hasil akurasi perhitungan tinggi. Pengukuran dilakukan pada tahap awal yaitu perancangan agar dapat sedini mungkin perubahan dilakukan pada rancangan sebelum diimplementasikan menjadi kode program.

2.2 Rekayasa Perangkat Lunak

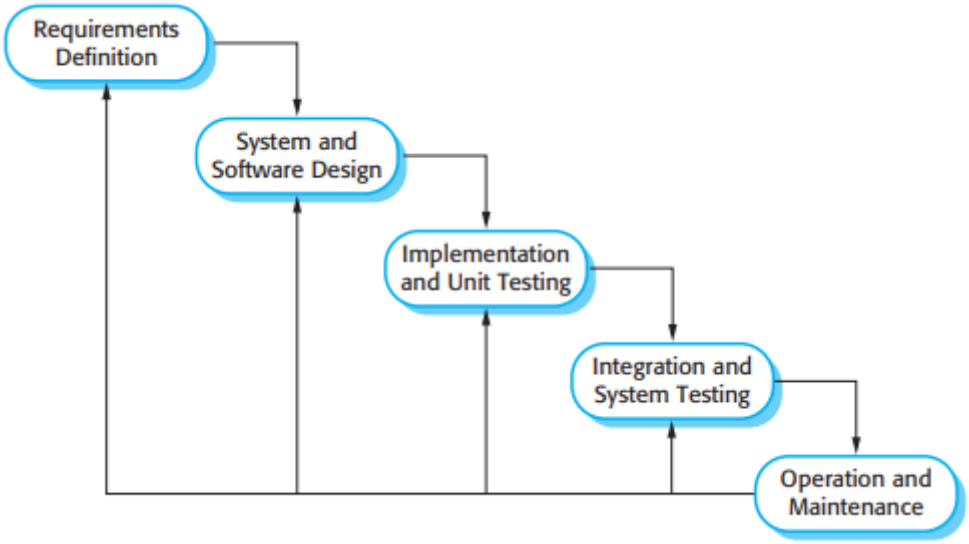
Rekayasa perangkat lunak adalah cabang ilmu dengan bahasan keseluruhan aspek pengembangan perangkat lunak, dimulai dari tahap analisis kebutuhan hingga tahap pemeliharaan perangkat lunak (Sommerville, 2011). Adapun istilah dari perangkat lunak, yaitu:

1. *Engineering Discipline*
   Engineers membuat suatu hal bekerja dengan menerapkan teori, metode dan alat bantu yang sesuai. Meskipun begitu *engineers* tetap selektif dalam menggunakannya dan selau mencoba untuk menyelesaikan masalah meskipun tanpa teori dan metode yang memadai. Mereka juga harus tetap bekerja dibawah organisasi dan batas keuangan agar dapat menemukan solusi untuk batasan itu.
2. *All Aspect of Software Production*
   Pengembangan aspek perangkat lunak tidak hanya tentang proses teknik, tapi juga mencakup aktifitas dari pengembangan perangkat lunak seperti manajemen proyek perangkat lunak dan pengembangan teori, alat, juga metode yang mendukung proses produksi

2.3 Pengembangan Perangkat Lunak

2.3.1 Model *Software Development Life Cycle*

Untuk pengembangan perangkat lunak pada penelitian ini, penulis menggunakan *Software development life cycle* (SDLC) model *waterfall.* Disebut demikian karena SDLC mode *waterfall* karena bentuknya yang mirip dengan air terjun dan direpresentasikan urut dari satu tahap ke tahap yang lain seperti pada dalam Gambar 2.1. Alur pada model ini pada prinsipnya harus melakukan penjadwalan serta perencanaan seluruh kegiatan pembanguan sebelum pengeerjaannya. Karena tahapnya harus berurutan, maka sangat penting dilakukan dokumentasi pada setiap tahap agar tahap yang lain tidak boleh dilakukan sebelum tahap sebelumnya selesai. Penggunaan SDLC Model *waterfall* memungkinan hanya bisa digunakan ketika kebutuhan telah diidentifikasi dengan diawal dan tanpa adanya perubahan yang berarti selama proses pengembangan sistem dilakukan (Sommerville, 2011).



Gambar 2.1 Waterfall Model

Sumber: (Sommerville, 2011)

Tahapan dari SDLC *waterfall* model ini menjelaskan bagaiman kegiatan pengembangan terjadi. Adapun 5 tahapan pada SDLC *waterfall* model, yaitu: analisis kebutuhan, perancangan sistem, implementasi dan pegujian unit, pengujian integrasi dan terakhi adalah maintenance (Sommerville, 2011). Analisis kebutuhan mendefinisikan layanan sistem, batasan dan tujuan yang dituju berdasarkan konsultasi dengan pihak pengguna. Ketika pendefinisian telah dilakukan dengan rinci maka dapat dipersiapkan sebagai spesifikasi sistem.

Setelah analisis kebutuhan terbentuk, maka langkah kedua adalah perancangan sistem. Perancangan sistem merupakan tahap dimana pengembang melakukan perancangan untuk sistem yang kemudian desain sistem mengalokasikan kebutuhan perangkat keras maupun perangkat lunak dengan mendirikan arsitektur sistem secara keseluruhan. Perancangan perangkat lunak melibatkan identifikasi dan penggambaran abstraksi sistem dan hubungan mereka.

Kemudian adapun tahap ketiga, atau tahap implementasi dan pengujian unit. Ketika sampai pada tahap ini, perancangan perangkat lunak dibentuk menyerupai sebuah rangkaian program atau unit program yang kemudian diintegrasikan menjadi sebuah sistem berdasarkan dengan rancangan awal. pengujian unit melibatkan proses verifikasi bahwa setiap unit telah memenuhi spesifikasinya.

Tahap selanjutnya adalah tahap pengujian integrasi yaitu pengujian sistem dengan mengintegrasikan unit program dan kemudian dilakukan pengujian sebagai satu sistem yang lengkap dan penuh untuk menentukan bahwa kebutuhan perangkat lunak telah terpenuhi dan berfungsi dengan baik sesuai dengan kebutuhan. Setelah dilakukan pengujian, sistem akan dikirim disampaikan ke pelanggan.

Pada tahap terakhir yaitu tahap pemeliharaan atau tahap *maintenance* merupakan tahap paling panjang dalam siklus hidup perangkat lunak setelah dilakukan pemasangan sistem dan digunakan secara praktis lingkungan pengguna. *Maintenance* melibatkan pemindaan kesalahan yang tidak ditemukan pada fase awal dari siklus hidup, memperbarui implementasi di unit sistem dan mengoptimalkan layanan bagi sistem sebagai halnya kebutuhan baru ditemukan.

2.3.2 Pendekatan Berorientasi Objek

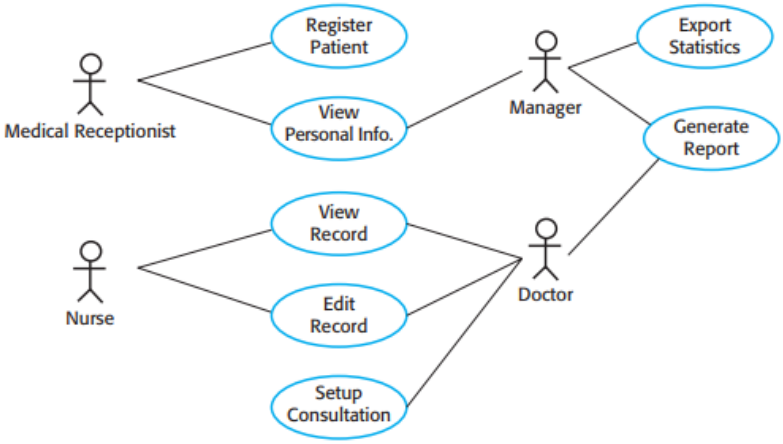
Pendekatan berorientasi objek untuk pengembangan perangkat lunak pertama kali diusulkan pada tahun 1960 akhir dan mengambil hampir 20 tahun untuk digunakan secara luas (Pressman, 2010). Pendekatan ini dibuat berdasarkan hubungan antara operasi dan objek yang terdapat pada *state* yang mengontrol *state* tersebut (Sommerville, 2011). Pendekatan berorientasi objek mengakibatkan penggunaan ulang (komponen program), dan penggunaan ulang mempengaruhi pengembangan *software* menjadi lebih cepat dan program berkualitas tinggi (Pressman, 2010). Perangkat lunak berorientasi objek lebih mudah pemeliharaannya karena struktur dipisahkan dengan sendirinya. Hal ini membuat efek samping yang lebih sedikit saat modifikasi perangkat lunak harus dilakukan. Selain itu, penyesuaian sistem berorientasi objek lebih mudah bagi sistem berskala besar seperti sistem yang dapat digunakan untuk membangun subsistem yang kemudian dapat digunakan ulang.

2.3.3 Pemodelan Berorientasi Objek

Unified modeling language (UML) adalah kumpulan tipe diagram berbeda dan dapat digunakan untuk memodelkan perangkat lunak. UML muncul pada tahun 1990-an pada pemodelan berorientasi objek dengan notasi berorientasi objek sama d bahasa grafis yang digunakan dalam pengembangan berorientasi objek yang sama kemudian diintegrasikan dan menjadi UML. *Unified modeling language (UML)* dapat dideskripsikan sebagai bahasa grafis yang dipakai dalam pembangunan perangkat lunak berorientasi objek dan melingkupi beberapa macam model sistem yang memberikan pandangan berbeda terhadap suatu sistem. UML sudah merupakan standar *de facto* untuk pemodelan berorientasi objek (Sommerville, 2011). Banyak tipe diagram UML yang mendukung pembangunan banyak jenis model sistem. Pada penelitian kali ini penulis memakai macam diagram UML antara lain:

2.3.3.1 Use Case Diagram

Use case diagram memberikan gambaran yang cukup sederhana dari suatu hubungan antara sistem dengan lingkungannya (Sommerville, 2011). Pemodelan *use case* biasa dimanfaatkan untuk penunjang elisitasi kebutuhan. *Use case* dapat dimanfaatkan untuk sketsa sederhana guna menggambarkan tentang harapan pengguna pada suatu sistem. Masing-masing *use case* mengilustrasikan tugas yang memperlihatkan hubungan antara eksternal dengan sistem. *Use case diagram* dapat dilihat pada contoh dalam Gambar 2.2

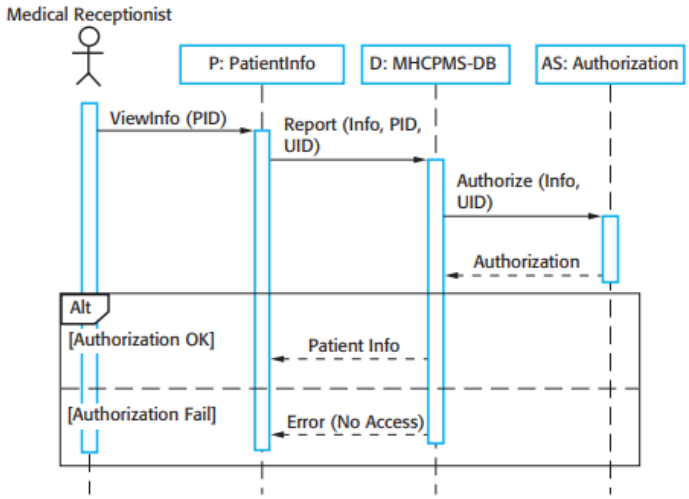


Gambar 2.2 *Use Case* Diagram

Sumber: (Sommerville, 2011)

2.3.3.2 Sequence Diagram

Sequence diagram dibutuhkan untuk memodelkan korelasi antara aktor dan objek pada suatu sistem dengan hubungan antar objek itu sendiri (Sommerville, 2011). Banyak sintaks yang dimiliki UML untuk *sequence diagram*, maka mengizinkan beragam jenis interaksi berbeda yang kemudian dapat dimodelkan. *Sequence diagram* mengindikasikan urutan hubungan yang terjadi selama *use case.* Objek dan juga aktor yang berperan dicantumkan pada bagian atas diagram dengan garis putus-putus vertikal dari aktor dan objek ke bawah. Panah beranotasi menunjukkan interaksi antar objek. Bagian persegi panjang yang terdapat pada bagian garis putus-putus vertikal menyatakan *life-line* dari objek. Anotasi yang terdapat pada panah menyatakan panggilan ke objek, parameter dari objek juga nilai kembalinya. Pada kotak yang berjudul *alt* menunjukkan kondisi alternatif.*Sequence diagram* dapat dilihat pada contoh seperti dalam Gambar 2.3.



Gambar 2.3 Sequence Diagram

Sumber: (Sommerville, 2011)

2.3.3.3 Class Diagram

Class diagram digunakan saat membangun sistem dengan pendekatan berorientasi objek untuk mengindikasikan asosiasi dan kelas diantara kelas-kelas pada suatu sistem (Sommerville, 2011). Dengan gambaran umum, kelas objek merupakan ilustrasi dari satu jenis objek sistem dan asosiasi menyatakan jika terdapat hubungan antar kelas. Interaksi antara kelas yang satu dengan kelas yang lain menunjukkan adanya hubungan antara kelas-kelas. Maka dari itu, setiap kelas dalam *class diagram* harus terkait. Ketika menujukkan asosiasi antar kelas, maka harus didefinisikan secara rinci yaitu dengan menambahkan informasi tentang atribut dan operasi. Atribut berfungsi sebagai karakteristik suatu objek sedangkan operasi adalah tempat untuk mengakses suatu objek (Sommerville, 2011). Saat proses awal pengembangan objek diilustrasikan dengan sesuatu di dunia nyata, lalu untuk implementesi ketika dikembangkan maka perlu menetapkan tambahan objek implementasi yang kemudian dapat digunak untuk menentukan kebutuhan fungsional sistem.

Adapun aturan yang terdapat pada *class diagram:*

1. Nama kelas objek terletak pada bagian atas.
2. Atribut kelas terletak pada bagian tengah. Atribut harus disertai dengan nama atribut dan, secara opsional, tipenya.
3. Operasi (pada bahasa pemrograman Java dan bahasa pemrograman OO lainnya disebut *method*) yang berhubungan dengan kelas objek terdapat di bagian bawah persegi panjang.

2.4 Kualitas Perangkat Lunak

Kualitas perangkat lunak didasari oleh kesesuaian spesifikasi produk dengan produk aslinya. Perkiraan yang mendasari adalah jika produk dapat seutuhnya ditetapkan dan kebijakan dapat diputuskan untuk dapat memeriksa produk yang diproduksi dengan spesifikasinya (Sommerville, 2011). Pengukuran kualitas perangkat lunak adalah proses subjektif dimana tim manajemen kualitas harus memakai penilaian dan pandangan mereka untuk menentukan apakah tingkat kualitas yang dapat diterima telah terpenuhi. Tim manajemen kualitas harus meninjau kembali apakah sistem perangkat lunak telah selaras dengan tujuan yang dimaksudkan.

Kualitas perangkat lunak tidak hanya persoalan tentang apakah fungsi perangkat lunak telah diterapkan dengan benar, tetapi juga terkait dengan atribut sistem non-fungsional (Sommerville, 2011). Adapun 15 macam atribut kualitas perangkat lunak yang penting, seperti ditunjukkan pada Gambar 2.4.

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Gambar 2.1 Atribut kualitas perangkat lunak

Sumber: (Sommerville, 2011)

2.5 *Testability* Perangkat Lunak

Testability merupakan faktor penting dari kualitas perangkat lunak. *Testability* menunjukan kemampuan perangkat lunak menguji suatu program untuk memastikan bahwa program tersebut telah melakukan fungsi yang dimaksudkan (Pressman, 2010). Memperkirakan nilai *testability* pada tahap desain adalah kriteria penting bagi perancang untuk membuat desain lebih dapat diuji. Maka dari itu perlu dilakukan identifikasi faktor-faktor testabilitas untuk menetapkan nilai *testability.* Faktor-faktor yang mempengaruhi testabilitas yaitu *effectiveness* dan *reusability*. Ketiga faktor tersebut berkorelasi dengan bantuan langkah-langkah statistik (Huda, dkk, 2015). *Testability* memiliki standar yang membantu perancang untuk mengembangkan rancangan pada fase awal proses pembangunan perangkat lunak.

Perancangan dan pengembangan berorientasi objek merupakan model pilihan bagi banyak pengembang perangkat lunak dan dengan berjalannya waktu pendekatan berorientasi objek menghapus pendekatan klasik (Pressman, 2010). Maanfaat lain dari penggunaan desain berorientasi objek antara lain dukungan untuk penggunaan kembali perangkat lunak.

2.6 Properti Desain Berorientasi Objek

Desain dan pengembangan berorientasi objek dalam lingkungan pengembang perangkat lunak menjadi model pengembangan yang sangat disukai untuk rancangan sistem berasio besar. Teknologi ini merokemendasikan dukungan untuk menghadirkan hasil produk perangkat lunak dengan kualitas yang lebih tinggi dan biaya pemeliharaan menjadi lebih rendah (Huda, dkk, 2015). Dimana keuntungan utama berorientasi objek salah satunya adalah dukungannya untuk testabilitas (pengujian) perangkat lunak, yang dapat dicapai baik melalui penggunaan kembali kelas yang sederhana pada *library* atau melalui warisan di antara hubungan.

Adapun kualitas penting yang diperhitungkan sebagai acuan kualitas internal desain berorientasi objek yang mendukung dalam konteks pengukuran.

Ide-ide ini secara relevan termasuk properti desain yaitu *cohesion, inheritance, coupling* dan *encapsulation. Encapsulation* adalah metode untuk memenuhi abstraksi data dan penyembunyian informasi dengan menyembunyikan spesifikasi internal suatu objek. *Inheritance* adalah distribusi operasi dan atribut antar kelas. Kopling menunjukkan hubungan atau interdependensi antar modul. *Cohesion* merujuk pada keandalan internal di dalam elemen desain.

2.7 Metrik Desain Berorientasi Objek

Object-oriented metrics (OOM) berfungsi vital dalam industri perangkat lunak karena kebanyanyakan pengembangkan mengembangkan proyek perangkat lunak dengan konsep berorientasi objek (Padhy, Singh dan Satapathy, 2018). Menghitung nilai kualitasd dan kuantitas dari perangkat lunak adalah tugas yang menantang bagi pengembang. Definisi dari metrik sendiri menurut KBBI adalah ilmu yang berhubungan dengan ukuran (Kemdikbud, 2016). Banyak metrik perangkat lunak telah dikembangkan dan digunakan untuk prediksi kesalahan, yang dapat memperkirakan ukuran, kompleksitas, kohesi, dan kopling. Beberapa metrik yang dapat dihitung dari informasi desain adalah *Quality Model for Object-Oriented Design* (QMOOD) seperti yang ada pada Tabel 2.2 (J. Bansiya, C.G. Davis, 2002).

Tabel 2.1Metrik *Quality Model for Object-Oriented Design*

Metric	**Nama**	**Deskripsi**
DSC	*Design Size in Classes (Design Size Metric)*	Metrik ini adalah hitungan dari jumlah total seluruh kelas dalam desain.
NOH	*Number of Hierarchies (Hierarchies Metric)*	Metrik ini adalah hitungan jumlah kelas hierarki dalam desain.
ANA	*Average Number of Ancestors (Abstraction Metric)*	Metrik ini menunjukkan jumlah rata-rata kelas dari mana kelas mewarisi informasi.
DAM	*Data Access Metric (Encapsulation Metric)*	Metrik ini adalah rasio jumlah atribut *private* dan *protected* dengan jumlah total atribut yang dideklarasikan di kelas. Nilai tinggi untuk DAM diinginkan. (Kisaran 0 hingga 1)
DCC	*Direct Class Coupling (Coupling Metric)*	Metrik ini menghitung jumlah kelas yang berbeda yang berelasi langsung dengan kelas. Metrik ini mencakup

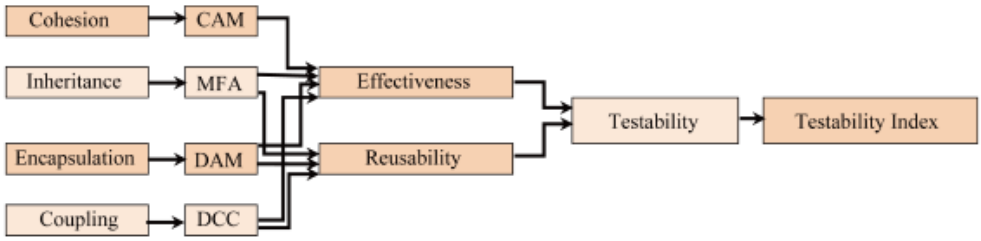
		kelas-kelas yang secara langsung berelasi dengan deklarasi atribut dan pengiriman pesan (parameter) dalam *method*.
MOA	*Measure of Aggregation (Compotition Metric)*	Metrik ini mengukur tingkat hubungan bagian-utuh yang diwujudkan dengan menggunakan atribut.
MFA	*Measure of Functional Abstraction (Inheritance Metric)*	Metrik ini adalah rasio jumlah *method* yang diwarisi oleh kelas dengan jumlah *method* yang dapat diakses oleh *method* anggota kelas. (Kisaran 0 hingga 1)
CIS	*Class Interface Size (Messaging Metric)*	Metrik ini mengukur jumlah metode publik di kelas.
NOM	*Number of Methods (Complexity Metric)*	Metrik ini mengukur jumlah total metode di kelas.

2.8 Metrik *Testability*

Ada beberapa macam metric berorientasi objek yang meneliti tentang pengukuran tingkat *testability* perangkat lunak. Pada versi *CK Metrics* terdiri dari 6 metrik yang lauyak digunakan untuk memperhitungkan faktor testabilitas dari perangkat lunak, yaitu *Number of Children* (NOC), *Weighted Method per class* (WMC), *Depth of Inheritance Tree* (DIT), *Coupling between Objects* (CBO), *Lack of Cohesion Metrics* (LCOM) dan *Response for a class* (RFC) (Suri, Harsha, 2015). Metrik CK tidak hanya digunakan oleh peneliti untuk memperhitungkan ukuran dan kualitas, namun juga digunakan sebagai acuan untuk menghitung algoritma pembelajaran mesin.

