Rekayasa Pembalikan Kode Berorientasi Objek ke Desain Kelas Menggunakan Struktur Data Graf

Paulus Mudjihartono[1], Findra Kartika Sari Dewi[2]
Program Studi Teknik Informatika, Universitas Atma Jaya Yogyakarta
Jl. Babarsari No. 43, Yogyakarta 55281, Indonesia
Email: paul235@staff.uajy.ac.id[1], findra@staff.uajy.ac.id[2]

Abstract. Reverse Engineering of Object Oriented Code to Class Design Using Graph Data Structure. *This time the software engineering methodology still follows the process of standard system development consisting of four phases of the system initialization, system analysis, system design, and system implementation. The phases occur sequentially so that the implementation is always done after the design is complete. This research is developing software Code Converter (COCON) to automate the conversion of object-oriented code to class design using graph data structures. COCON only requires an object-oriented code file as input and produce results in the form of a file containing a list of classes and relations between classes that read from input file. COCON help users to find out the class design of object-oriented code and becomes the basis for drawing class diagrams of object-oriented code.*

Keywords: *reverse engineering, class design, relation, OOP, graph*

Abstrak. Pada saat ini metodologi rekayasa perangkat lunak masih mengikuti proses pengembangan sistem yang standar yang terdiri dari empat fase yakni inisialisasi sistem, analisis sistem, desain sistem dan implementasi sistem. Keempat fase tersebut terjadi secara berurutan sehingga fase implementasi selalu dilakukan setelah fase desain selesai. Penelitian ini membangun perangkat lunak *Code Converter* (COCON) yang berfungsi untuk mengotomasi konversi kode program berorientasi objek ke desain kelas dengan struktur data graf. COCON hanya membutuhkan sebuah berkas kode program berorientasi objek sebagai masukan dan memberikan hasil berupa sebuah berkas yang berisi daftar kelas dan relasi antar kelas yang dibaca dari berkas masukan. COCON membantu pengguna untuk mengetahui desain kelas dari kode program berorientasi objek dan menjadi dasar dalam penggambaran diagram kelas dari kode berorientasi objek.

Kata Kunci: rekayasa pembalikan, desain kelas, relasi, PBO, graf

1. Pendahuluan

1.1. Latar Belakang

Pada saat ini metodologi rekayasa perangkat lunak masih mengikuti proses pengembangan sistem yang standar yang terdiri dari empat fase yakni inisialisasi sistem, analisis sistem, desain sistem dan implementasi sistem (Bentley & Whitten, 2007). Keempat fase tersebut terjadi secara berurutan sehingga fase implementasi selalu dilakukan setelah fase desain selesai. Ketika desain berubah, maka implementasi juga harus berubah. Saat ini sudah ada beberapa perangkat lunak yang dapat digunakan untuk menghasilkan kode program berorientasi objek secara otomatis dari desain kelasnya. Sehingga ketika desain kelas diubah, secara otomatis pula bisa diperoleh kode program yang berubah mengikuti desain kelas yang baru.

Ketika seorang *programmer* diberikan sebuah kode program tanpa desain kelas yang terdefinisi secara gamblang dalam bentuk gambar atau denah, maka sulit baginya untuk

mengetahui relasi kelas yang terlibat dalam program tersebut. Ketika hal ini terjadi, maka untuk memodifikasi kelas juga akan menjadi hal yang sangat sulit dan mungkin malah merusak kelas dan relasinya secara global. Oleh karena itu, dibutuhkan sebuah perangkat lunak yang mampu mengkonversi kode program berorientasi objek ke desain kelas sehingga akan mudah bagi seorang *programmer* untuk melihat bagaimana desain kelas dan relasi antar kelas yang terlibat.

Penelitian ini akan menghasilkan sebuah perangkat lunak yang mampu mengkonversi kode program berorientasi objek ke desain kelas dengan struktur data graf. Desain kelas akan terbatas sampai dengan kode struktur data graf yang mendefinisikan kelas dan relasi antar kelas yang terdapat dalam kode program berorientasi objek yang terdefinisi.

1.2. Perumusan Masalah

Perumusan masalah dari penelitian ini adalah bagaimana merekayasa pembalikan kode berorientasi objek ke desain kelas dengan struktur data graf?

1.3. Tujuan Penelitian

Tujuan penelitian ini adalah merekayasa perangkat lunak pembalik kode berorientasi objek ke desain kelas dengan struktur data graf.