Matrik lainnya yaitu menghitung *testability* menggunakan *multivariate regression* yaitu model regresi berganda yang menyatakan korelasi antara *testability,* properti *Object-Oriented Design* (OOD) dan metrik (Huda, dkk, 2015). *Class diagram* pada tahap perancangan digunakan untuk mengidentifikasi poin-poin dari metrik desain. Adapun metrik yang bertindak sebagai variabel independen pada *effectiveness* yaitu *encapsulation*, *inheritance* , *couplin* dan *coheson*. Kemudian, metrik yang bertindak sebagai variabel independen pada *reusability* adalah *encapsulation*, *inheritance* dan juga *coupling*. Sementara *testability* bertindak sebagai variabel dependen yang ditunjukkan pada Gambar

2.5. Memperkirakan nilai *testability* sangatlah penting dan dapat membantu mendapatkan indeks *testability* untuk kualitas yang tinggi dari desain perangkat lunak.



Gambar 2.4 Korelasi antara faktor *testability* dan *design metrics*

Sumber: (Huda, dkk, 2015)

Pada penelitian ini digunakan *multivariate regression* atau model regresi berganda untuk menghitung *testability* melalui *class diagram* pada tahap perancangan. Dengan menggunakan model regresi, penelitian sudah mengembangkan model efektivitas dan model reusabilitas yang terdapat pada persamaan dan dapat membentuk dasar yang kuat untuk perkiraan pengembangan nilai *testability* yang dapat dilihat pada persamaan 1:

$$Testability = \alpha 0 \pm \beta 1 \times Effectiveness \pm \beta 2 \times Reusability \quad (1)$$

Dengan menggunakan SPSS (*Statistical Package for the Social Sciences)*, semua data yang diperoleh dari penelitian sebelumnya dan berdasarkan matrik-matrik desain, koefisien intersepsi regresi, dari tiap metrik dihitung, maka, model *testability* regresi berganda telah dikembangkan dengan persamaan 2:

$$Testability = 59.524 - 4.671 \times Effectiveness + 0.806 \times Reusability \quad (2)$$

2.8.1 *Cohesion Among Methods of Class* (CAM)

Metrik ini digunakan untuk mengukur koherensi antara *method* kelas yang berdasarkan daftar parameter *method*. Metrik CAM mengukur tingkat perpotongan parameter dijumlah dengan nilai maksimum dari semua tipe parameter yang ada di kelas (Fujita, H. & Pisanelli, D. M., 2007). Dapat ditarik kesimpulan bahwa *metodh* pada suatu kelas, memiliki peluang akses ke tipe parameter yang serupa dengan jenis parameter dan proses informasi yang terkait erat. Berikut adalah CAM Metric yang dirumuskan pada persamaan 3 seperti berikut:

$$CAM = \frac{\sum_{m \times dt}^{p}}{\sum cm} \quad (3)$$

Keterangan:
p = total parameter berbeda yang terdapat padasetiap method
m = Jumlah *metodh* seluruh kelas
dt = total tipe data yang berbeda pada kelas
cm = kelas degan minimal satu *methdo tersedia*

2.8.2 *Measure of Functional Abstration* (MFA)

Metrik MFA merupakan perbandingan dari total *method* yang diwarisi oleh kelas dengan jumlah *method* yang dapat diakses oleh *method* anggota kelas. Diinpresentasikan dengan rata-rata dari keseluruhan kelas pada *class diagram* (dengan minimal satu method tersedia) dari perbandingan jumlah *method* yang diwarisi oleh kelas tersebut dengan jumlah total *method* yang tersedia pada kelas seperti dalam persamaan berikut (Fujita & Pisanelli, 2007):

$$MFA = \frac{\sum \frac{mw}{mt}}{\sum cm} \quad (5)$$

Keterangan:

Mw = method yang diwarisi

Mt = method yang tersedia

Cm = kelas dengan minimal satu method tersedia

2.8.3 *Direct Access Metric* (DAM)

Metrik ini adalah perbandingan total atribut *protected* dan *private* dengan total atribut yang dideklarasikan di kelas. Diinpresentasikan dengan rata-rata dari perbandingan total atribut *private* dan *protcted* dan total atribut yang telah dideklarasikan di kelas kemudian dibagi dengan seluruh kelas dengan minimal satu atribut yang tersedia. Jika dirumuskan maka seperti dalam Persamaan 5 (Fujita, H. & Pisanelli, D. M., 2007):

$$DAM = \frac{\sum att}{\sum cdam} \quad (6)$$

Keterangan:

dam = perbandingan total atribut *private* dan *protected* dengan total atribut yang dideklarasikan di kelas)
cdam = kelas dengan minimal satu atribut

2.8.4 *Direct Class Coupling* (DCC)

Metrik ini mengukur total kelas yang berbeda yang berhubungan secara langsung dengan kelas. DCC merangkum kelas-kelas yang secara langsung berhubungan dengan deklarasi atribut dan pengiriman pesan (parameter) dalam *method*. Jika diterapkan pada rancangan *class diagram* maka dihitung rata-rata dari jumlah DCC per kelas dibagi dengan jumlah seluruh kelas yang ada. Jika dirumuskan maka seperti dalam Persamaan 6 berikut (Fujita, H. & Pisanelli, D. M., 2007)):

$$DCC = \frac{\sum dcc}{\sum c} \quad (6)$$

Keterangan:
dcc = total kelas yang berbeda yang berelasi langsung dengan kelas (pada masing-masing kelas)
c = jumlah total keseluruhan kelas yang ada pada *class diagram*

2.9 Pengujian Perangkat Lunak

Pengujian merupakan tahap yang bertujuan untuk menyatakan bahwa program melaksanakan apa yang ditujukan dan untuk mendeteksi ketidaksempurnaan yang yang terdapat pada program sebelum digunakan (Sommerville, 2011). Pengujian dilakukan dengan menjalankan program menggunakan data buatan untuk memeriksa apakah terdapat kesalahan, keganjilan, ataupun informasi tentang atribut non-fungsional program. Adapun dua tujuan berbeda pada proses pengujian, yaitu:

1. Tujuan pertama digunakan untuk menunjukkan kepada pengembang dan pengguna bahwa perangkat lunak telah memenuhi persyaratan. Tujuan ini lebih mengarah ke pengujian validasi, di mana diharapkan sistem berfungsi dengan baik ketika diberikan serangkaian kasus uji yang menggambarkan penggunaan yang diharapkan perangkat luanak.
2. Tujuan kedua digunakan untuk menemukan keadaan dimana perilaku perangkat lunak tidak sesuai dengan spesifikasinya. Tujuan ini lebih mengarah ke pengujian ketidaksempurnaan. Kasus uji yang digunakan sengaja dikaburkan dan tidak perlu merefleksikan sebagaimana mestinya perangkat lunak digunakan.

Diantara kedua tujuan pengujian diatas, tidak ada batasan yang pasti. Ketika pengujian validasi dilakukan, maka penguji akan mendeteksi ketidaksempurnaan dalam sistem; ketika pengujian cacat dilakukan, maka beberapa tes akan menunjukkan bahwa program memenuhi kebutuhan.

Jenis-jenis pengujian perangkat lunak yang dilakukan pada penelitian ini adalah sebagai berikut:

2.9.1 Pengujian Unit

Pengujian unit adalah proses pengujian komponen program mencakup metode dan kelas objek (Sommerville, 2011). Pengujian ini berpusat pada tujuan untuk verifikasi unit terkecil dari perancangan perangkat lunak yang berupa modul atau komponen perangkat lunak (Pressman, 2010). Ketika menguji kelas objek, maka kasus uji harus didesain untuk memenuhi keseluruhan cakupan fitur objek. Yang berarti harus dilakukan:

1. Menguji semua operasi yang terkait dengan objek
2. Mengatur dan memeriksa nilai semua atribut yang terkait dengan objek
3. Menempatkan objek ke dalam semua kemungkinan yang berarti harus mensimulasikan tindakan yang menyebabkan perubahan keadaan.

Pengujian unit mengarah pada *white-box,* dan dilakukan secara paralel untuk beberapa komponen.

Setelah pengembangan *code* dilakukan, ditinjau dan diverifikasi untuk korespondensi dengan desain tangka komponen, desain kasus uji unit dimulai.

Tiap-tiap kasus uji harus dipadukan dengan satu set hasil yang diharapkan. Karena komponen bukanlah program yang berdiri sendiri, maka *driver* dan / atau

stub harus diterapkan untuk setiap pengujian unit. *Driver* tidak melebihi program utama yang menerima data kasus uji, mengirimkan data tersebut ke komponen (untuk diuji), dan mencetak hasil yang signifikan. Fungsi *stub* untuk menggantikan modul yang lebih rendah (disebut oleh) komponen yang akan diuji.

2.9.1.1 Pengujian *White Box*

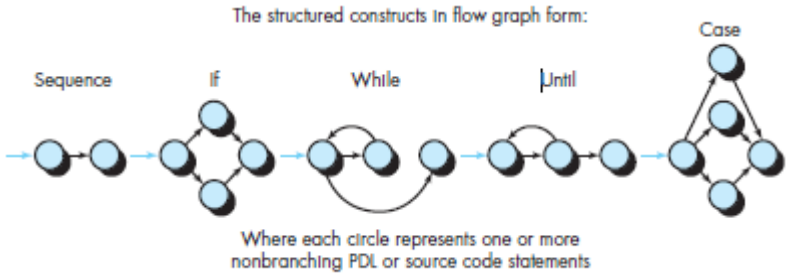
Pengujian *white-box* merupakan pendekatan untuk pengujian program yang berdasarkan pada pengetahuan tentang struktur program dan komponennya (Sommerville, 2011). Akses ke kode sumber sangat penting untuk pengujian *white-box*.

Aspek-aspek pengujian *white-box* (Kurniawan, 2015) :

a. Memastikan semua jalur algoritme telah diuji setidaknya satu kali
b. Menguji keseluruhan keputusan lojik (*true* atau *false*)
c. Menjalankan keseluruhan loop pada batasan yang telah ditentukan
d. Memvalidasi struktur data internal

Pada pengujian *white-box* terdapat dua jenis yaitu pengujian struktur kontrol *(control structure testing)* dan pengujian jalur dasar (*basic path testing)* (Kurniawan, 2015). Pengujian struktur kontrol sebagai pelengkap untuk pengujian jalur dasar dimana terdapat pengujian kondisi dan juga pengujian *loop.* Pengujian jalur dasar adalah pengujian *white-box* yang dibuat berdasarkan ukuran tingkat kompleksitas dari algoritme hasil perancangan. Notasi-notasi yang digunakan pada pengujian *white box* antara lain:

1. Notasi *Flow Graph*
   Notasi *flow graph* merupakan notasi yang digunakan untuk menggambarkan jalur eksekusi dari kode program dengan menggunakan notasi *node* (disimbolkan dengan lingkaran) dan *edge* (penghubung yang disimbolkan dengan anak panah)



Gambar 2.2 Notasi *Flow Graph*
Sumber: Pressman(2010))

2. *Cyclomatic Complexity*
   Cyclomatic Complexity merupakan matriks yang digunakan pada perangkat lunak untuk mengukur nilai kompleksitas pada perangkat lunak yang telah didefinisikan sebelumnya (Pressman, 2010). Nilai yang diperoleh dari *cyclomatic complexity* ini ialah banyaknya kemungkinan jalur yang digunakan pada kode program. Terdapat tiga persamaan yang dapat dipilih untuk menghitung nilai kompleksitas perangkat lunak, antara lain:

a. V(G) = E - N + 2, E merupakan jumlah tepi pada *flow graph* dan N merupakan jumlah simpul pada *flow graph*
b. V(G) = P + 1, P merupakan jumlah predikat *node* pada *flow graph*
c. Jumlah dari region pada *flow graph.*

3. *Independent Path*
   Sebagai jalur yang melalui program, setidaknya satu rangkaian proses atau kondisi yang baru. Sekurang-kurangnya terdapat satu tepi yang belum melintasi jalur sebelumnya (A.S & Shalahuddin, 2013).

2.9.2 Pengujian Integrasi

Pengujian integrasi merupakan teknik sistematis untuk membuat struktur program yang sementara pada saat melakukan uji untuk menemukan kesalahan terkait dengan *interfacing* (Pressman, 2010). Tujuan pengujian ini adalah untuk mengambil komponen yang telah diuji unit dan membuat struktur program yang ditentukan oleh desain. Meskipun unit telah diuji masing-masing dan dapat bekerja dengan baik, namun harus diuji juga ketika unit-unit disatukan, karena satu unit akan berdampak pada unit lainnya. Pengujian integrasi berfokus untuk mengevaluasi interaksi di antara unit tersebut. terdapat beberapa teknik dalam melakukan pengujian integrasi yaitu *top-down integration, bottom-up integration, regression testing,* dan *smoke testing* (Pressman, 2001).

Pada penelitian ini pengujian integrasi menggunakan pendekatan *top-down integration*. Pendekatan *top-down* adalah pendekatan untuk melakukan pengujian integrasi melalui hirarki kontrol dengan bergerak ke arah bawah. Pengujian dimulai dari modul kontrol utama menuju ke modul di bawahnya, modul yang berada di bawah dimasukkan ke dalam struktur baik secara *breadth-first* maupun *depth-first.* Strategi integrasi *top-down* memverifikasi kontrol utama di awal proses pengujian. Proses integrasi dilakukan dalam serangkaian lima langkah:

1. *Driver* tes menggunakan modul kontrol utama dan semua komponen yang berada di bawah modul kontrol utama diganti dengan *stub*.
2. *Stub* diganti satu per satu dengan komponen yang sebenarnya.
3. Pengujian dilakukan karena setiap komponen terintegrasi.
4. Setelah rangkaian tes selesai, *stub* lainnya diganti dengan komponen asli.
5. Pengujian regresi dapat dilakukan untuk memastikan bahwa kesalahan baru belum diperkenalkan.

2.9.3 Pengujian Validasi

Pengujian validasi dilakukan untuk memastikan bahwa semua kebutuhan yang didefinisikan telah terpenuhi oleh perangkat lunak(Pressman, 2001). Validasi didapatkan melalui rangkaian *black-box testing* yang menunjukkan ketepatan dengan kebutuhan. *Test plan* menguraikan kelas-kelas uji dan prosedur uji menentukan kasus-kasus uji tertentu yang akan digunakan untuk menunjukkan ketepatan dengan kebutuhan. Baik rencana dan prosedur dibuat untuk memastikan bahwa semua kebutuhan fungsional telah terpenuhi, semua karakteristik perilaku tercapai, semua kebutuhan kinerja tercapai, dan

dokumentasi benar. Dokumen spesifikasi kebutuhan perangkat lunak mengandung bagian yang disebut kriteria validasi. Informasi yang terkandung dalam bagian itu menjadi dasar untuk pendekatan pengujian validasi.

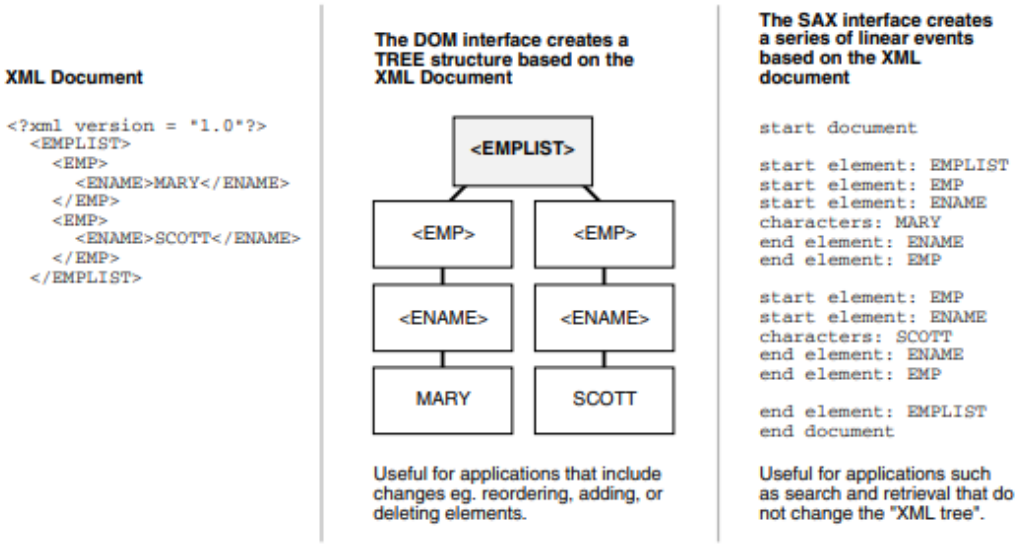
2.9.3.1 Pengujian Black Box

Pengujian *black-box* merupakan suatu pendekatan untuk pengujian di mana penguji tidak mengakses kode program atau komponen dari suatu sistem. Pengujian berdasarkan kebutuhan sistem (Sommerville, 2011). Pada pengujian *black-box* dibutuhkan kasus uji yaitu spesifikasi input untuk tes dan ekspektasi output dari sistem, ditambah dengan pernyataan tentang apa yang sedang diuji. Data uji merupakan masukan yang telah dirancang untuk menguji suatu sistem. Kadang-kadang data uji dapat dihasilkan secara otomatis, tetapi pembuatan kasus uji otomatis tidak mungkin dilakukan, karena orang-orang yang memahami apa yang seharusnya dilakukan sistem harus terlibat untuk menentukan hasil tes yang diharapkan.

2.10 Teknologi Pengembangan

2.10.1 JAVA API for XML Processing (JAXP)

JAXP adalah antarmuka standar untuk memproses XML dengan aplikasi Java (Ashdown, Greenberg, & Melnick, 2014). JAXP mendukung standar DOM dan SAX. DOM API memparsing dokumen XML dan membangun representasi *tree* dari dokumen dalam memori, menyediakan kemampuan pengguna untuk mengakses dan memanipulasi seperti menambah, membaca, mengganti elemen atau, dan menghapus. DOM API lebih mudah digunakan karena menyediakan struktur *tree* yang familiar .Sedangkan SAX API memproses dokumen XML sebagai aliran peristiwa, yang berarti program tidak dapat mengakses lokasi acak dalam dokumen. SAX berguna untuk operasi pencarian dan program lain yang tidak perlu manipulasi *XML tree.* SAX lebih cepat dari DOM ketika mengambil dokumen XML dari database. Pada penelitian ini JAXP digunakan untuk memproses *class diagram* sebagai inputan untuk perhitungan nilai *reusability* dalam format XML untuk selanjutnya masuk kedalam perhitungan metric untuk perhitungan nilai *reusability.* Perbandingan DOM dan SAX API ditunjukkan dalam Gambar 2.6.

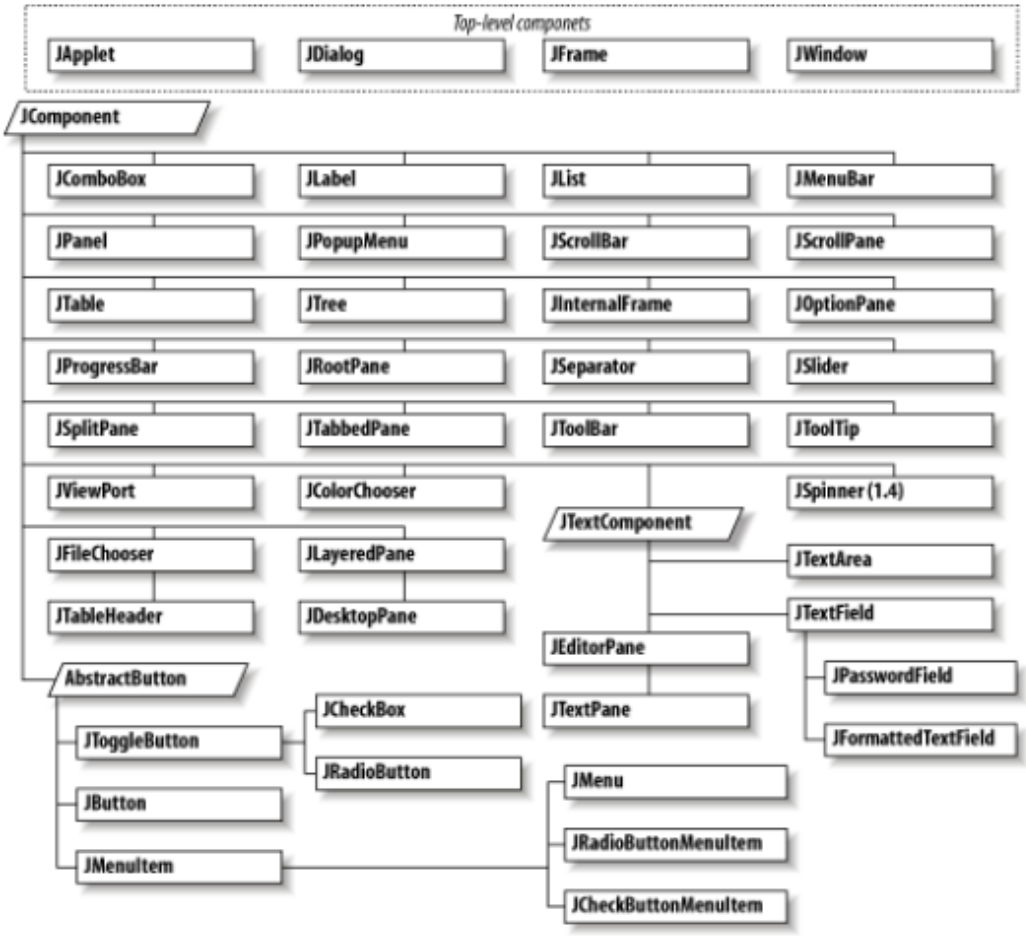


Gambar 2.5 Perbedaan SAX API dan DOM API

Sumber: (Ashdown, Greenberg, & Melnick, 2014)

2.10.2 *Swing*

Swing adalah toolkit GUI yang diciptakan Sun Microsystems yang merupakan generasi baru untuk memungkinkan pengembangan aplikasi *enterprise* di Java (Cole, dkk, 2002). Dengan menggunakan Swing, pemrogram dapat membuat aplikasi Java skala besar dengan beragam komponen yang kuat. Selain itu, dapat dengan mudah memodifikasi komponen-komponen ini untuk mengontrol penampilan dan perilakunya. Swing adalah bagian dari produk-produk Java yaitu Java Foundation Classes (JFC), yang merupakan gabungan banyak fitur dari Netscape's Internet Foundation Class (IFC) serta aspek desain dari divisi Taligent IBM dan Desain Mercusuar. Swing membutuhkan setidaknya JDK 1.1.5 untuk menjalankan. Swing dibangun berdasarkan model yang diperkenalkan dalam seri JDK 1.1. Selain itu harus memiliki *browser* yang mendukung Java 1.1 untuk mendukung applet Swing. Komponen hirarki Swing ditunjukkan dalam Gambar 2.7.



Gambar 2.6 Komponen hirarki Swing

Sumber: (Cole, dkk, 2002)

2.11 Perhitungan Efisiensi

Efisiensi merupakan salah satu jenis kebutuhan non-fungsional yang termasuk dalam kebutuhan produk, efisiensi dapat berupa kebutuhan performa dan kebutuhan ruang (Sommerville, 2011). Perhitungan efisiensi terdapat dua macam yaitu efisiensi waktu (*time based efficiency)* dan efisiensi secara keseluruhan (*overall relative efficiency)* (Sergeev, 2010). Efisiensi waktu dapat dihitung dari kecepatan pengguna ketika menyelesaikan suatu tugas. Sedangkan efisiensi secara keseluruhan dapat dihitung dari presentase waktu yang diperlukan pengguna untuk berhasil menyelesaikan tugas dalam kaitannya dengan total waktu yang diperlukan oleh semua pengguna. Persamaan 2.6 adalah perhitungan efisiensi waktu dan Persamaan 2.7 adalah perhitungan efisiensi secara keseluruhan.

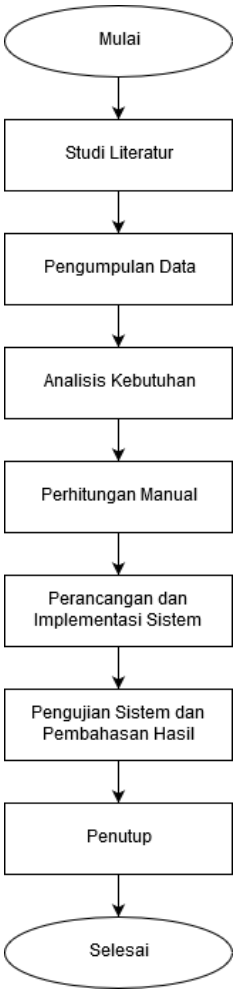
$$\bar{P}_t = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N}\frac{n_{ij}}{t_{ij}}}{NR} \quad (2.6)$$

$$\bar{P} = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} n_{ij}t_{ij}}{\sum_{j=1}^{R}\sum_{i=1}^{N} t_{ij}} * 100\% \quad (2.7)$$

Keterangan:
$\bar{P}t$: *Time Based Efficiency*
$\bar{P}$ *: Overall Relative Efficiency*
N : Jumlah skenario (tugas)
R : Jumlah responden (pengguna)
nij : tugas i yang diselesaikan pengguna j (nilainya jika berhasil adalah 1 dan gagal bernilai 0)
tij : waktu yang diperlukan responden untuk menyelesaikan skenario

BAB 3 METODOLOGI PENELITIAN

Metodologi penelitian ini berfungsi untuk memudahkan pemahaman alur penelitian. Langkah dari penelitian ini adalah studi literatur, pengumpulan data, analisis kebutuhan, perhitungan manual, perancangan dan implementasi sistem, pengujian dan pembahasan hasil, lalu penutup yang berisi kesimpulan dan saran. Untuk metode pengembangan aplikasinya, penulis menggunakan *SDLC waterfall model.* Diagram alir dari metodologi penelitian ini ditunjukkan dalam Gambar 3.1.



Gambar 3.1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Langkah pertama yaitu studi literatur. Penulis mengkaji landasan teori terkait dan sesuai dengan topik penelitian. Landasan teori yang dikaji bersumber dari jurnal internasional, buku, artikel serta penelitian yang sebelumnya telah dilakukan. Adapun teori-teori yang dipelajari oleh penulis antara lain: rekayasa perangkat lunak, pengembangan perangkat lunak, model SDLC, pendekatan berorientasi objek, pemodelan berorientasi objek, kualitas perangkat lunak, *reusability* perangkat lunak, properti desain berorientasi objek, metrik desain

berorientasi objek*,* metrik *testability, effectiveness* dan *reusability,* pengujian perangkat lunak, dan teknologi pengembangan yaitu JAXP dan *Swing*.

3.2 Analisis Kebutuhan

Langkah kedua adalah analisis kebutuhan yang menjelaskan deskripsi umum sistem, elisitasi kebutuhan, identifikasi karakteristik pengguna, kebutuhan fungsionalitas dan non-fungsionalitas sistem, lingkungan pembangunan kakas bantu perhtiungan nilai *testability.* Dalam pembangunan sistem ini menggunakan pendekatan berorientasi objek sehingga kebutuhan fungsionalitas yang telah didefinisikan dimodelkan dengan *use case diagram,* dan kemudian di spesifikasikan dengan *use case scenario.*

3.3 Perhitungan Manual

Pada tahap perhitungan manual dilakukan pengukuran untuk menghitung nilai dari *coupling, cohesion, encapsulation,* dan *inheritance*. Perhitungan manual dilakukan untuk memberikan perincian dan menjelaskan tentang pemahaman perhitungan nilai *testability.*

3.4 Perancangan dan Implementasi Sistem

Langkah ke empat adalah perancangan sistem yang berdasarkan hasil rekayasa kebutuhan. Pada penelitian ini menggunakan pendekatan yang berorientasi objek maka rancangan dimodelkan dengan *sequence diagram* dan *class diagram,* selain itu juga terdapat perancangan algoritme dan perancangan antarmuka. Pada tahap ini juga dilakukan perancangan antarmuka pengguna. Kemudian implementasi sistem sesuai dengan rancangan yang telah dibuat. Pada implementasi dijelaskan spesifikasi sistem, batasan sistem, kode sumber, dan hasil implementasi antarmuka. Implementasi sistem dibuat menggunakan bahasa pemrograman *Java* dan berbasis *dekstop*.

3.5 Pengujian Sistem dan Pembahasan Hasil

Selanjutnya dilakukan langkah pengujian terhadap hasil implementasi sistem. Pengujian yang dilakukan yaitu pengujian unit menggunakan teknik pengujian *white-box,* pengujian integrasi menggunakan teknik *top-down,* dan pengujian validasi menggunakan teknik pengujian *black-box* untuk kebutuhan fungsional maupun non-fungsional. Selain itu pada bagian ini juga dilakukan pembahasan hasil pengujian.

3.6 Penutup

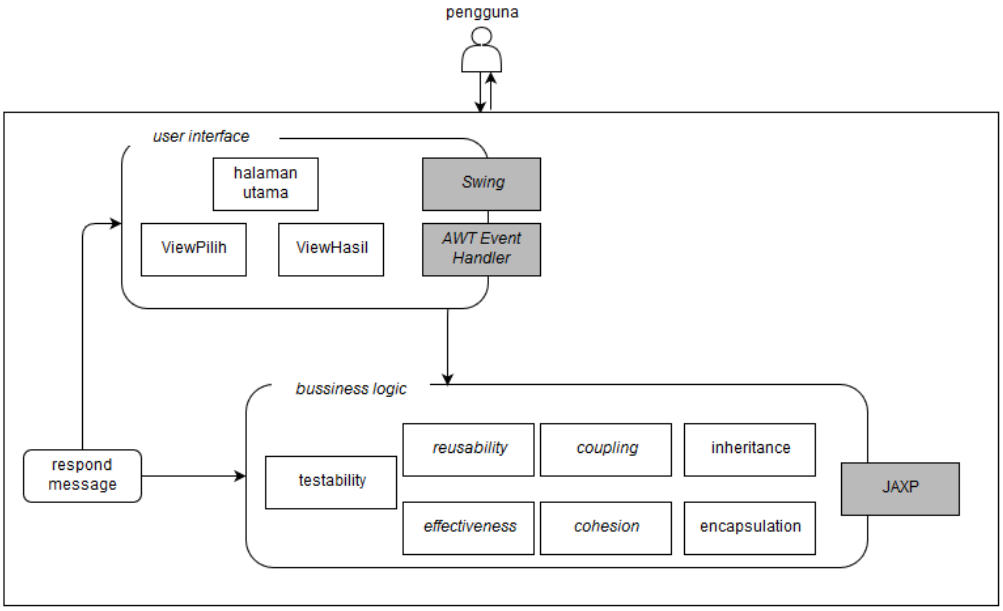
Pada tahap penutup ini dilakukan penarikan kesimpulan dari hasil penelitian yang telah dilakukan yaitu berdasarkan rumusan masalah yang telah diuraikan. Selain itu juga diuraikan saran untuk penelitian selanjutnya yang merupakan masukan dan juga evaluasi dari kesalahan yang mungkin terjadi pada penelitian ini

BAB 4 ANALISIS KEBUTUHAN

4.1 Deskripsi Umum Sistem

Langkah pertama yang dilakukan pada saat mengembangkan perangkat lunak yaitu analisis kebutuhan. Pengembangan sistem kakas bantu untuk menghitung nilai *testability* menggunakan sebuah rancangan perangkat lunak yaitu *class diagram.* Perhitungan nilai *testability* pada penelitian ini menggunakan metrik properti desain berorientasi objek dengan teknik regresi berganda. Properti desain berorientasi objek yang mempengaruhi nilai *testability* adalah *encapsulation, coupling, inheritance, dan cohesion.* Metrik *Data Access Metric* (DAM) digunakan untuk mendapatkan poin *encapsulation*, metrik *Measure of Functional Abstraction* (MFA) digunakan untuk menadapatkan poin *inheritance,* metrik *Direct Class Coupling* (DCC) digunakan untuk mendapatkan poin *coupling* dan metrik *Cohesion Among Method of the Class* (CAM) digunakan untuk mendapatkan poin *cohesion.* Metrik-metrik tersebut berperan sebagai variabel independen dan *testability* sebagai variabel dependen.

Kakas Bantu Perhitungan Nilai *Testability* adalah aplikasi yang dikembangan berbasis *dekstop.* Perhitungan nilai *testability* dapat dilakukan jika pengguna memasukan rancangan ssitem berupa *class diagram* dengan format XML. Setelah pengguna melakukan proses tersebut, sistem akan memindai komponen-komponen yang dibutuhkan untuk perhitungan nilai *testability.* Penggunaan JAXP pada penelitian ini untuk memproses XML dan mendapatkan komponen-komponen yang diperlukan.



Gambar 4.1 Arsitektur Sistem Kakas Bantu Nilai *Testability*

Gambar 4.1 merupakan gambaran bagaimana proses dari sistem berjalan, sistem dan pengguna saling berhubungan melalui antarmuka pengguna yang dibangun dengan komponen *Swing* dan *event-handler* menggunakan AWT untuk memberi masukan maupun menerima keluaran. Masukan dari pengguna kemudian dikelola oleh *business logic* pada sistem dimana JAXP memproses berkas XML yang kemudian hasilnya digunakan untuk melakukan perhitungan poin *encapsulation, inheritance, coupling, cohesion, reusability, effectiveness* dan *testability.* Setelah berkas masukan diproses langkah selanjutnya yaitu sistem memberi *respond message* yang akan ditampilkan pada antarmuka pengguna.

4.2 Elisitasi Kebutuhan

Pada tahap ini kebutuhan pengguna diuraikan yang kemudian dipenuhi oleh sistem. Elisitasi kebutuhan dilakukan dengan mempelajari pustaka yang berhubungan dengan pengukuran nilai *testability.*

1. Memilih file *class diagram* perangkat lunak dengan format XML.
2. Menghitung nilai *testability* dari *class diagram* perangkat lunak.
3. menghitung nilai *effectiveness* dari *class diagram* perangkat lunak.
4. menghitung nilai *reusability* dari *class diagram* perangkat lunak.
5. Melihat hasil dari pengukuran *testability* dari *class diagram* perangkat lunak.

4.3 Identifikasi Pengguna

Pengguna dari sistem perhitungan nilai *testability* ini adalah tim pengembang perangkat lunak khususnya *desainer* dan *tester* perangkat lunak yang memahami istilah-istilah pada sistem yaitu *encapsulation, inheritance, coupling, cohesion class diagram, design, reusability, effectiveness,* dan *testability.* Sistem hanya mempunyai satu pengguna yaitu perancang dari perangkat lunak sehingga pada penulisan ini penulis menggunakan istilah ‘pengguna’ untuk menyebut perancang sebagai pengguna. Sistem ini digunakan oleh pengguna untuk menghitung rancangan dari perangkat lunak yang telah dibuat memiliki seberapa nilai *testability*. Tabel 4.1 menunjukkan deskripsi yang dapat dilakukan oleh pengguna pada sistem ini.

Tabel 4.1 Identifikasi Pengguna

No	Identifikasi pengguna	deksripsi
1.	Pengguna	Yang dapat dilakukan pengguna pada sistem ini adalah memilih berkas *class diagram* rancangan perangkat lunak berformat XML, menghitung nilai *testability* dari *class diagram* rancangan perangkat lunak, menghitung nilai *effectiveness* dari *class diagram* rancangan perangkat lunak, menghitung nilai *reusability* dari *class diagram*

		rancangan perangkat lunak, melihat hasil dari pengukuran *testability class diagram* rancangan perangkat lunak.

4.4 Spesifikasi Kebutuhan Sistem

4.4.1 Kebutuhan Fungsional

Setiap kebutuhan fungsional memiliki kode kebutuhan F_TEST_XXX. Dimana F menunjukkan kebutuhan fungsional, TEST merupakan nama sistem yang dikembangkan dan XXX menjukkan spesifikasi nomor kebutuhan. Dengan adanya spesifikasi kebutuhan, diharapkan sistem mampu memenuhi kebutuhan yang telah diuraikan pada Tabel 4.2 berikut dengan spesifikasinya.

Tabel 4.2 Spesifikasi Kebutuhan Fungsional Sistem

Kode kebutuhan	Use Case	Spesifikasi Kebutuhan
F_TEST_001	Cari Berkas	Sistem menyediakan layanan bagi pengguna untuk memasukkan *file* dari *class diagram* rancangan perangkat lunak.
		1.1.Sistem mempunyai tombol *Browse* dan juga t*extfield* yang nantinya terisi otomatis oleh *file* masukkan dari pengguna. 1.2.Sistem menampilkan dialog untuk cari berkas. 1.3.Hanya berkas dengan format XML saja bisa dipilih dan ditampilkan pada direktori. 1.4.Berkas masukan berformat XML dan berstruktur *simple* sajalah yang dapat diterima sistem.
F_TEST_002	Hitung *Testability*	Sistem menyediakan layanan untuk menghitung *testability* dari *class diagram* rancangan perangkat lunak dengan masukan dari pengguna.
		2.1. Sistem *mengukur testability* dari *class diagram* rancangan perangkat lunak menggunakan teknik regresi bergada yang diambil dari nilai *effectiveness* dan *reusability*

Tabel 4.2 Spesifikasi Kebutuhan Fungsional Sistem (lanjutan)

		2.2. Sistem mempunyai tombol untuk menjalankan proses perhitungan. 2.3. Sistem tidak akan bisa melakukan perhitungan apabila tidak ada *file* XML dari *class diagram* rancangan perangkat lunak yang dipilih dan tidak berformat *simple*.
F_TEST_003	Hitung *Effectiveness*	Sistem menyediakan layanan untuk menghitung nilai *effectiveness* dari *class diagram* rancangan perangkat lunak dengan masukan dari pengguna.
		3.1. Sistem mengukur nilai *effectiveness* dari *class diagram* rancangan perangkat lunak dengan teknik regresi berganda yang diambil dari nilai *encapsulation, inheritance, coupling* dan *cohesion*. 3.2. Sistem mempunyai tombol untuk menjalankan proses perhitungan. 3.3. sistem tidak akan bisa melakukan perhitungan apabila tidak ada file XML dari *class diagram* rancangn perangkat lunak yang dipilih dan tidak berformat *simple.*
F_TEST_004	Hitung *Reusability*	Sistem menyediakan layanan untuk menghitung *reusability* dari *class diagram* rancangan perangkat lunak dengan masukan dari pengguna.
		4.1. Sistem mengukur *reusability* dari *class diagram* rancangan perangkat lunak dengan teknik regresi berganda yang diambil dari nilai *coupling*, *encapsulation* dan *inheritance* untuk mengukur *reusability class diagram* rancangan perangkat lunak. 4.2. Sistem mempunyai tombol untuk menjalankan proses perhitungan. 4.3. sistem tidak akan bisa melakukan perhitungan apabila tidak ada file XML dari *class diagram* rancangan perangkat lunak yang dipilih dan tidak berformat *simple*.

Tabel 4.2 Spesifikasi Kebutuhan Fungsional Sistem (lanjutan)

F_TEST_005	Lihat Hasil	Sistem menyediakan layanan untuk menampilkan hasil perhitngan dari *testability* bagi pengguna.
		5.1. Sistem menyediakan halaman untuk melihat hasil dari perhitungan *testability.* 5.2. Sistem dapat menampilkan hasil dari nilai *testability* beserta dengan rincian perhitungannya.

4.4.2 Kebutuhan Non Fungsional

Setiap kebutuhan non-fungsional memiliki kode kebutuhan NF_TEST_XXX. Dimana NF menunjukkan kebutuhan non-fungsional, TEST merupakan nama sistem yang dikembangkan dan XXX menjukkan spesifikasi nomor kebutuhan. Dengan adanya spesifikasi kebutuhan, diharapkan sistem mampu memenuhi kebutuhan yang telah diuraikan pada Tabel 4.3 berikut dengan spesifikasinya.

Tabel 4.3 Deskripsi Kebutuhan Non Fungsional

Kode Kebutuhan	parameter	Deskripsi
NF_TEST_001	Efisiensi	Tingkat efisiensi yang dimiliki sistem yaitu 100% agar dapat mencapai keberhasilan perhitungan dan lebih cepat dari perhitungan manual. Sistem mempunyai tingkat efisiensi yaitu 100% untuk keberhasilan melakukan perhitungan dan lebih cepat dari perhitungan manual.

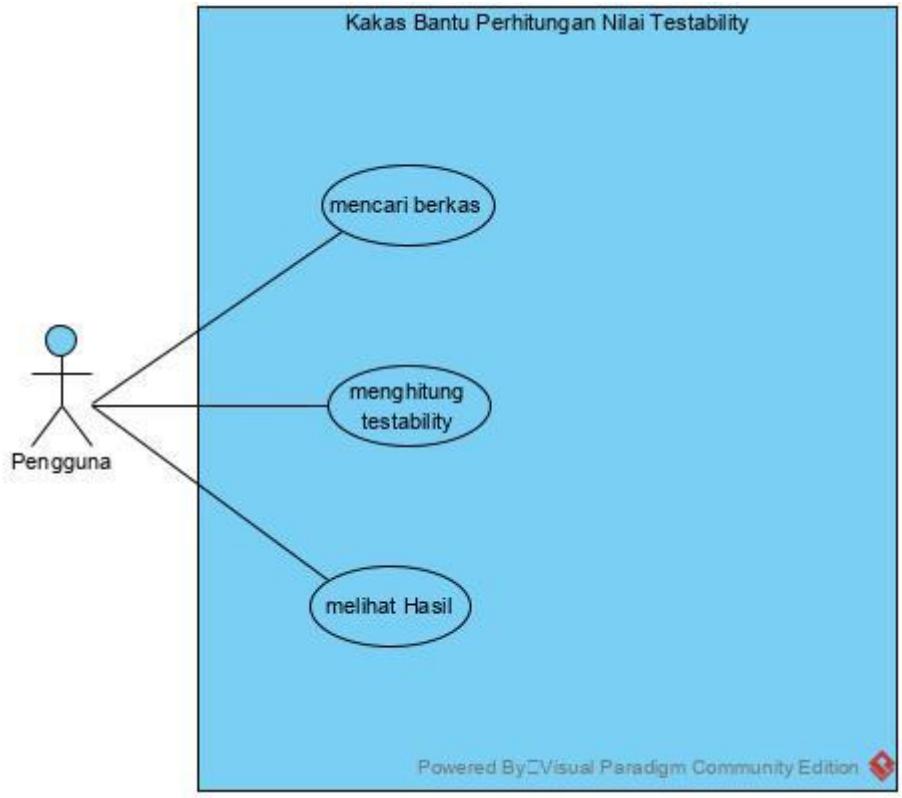
4.5 Lingkungan Pengembangan

Spesifikasi lingkungan pengembangan sistem kakas bantu nilai *testability* adalah sebagai berikut:

1. Menggunakan bahasa pemrograman *Java.*
2. Menggunakan IDE NetBeans versi 8.2.
3. Menggunakan JAXP untuk pemroses XML.
4. Menggunakan sistem operasi Windows 10.