1.4. Manfaat Penelitian

Manfaat penelitian ini secara umum, yaitu: (1) Mengetahui desain kelas dari kode program berorientasi objek. Jika hanya diketahui kode program berorientasi objek saja, maka akan sulit untuk mengetahui apa saja kelas dan relasi antar kelas yang terdefinisi dalam kode program tersebut. Penelitian ini akan menghasilkan kode yang mendefinisikan desain kelas dari kode program berorientasi objek yang terdefinisi. (2) Menjadi dasar dalam penggambaran diagram kelas dari kode berorientasi objek. Penggambaran diagram kelas akan semakin memperjelas objek kelas dan relasi antar kelas yang dimiliki, dari kode kode program berorientasi objek yang terdefinisi.

2. Tinjauan Pustaka

2.1. Konsep Dasar Pemrograman Berorientasi Objek

2.1.1. Enkapsulasi

Enkapsulasi adalah suatu cara untuk menyembunyikan informasi detail (*information hiding)* dari suatu kelas. *Information hiding* adalah proses penyembunyian informasi dari suatu kelas dengan memberikan akses kontrol *private* pada atribut atau *method* sehingga anggota atribut atau *method* tersebut tidak dapat diakses dari luar kelas. Tanpa *information hiding*, anggota kelas seperti atribut dan *method* dapat secara langsung diakses dari luar kelas yang melingkupinya, namun dengan *information hiding* perlu adanya *interface* untuk menjadi perantara agar atribut dan *method* yang sudah disembunyikan tersebut tetap dapat diakses dan/atau dimodifikasi dari luar kelas.

2.1.2. Pewarisan

Dalam pewarisan, setelah sebuah objek diciptakan, objek tersebut dapat digunakan sebagai fondasi untuk objek yang sama yang memiliki perilaku atau karakteristik yang sama. Objek bisa dibentuk sebagai suatu kelas. Kelas bisa diatur dalam hierarki kelas atau subkelas. Pewarisan adalah metode untuk mewariskan ciri dari suatu objek dari kleas ke subkelas dalam hierarki. Jadi objek yang baru dapat diciptakan dengan mewariskan ciri dari kelas yang sudah

ada. Kelas yang menurunkan ciri disebut kelas dasar *(base class)*, sedangkan yang kelas baru yang mewarisi sifat kelas dasar disebut kelas turunan *(derived class)*. Pewarisan sifat hanya berlaku searah yaitu dari kelas dasar ke kelas turunan. Sifat yang dimiliki oleh kelas turunan tidak diwariskan ke kelas dasar. Kelas turunan tidak hanya bisa memiliki kelas yang diwarisi dari kelas induknya, tapi juga dapat memiliki sifat sendiri yang tidak ditemukan di kelas induknya.

2.1.3. Polimorfisme

Polimorfisme adalah pemikiran bahwa objek dinamis suatu kelas dasar dapat berperilaku seperti kelas turunan. Ketika objek dinamis menunjuk kelas dasar, objek tersebut berperilaku seperti kelas dasar, tetapi ketika objek tersebut menunjuk kelas turunan, objek tersebut berperilaku seperti kelas turunan. Objek dapat memiliki beberapa bentuk, tergantung kelas mana yang sedang ditunjuk.

2.2. Kelas, Generalisasi dan Spesialisasi

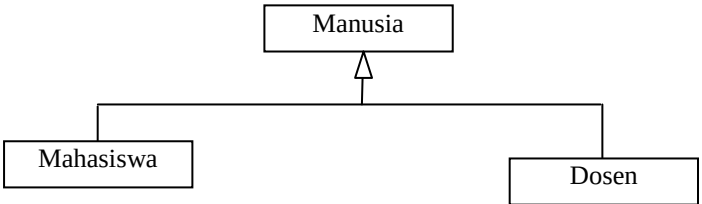
Kelas adalah set objek yang berbagi atribut dan perilaku yang sama. Sedangkan Generalisasi/Spesialisasi adalah sebuah teknik dimana atribut dan perilaku dari beberapa kelas dikelompokkan (atau diabstraksi) kedalam kelas mereka sendiri, disebut dengan *supertype*. Atribut dan perilaku dari kelas *supertype* kemudian diwarisi oleh kelas-kelas *subtype*.