4.6 Pemodelan Kebutuhan

4.6.1 Use Case Diagram



Gambar 4.2 Use Case Diagram Sistem

Kebutuhan fungsional yang telah dijabarkan pada sub bab sebelumnya lalu dimodelkan kedalam *use case diagram* dalam Gambar 4.2. Kemudian penjelasan secara detail tentang perilaku setiap *use case* selanjutnya dijelaskan kedalam *use case scenario* yang diuraikan pada Tabel 4.4 sampai dengan Tabel 4.9. Pada *use case diagram* tersebut menunjukkan Pengguna adalah aktor tungal dalam sistem yang dapat menggunakan semua kebutuhan sistem. Kebutuhan sistem berjumlah 9 yaitu:

1. Mencari berkas (F_TEST_001)
2. Hitung *testability* (F_TEST_002)
3. Melihat hasil (F_TEST_003)

4.6.2 *Use Case Scenario*

Berikut adalah penjelasan lebih rinci perilaku pengguna dengan sistem dari *use case scenario*:

1. *Use Case Scenario* Mencari Berkas (F_TEST_001)

   Cari Berkas dipergunakan ketika pengguna ingin menghitung nilai *testability.* Halaman ini digunakan untuk mencari *file* yang kemudian akan

dihitung oleh sistem. Proses skenario dari *use case* cari berkas akan dijelaskan pada Tabel 4.4.

Tabel 4.4 *Use case scenario cari berkas*

Cari Berkas	
Kode	F_TEST_001
Objectives	Menjadi fungsi untuk pengguna agar dapat memilih berkas *class diagram* rancangan perangkat lunak pada direktori lokal.
Actor	Pengguna
Pre Conditions	1. Halaman cari berkas ditampilkan.
Main Flow	1. Pengguna menekan tombol Cari Berkas 2. Sistem menampilkan dialog pilih file dan hanya menampilkan berkas dengan format XML. 3. Pengguna memilih berkas yang sesuai dan menekan tombol *Open.*
Alternative Flow	1. Pengguna menekan tombol *Cancel* untuk membatalkan pemilihan berkas.
Post Condition	Alamat direktori berkas yang dipilih ditampilkan.

2. *Use Case Scenario Ukur Testability* (F_TEST_002)

Perilaku dari interaksi *use case* ukur *testability* dengan Pengguna dijelaskan secara rinci pada Tabel 4.5.

Tabel 4.5 *Use case scenario* ukur *testability*

Ukur *Testability*	
Kode	F_TEST_002
Objectives	Menjadi fungsi untuk pengguna mengukur *testability* dari *class diagram* perangkat lunak yang telah dipilih.
Actor	Pengguna
Pre Conditions	1. Halaman cari berkas ditampilkan. 2. File yang akan diukur telah dipilih.
Main Flow	1. Pengguna menekan tombol Hitung.

Tabel 4.5 *Use case scenario* ukur *testability* (lanjutan)

	2. Sistem melakukan parsing berkas XML, mengukur *reusability*
Alternative Flow	1. Apabila berkas tidak berstruktur *simple* dari *Visual Paradigm* maka sistem akan menampilkan notifikasi bahwa berkas XML tidak sesuai. 2. Apabila tidak ada sebuah berkas yang dipilih maka sistem akan memberi notifikasi bahwa berkas belum dipilih.
Post Conditions	Nilai *testability* telah dihitung.

3. *Use Case Scenario* Lihat Hasil (F_TEST_003)

   Perilaku dari interaksi *use case* ukur lihat hasil dengan Pengguna dijelaskan secara rinci pada Tabel 4.8.

Tabel 4.6 *Use case scenario* lihat hasil

Lihat Hasil	
Kode	F_TEST_003
Objectives	Menjadi fungsi untuk pengguna melihat hasil dari pengukuran *testability*, pengukuran *effectiveness*, pengukuran *reusability*, poin *cohesion*, poin *coupling*, poin *inheritance*, dan poin *encapsulation class diagram* rancangan perangkat lunak
Actor	Pengguna
Pre Conditions	1. Halaman cari berkas ditampilkan. 2. File yang akan diukur telah dipilih.
Main Flow	1. Pengguna menekan tombol navigasi yang menunju ke halaman lihat hasil.
Alternative Flow	-
Post Conditions	Hasil dari pengukuran *testability*, pengukuran *effectiveness*, pengukuran *reusability*, poin *cohesion*, poin *coupling*, poin *inheritance*, poin *encapsulation*, dan rincian rumus pengukuran ditampilkan.

4.7 Perhitungan Manual

Perhitungan manual dilakukan untuk memberikan perincian dan menjelaskan tentang pemahaman perhitungan nilai *testability* perangkat lunak dengan rumus teknik regresi berganda menggunakan 4 variabel independen yaitu *cohesion, coupling, inheritance*¸ dan *encapsulation. .* Adapun nilai konstanta bagi masing-masing variabel telah ditentukan sebelumnya, yaitu 8,783 untuk *effectiveness* dan -37,111 untuk *reusability*. Sedangkan nilai konstanta dari *testability* telah didefinisikan pada Gambar 4.3.

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	59.524	48.835		1.219	0.347
	Effectiveness	−4.671	3.739	−0.563	−1.249	0.338
	Reuseability	0.806	0.351	1.034	2.294	0.149

Gambar 4.3 Nilai Konstanta *Testability*

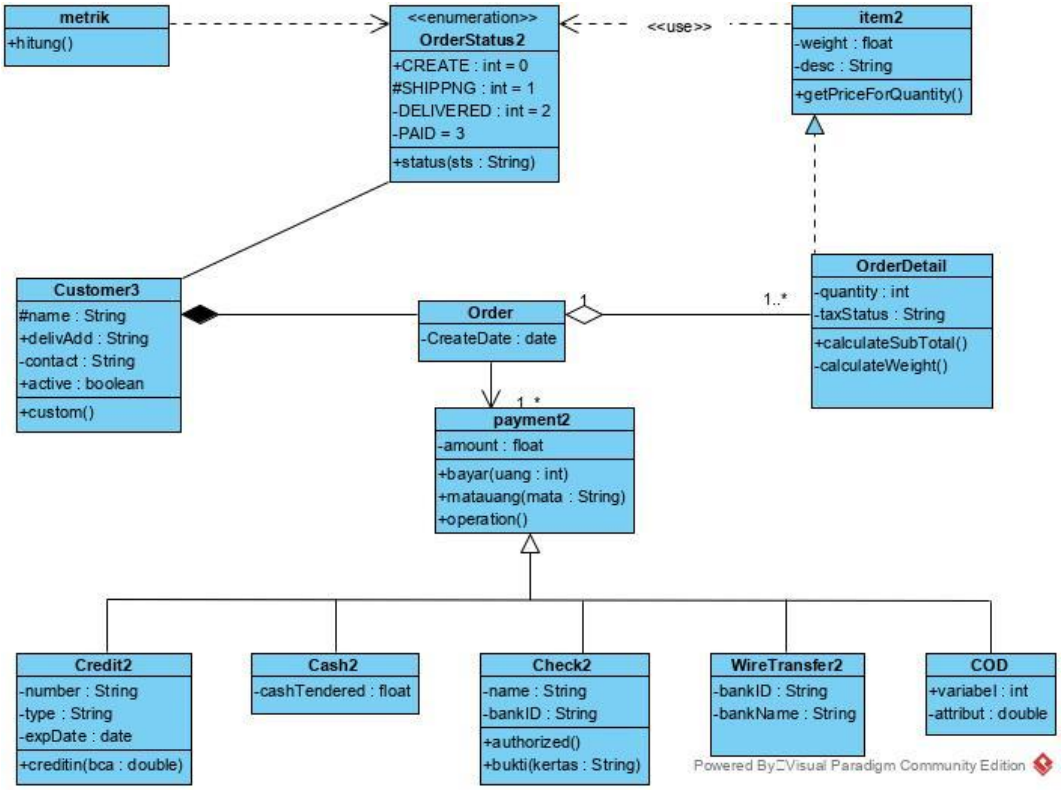
$$Effectiveness = 8{,}783 - 1{,}614 * encapsulation + 11{,}141 * inheritance - 0{,}866 * coupling - 6{,}477 * cohesion$$

$$Reusability = -37{,}111 + 3{,}973 * coupling + 32{,}500 * inheritance + 20{,}709 * encapsulation$$

$$Testability = 59.524 - 4.671 \times Effectiveness + 0.806 \times Reusability$$

Dimana,
Cohesion : Dihitung menggunakan metric CAM
Coupling : Dihitung menggunakan metrik DCC
Inheritance : Dihitung menggunakan metrik MFA
Encapsulation : Dihitung menggunakan metrik DAM

Gambar 4.4 merupakan contoh *class diagram* yang digunakan untuk perhitungan.



Gambar 4.4 Contoh *class diagram*

Dari contoh *class diagram* didapatkan data seperti pada Tabel 4.13 dan Table 4.14:

Tabel 4.7 Data set *effectiveness* sistem

Effectiveness			
Cohesion	***Coupling***	***Inheritance***	***Encapsulation***
0,08/0,8=0,1	26/12 = 2,17	4,35/0,8=0,54	9,75/11=0,89

Tabel 4.8 Data set *reusability* sistem

Reusability		
Coupling	***Inheritance***	***Encapsulation***
26/12 = 2,17	4,35/0,8=0,54	9,75/11=0,89

Dengan menggunakan data set dari Tabel 4.13, maka dapat dilakukan pengukuran nilai *effectiveness* sistem sebagai berikut:

$$Effectiveness = 8{,}783 - 1{,}614 * 0{,}89 + 11{,}141 * 0{,}54 - 0{,}866 * 2{,}17 - 6{,}477 * 0{,}1$$

$$Effectiveness = 8{,}783 - 1{,}436 + 6{,}016 - 1{,}879 - 0{,}6477$$

$$Effectiveness = 12{,}132$$

Dengan menggunakan data set dari Tabel 4.14, maka dapat dilakukan pengukuran nilai *reusability* sistem sebagai berikut:

$$Reusability = -37{,}111 + 3{,}973 * 2{,}17 + 32{,}500 * 0{,}54 + 20{,}709 * 0{,}89$$

$$Reusability = -37{,}111 + 8{,}608 + 17{,}672 + 18{,}356$$

$$Reusability = 7{,}525$$

Setelah nilai *effectiveness* dan *reusability* dari sistem telah diukur, maka nilai *testability* sistem dapat diperoleh dengan menggunakan rumus pengukuran *testability* sebagai berikut:

$$Testability = 59{,}524 - 4{,}671 * 12{,}132 + 0{,}806 * 7{,}525$$

$$Testability = 59{,}524 - 56{,}668 + 6{,}065$$

$$Testability = 8{,}921$$

BAB 5 PERANCANGAN DAN IMPLEMENTASI SISTEM

5.1 Perancangan Sistem

Perancangan sistem dibuat sesuai dengan hasil analisis kebutuhan. Pada bagian perancangan sistem ini terdapat perancangan arsitektur, algoritme, dan juga antarmuka. Perancangan sistem ini yang digunakan sebagai acuan tahap selanjutnya yaitu implementasi.

5.1.1 Perancangan Arsitektur

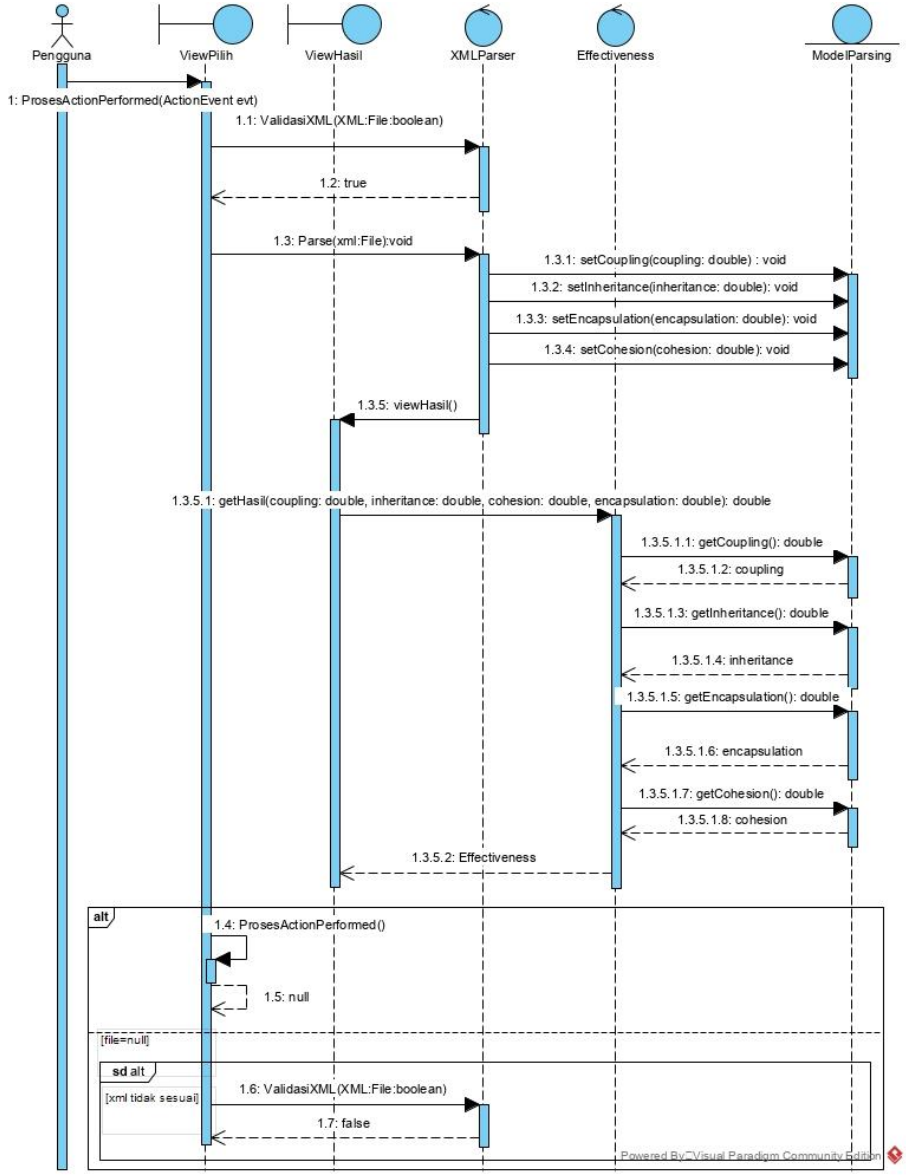
Karena dalam pengembangan ini menggunakan pendekatan berorientasi objek, maka pada tahap perancangan arsitektur dimodelkan dengan *sequence diagram* dan *class diagram.*

5.1.1.1 Sequence Diagram

1. Ukur *Effectiveness*

   Gambar 5.1 menjelaskan alur pengguna berinteraksi meminta sistem untuk melakukan pemrosesan berkas *class diagram* rancangan perangkat lunak dengan format *XML* untuk mendapatkan nilai *effectiveness*nya setelah mendapatkan nilai poin *coupling,* nilai poin *inheritance,* nilai poin encapsulation, dan nilai poin *cohesion* sesuai rumus persamaan. Terdapat delapan objek yang terlibat dalam interaksi ini yaitu *ViewPilih* dan *ViewHasil* dan lihatIndeks sebagai *boundary, Coupling, Inheritance, Encapsulation, Cohesion,* dan *Effectiveness* sebagai *controller,* dan *XMLParser* sebagai *model*.

   Ketika *method prosesActionPerformed* dipanggil dari *ViewPilih* maka berkas XML yang dipilih diseleksi apakah struktur nya dari struktur *simple Visual Paradigm.* Jika struktur sesuai, selanjutnya berkas akan di*parsing* untuk diambil elemen yang dibutuhkan pada pengukuran *effectiveness* yang dilakukan oleh *XMLParser.* Setelah elemen yang dibutuhkan didapat, kemudian ke kelas *ViewHasil* menjankan konstruktornya dan memanggil *getHasil dari kelas Effectiveness* untuk mengukur *effectiveness class diagram*. *Effectiveness* didapat dari rumus yang memanggil *getHasil* dari kelas *Coupling, getHasil* dari kelas *Inheritance, getHasil* dari kelas *Encapsulation,* dan *getHasil* dari kelas *Cohesion.* Selanjutnya hasil *effectiveness* ditampilkan pada *ViewHasil.* Jika struktur XML tidak sesuai maka proses pengukuran tidak akan dilakukan. Selain itu apabila tidak ada berkas yang dipilih maka tidak akan dilakukan proses pengukuran juga, hal ini dicek oleh *ViewPilih* saat melalui *method prosesActionPerformed.*



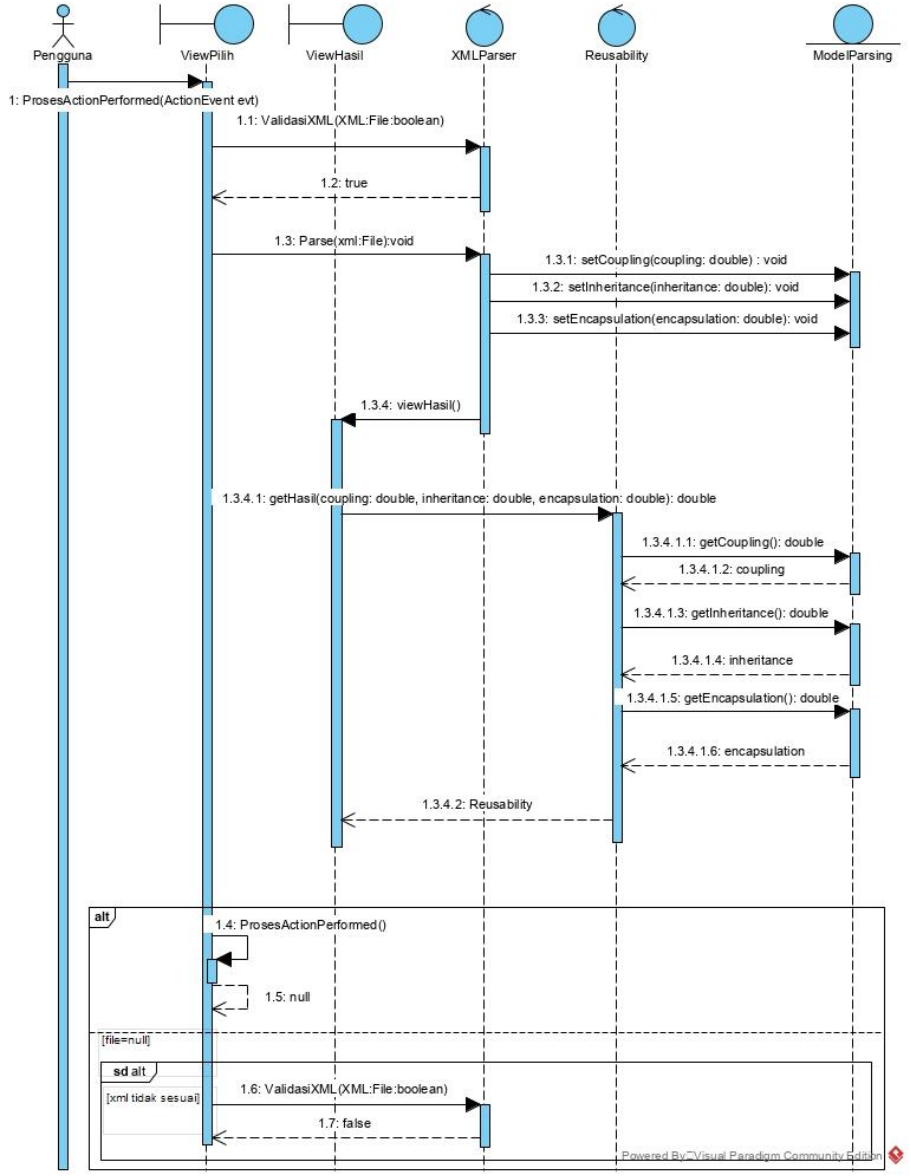
Gambar 5.1 *Sequence diagram* ukur *effectiveness*

2. Ukur *Reusability*

Gambar 5.2 menjelaskan alur pengguna berinteraksi meminta sistem untuk melakukan pemrosesan berkas *class diagram* rancangan perangkat lunak dengan format *XML* untuk mendapatkan nilai *reusability*nya setelah mendapatkan nilai poin *coupling,* nilai poin *inheritance,* dan nilai poin *encapsulation* sesuai rumus persamaan. Terdapat delapan objek yang terlibat dalam interaksi ini yaitu *ViewPilih* dan *ViewHasil* dan lihatIndeks sebagai *boundary, Coupling, Inheritance, Encapsulation,* dan *Reusability* sebagai *controller,* dan *XMLParser* sebagai *model*.