2.3. Graf

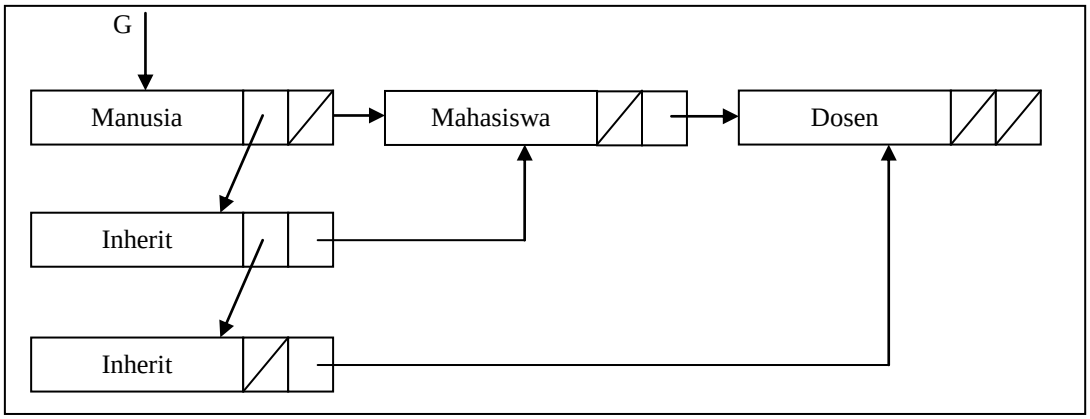
Graf adalah representasi grafik yang melibatkan sekumpulan objek yang dilambangkan dengan simpul atau verteks, dan relasi-relasi biner antar objek-objek diskret tersebut yang dilambangkan dengan rusuk atau sisi atau *edge* (Langsam & Tenenbaum, 2000). Ada empat jenis graf, yakni graf tidak berarah *(undirected graph)*, graf berarah *(directed graph)*, graf berbobot *(weighted graph)* dan graf berarah dan berbobot *(weighted and directed graph)* (Standish, 1995).

2.4. Desain Kelas dalam Struktur Graf

Dari gambar 1 dapat dilihat ada 3 kelas, yaitu kelas Manusia, kelas Mahasiswa dan kelas Dosen. Dari diagram kelas yang memiliki relasi sedemikian rupa, jika dikonversikan dalam graf, maka akan didapatkan diagram pada gambar 2.



Gambar 1. Hubungan Generalisasi/Spesialisasi Menggunakan UML



Gambar 2. Desain Kelas dalam Struktur Graf

3. Metodologi Penelitian

3.1. Pengumpulan Bahan

Pengumpulan bahan bertujuan untuk memperoleh literatur yang lengkap tentang bahan yang sedang diteliti. Pengumpulan bahan dilakukan dengan mencari buku, jurnal, tesis yang berhubungan dengan bahan yang sedang diteliti. Pengumpulan bahan dapat memanfaatkan perpustakaan yang sudah ada dan mengakses situs-situs internet yang telah mempublikasikan hasil penelitian. Berdasarkan bahan-bahan yang sudah diperoleh kemudian dilakukan pengembangan terhadap algoritma pembalikan kode berorientasi objek ke desain kelas dengan struktur data graf yang akan diteliti.

3.2. Pengumpulan Data

Data berupa kode berorientasi objek yang valid, bebas dari error, yang menggunakan bahasa pemrograman C#.NET. Dari ini diambil dari kode program hasil dari penelitian lain, yang digunakan sebagai sampel percobaan. Dari data ini akan dilakukan penyesuaian berupa penggabungan beberapa kelas kedalam sebuah berkas untuk dapat dijadikan parameter masukan untuk COCON.

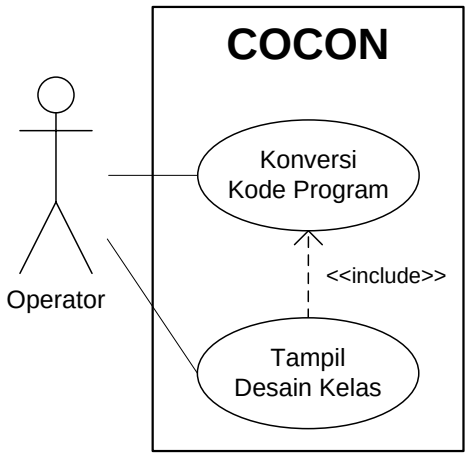
3.3. Perancangan Perangkat Lunak

3.3.1. Kebutuhan Antarmuka

Pengguna akan berinteraksi dengan COCON dengan antarmuka GUI (*Graphical User Interface*). Sebagai sarana *input* digunakan *mouse* dan *keyboard*, sedangkan sarana *output* digunakan monitor komputer. Perangkat keras yang diperlukan oleh COCON adalah Personal Computer, RAM minimal 256 MB, Keyboard dan Mouse. Perangkat lunak lain yang digunakan untuk mendukung berjalannya perangkat lunak COCON adalah Microsoft Windows XP/Vista/7 dan Microsoft Visual Studio .NET 2005.