Ketika *method prosesActionPerformed* dipanggil dari *ViewPilih* maka berkas XML yang dipilih diseleksi apakah struktur nya dari struktur

simple Visual Paradigm. Jika struktur sesuai, selanjutnya berkas akan di*parsing* untuk diambil elemen yang dibutuhkan pada pengukuran *reusability* yang dilakukan oleh *XMLParser.* Setelah elemen yang dibutuhkan didapat, kemudian ke kelas *ViewHasil* menjankan konstruktornya dan memanggil *getHasil dari kelas Reusability* untuk mengukur *reusability. Reusability* didapat dari rumus yang memanggil *getHasil* dari kelas *Coupling, getHasil* dari kelas *Inheritance,* dan *getHasil* dari kelas *Encapsulation.* Selanjutnya hasil *reusability* ditampilkan pada *ViewHasil.* Jika struktur XML tidak sesuai maka proses pengukuran tidak akan dilakukan. Selain itu apabila tidak ada berkas yang dipilih maka tidak akan dilakukan proses pengukuran juga, hal ini dicek oleh *ViewPilih* saat melalui *method prosesActionPerformed.*

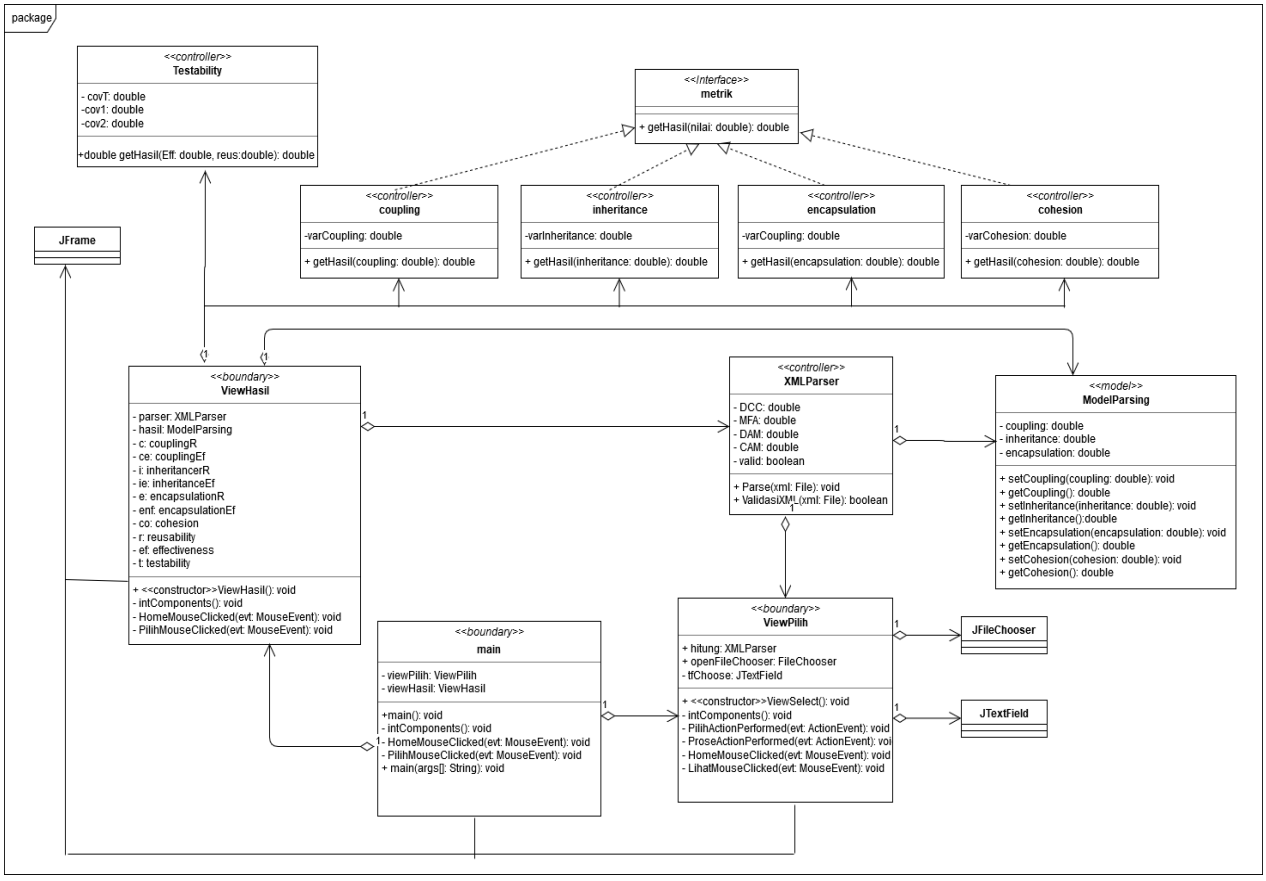


Gambar 5.2 *Sequence* diagram ukur *reusability*

5.1.1.2 Class Diagram

Class diagram dari sistem pengukuran nilai *reusability* ditunjukkan dalam Gambar 5.3. Didalam *class diagram* ini terdapat satu kelas *interface,* tiga kelas *boundary,* empat kelas *controller,* satu kelas *model*, dan tiga kelas dari penggunaan *API Swing.* Untuk lebih rinci penjelasan dari setiap kelasnya adalah sebagai berikut:

1. Kelas *interface* metrik sebagai kelas abstraksi dari kelas *Inheritance*, kelas *Coupling*, kelas *Encapsulation*, dan kelas *cohesion*. Dalam kelas ini terdapat satu operasi abstrak yaitu getHasil.



Gambar 5.3 *Class Diagram*

2. Kelas *controller Coupling* bertugas untuk mengukur poin *coupling* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varCoupling* yang menyimpan nilai koefisian *coupling* untuk pengukuran *reusability* dan *effectiveness.* Dalam kelas ini juga terdapat satu operasi yang merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*
3. Kelas *controller Inheritance* bertugas untuk mengukur poin *inheritance* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varInheritance* yang menyimpan nilai koefisian *inheritance* untuk pengukuran

reusability dan *effectiveness.* Dalam kelas ini juga terdapat satu operasi yang merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*

4. Kelas *controller Encapsulation* bertugas untuk mengukur poin *encapsulation* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varEncapsulation* yang menyimpan nilai koefisian *encapsulation* untuk pengukuran *reusability* dan *effectiveness.* Dalam kelas ini juga terdapat satu operasi yang merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*
5. Kelas *controller cohesion* bertugas untuk mengukur poin *cohesion* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varCohesion* yang menyimpan nilai koefisian *cohesion* untuk pengukuran *reusability* dan *effectiveness.* Dalam kelas ini juga terdapat satu operasi yang merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*
6. Kelas *controller Testability* bertugas untuk mengukur nilai *testability* dari berkas masukan sesuai persamaan rumus mengukur *testability*. Dalam kelas ini terdapat tiga atribut yaitu *CovT, cov1,* dan *cov2* yang bertugas menyimpan nilai kovarian *testability* dan memiliki satu operasi yaitu *getHasil*. Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*
7. Kelas model *XMLParser* adalah kelas yang menyimpan hasil dari *parsing XML* yaitu elemen-elemen yang dibutuhkan untuk mengukur *reusability.* Dalam kelas ini terdapat empat atribut yaitu DCC, MFA, DAM, dan CAM yang menyimpan nilai dari elemen hasil *parsing*, dan juga atribut valid untuk menyimpan nilai validasi. Di dalam kelas ini terdapat lima operasi yaitu *getCoup, getInher, getEncap, getCoh, Parser,* dan ValidasiXML. Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewPilih* dan *ViewHasil.*
8. Kelas Jframe adalah bawaan dari API Swing. Komponen *Frame* ini digunakan oleh kelas *boundary* yaitu *main, ViewPilih,* dan *ViewHasil.*
9. Kelas *boundary ViewPilih* bertugas menampilkan antarmuka pengguna untuk melakukan pemilihan berkas rancangan perangkat lunak yang akan diukur nilai *testability*nya. Dalam kelas ini terdapat satu konstruktor dan empat operasi yaitu *PilihActionPerformed, ProsesActionPerformed, HomeMouseClicked,* dan *LihatMouseClicked.* Kelas ini memiliki hubungan agregasi dengan kelas *model XMLParser* dan kelas *boundary ReusMain.* Kelas ini menggunakan komponen *Frame, Filechooser, dan Textfield* dari API Swing.
10. Kelas JFileChoosen adalah bawaan dari API Swing. Komponen FileChooser digunakan oleh kelas boundary *ViewPilih*
11. Kelas JTextField adalah bawaan dari API Swing. Komponen TextField digunakan oleh kelas boundary *ViewPilih.*
12. Kelas *boundary ViewHasil* bertugas menampilkan antarmuka pengguna untuk melihat hasil pengukuran *testability* rancangan perangkat lunak. Dalam kelas ini terdapat satu konstruktor dan dua operasi yaitu *PilihMouseClicked* dan

HomeMouseClicked. Kelas ini memiliki hubungan agregasi dengan kelas model *XMLParser,* kelas *Coupling, Inheritance, Encapsulation, Cohesion, Effectiveness, Reusability, Testability* dan kelas *boundary main.* Kelas ini menggunakan komponen *Frame* dari API Swing.

13. Kelas *boundary main* bertugas sebagai antarmuka utama untuk navigasi ke panel-panel lain. Dalam kelas ini terdapat satu konstruktor dan empat operasi yaitu *initComponents*, *PilihMouseClicked*, *LihatMouseClicked*, dan *main.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewPilih* dan *ViewHasil.* Kelas ini menggunakan komponen *Frame* dari API Swing.

5.1.2 Perancangan Algoritme

Pada perancangan algoritme sistem pengukuran nilai *testability* ini penulis mengambil 3 buah sampel operasi utama yaitu operasi Pilih pada kelas *ViewPilih,* operasi *ValidasiXML* pada kelas *XMLParser,* dan operasi *getHasil* pada kelas *Testability.*

5.1.2.1 Kelas ViewPilih

Rancangan algoritme operasi *PilihActionPerformed* ditunjukkan dalam Kode Program 5.1.

No	ViewPilih.java
1	Private void PilihActionPerformed(evt: ActionEvent)
2	String[] splitAlamat
3	String alamat
4	IntreturnValue = FileChooser CALL showOpenDialog(this: object)
5	if (returnValue = JFileChooser CALL APPROVE_OPTION)
6	FILENAME = FileChooser CALL getSelectedFile
7	String separator = "\\"
8	splitAlamat = FILENAME replace with separator
9	split by "\\\\"
10	alamat = splitAlamat[0] + "\\"
11	for (int i = 1; i < splitAlamat.length; i++)
12	if (i == splitAlamat.length - 1)
13	alamat = alamat CONCAT(splitAlamat[i])
14	else alamat = alamat concat(splitAlamat[i] + "/")
15	FieldPilih CALL setText(alamat)
16	Else tfChoose.setText("No file Chosen!")
17	END IF

Kode Program 5.1 *Pseudocode* operasi *PilihActionPerformed*

5.1.2.2 Kelas *XMLParser*

Rancangan algoritme operasi ValidasiXML ditunjukkan dalam kode program 5.2.

No	**XMLParser.java**
1	Boolean ValidasiXML (xml: File)
2	DocumentBuilderFactory dbFactory = DocumentBuilderFactory CALL newInstance

```
3   DocumentBuilder dBuilder = dbFactory CALL
    newDocumentBuilder
4   Document doc = dBuilder CALL parse(xml: File)
5   doc CALL getDocumentElement, CALL normalize
6   doc CALL getDocumentElement Nodelist strc = doc CALL
    getElementsByTagName("Project")
7   Node nNode = strc CALL item(0)
8   Element eElemenet = (Element) nNode
9   IF ("simple" != eElement CALL
10  getAttribute(XMLstructure: String)
11      THEN valid = false
12  ELSE valid = true
13  END IF
14  Return valid
```

Kode Program 5.2 *Pseudocode* operasi *ValidasiXML*

5.1.2.3 Kelas *Testability*

Rancangan algoritma operasi *getHasil* ditunjukkan dalam Kode Progam 5.4.

```
no  Testability.java
1   double CovT = 59.524
2   double cov1 = 4.671
3   double cov2 = 0.806
4   double getHasil(Eff: double, reus:double)
5   double testability = 0
6   testability = CovT - (cov1 * Effectiveness) + (cov2
    * Reusability);
7   return testability;
```

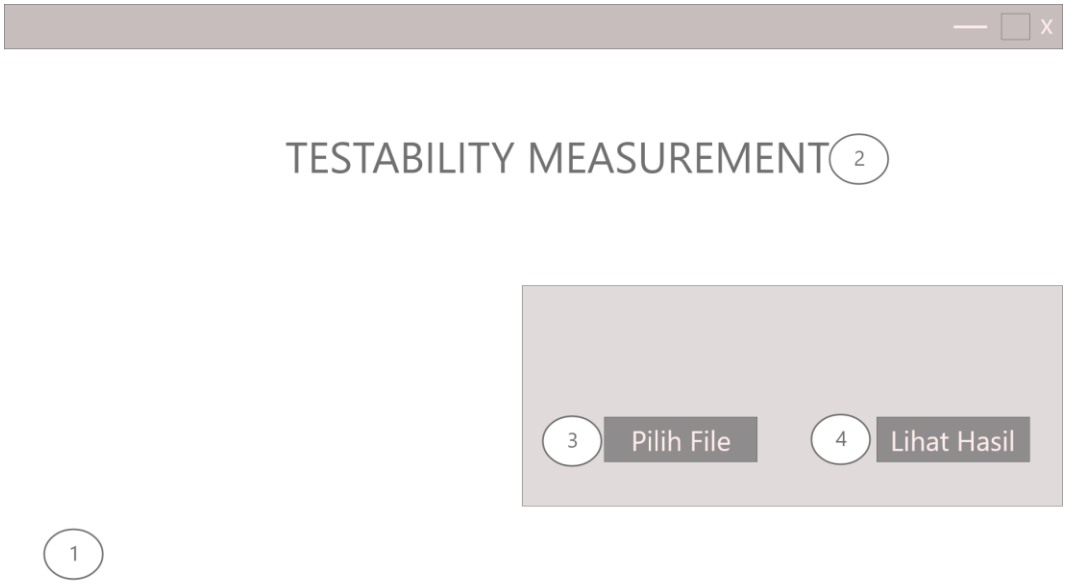
Kode Program 5.3 *Pseudocode* operasi *getHasil*

5.1.3 Perancangan Antarmuka Pengguna

Pada pengembangan sistem pengukuran nilai *testability* ini terdapat empat halaman antarmuka, yaitu, halaman menu awal, halaman cari berkas, halaman lihat hasil, dan halaman lihat indeks *testability.* Berikut adalah perancangan antarmuka pengguna berupa *mock-up* yang telah dirancang dengan menggunakan layout.

5.1.3.1

Rancangan antarmuka pengguna halaman menu awal berupa *mock-up* ditampilkan dalam Gambar 5.3.



Gambar 5.4 *Mock-up* halaman menu utama

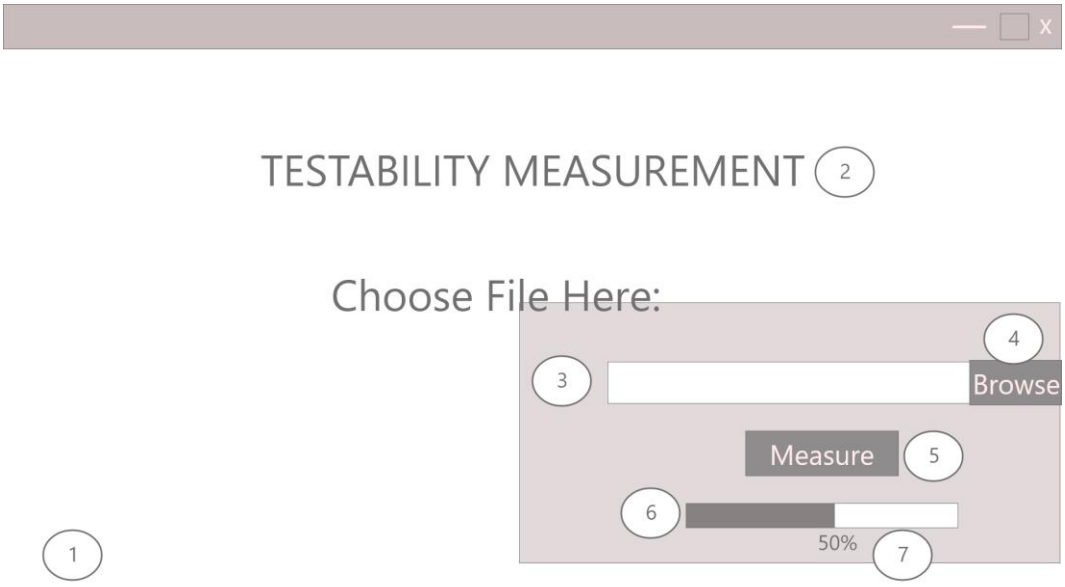
Pada Tabel 5.1 merupakan keterangan komponen yang terdapat dalam halaman utama.

Tabel 5.1 Keterangan *mock-up* halaman pilih file

No	Nama komponen	Tipe komponen	keterangan
1	Frame halaman menu utama	Frame	Kontainer yang digunakan menampung komponen isi pada halaman utama.
2	Judul	Label	Nama sistem
3	Menu 2	Button	Navigasi menuju ke menu pilih file
4	Menu 3	Button	Navigasi ke menu Lihat Hasil

5.1.3.2 Antarmuka Pengguna Halaman Pilih File

Rancangan antarmuka pengguna halaman pilih file berupa mock-up ditampilkan dalam Gambar 5.5.



Gambar 5.5 *Mock-up* halaman pilih file

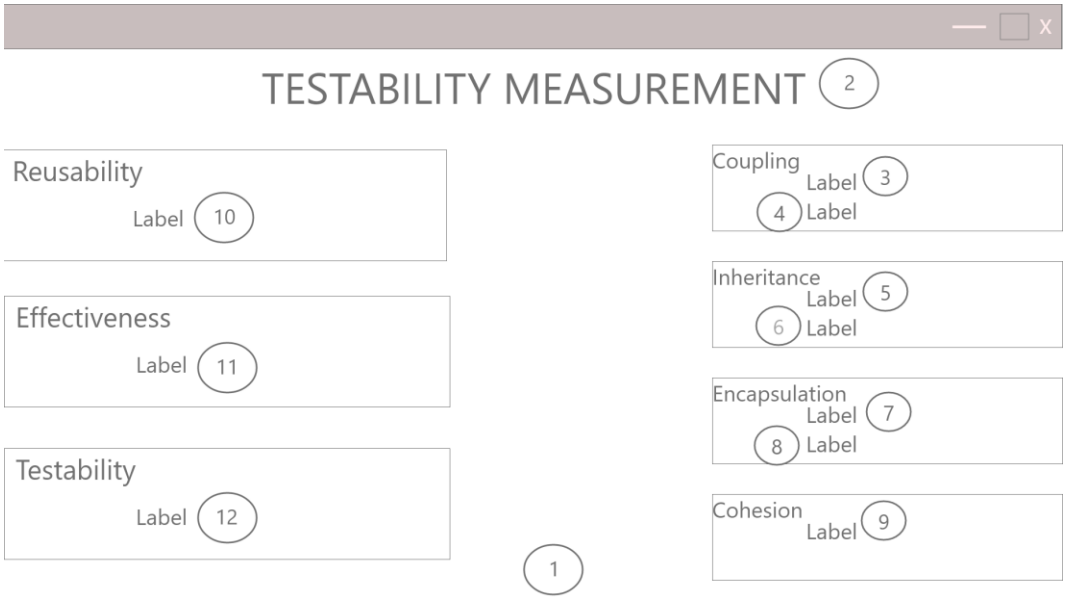
Pada Tabel 5.2 merupakan keterangan komponen-komponen yang harus ada dalam halaman pilih file.

Tabel 5.2 keterangan mock-up halaman pilih file

No	Nama komponen	Tipe komponen	keterangan
1	Frame halaman menu utama	Frame	Kontainer yang digunakan menampung komponen isi pada halaman utama.
2	Judul	Label	Nama sistem dan navigasi menuju halaman menu utama
3	*Text Field* alamat direktori	*Text Field*	*Field* untuk mengisi alamat direktori berkas yang telah dipilih
4	Tombol *Browse*	Button	Tombol untuk memilih berkas
5	Tombol *measure*	Button	Tombol yang digunakan untuk melakukan pemrosesan berkas masukan
6	*Progress Bar*	*Progress Bar*	Indikator yang menunjukkan *progress loading* perhitungan
7	Angka keterangan progress	*Label*	Indikator yang menunjukkan *progress loading* perhitungan berupa angka

5.1.3.3 Antarmuka pengguna halaman lihat hasil

Rancangan antarmuka pengguna halaman pilih file berupa mock-up ditampilkan dalam Gambar 5.6.



Gambar 5.6 *mock-up halaman lihat hasil*

Pada Tabel 5.3 merupakan keterangan komponen-komponen yang harus ada dalam halaman pilih file.

Tabel 5.3 keterangan mock-up halaman lihat hasil

No	Nama komponen	Tipe komponen	keterangan
1	Frame halaman menu utama	Frame	Kontainer yang digunakan menampung komponen isi pada halaman utama.
2	Judul	Label	Nama sistem dan navigasi menuju halaman menu utama
3	Label	label	Berisikan hasil perhitungan *coupling* nilai *reusability* dari *class diagram*
4	Label	label	Berisikan hasil perhitungan *coupling* nilai *effectiveness* dari *class diagram*
5	Label	Label	Berisikan hasil perhitungan *inheritance* nilai *reusability* dari *class diagram*

Tabel 5.4 keterangan mock-up halaman lihat hasil (lanjutan)

6	Label	Label	Berisikan hasil perhitungan *inheritance* nilai *effectiveness* dari *class diagram*
7	Label	Label	Berisikan hasil perhitungan *encapsulation* nilai *reusability* dari *class diagram*
8	Label	Label	Berisikan hasil perhitungan *encapsulation* nilai *effectiveness* dari *class diagram*
9	Label	Label	Berisikan hasil perhitungan *cohesion* dari *class diagram*
10	Label	Label	Berisikan hasil perhitungan *reusability* dari *class diagram*
11	Label	Label	Berisikan hasil perhitungan effectiveness dari *class diagram*
12	Label	Label	Berisikan hasil perhitungan *testability* dari *class diagram*

5.2 Implementasi Sistem

Pada implementasi sistem dijelaskan mengenai spesifikasi sistem, lingkungan pengembangan, batasan implementasi, kode program, dan hasil implementasi dalam bentuk antarmuka pengguna.

5.2.1 Spesifikasi Sistem

Aplikasi untuk mengukur nilai *testability* dikembangkan sesuai dengan algoritme pada *testability. Testability* berkorelasi dengan *effectiveness* dan *reusability* yang dipengaruhi oleh atribut rancangan perangkat lunak yaitu *coupling, inheritance, cohesion* dan *encapsulation. Metric* yang terlibat dalam persamaan adalah DCC untuk *coupling,* MFA untuk *inheritance,* CAM untuk *cohesion,* dan DAM untuk *encapsulation.* Spesifikasi sistem berdasarkan lingkungan perangkat keras, lingkungan perangkat lunak dan lingkungan sistem operasi dijelaskan berikut ini.

5.2.1.1 Lingkungan Perangkat Keras

Pada pengembangan aplikasi pengukuran *testability* ini menggunakan perangkat keras dengan spesifikasi seperti pada Tabel 5.4.