3.3.2. Kebutuhan Fungsionalitas

Secara umum, fungsi-fungsi yang dimiliki oleh COCON dapat dilihat di gambar 3, dan deskripsinya dapat dilihat di tabel 1 dan tabel 2.



Gambar 3. Diagram *Use Case*

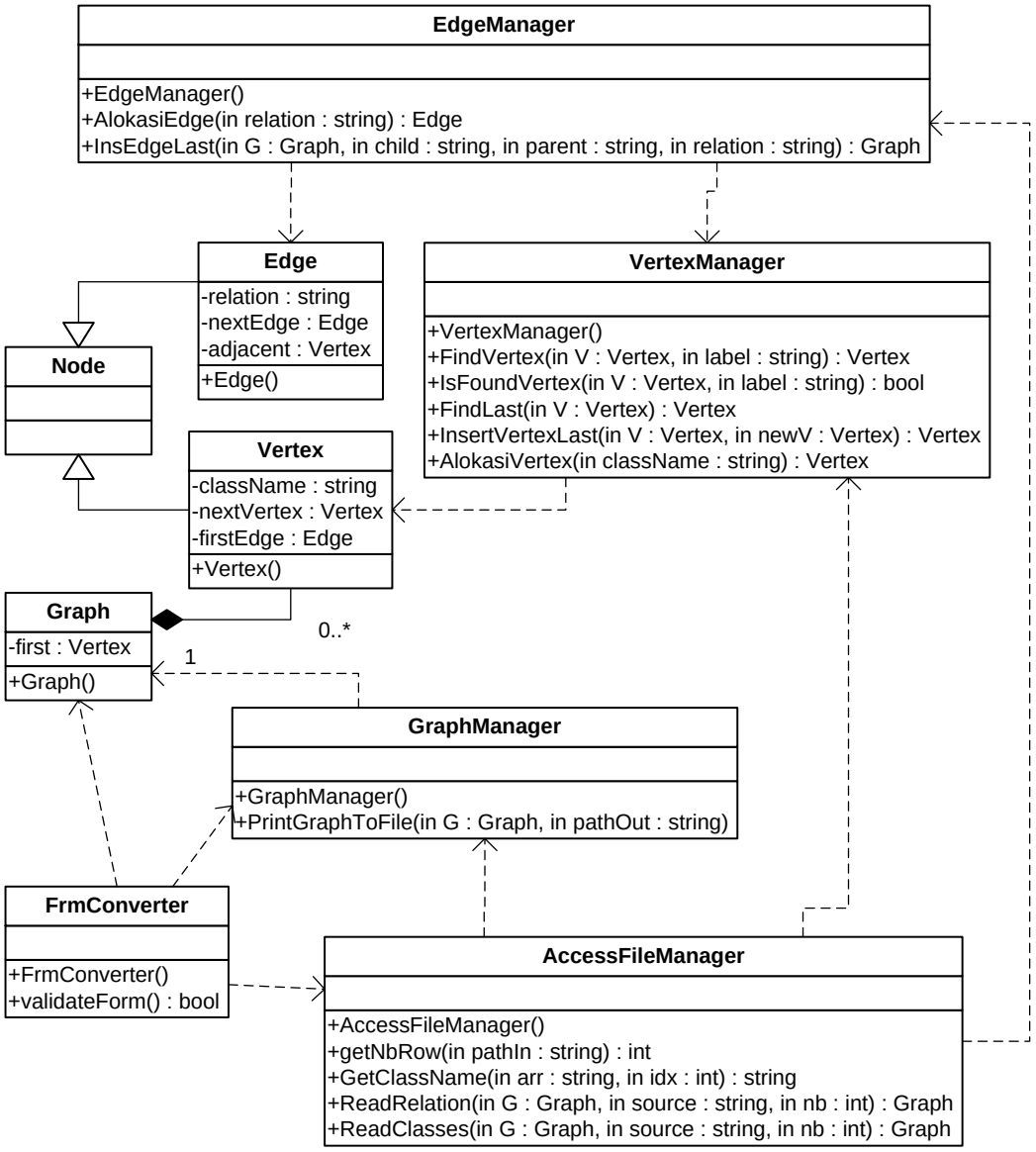
Tabel 1. Skenario *Use Case* Konversi Kode Program