Tabel 5.5 Lingkungan Perangkat Keras

System Modell	Asus A456U
Processor	Intel Core i5-6200U CPU @ 2.30GHz 240 GHz
Graphic	NVIDIA-GEFORCE 940MX
Memory	4GB
HDD	1TB

5.2.1.2 Lingkungan Perangkat Lunak

Pada pengembangan aplikasi pengukuran *testability* ini menggunakan perangkat lunak dengan spesifikasi seperti pada Tabel 5.5.

Tabel 5.6 Lingkungan perangkat lunak

IDE	Netbeans 8.2
Programming Language	Java
Library/API/Framework	JAXP, IO, AWT, Swing
XML maker	Visual Paradigm 16.0 Community Edition
Modeling Tool	Draw.io

5.2.1.3 Lingkungan Sistem Operasi

Pengembangan aplikasi pengukuran *reusability* ini menggunakan Sistem Operasi Windows 10.

5.2.2 Batasan Implementasi

Berikut ini adalah batasan dalam implementasi sistem pengukuran nilai *testability*:

1. Rancangan perangkat lunak hanya dalam bentuk *class diagram.*
2. Masukan yang diterima adalah dokumen XML dengan struktur *simple* dari Visual Paradigm.
3. Sistem tidak menentukan batasan nilai *testability* yang baik atau buruk, hanya mengukur nilai saja.

5.2.3 Kode Sumber

Berikut ini adalah beberapa contoh kode sumber dari hasil implementasi aplikasi pengukuran *testability.* Contoh yang diuraikan disini meliputi kode sumber dari kelas *ViewPilih,* kelas *XMLParser,* dan *testability.*

Pada Kode Program 5.4 adalah contoh kode sumber dari kelas *ViewPilih:*

No	ViewPilih.java
1	`private void PilihActionPerformed(java.awt.event.ActionEvent evt) {`
2	`String[] splitAlamat;`
3	`String alamat;`
4	`int returnValue = openFileChooser.showOpenDialog(this);`
5	`if (returnValue == JFileChooser.APPROVE_OPTION) {`
6	`FILENAME = openFileChooser.getSelectedFile().toString();`
7	`String separator = "\\";`
8	`splitAlamat = FILENAME.replaceAll(Pattern.quote(separator), "\\\\").split("\\\\");`
9	`alamat = splitAlamat[0] + "\\";`
10	`for (int i = 1; i < splitAlamat.length; i++) {`
11	`if (i == splitAlamat.length - 1) {`
12	`alamat = alamat.concat(splitAlamat[i]);`
13	`} else {`
14	`alamat = alamat.concat(splitAlamat[i] + "/");`
15	`}`
16	`}`
17	`FieldPilih.setText(alamat);`
18	`} else {`
19	`tfChoose.setText("No file Chosen!");`
20	`}`
21	`}`

Kode Program 5.4 Kode sumber kelas *ViewPilih* operasi *PilihActionPerformed*

Pada kode program 5.5 adalah contoh kode sumber dari kelas *XMLParser*

No	XMLParser.java
1	`public boolean ValidasiXML(File xml) throws ParserConfigurationException, SAXException, IOException {`
2	`DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();`
3	`DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();`
4	`Document doc = dBuilder.parse(xml);`
5	`doc.getDocumentElement().normalize();`
6	`doc.getDocumentElement();`
7	`NodeList strc=doc.getElementsByTagName("Project"); Node nNode = strc.item(0);`
8	`Element eElement = (Element) nNode;`
9	`if(!"simple".equalsIgnoreCase`
10	`(eElement.getAttribute("Xml_structure"))|| strc==null) {`
	`valid = false;`
	`} else {`
11	`valid = true;`
12	`}`
13	`return valid;`

Kode Program 5.5 Kode sumber kelas *XMLParser* operasi *ValidasiXML*

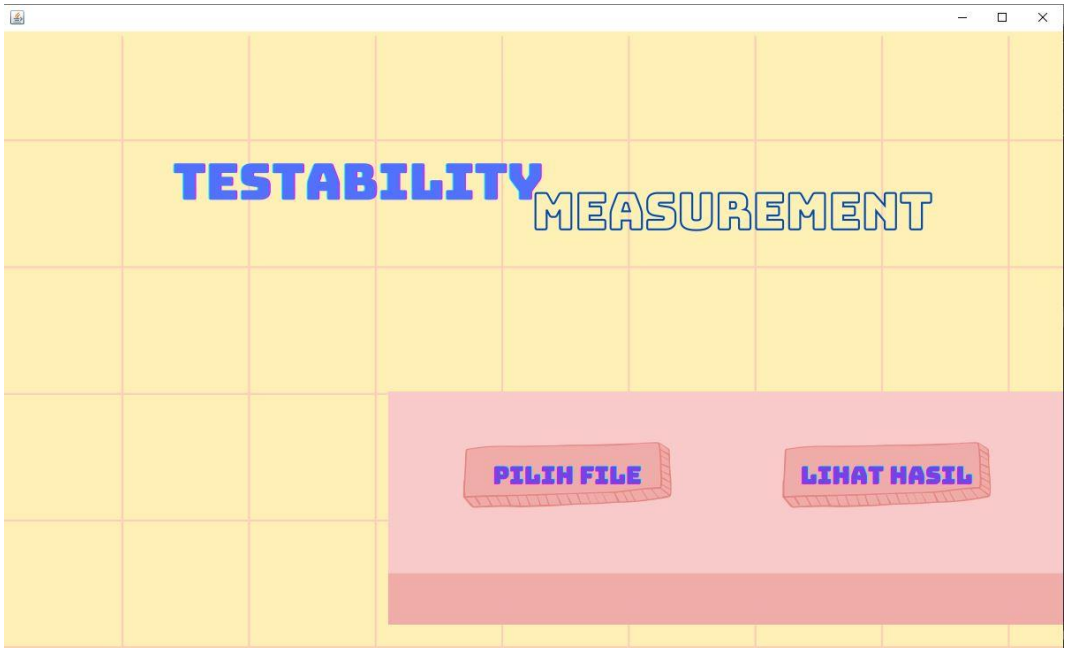
Pada Kode Program 5.6 adalah contoh kode sumber dari kelas *Testability:*

No	Testability.java
1	public class Testability {
2	double CovT = 59.524;
3	double cov1 = 4.671;
4	double cov2 = 0.806;
5	public double getHasil(double Effectiveness, double Reusability) {
6	double testability = 0;
7	testability = CovT - (cov1 * Effectiveness) + (cov2 * Reusability);
8	return testability;
9	}
10	}

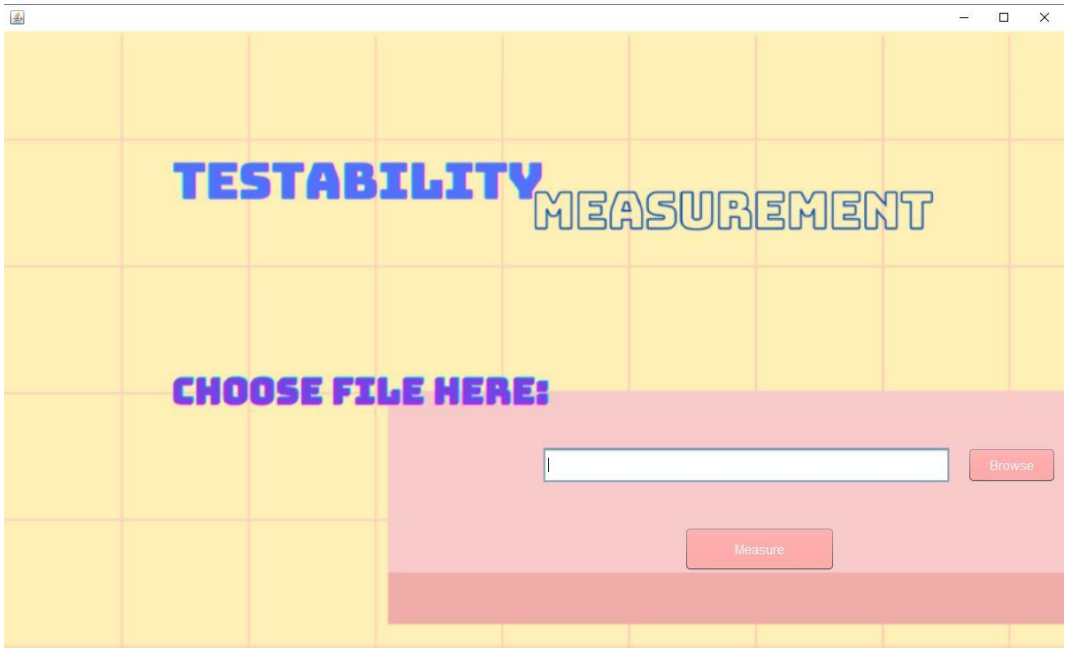
Kode Program 5.6 Kode sumber kelas *Testability* operasi *getHasil*

5.2.4 Hasil Implementasi

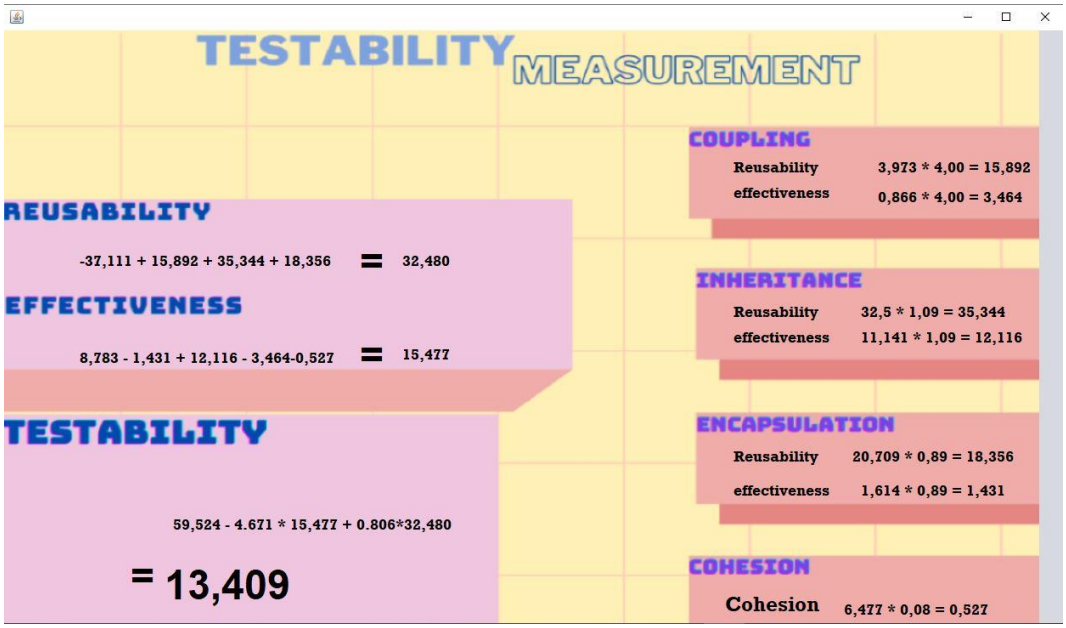
Hasil dari implementasi aplikasi pengukuran *testability* dan bagaimana kebutuhan fungsional seperti cari berkas, ukur *testability,* ukur *effectiveness,* ukur ukur *reusability,* dan lihat hasil ditampilkan pada bagian ini. Gambar 5.7 merupakan halaman awal ketika aplikasi dibuka. Gambar 5.8 merupakan halaman seleksi berkas dan mulai pengukuran yang merupakan hasil dari implementasi kebutuhan *browse* berkas, dan proses perhitungan. Gambar 5.9 adalah halaman lihat hasil yang merupakan hasil dari implementasi kebutuhan lihat hasil. Pada halaman tersebut menampilkan hasil dari pengukuran *testability* beserta dengan rincian pengukurannya.



Gambar 5.7 Tampilan halaman awal aplikasi



Gambar 5.8 Tampilan halaman pilih file



Gambar 5.9 Tampilan halaman lihat hasil

BAB 6 PENGUJIAN SISTEM DAN PEMBAHASAN HASIL

6.1 Pengujian Unit

Pengujian unit pada bagian ini menggunakan metode *white box testing.* Pengujian unit diambil tiga buah operasi antara lain operasi *PilihActionPerformed* dari kelas *ViewPilih, ValidasiXML* dari kelas *XMLParser,* dan *getHasil* dari kelas *Testability*

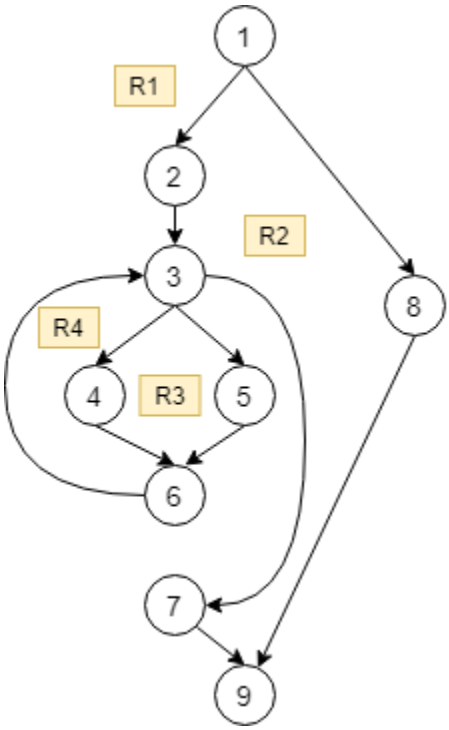
6.1.1 Pengujian Unit Operasi *PilihActionPerformed*

Identifikasi jalur kasus uji untuk operasi *PilihActionPerformed* dari kelas *ViewPilih* seperti dalam Tabel 6.1

Tabel 6.1 Algoritme operasi *PilihActionPerformed*

```
1   Private void PilihActionPerformed(evt: ActionEvent)
1   String[] splitAlamat
1   String alamat
1   IntreturnValue = FileChooser CALL showOpenDialog(this:
1   object)
2   if (returnValue = JFileChooser CALL APPROVE_OPTION)
2         FILENAME = FileChooser CALL getSelectedFile
2         String separator = "\\"
2         splitAlamat = FILENAME replace with separator
2         split by "\\\\"
2         alamat = splitAlamat[0] + "\\"
3         for (int i = 1; i < splitAlamat.length; i++)
4               if (i == splitAlamat.length - 1){
4                     alamat = alamat CONCAT(splitAlamat[i])
5               else alamat = alamat CONCAT(splitAlamat[i] +
5               "/")
6               END IF
7         FieldPilih CALL setText(alamat)
8   Else tfChoose.setText("No file Chosen!")
9   END IF
```

Berdasarkan Tabel 6.1 didapatkan *flow graph* untuk operasi *PilihActionPerformed* seperti dalam Gambar 6.1



Gambar 6.1 *Flow graph* operasi *PilihActionPerformed*

Berdasarkan Gambar 6.1 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 4

b. V(G) = E – N + 2 = 11-9+2 = 4

c. V(G) = 3 + 1 = 3+1 = 4

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

a. Jalur 1 = 1-8-9

b. Jalur 2 = 1-2-3-4-6-7-9

c. Jalur 3 = 1-2-3-5-6-7-9

d. Jalur 4 = 1-2-3-7-9

Dari *independent path* yang telah didapat, kasus uji untuk operasi *PilihActionPerformed* adalah seperti pada Tabel 6.2

Tabel 6.2 Hasil Pengujian unit operasi PilihActionPerformed

Jalur	Prosedur Uji	Ekspektasi Hasil	Hasil Aktual	Status
1	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=0	tfChoose = null	tfChoose = null	Valid
2	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=1, FILENAME= "C:\\Users/Asus/Desktop/project1.xml"	Alamat direktori berkas dipecah kemudian digabung kembali dan pada kata terakhir tidak ditambah “/”	Alamat direktori berkas dipecah kemudian digabung kembali dan pada kata terakhir tidak ditambah “/”	Valid
3	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=1, FILENAME= "C:\\Users/Asus/Desktop/project1.xml"	Alamat direktori berkas dipecah kemudian digabung kembali dengan karakter penghubung “/”	Alamat direktori berkas dipecah kemudian digabung kembali dengan karakter penghubung “/”	Valid
4	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=1, FILENAME= "C:\\Users/Asus/Desktop/project1.xml"	FieldPilih = “C:\\Users/Asus/Desktop/project1.xml”	FieldPilih = “C:\\Users/Asus/Desktop/project1.xml”	Valid

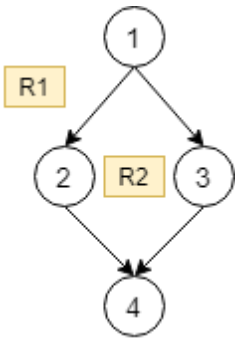
6.1.2 Pengujian Unit Operasi *ValidasiXML*

Identifikasi jalur kasus uji untuk operasi *ValidasiXML* dari kelas *XMLParser* seperti dalam Tabel 6.3.

Tabel 6.3 Algoritme operasi *ValidasiXML*

1	Boolean ValidasiXML (xml: File)
1	DocumentBuilderFactory dbFactory =
1	DocumentBuilderFactory CALL newInstance
1	DocumentBuilder dBuilder = dbFactory CALL
1	newDocumentBuilder
1	Document doc = dBuilder CALL parse(xml: File)
1	doc CALL getDocumentElement, CALL normalize
1	doc CALL getDocumentElement *Node*list strc = doc CALL
1	getElementsByTagName("Project")
1	*Node nNode = strc CALL item(0)*
1	Element eElemenet = (Element) n*Node*
2	IF ("simple" != eElement CALL getAttribute(XMLstructure:
2	String)
2	THEN valid = false
3	ELSE valid = true
4	END IF
4	Return valid

Berdasarkan gambar 6.3 didapatkan *flow graph* untuk operasi *ValidasiXML* seperti dalam Gambar 6.2.



Gambar 6.2 *Flow graph* operasi *ValidasiXML*

Berdasarkan Gambar 6.4 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 2

b. V(G) = E – N + 2 = 4-4+2 = 2

c. V(G) = P + 1 = 1+1 = 2

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

a. Jalur 1 = 1-2-4

b. Jalur 2 = 1-3-4

Dari *indepent path* yang telah didapat, kasus uji untuk operasi *ValidasiXML* adalah seperti pada Tabel 6.4.

Tabel 6.4 Hasil pengujian unit operasi *ValidasiXML*

Jalur	Prosedur Uji	Ekspektasi Hasil	Hasil Aktual	Status
1	Kelas driver memanggil method ValidasiXML() dengan variabel doc = "C:\\Users/Asus/Desktop/project1.xml"	valid = false	valid = false	Valid
2	Kelas driver memanggil method ValidasiXML() dengan variabel doc = "C:\\Users/Asus/Desktop/project1.xmll"	valid = true	valid = true	Valid

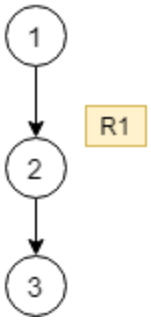
6.1.3 Pengujian Unit Operasi *getHasil*

Identifikasi jalur kasus uji untuk operasi *getHasil* dari kelas *testability* seperti dalam Tabel 6.5.

Tabel 6.5 Algoritme operasi *getHasil*

	`double CovT = 59.524`
	`double cov1 = 4.671`
	`double cov2 = 0.806`
	`double getHasil(Eff: double, reus:double)`
1	`double testability = 0`
2	`testability = CovT - (cov1 * Effectiveness) + (cov2 * Reusability);`
3	`return testability;`

Berdasarkan Tabel 6.5 didapatkan *flow graph* untuk operasi *getHasil* seperti dalam Gambar 6.3



Gambar 6.3 *Flow Graph* Operasi *getHasil*

Berdasarkan Gambar 6.6 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 1
b. V(G) = E – N + 2 = 1–2+2 =1
c. V(G) = P + 1 = 0+1 = 1

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

a. Jalur 1 = 1-2

Dari *indepent path* yang telah didapat, kasus uji untuk operasi *getHasil* adalah seperti pada Tabel 6.6

Tabel 6.6 Hasil pengujian unit operasi *getHasil*

Jalur	**Prosedur Uji**	**Ekspektasi Hasil**	**Hasil Aktual**	**Status**
1	Kelas driver memanggil method getHasil() dengan argumen effectiveness = 12 dan reusability = 8	Testability = 9,919	Testability = 9,919	Valid

6.2 Pengujian Integrasi

Pengujian integrasi adalah pengujian dengan menggabungkan unit-unit untuk membentuk satu fungsional. Pengujian integrasi pada penulisan ini dilakukan dengan cara menguji hubungan antar kelas menggunakan pendekatan *top-down* yang meliputi kelas *Reusability dan* kelas *effectiveness* untuk menjalankan fungsi ukur *testability.* Pada Tabel 6.7 menunjukkan identifikasi kelas/modul yang diuji.

Tabel 6.7 Identifikasi kelas pengujian integrasi

Operasi	Kelas	Tujuan
getHasil (double Eff, double reus)	*Testability*	Mengukur nilai *Testability* berdasarkan masukan pengguna
getHasil (double coh, double coup, double inher, double encap)	Effectiveness	
getHasil(double Coup, double Inher, double Encap)	Reusability	
getHasil (double cohesion)	Cohesion	

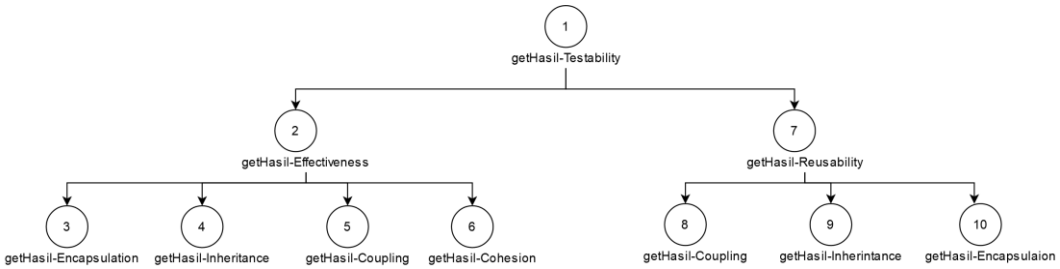
Tabel 6.7 Identifikasi kelas pengujian integrasi (lanjutan)

getHasil (double coupling)	Coupling	
getHasil (double inheritance	Inheritance	
getHasil (double encapsulation)	Encapsulation	

Pada Tabel 6.7 maka dapat ditunjukkan model pengujian integrasi dalam Gambar 6.4. Berdasarkan model *top-down* pada Gambar 6.4 menunjukan hubungan antara *node*1 ke *node*2 dan *node7.* Pada hubungan *node*1, *node*2, *node*3 *operasi getHasil* dari *Coupling* akan memberikan hasil poin *coupling* yang menjadi masukan parameter *Coup* dari operasi *getHasil* pada kelas *Effectiveness*. Kemudian hubungan antara *node*1, node2 dan *node*3 ke *node*4 pada operasi *getHasil* pada *cohesion* akan memberikan hasil *cohesion* yang menjadi masukan parameter *coh* dari operasi *getHasil* pada kelas *Effectiveness*. Lalu hubungan *node*1, node2, node3, *node*4, ke *node*5 pada operasi *getHasil* pada *Encapsulation* akan memberikan hasil poin *encapsulation* yang menjadi masukan parameter *Encap* dari operasi *getHasil* pada kelas *Effectiveness*. Dan hubungan *node*1, node2, node3, *node*4, *node*5 ke *node*6 pada operasi *getHasil* pada *inheritance* akan memberikan hasil poin *inheritance* yang menjadi masukan parameter *Inher* dari operasi *getHasil* pada kelas *Effectiveness*.

Untuk pengujian integrasi pada *Reusability* maka ditunjukkan dengan hubungan antara hubungan *node*1, *node7*, *node*8 *operasi getHasil* dari *Coupling* akan memberikan hasil poin *coupling* yang menjadi masukan parameter *Coup* dari operasi *getHasil* pada kelas *Reusability*. Lalu hubungan *node*1, node7 dan node8 ke *node*9 pada operasi *getHasil* pada *Encapsulation* akan memberikan hasil poin *encapsulation* yang menjadi masukan parameter *Encap* dari operasi *getHasil* pada kelas *Reusability*. Dan hubungan *node*1, node7, node8 dan *node*9 ke *node*6 pada operasi *getHasil* pada *inheritance* akan memberikan hasil poin *inheritance* yang menjadi masukan parameter *Inher* dari operasi *getHasil* pada kelas *Reusability*.

Maka didapatkan langkah pengujian integrasi nilai *Testability* seperti pada Tabel 6.7.



Gambar 6.4 Model *Top-Down Testing*

Tabel 6.8 Pengujian Integrasi

No	Langkah Pengujian	Keterangan
1	Node1 + node2	*Operasi getHasil*(*double Effectiveness)* pada kelas *Effectiveness* dipanggil pada kelas *Testability*
2	Node1 + node 2 + node 3	*Operasi getHasil*(*double encapsulation)* pada kelas *Encapsulation* dipanggil operasi *getHasil(double EncapEf, double InherEf, double CoupEf, double Coh)* pada kelas *Effectiveness*
3	Node1 + node 2 + node 3 + node 4	*Operasi getHasil* (*double inheritance)* pada kelas *inheritance* dipanggil operasi *getHasil(double EncapEf, double InherEf, double CoupEf, double Coh)* pada kelas *Effectiveness* yang telah terdapat nilai *getHasil(double encapsulation)* pada kelas *Encapsulation*
4	Node1 + node 2 + node 3 + node 4 + node 5	*Operasi getHasil* (*double coupling)* pada kelas *coupling* dipanggil operasi *getHasil(double EncapEf, double InherEf, double CoupEf, double Coh)* pada kelas *Effectiveness* yang telah terdapat nilai *getHasil(double inher)* pada kelas *Cohesion*
5	Node1 + node 2 + node 3 + node 4 + node 5 + node 6	*Operasi getHasil* (*double inheritance)* pada kelas *inheritance* dipanggil operasi *getHasil*(*double Coup, double Coh, double Inher, double Encap*) pada kelas *Effectiveness* yang telah terdapat nilai *getHasil(double inher)* pada kelas *Encapsulation*
6	Node 1 + node 7	*Operasi getHasil*(*double Reusability)* pada kelas *Reusability* dipanggil pada kelas *Testability*
7	Node 1 + node 7 + node 8	*Operasi getHasil*(*double coupling)* pada kelas *Coupling* dipanggil operasi *getHasil*(*double Coup, double Inher, double Encap*) pada kelas *Reusability*
8	Node 1 + node 7 + node 8 + node 9	*Operasi getHasil* (*double encapsulation)* pada kelas *encapsultaion* dipanggil

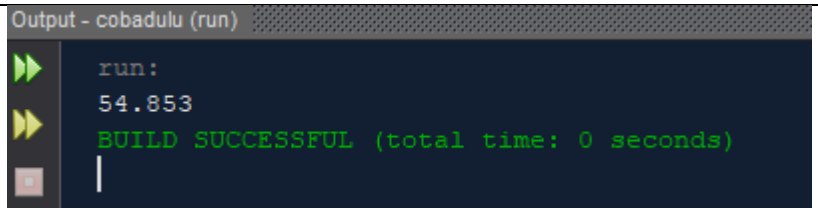
Tabel 6.8 Pengujian Integrasi (lanjutan)

		operasi *getHasil*(*double Coup, double Inher, double Encap*) pada kelas *Reusability* yang telah terdapat nilai *getHasil(double coup)* pada kelas *Coupling*
9	Node 1 + node 7 + node 8 + node 9 + node 10	*Operasi getHasil* (*double inheritance)* pada kelas *inheritance* dipanggil operasi *getHasil*(*double Coup, double Inher, double Encap*) pada kelas *Reusability* yang telah terdapat nilai *getHasil(double encap)* pada kelas *Encapsulation*

Setelah langkah pengujian integrasi dilakukan seperti yang dijelaskan pada Tabel 6.8, maka dibuat stub pengganti dalam operasi getHasil dari kelas *Testability.* Tabel 6.9 akan menjelaskan penggunaan stub guna menjalankan langkah 1.

Tabel 6.9 Langkah 1 Pengujian Integrasi

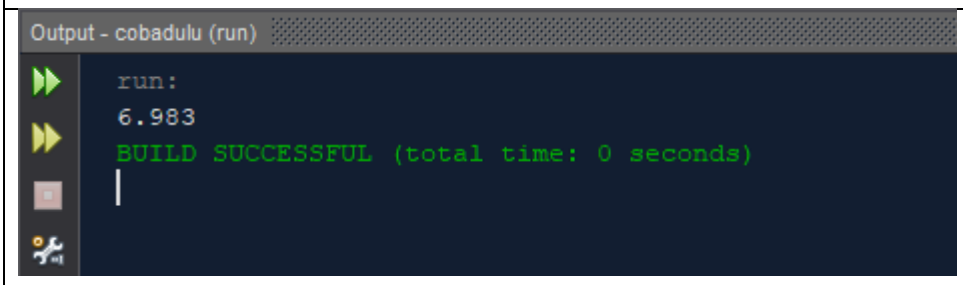
```
public class Testability {
      double CovT = 59.524;
      double cov1 = 4.671;
      double cov2 = 0.806;
      public double getHasil(double Effectiveness, double
      Reusability) {
        double testability = 0;
            testability = CovT - (cov1 * Effectiveness) + (cov2
            * Reusability);
              return testability;
      }
}
public class integration_testing {
    public static void main(String[] args) {
      Testability t = new Testability();
        double stubEf = 1; // stub untuk mencoba langkah 1
      System.out.println(t.getHasil(stubEf,0));
    }
}
```



Penggunaan *stubEf* atau stub kelas *Effectiveness* dapat menghitung hasil integrasi pada langkah 1 yang ditunjukkan pada Tabel 6.9 dengan nilai 54.853 atau tidak 0 pada *getHasil Testability*. Langkah 2 akan ditunjukkan pada tabel 6.10 dengan penggunaan *stubEncap* pada kelas *Effectiveness.*

Tabel 6.10 Langkah 2 Pengujian Integrasi

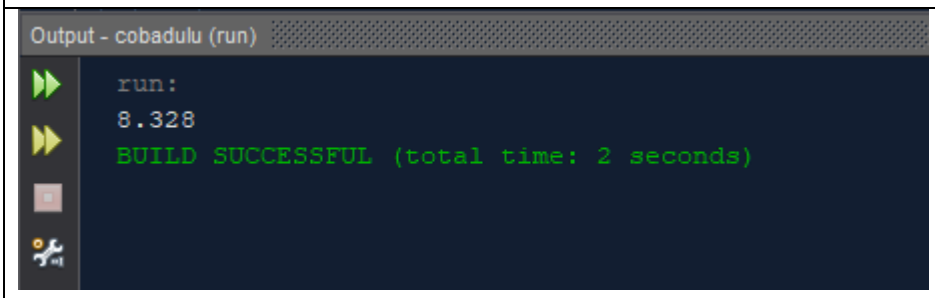
```
public class Effectiveness {
        double CovEf = 8.783;
 public double getHasil(double CoupEf, double InherEf, double
EncapEf, double Coh) {
        double effectiveness = 0;
        effectiveness = CovEf - EncapEf + InherEf - CoupEf - Coh;
        return effectiveness;
}
}
public static void main(String[] args) {
                Effectiveness ef = new Effectiveness();
            double stubEncap = 1.8; // stub untuk mencoba langkah
            2.
            System.out.println(ef.getHasil(stubEncap,  0,0, 0));
```



Penggunaan *stubEncap* atau stub kelas *Encapsulation* dapat menghitung hasil integrasi pada langkah 2 yang ditunjukkan pada Tabel 6.10 dengan nilai 6.983 atau tidak 0 pada *getHasil Effectiveness*. Langkah 3 akan ditunjukkan pada tabel 6.11 dengan penggunaan *stubInher* pada kelas *Effectiveness.*

Tabel 6.11 Langkah 3 Pengujian Integrasi

```
public class Effectiveness {
        double CovEf = 8.783;
 public double getHasil(double CoupEf, double InherEf, double
EncapEf, double Coh) {
        double effectiveness = 0;
        effectiveness = CovEf - EncapEf + InherEf - CoupEf - Coh;
        return effectiveness;
}
}
public static void main(String[] args) {
                Effectiveness ef = new Effectiveness();
            double stubEncap = 1.8;
            double stubInher = 1.345; // stub untuk mencoba
            langkah 3
System.out.println(ef.getHasil(stubEncap, stubInher,0, 0));
```



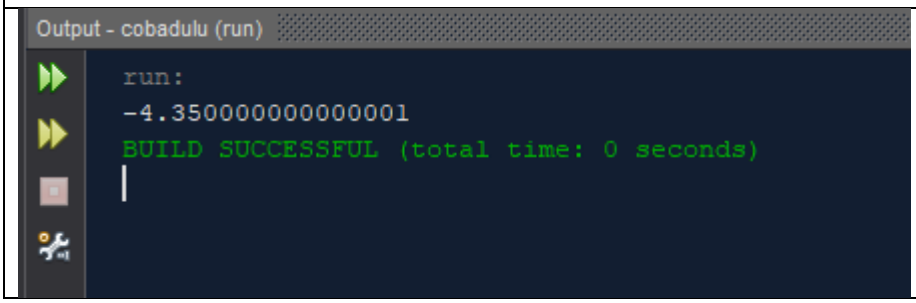
Penggunaan *stubInher* atau stub kelas *inheritance* dengan *stubEncap* dapat menghitung hasil integrasi pada langkah 3 yang ditunjukkan pada Tabel 6.11 dengan nilai 8.328 atau tidak 0 pada *getHasil Effectiveness*. Langkah 4 akan ditunjukkan pada tabel 6.12 dengan penggunaan *stubCoup* pada kelas *Effectiveness.*

Tabel 6.12 Langkah 4 Pengujian Integrasi

```
public class Effectiveness {
        double CovEf = 8.783;
 public double getHasil(double CoupEf, double InherEf, double
EncapEf, double Coh) {
        double effectiveness = 0;
        effectiveness = CovEf - EncapEf + InherEf - CoupEf - Coh;
        return effectiveness;
}
}
public static void main(String[] args) {
```

Tabel 6.12 Langkah 4 Pengujian Integrasi (lanjutan)

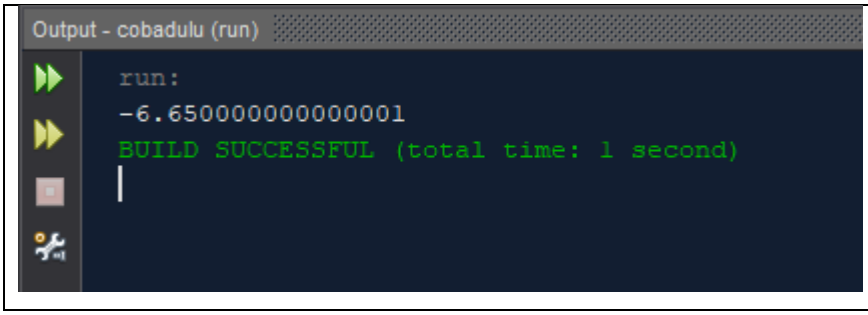
```
Effectiveness ef = new Effectiveness();
            double stubEncap = 1.8;
            double stubInher = 1.345;
            double stubCoup = 12.678; //stub untuk mencoba langkah 4

System.out.println(ef.getHasil(stubEncap,    stubInher,stubCoup,
0));
```



Penggunaan *stubCoup* atau stub kelas *Coupling* dengan *stubEncap* dan *stubInher* dapat menghitung hasil integrasi pada langkah 4 yang ditunjukkan pada Tabel 6.12 dengan nilai -4.35 atau tidak 0 pada *getHasil Effectiveness*. Langkah 5 akan ditunjukkan pada tabel 6.13 dengan penggunaan *stubCoh* pada kelas *Effectiveness.*

Tabel 6.13 Langkah 5 Pengujian Integrasi

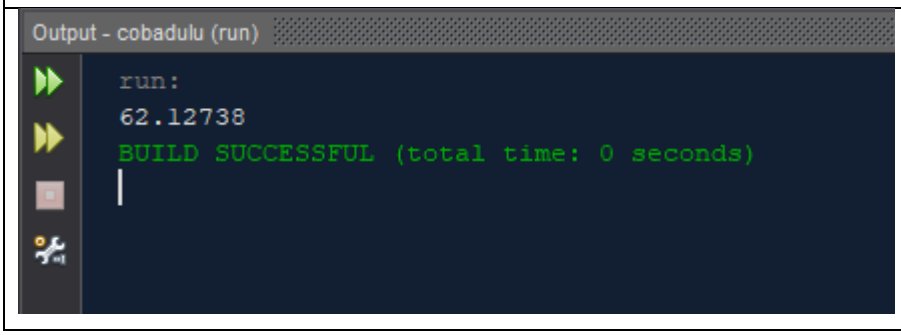
```
public class Effectiveness {
        double CovEf = 8.783;
 public double getHasil(double CoupEf, double InherEf, double
EncapEf, double Coh) {
        double effectiveness = 0;
        effectiveness = CovEf - EncapEf + InherEf - CoupEf - Coh;
        return effectiveness;
}
}
public static void main(String[] args) {
                Effectiveness ef = new Effectiveness();
            double stubEncap = 1.8;
            double stubInher = 1.345;
            double stubCoup = 12.678;
            double stubCoh = 2.3;    //stub untuk mencoba langkah 5
System.out.println(ef.getHasil(stubEncap,    stubInher,stubCoup,
stubCoh));
```



Penggunaan *stubCoh* atau stub kelas *Cohesion* dengan *stubEncap, stubInher, stubCoup* dapat menghitung hasil integrasi pada langkah 5 yang ditunjukkan pada Tabel 6.13 dengan nilai -6.65 atau tidak 0 pada *getHasil Effectiveness*. Langkah 6 akan ditunjukkan pada tabel 6.14 dengan penggunaan *stubR* pada kelas *Testability.*

Tabel 6.14 Langkah 6 Pengujian Integrasi

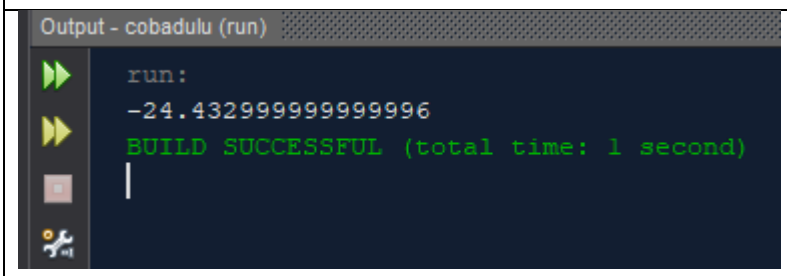
```
public class Testability {
      double CovT = 59.524;
      double cov1 = 4.671;
      double cov2 = 0.806;
      public double getHasil(double Effectiveness, double
      Reusability) {
        double testability = 0;
            testability = CovT - (cov1 * Effectiveness) + (cov2
            * Reusability);
              return testability;
      }
}
public class integration_testing {
    public static void main(String[] args) {
      Testability t = new Testability();
        double stubR = 3.23; // stub untuk mencoba langkah 6
      System.out.println(t.getHasil(stubEf,0));
    }
```



Penggunaan *stubR* atau stub kelas *Reusability* dapat menghitung hasil integrasi pada langkah 6 yang ditunjukkan pada Tabel 6.14 dengan nilai 62.1238 atau tidak 0 pada *getHasil Testability*. Langkah 7 akan ditunjukkan pada tabel 6.15 dengan penggunaan *stubCoup* pada kelas Reusability.

Tabel 6.15 Langkah 7 Pengujian Integrasi

```
public class Reusability {
    double Cov = -37.111;
    public double getHasil(double CoupR, double InherR, double
EncapR) {
        double reusability = 0;
        reusability = Cov + CoupR + InherR + EncapR;
        return reusability;
    }
}
public class integration_testing {
    public static void main(String[] args) {
      Reusability r = new Reusability();
      double stubCoup = 12.678; //stub untuk mencoba langkah 7
            System.out.println(r.getHasil(stubCoup, 0,0));
```



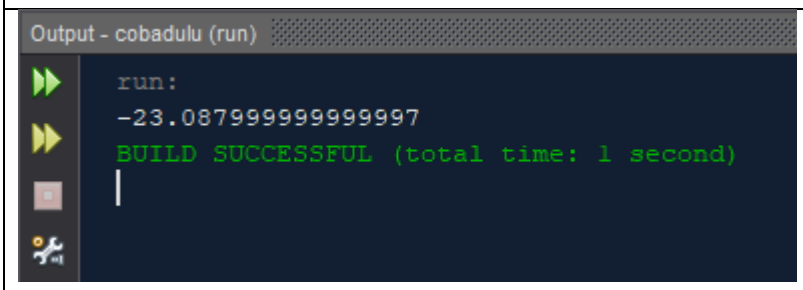
Penggunaan *stubCoup* atau stub kelas *Coupling* dapat menghitung hasil integrasi pada langkah 7 yang ditunjukkan pada Tabel 6.15 dengan nilai -24.433 atau tidak 0 pada *getHasil Effectiveness*. Langkah 8 akan ditunjukkan pada tabel 6.16 dengan penggunaan *stubInher* pada kelas *Reusability.*

Tabel 6.16 Langkah 8 Pengujian Integrasi

```
public class Reusability {
    double Cov = -37.111;
    public double getHasil(double CoupR, double InherR, double
EncapR) {
        double reusability = 0;
        reusability = Cov + CoupR + InherR + EncapR;
        return reusability;
    }
```

Tabel 6.16 Langkah 8 Pengujian Integrasi (lanjutan)

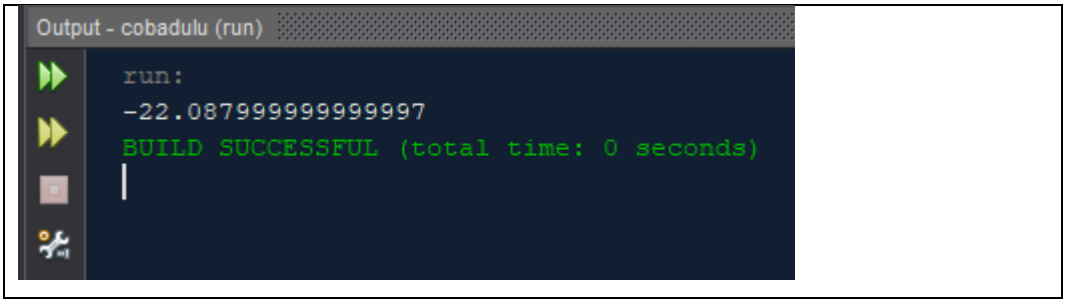
```
}
public class integration_testing {
    public static void main(String[] args) {
      Reusability r = new Reusability();
      double stubCoup = 12.678;
      double stubInher = 1.345; //stub untuk mencoba langkah 8
System.out.println(r.getHasil(stubCoup, stubInher,0));
```



Penggunaan *stubInher* atau stub kelas *inheritance* dengan *stubCoup* dapat menghitung hasil integrasi pada langkah 8 yang ditunjukkan pada Tabel 6.16 dengan nilai -23.088 atau tidak 0 pada *getHasil Effectiveness*. Langkah 9 akan ditunjukkan pada tabel 6.17 dengan penggunaan *stubEncap* pada kelas *Reusability.*

Tabel 6.17 Langkah 9 Pengujian Integrasi

```
public class Reusability {
    double Cov = -37.111;
    public double getHasil(double CoupR, double InherR, double
EncapR) {
        double reusability = 0;
        reusability = Cov + CoupR + InherR + EncapR;
        return reusability;
    }
}
public class integration_testing {
    public static void main(String[] args) {
      Reusability r = new Reusability();
      double stubCoup = 12.678;
      double stubInher = 1.345; //stub untuk mencoba langkah 8
      double stubEncap = 1; //stub untuk mencoba langkah 9
System.out.println(r.getHasil(stubCoup, stubInher,stubEncap));
```



Penggunaan *stubEncap* atau stub kelas *encapsulation* dengan *stubCoup* dan *stubInher* dapat menghitung hasil integrasi pada langkah 9 yang ditunjukkan pada Tabel 6.17 dengan nilai -22.088 atau tidak 0 pada *getHasil Effectiveness*.