Nama *Use Case*	Konversi Kode Program
Aktor	Operator
Deskripsi Singkat	*Use case* ini digunakan pengguna untuk mengkonversi kode program menjadi desain kelas. Konversi kode program dikenakan pada sebuah berkas yang berisi kode program valid dan bisa memuat banyak kelas sekaligus.
Pre Condition	-
Flow of Event	1. Sistem menampilkan antarmuka konversi kode program. 2. Aktor memilih berkas kode program yang akan dikonversi. 3. Aktor memilih lokasi dan memasukkan nama berkas desain tujuan konversi. 4. Aktor menekan tombol Konversi. 5. Sistem membaca nama kelas dan relasi antar kelas didalam kode program. 6. Sistem menciptakan berkas yang berisikan desain kelas. 7. *Use case* selesai.
Post Condition	Berkas yang berisi desain kelas tercipta.
Alternative Flow	1. Pada langkah 6, jika berkas desain kelas gagal diciptakan, maka sistem akan menampilkan pesan bahwa desain gagal disimpan.

Tabel 2. Skenario *Use Case* Tampil Desain Kelas

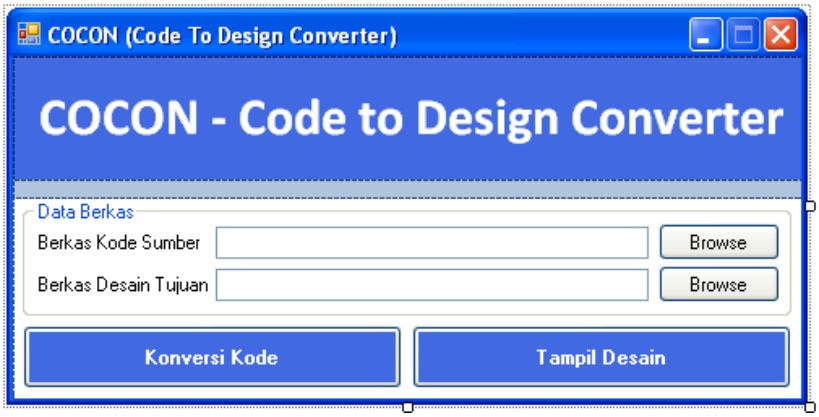
Nama *Use Case*	**Tampil Desain Kelas**
Aktor	Operator
Deskripsi Singkat	*Use case* ini digunakan oleh aktor untuk menampilkan berkas desain kelas yang terbentuk dari proses konversi kode program.
Pre Condition	Berkas desain kelas berhasil terbentuk dari proses konversi kode program.
Flow of Event	1. Aktor menekan tombol Tampil Desain. 2. Sistem menampilkan isi berkas desain yang terbentuk dari proses konversi kode program. 3. *Use case* selesai.
Post Condition	Isi berkas desain kelas ditampilkan.
Alternative Flow	1. Pada langkah 2, jika berkas desain kelas tidak ditemukan, maka sistem akan menampilkan pesan bahwa berkas desain tidak ditemukan.

3.4. Perancangan Class Diagram



Gambar 4. ***Class Diagram***

3.5. Perancangan Antarmuka



Gambar 5. ***Form Converter***

Antarmuka ini terdiri sebuah *form* yang digunakan untuk mendata lokasi berkas kode sumber dan desain tujuan, serta tombol untuk memproses berkas yang telah dimasukkan. *Textbox* atas digunakan untuk menerima masukan lokasi dan nama berkas kode sumber, *textbox* bawah digunakan untuk menerima masukan lokasi dan nama berkas desain tujuan. Dua *textbox* ini tidak harus diisi dengan ketikan, namun bisa melalui tombol Browse yang ada disebelah kanan masing-masing *textbox*. Tombol Konversi Kode digunakan untuk memproses berkas kode sumber menjadi berkas desain tujuan, sedangkan tombol Tampil Desain digunakan untuk menampilkan isi dari berkas desain tujuan.