6.3 Pengujian Validasi

Pengujian validasi pada bagian ini menggunakan *black box testing* untuk menguji apakah sistem yang dibangun telah memenuhi kebutuhan fungsional dan non-fungsional. Pengujian validasi untuk kebutuhan fungsional terdapat 5 kebutuhan antara lain cari berkas, ukur *Testability,* ukur *Effectiveness,* ukur *reusability,* dan lihat hasil. Sedangkan pengujian validasi untuk kebutuhan non-fungsional terdapat satu kebutuhan yaitu efisiensi.

6.3.1 Pengujian Validasi Cari Berkas

Pengujian validasi kebutuhan cari berkas ditunjukkan pada Tabel 6.7 dan Tabel 6.8. Tabel 6.7 adalah hasil pengujian validasi dari *main flow kebutuhan Browse* Berkas dan Tabel 6.8 adalah hasil pengujian validasi dari *alternative flow* kebutuhan cari berkas.

Tabel 6.18 Pengujian validasi *main flow browse* berkas

Kode kebutuhan	F_TEST_001
Kasus uji	Melakukan Cari Berkas
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional cari Berkas
Data masukan	Berkas XML
Prosedur	1. Membuka halaman pilih file 2. Menekan pada tombol *Browse* 3. Memilih sebuah berkas XML 4. Menekan tombol *Open*
Hasil yang diharapkan	Sistem mengisi *text field* dengan alamat direktori berkas yang dipilih
Hasil	Sistem mengisi *text field* dengan alamat direktori berkas yang dipilih
Status	Valid

Tabel 6.19 Pengujian validasi *alternative flow browse* berkas

Kode kebutuhan	F_TEST_001
Kasus uji	Melakukan Cari Berkas
Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional Cari Berkas
Data masukan	-
Prosedur	1. Membuka halaman pilih file 2. Menekan pada tombol *Browse* 3. Memilih sebuah berkas XML 4. Menekan tombol *Cancel*
Hasil yang diharapkan	*Text field* bernilai kosong
Hasil	*Text field* bernilai kosong
Status	Valid

6.3.2 Pengujian Validasi Ukur *Testability*

Pengujian validasi kebutuhan Ukur *Testability* ditunjukan pada Tabel 6.9 dan Tabel 6.10. Pada Tabel 6.9 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur *Testability* dan pada Tabel 6.10 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur *Testability*.

Tabel 6.20 Pengujian validasi *main flow* ukur *testability*

Kode kebutuhan	F_TEST_002
Kasus uji	Mengukur *Testability*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur *Testability*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur *testability* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur *testability* berkas XML *class diagram* perangkat lunak
Status	Valid

Tabel 6.21 Pengujian Validasi *Alternative Flow* Ukur *testability*

Kode kebutuhan	F_TEST_002
Kasus uji	Mengukur *Testability*
Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional Ukur *Testability*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Hasil	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Status	Valid

6.3.3 Pengujian Validasi Ukur *Effectiveness*

Pengujian validasi kebutuhan Ukur *Effectiveness* ditunjukan pada Tabel 6.11 dan Tabel 6.12. Pada Tabel 6.11 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur *Effectiveness* dan pada Tabel 6.12 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur *Effectiveness*.

Tabel 6.22 Pengujian validasi *main flow* ukur *effectiveness*

Kode kebutuhan	F_TEST_003
Kasus uji	Mengukur *Effectiveness*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur *Effectiveness*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open*

	5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur *effectiveness* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur *effectiveness* berkas XML *class diagram* perangkat lunak
Status	Valid

Tabel 6.23 Pengujian Validasi *Alternative Flow* Ukur *effectiveness*

Kode kebutuhan	F_TEST_003
Kasus uji	Mengukur *Effectiveness*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur *Effectiveness*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur *effectiveness* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur *effectiveness* berkas XML *class diagram* perangkat lunak
Status	Valid

6.3.4 Pengujian Validasi Ukur *Reusability*

Pengujian validasi kebutuhan Ukur *Reusability* ditunjukan pada Tabel 6.13 dan Tabel 6.14. Pada Tabel 6.13 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur *Reusabilityy* dan pada Tabel 6.14 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur *Reusability*.

Tabel 6.24 Pengujian validasi *main flow* ukur *reusability*

Kode kebutuhan	F_TEST_004
Kasus uji	Mengukur *Reusability*

Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur *Reusability*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur *reusability* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur *reusability* berkas XML *class diagram* perangkat lunak
Status	Valid

Tabel 6.25 Pengujian Validasi *Alternative Flow* Ukur *Reusability*

Kode kebutuhan	F_TEST_003
Kasus uji	Mengukur *Reusability*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur *Reusability*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur *reusability* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur *reusability* berkas XML *class diagram* perangkat lunak
Status	Valid

6.3.5 Pengujian Validasi Lihat Hasil

Berikut ini merupakan pengujian validasi untuk kebutuhan Lihat Hasil. Pada Tabel 6.15 merupakan hasil pengujian validasi dari *main flow* kebutuhan Lihat Hasil.

Tabel 6.26 Pengujian validasi *main flow* lihat hasil

Kode kebutuhan	F_TEST_005
Kasus uji	Melihat Hasil Pengukuran
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Lihat Hasil
Data masukan	-
Prosedur	1. Menekan tombol navigasi yang menuju ke halaman Lihat Hasil
Hasil yang diharapkan	Sistem menampilkan halaman Lihat Hasil yang berisi hasil dari pengukuran *Testability* beserta rinciannya
Hasil	Sistem menampilkan halaman Lihat Hasil yang berisi hasil dari pengukuran *testability* beserta rinciannya
Status	Valid

6.3.6 Pengujian Validasi Efisiensi Sistem

Pengujian efisiensi sistem dilakukan untuk mengetahui perbandingan waktu perhitungan *reusability* rancangan perangkat lunak dengan menggunakan sistem dan perhitungan manual. Pada pengujian ini dilakukan oleh seorang pengguna menggunakan empat berkas uji dengan jumlah kelas yang berbeda-beda. Berkas uji yang digunakan sebagai masukan adalah dari proyek perangkat lunak *open source.* Pada Tabel 6.27 adalah hasil dari pengujian efisiensi sistem.

Tabel 6.27 Hasil Pengujian Efisiensi Sistem

Tugas	Jumlah kelas	*Computation Time System* (dalam detik)	Perhitungan Manual (dalam detik)
Mengukur *Testability* berkas uji I	7	1	101
Mengukur *Testability* berkas uji II	10	2	189
Mengukur *Testability* berkas uji III	22	3	247
Mengukur *Testability* berkas uji IV	39	2	337
Time based efficiency		0,583 tugas/detik	0,0055 tugas/detik
Overall relative efficiency		100%	100%

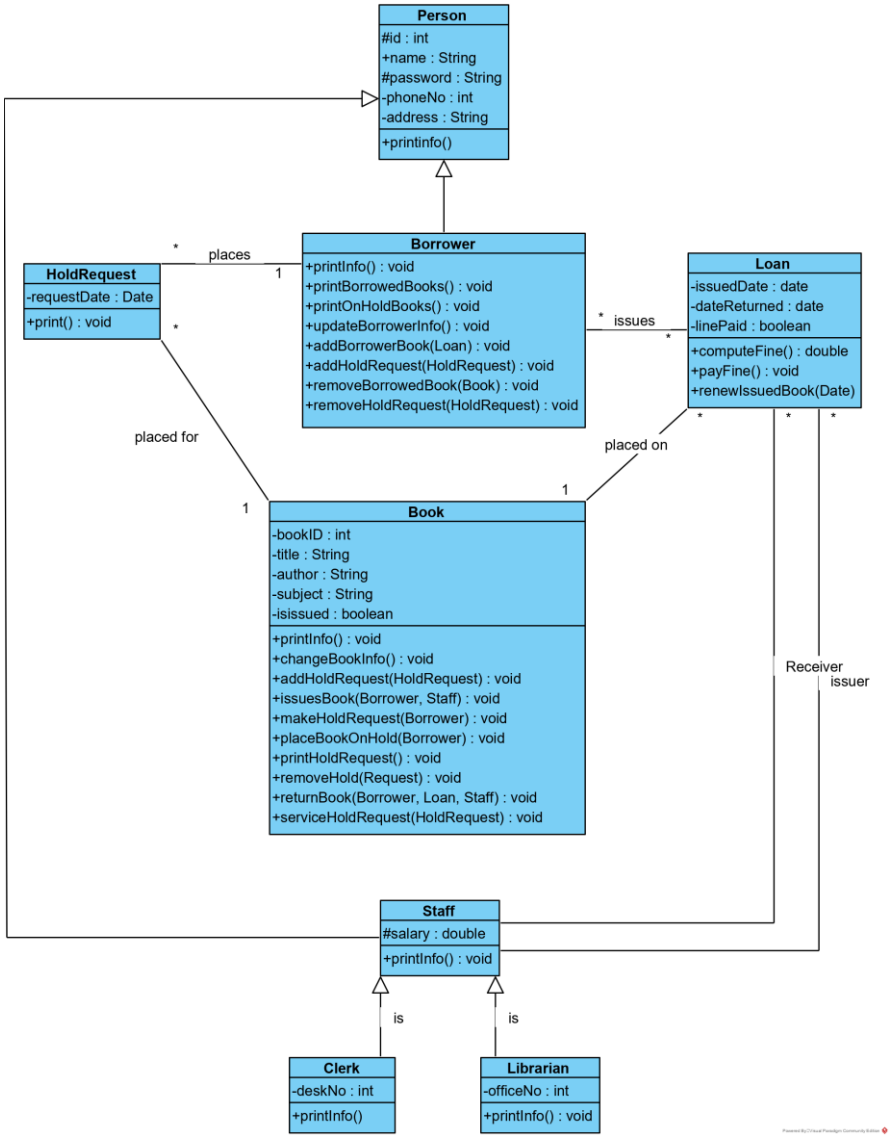
Hasil yang telah ditunjukkan pada Tabel 6.27 untuk *computation time system* pada berkas uji sederhana dan kompleks menghasilkan *time based efficiency* 0,583 tugas/detik sementara perhitungan secara manual menghasilkan 0,0055 tugas/detik. Untuk hasil *overall relative efficiency* perhitungan sistem dan perhitungan manual sama-sama menghasilkan 100% karena pengguna berhasil menyelesaikan semua tugas. Berdasarkan nilai tersebut maka efisiensi perhitungan menggunakan sistem lebih baik daripada perhitungan manual. Dari hasil yang didapatkan maka dapat disimpulkan efisiensi perhitungan *reusability* menggunakan sistem telah memenuhi kebutuhan non-fungsional yaitu sistem memiliki efisiensi lebih dari 80% yaitu 100% dapat melakukan perhitungan dan secara waktu lebih cepat dari perhitungan manual baik untuk berkas uji sederhana maupun kompleks.

6.4 Pembahasan Hasil

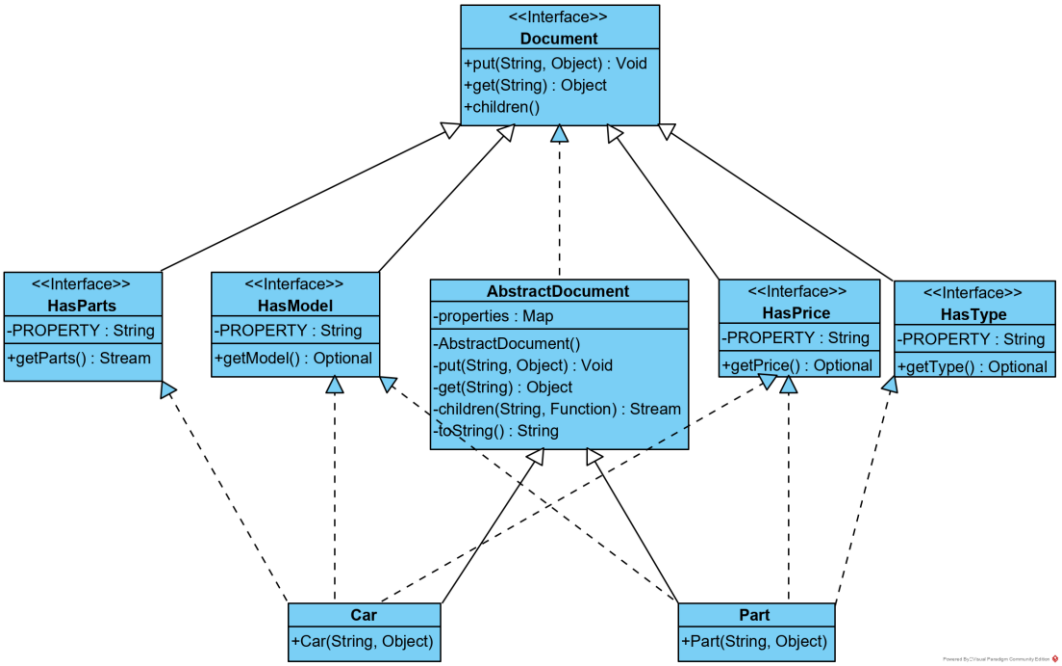
Berdasarkan pengujian yang telah dilakukan pada sistem perhitungan *Testability* menghasilkan status valid untuk keseluruhan kasus uji, sehingga sistem yang dibangun dapat digunakan untuk kakas bantu perhitungan nilai *Testability.* Pengujian unit menguji tiga *method* dari sistem yaitu *method PilihActionPerformed* dari kelas *ViewPilih, method* ValidasiXML dari kelas *XMLParser,* dan *method getHasil* dari kelas *Testability,* semua kasus uji yang dihasilkan mendapatkan status valid. Pengujian integrasi menguji integrasi antara *method getHasil* dari kelas *Testability,* kelas *effectiveness* (dengan parameter kelas *encapsulation*, kelas *inheritance*, kelas *coupling,* kelas *cohesion),* dan kelas *reusability* (dengan parameter kelas *encapsulation*, kelas *inheritance*, kelas *coupling)* menghasilkan status valid. Untuk pengujian validasi telah diuji semua kebutuhan fungsional termasuk kondisi alternatifnya dan mendapatkan hasil valid untuk semua kasus uji. Selain kebutuhan fungsional, pengujian validasi juga menguji kebutuhan non-fungsional yang mendapatkan hasil valid.
Pada bagian ini akan dijelaskan sistem yang diuji rancangan *class diagram* nya yaitu sistem *Library Management System JAVA dan Java Design Pattern* yang merupakan proyek *open source. Class diagram* dari sistem *Library Management System JAVA* adalah seperti dalam Gambar 6.5. Dalam sistem *Library Management System JAVA* terdapat 8 kelas dan dihasilkan *coupling* sebesar 5, *inheritance* sebesar 0,4, *encapsulation* sebesar 0,97, dan *cohesion* sebesar 0,13 sehingga pada perhitungan ini didapatkan nilai *reusability*-nya sebesar 15,871 dan nilai *effectiveness*-nya sebesar 6,470 kemudian dapat diketahui nilai *testability* dari sistem *LMS* adalah 42,096. Sedangkan *class diagram* dari sistem *Java Design Pattern* adalah seperti dalam Gambar 6.6. Dalam sistem *Java Design Pattern* terdapat 8 kelas dan dihasilkan *coupling* sebesar 3,5, *inheritance* sebesar 0,8, *encapsulation* sebesar 1 dan *cohesion* sebesar 0 sehingga pada perhitungan ini didapatkan nilai *reusability*-nya sebesar 23,504 dan nilai *effectiveness*-nya sebesar 13,051. Kemudian dapat diketahui nilai *testability* dari sistem *Career Information Management* adalah 17,508.

Nilai 0 pada *cohesion* dikarenakan sistem *Java Design Pattern* tidak memenuhi satu rumus pada *cohesion*. Tidak ada batasan nilai *testability* seberapa yang termasuk golongan baik atau tidak baik karena belum ada penelitian lebih

lanjut mengenai batasan tersebut. Namun sesuai dengan penelitian "*Metric Based Testability Estimation Model for Object Oriented Design: Quality Perspective"* oleh (Huda, dkk, 2015) bahwa semakin tinggi nilai *testability* maka semakin tinggi juga rating sistem tersebut.



Gambar 6.5 *Class Diagram* Sistem *Library Management System*



Gambar 6.6 Java Design Pattern

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil penelitian dari pengembangan aplikasi kakas bantu perhitungan nilai *testability* berdasarkan rancangan perangkat lunak diperoleh kesimpulan sebagai berikut:

1. Pengembangan aplikasi kakas bantu perhitungan nilai *testability* dengan menerapkan *waterfall model* karena pada penelitian ini keseluruhan kebutuhan telah didapatkan diawal dan tidak terjadi perubahan yang signifikan selama proses pengembangan aplikasi. Pada tahap analisis kebutuhan dilakukan proses pengkajian kepustakaan yang kemudian didapatkan kebutuhan fungsional berjumlah 3 dan kebutuhan non-fungsional berjumlah 1. Pada tahap perancangan dihasilkan perancangan algoritme, perancangan arsitektur, dan perancangan antarmuka pengguna. Dari rancangan arsitektur dihasilkan *sequence diagram* berjumlah 3 yang pada dokumen ini dituliskan sampel 2 *sequence diagram* dan *class diagram* yang memuat satu kelas *interface,* tiga kelas *boundary,* enam kelas *controller,* satu kelas *model*, dan tiga kelas dari *API Swing*. Kemudian pada tahap implementasi berhasil mengimplementasikan rancangan arsitektur, rancangan algoritme, dan rancangan antarmuka pengguna. Pada tahap pengujian dilakukan pengujian unit dengan teknik *basis path* yang berhasil menguji semua jalur uji, pengujian integrasi menggunakan pendekatan *top-down* berhasil menunjukkan modul telah terintegrasi, dan pengujian validasi berhasil memvalidasi seluruh kebutuhan fungsional yang berjumlah 5.
2. Pada pengujian efisiensi mendapatkan hasil sistem dapat melakukan perhitungan *testability* 100% berhasil dan lebih cepat dari perhitungan manual sehingga sistem telah memenuhi kebutuhan non-fungsionalnya.

7.2 Saran

Saran untuk pengembangan lanjut aplikasi kakas bantu perhitungan nilai *testability* berdasarkan rancangan perangkat lunak adalah sebagai berikut:

1. Sistem dapat dikembangkan supaya *compatible* dengan standar xml yang lain karena untuk saat ini masih tidak fleksibel untuk menerima berkas xml yang di parsing kerena hanya mampu memparsing xml dari Visual Paradigm dengan format *simple*.
2. Metrik-metrik yang digunakan dalam pengukuran *testability* menggunakan metrik *Quality Model for Object-Oriented Design* (QMOOD) yang masih bisa ditingkatkan menggunakan metrik-metrik yang lebih mutakhir.

DAFTAR REFERENSI

Ashdown, L., Greenberg, J., & Melnick, J. (2014). *Oracle XML Developer's Kit Programmer's Guide 11.1 Release.* Oracle.

Bansiya, J. dan Davis, C.G., 2002. A Hierarchical Model for Object-Oriented Design Quality Assessment. 28(1), hal.4–17.

Booch, G., Jacobson, I. dan Rumbaugh, J., 1999. *The Unified Modeling Language Reference Manual*. [daring] Tersedia pada: <http://www.citeulike.org/group/1374/article/3944245>.

Cole, B., dkk. (2002). *Java Swing, 2nd Edition.* Sebastopol: O'Reilly.

Fujita, H. & Pisanelli, D. M., 2007. *New Trends in Software Methodologies, Tools and Techniques.* 161 penyunt. Netherlands: IOS Press.

Huda, M., Sharma Arya, Y. D. and Hasan Khan, M., 2015. Metric Based *Testability* Estimation Model for Object Oriented Design: Quality Perspective. *Journal of Software Engineering and Applications*, 08(04), pp. 234-243.

Goyal, N. dan Gupta, E.D., 2014. Reusability Calculation of Object Oriented Software Model by Analyzing CK Metric. 3(7), hal.2466–2470.

Huda, M., Sharma Arya, Y. D. and Hasan Khan, M., 2015. Quantifying *Reusability* of Object Oriented Design: A *Testability* Perspective. *Journal of Software Engineering and Applications*, 8, 175-183.

Huda, M., Sharma Arya, Y. D. and Hasan Khan, M., 2015. *Effectiveness* Factor Of Object Oriented Design: A *Testability* Perspective. *International Journal of Software Engineering & Applications* (*IJSEA*), 6, 41-49.

K. Qian, X. Fu, L. Tao, and C. Xu, *Software Architecture And Design Illuminated. Jones & Bartlett Learning*, 2009

P. Bourque and R. E. Fairley, Guide to the Software Engineering Body of Knowledge, Version 3.0. IEEE Computer Society, 2014.

Pressman, R.S., 2001. *Software Engineering: A Practitioner's Approach*.

R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003

Sandhu, P.S., Kaur, H. dan Singh, A., 2018. Modeling of Reusability of Object Oriented Software System. hal.1–5.

Sergeev, A., 2010. *Efficiency.* [Online] Available at: http://ui-designer.net/usability/efficiency.htm [Diakses 12 2 2019].

Sommervile, Ian. 2011. *Software Engineering 9th Edition.* US America: Pearson Education, Inc.

Suri, Harsha, 2015. Object Oriented Software *Testability* (OOST) Metrics Analysis. *International Journal of Computer Applications Technology and Research,* 4(5), pp. 359-367.

Ubaidillah, M., Pradana, F. & Priyambadha, B., 2017. Kakas Bantu Perhitungan Nilai Kopling Menggunakan Metrik Cognitive Weighted *Coupling* Between Object. *KINETIK,* II(1), pp. 63-62.