3.6. Pembuatan Perangkat Lunak

Hasil rancangan *flowchart* kemudian diimplementasikan dengan menggunakan bahasa pemrograman C#. Pembuatan perangkat lunak dilakukan dalam tahap berikut: (1) Pemindaian kode program untuk mencari kelas apa saja yang terdapat dalam kode program. (2) Pembentukan struktur data graf untuk kelas-kelas yang diperoleh dari pemindaian dilangkah pertama. (3) Pemindaian kode program untuk mencari relasi apa saja yang terlibat diantara kelas-kelas yang diperoleh dari pemindaian dilangkah pertama. Pembentukan struktur data graf untuk relasi antar kelas yang diperoleh dari pemindaian dilangkah ketiga.

3.7. Pengujian Perangkat Lunak

Perangkat lunak yang sudah jadi kemudian diuji dengan menggunakan kode program berorientasi objek sebagai parameter masukannya. Pada tahap ini dilakukan pengujian apakah perangkat lunak sudah bekerja sesuai dengan yang diinginkan. Revisi perangkat lunak dapat dilakukan jika program tidak bekerja sesuai dengan yang diinginkan.

3.7.1. *Pengujian Akurasi Konversi*

Pada percobaan ini akan digunakan kode program berorintasi objek dengan variasi kerumitan. Dalam percobaan ini akan dilihat apakah desain kelas yang dihasilkan oleh perangkat lunak sesuai dengan kode program yang menjadi parameter masukannya.

4. Hasil Dan Pembahasan

4.1. Berkas Kode Sumber

Berkas kode sumber yang dikonversi hanya terdiri sebuah berkas yang bisa memuat banyak kelas sekaligus. Didalam sebuah kelas, deklarasi maupun definisi atribut dan method tidak berpengaruh karena yang dipindai hanya sebatas nama-nama kelas yang terdapat didalam satu berkas masukan tersebut. Kelas-kelas yang terdeteksi akan dicatat dan dicari relasi *inheritance*-nya satu sama lain. Relasi *inheritance* yang ditangani adalah relasi *inheritance* tunggal. Kode program yang dikonversi berbahasa C#, sudah dikompilasi dan bebas dari error.

4.2. Berkas Desain Kelas

Berkas desain kelas hasil konversi kode sumber akan menampilkan daftar kelas dan daftar relasi *inheritance* antar kelas. Format yang digunakan dalam berkas desain kelas ini, pada bagian awal akan ditampilkan daftar nama kelas yang ditemukan di kode sumber, dan pada bagian akhir akan ditampilkan daftar relasi yang ditemukan antar kelas-kelas yang ada. Berkas ini berformat teks dengan ekstensi txt.

4.3. Proses Konversi

Proses konversi kode sumber dimulai dengan pembacaan nama kelas yang terdapat didalam kode sumber. Setiap kali ditemukan sebuah kelas, nama kelas tersebut akan diperiksa apakah nama kelas tersebut sudah pernah ditemukan sebelumnya. Jika belum, maka nama kelas tersebut akan dialokasikan ke sebuah *vertex* dan setelah itu segera dimasukkan kedalam *graph* yang menyimpan nama-nama kelas yang terdaftar sebelumnya. Setelah semua nama kelas tersimpan dalam *graph*, dilanjutkan dengan proses pembacaan relasi antar kelas. Jika ditemukan relasi *inheritance* antar satu kelas dengan kelas yang lain, maka akan dialokasikan sebuah *edge* yang akan menghubungkan kedua *vertex* yang saling berhubungan. Setelah semua relasi inheritance antar kelas tersimpan, maka proses konversi telah selesai dan hasilnya disimpan ke dalam berkas teks. Berkas teks ditulis berdasarkan *graph* yang terbentuk, dimana *vertex* mewakili kelas dan *edge* mewakili relasi.

4.4. Pengujian Dengan Variasi Kerumitan Kode Program

Pengujian dilakukan dengan variasi kerumitan kode program dari kode yang sederhana hingga kode yang cukup kompleks. Untuk kode kompleks akan diberikan contoh dimana terdapat tujuh kelas dan kelas-kelas tersebut memiliki beberapa relasi *inheritance*. Rangkuman ada pada tabel 3.

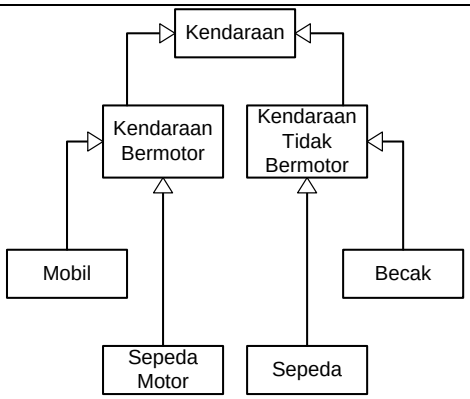
Tabel 3. Pengujian dengan Kode Progam Kompleks

Kode Sumber

```
using System;
public class Kendaraan
{
    string nama;
    int tahun;
    public Kendaraan()
    {
        nama = "";
        tahun = 0;
    }
}
public class KendaraanBermotor : Kendaraan
{
    string nama;
    int tahun;
    string mesin;
    public KendaraanBermotor()
    {
        nama = "";
        tahun = 0;
        mesin = "";
    }
}
public class KendaraanTidakBermotor : Kendaraan
{
    string nama;
    int tahun;
    public KendaraanTidakBermotor()
    {
        nama = "";
        tahun = 0;
    }
}
public class Mobil : KendaraanBermotor
{
    string nama;
    int tahun;
    string mesin;
    string audio;
    public Mobil()
    {
```

```
            nama = "";
            tahun = 0;
            mesin = "";
            audio = "";
        }
    }
    public class SepedaMotor : KendaraanBermotor
    {
        string nama;
        int tahun;
        string mesin;
        string cakram;
        public SepedaMotor()
        {
            nama = "";
            tahun = 0;
            mesin = "";
            cakram = "";
        }
    }
    public class Sepeda : KendaraanTidakBermotor
    {
        string nama;
        int tahun;
        int keranjang;
        public Sepeda()
        {
            nama = "";
            tahun = 0;
            keranjang = 0;
        }
    }
    public class Becak : KendaraanTidakBermotor
    {
        string nama;
        int tahun;
        string kap;
        public Becak()
        {
            nama = "";
            tahun = 0;
            kap = "";
        }
    }
```

Analisis kode sumber	Dari kode sumber dapat dilihat bahwa terdapat tujuh kelas yakni kelas Kendaraan, KendaraanBermotor, KendaraanTidakBermotor, Mobil, SepedaMotor, Sepeda dan Becak. Kelas Kendaraan berperan sebagai *base class* untuk kelas KendaraanBermotor dan KendaraanTidakBermotor. Kelas KendaraanBermotor berperan sebagai *base class* untuk kelas Mobil dan SepedaMotor. Kelas KendaraanTidakBermotor berperan sebagai *base class* untuk kelas Sepeda dan Becak. Ada enam relasi *inheritance* yaitu antara kelas Kendaraan dan KendaraanBermotor, Kendaraan dan KendaraanTidakBermotor, KendaraanBermotor dan Mobil, KendaraanBermotor dan SepedaMotor, KendaraanTidakBermotor dan Sepeda, KendaraanTidakBermotor dan Becak.
Daftar Kelas	1. Kendaraan 2. KendaraanBermotor 3. KendaraanTidakBermotor

	4. Mobil 5. SepedaMotor 6. Sepeda 7. Becak
Relasi *Inheritance*	1. KendaraanBermotor *inherit* Kendaraan 2. KendaraanTidakBermotor *inherit* Kendaraan 3. Mobil *inherit* KendaraanBermotor 4. SepedaMotor *inherit* KendaraanBermotor 5. Sepeda *inherit* KendaraanTidakBermotor 6. Becak *inherit* KendaraanTidakBermotor
Class Diagram	
Proses Konversi	Proses konversi kode sumber dimulai dengan pembacaan nama kelas yang terdapat didalam kode sumber. Nama kelas yang pertama kali terdeteksi adalah Kendaraan. Karena kelas Kendaraan belum pernah ditemukan sebelumnya, maka nama kelas tersebut dialokasikan ke sebuah *vertex* dan setelah itu segera dimasukkan kedalam *graph*. Kelas berikutnya yang terdeteksi secara berurutan adalah KendaraanBermotor, KendaraanTidakBermotor, Mobil, SepedaMotor, Sepeda dan Becak. Kelas-kelas ini juga belum pernah ditemukan sebelumnya, maka nama kelas tersebut dialokasikan ke *vertex* dan setelah itu segera dimasukkan kedalam *graph*. Setelah semua nama kelas tersimpan dalam *graph*, dilanjutkan dengan proses pembacaan relasi antar kelas. Dari pemindaian ulang kode sumber, ditemukan relasi *inheritance* antara beberapa kelas, maka dialokasikan *edge* yang menghubungkan *vertex* KendaraanBermotor ke *vertex* Kendaraan, *edge* yang menghubungkan *vertex* KendaraanTidakBermotor ke *vertex* Kendaraan, *edge* yang menghubungkan *vertex* Mobil ke *vertex* KendaraanBermotor, *edge* yang menghubungkan *vertex* SepedaMotor ke *vertex* KendaraanBermotor, *edge* yang menghubungkan *vertex* Sepeda ke *vertex* KendaraanTidakBermotor, *edge* yang menghubungkan *vertex* Becak ke *vertex* KendaraanTidakBermotor. Semua relasi *inheritance* antar kelas tersimpan, maka proses konversi telah selesai dan hasilnya disimpan ke dalam berkas teks. Penulisan berkas desain kelas dimulai dari pembacaan label-label *vertex* yang mewakili nama-nama kelas, yakni Kendaraan, KendaraanBermotor, KendaraanTidak-Bermotor, Mobil, SepedaMotor, Sepeda dan Becak. Setelah itu dilanjutkan dengan pembacaan relasi yang ditunjukkan dengan *edge*, yakni dari KendaraanBermotor ke Kendaraan, dari

	KendaraanTidakBermotor ke Kendaraan, dari Mobil ke KendaraanBermotor, dari SepedaMotor ke KendaraanBermotor, dari Sepeda ke KendaraanTidakBermotor, dan dari Becak ke KendaraanTidakBermotor. Hal ini menunjukkan bahwa kelas KendaraanBermotor *inherit* Kendaraan, KendaraanTidakBermotor *inherit* Kendaraan, Mobil *inherit* KendaraanBermotor, SepedaMotor *inherit* KendaraanBermotor, Sepeda *inherit* KendaraanTidakBermotor, dan Becak *inherit* KendaraanTidakBermotor. Proses pembacaan selesai.
Desain kelas	Daftar Kelas: Kendaraan KendaraanBermotor KendaraanTidakBermotor Mobil SepedaMotor Sepeda Becak Daftar Relasi: KendaraanBermotor *inherit* Kendaraan KendaraanTidakBermotor *inherit* Kendaraan Mobil *inherit* KendaraanBermotor SepedaMotor *inherit* KendaraanBermotor Sepeda *inherit* KendaraanTidakBermotor Becak *inherit* KendaraanTidakBermotor
Hasil Pengujian	Desain kelas sesuai dengan *class diagram*.

5. Kesimpulan

Berdasarkan pembahasan pada bab-bab sebelumnya, dapat ditarik kesimpulan sebagai berikut: (1) Struktur graf dengan presisi dapat digunakan sebagai representasi diagram kelas. (2) Proses konversi balik telah dapat dilakukan dengan benar dengan menggunakan struktur graf sebagai media penampungnya. (3) Rekayasa pembalikan kode telah berhasil dibuktikan dan dilakukan dengan baik.

6. Saran

Beberapa saran yang dapat ditarik dari proses analisa sampai pembuatan laporan penelitian ini adalah sebagai berikut: (1) COCON belum mempertimbangkan efisiensi komputasi, seperti alokasi sumberdaya prosesor dan waktu. Dengan memasukkan optimasi komputasi ke dalam algoritma pembalikan kodenya, eksekusi dapat dijalankan lebih efisien. (2) COCON belum dilengkapi dengan hasil yang berupa object persistence, untuk keperluan kemudahan pemanfaatan hasil, termasuk memanipulasinya ke dalam bentuk data lain.

Referensi

Bentley, L.D and Whitten, J.L., *Systems Analysis & Design for the Global Enterprise 7th ed*, McGraw-Hill, 2007.

Boggs, Wendy, Michael Boggs. *Mastering UML With Rational Rose*.2002.

Fowler, Martin, Kendall Scott. *UML Distilled – Second Edition*. Adisson Wesley. 1999.

Jacobson, Ivar, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley. 1998.

Kusniyati, Harni, *Modul Bahasa C++,* Pusat Pengembangan Bahan Ajar UMB.

Langsam, Augenstein, & Tenenbaum, *Data Structures Using C and C++ 2nd ed*., Prentice Hall Inc, 2000

Schach, Stephen R. *An Introduction to Object-Oriented Systems Analysis and Design with UML and the Unified Process*. Mc Graw-Hill. 2004.

Standish, Thomas A., *Data Structures, Algorithms, and Software Principles in C*, Addison Wesley, 1995