TESIS - KI142502

PENDETEKSIAN KLON SECARA SEMANTIK BERDASARKAN PERILAKU KODE

Bayu Priyambadha
5112201011

DOSEN PEMBIMBING
Dr. Ir. Siti Rochimah, MT.

PROGRAM MAGISTER
REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2015

THESIS - KI142502

SEMANTIC CLONE DETECTION BASED ON CODE BEHAVIOR

Bayu Priyambadha
5112201011

SUPERVISOR
Dr. Ir. Siti Rochimah, MT.

MASTER PROGRAM
SOFTWARE ENGINEERING
INFORMATICS ENGINEERING
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2015

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

Oleh :
BAYU PRIYAMBADHA
NRP. 5112201011

Dengan Judul :
Pendeteksian Klon Secara Semantik Berdasarkan Perilaku Kode

Tanggal Ujian : 3 Juni 2015
Periode Wisuda : 2015 Genap

Disetujui oleh :

1. Dr. Ir. Siti Rochimah, MT
   NIP. 196810021994032001 (Pembimbing)
2. Daniel Oranova, S,Kom. MSc. PD.Eng.
   NIP. 197411232006041001 (Penguji)
3. Rizky Januar Akbar, S.Kom., M.Eng.
   NIP. 198701032014041000 (Penguji)
4. Sarwosri, S.Kom., M.T.
   NIP. 197608092001122001 (Penguji)

Direktur Program Pasca Sarjana,

Prof. Dr. Ir. Adi Soeprijanto, M.T.
NIP. 196404051990021001

PENDETEKSIAN KLON SECARA SEMANTIK BERDASARKAN PERILAKU KODE

Nama mahasiswa : Bayu Priyambadha
NRP : 5112201011
Pembimbing I : Dr. Ir. Siti Rochimah, MT.

ABSTRAK

Kegiatan penyalinan dan penyisipan fragmen kode dari suatu kode sumber ke kode sumber yang lain, dengan atau tidak disertai modifikasi, disebut sebagai kloning kode. Proses tersebut sering dilakukan oleh pengembang perangkat lunak. Keberadaan kloning akan meningkatkan usaha yang harus dilakukan oleh pengembang perangkat lunak dalam proses pemeliharaan kode baik berupa tenaga, waktu dan biaya. Kloning juga dapat menimbulkan cacat pada perangkat lunak. Oleh karena itu kloning harus ditemukan. Penelitian tentang pendeteksian keberadaan kloning telah dilakukan. Deteksi kloning secara semantik adalah salah satu metode deteksi yang masih butuh pendalaman. Salah satu metode deteksi kloning secara semantik adalah berdasarkan perilaku kode. Perilaku kode dideteksi dengan melihat input, output dan efek dari sebuah metode. Metode dengan input, output dan efek yang sama akan menunjukkan bahwa secara fungsi metode tersebut adalah sama. Namun, metode deteksi tersebut tidak dapat digunakan pada metode yang sifatnya void, atau tidak memiliki nilai kembalian maupun parameter input. Proses melihat kesamaan tipe input, output dan efek hanya dilakukan dengan menggunakan pendekatan kesamaan *string*.

Pada penelitian ini diusulkan sebuah metode dalam proses deteksi kloning kode secara semantik. Dalam metode ini, proses deteksi dilakukan pada seluruh metode tanpa kecuali. Deteksi input, output dan efek pada sebuah metode *void* dilakukan dengan menggunakan PDG (*Program Dependence Graph*). Proses pencocokan kesamaan metode berdasarkan tipe input, output dan efek dilakukan berdasarkan informasi semantik yang dikandung. Untuk mendapatkan perilaku metode, uji coba dilakukan pada setiap metode. Uji coba tersebut dilakukan dengan memberikan data input secara random kepada metode yang sedang diuji. Output yang keluar dari metode ditangkap untuk dicatat sebagi bahan proses pencocokan kesamaan. Kelompok metode dengan nilai input dan output yang sama adalah metode yang memiliki kesamaan secara perilaku.

Dengan menerapkan metode tersebut, maka deteksi kloning kode pada sebuah kode sumber dapat dilakukan secara menyeluruh. Penggunaan metode deteksi klon secara semantik berdasarkan perilaku kode ini memiliki nilai koefisien Kappa sebesar 0,2128. Nilai tersebut dapat diinterpretasikan bahwa hasil tingkat kecocokan antara sistem dan pakar pada penelitian ini adalah cukup (*fair agreement*).

Kata kunci : deteksi kloning, semantik kloning, kloning perilaku, perawatan perangkat lunak.

SEMANTIC CLONE DETECTION BASED ON CODE BEHAVIOR

Nama mahasiswa : Bayu Priyambadha
NRP : 5112201011
Pembimbing I : Dr. Ir. Siti Rochimah, MT.

ABSTRACT

Activities copying and insertion fragments of a source code to another source code, with or without accompanying by the modification, are known as code cloning. The software developer is often doing this activity. The existence of cloning would increase the effort that must be made by software developers in the process of the maintenance of code in the form of manpower, time, and cost. Cloning can also cause defects in the software. Hence, cloning must be found. Research on detecting the presence of cloning has been done. Detection of cloned semantically is a detection method that still need a deepening. One of detection method of semantic cloning is based on the behavior of the code. The code behavior detected by looking at input, output and effects of the method. Methods with input, output, and the same effect will show that is a function of the method is the same. However, the detection method could not be used on a void method. The process of seeing the similarity type of input, output and effects only effected by using the string similarity approach.

This research proposed a method of semantic code clone detection. In this method, the detection process is performed on the entire method without exception. Detection of input, output and effects in a void method done using PDG (Program Dependence Graph). The process of matching similarity of method based on the type of input, output, and the effect is done based on semantic information is contained. On every method, testing has performed to get the behavior methods. The test is done with random input data to the method being tested. The result of method testing was recorded to observes the behavior of methods. Group method with the same value of input and output are methods that have similarities in behavior.

By applying the method, then the expected detection of cloned code on the source code can be done thoroughly. The use of clone detection methods semantically based on behavior has a value of the Kappa coefficient of 0.2128. This value can be interpreted that the results of the level of agreement between systems and experts in this study were fair agreement.

Keywords: clone detection, semantic clone, behavioral cloning, software maintenance.

KATA PENGANTAR

Segala puji bagi Allah yang telah memberikan rahmatnya sehingga penulis dapat menyelesaikan tesis dengan judul "Pendeteksian Klon Secara Semantik Berdasarkan Perilaku Kode". Tesis ini disusun untuk memenuhi sebagian persyaratan guna memperoleh gelar Magister Komputer di Institut Teknologi Sepuluh Nopember.

Melalui pengantar ini penulis ingin mengucapkan banyak terima kasih karena dalam penyusunan tesis ini penulis telah mendapat bantuan dan dorongan baik secara moril maupun materiil dari berbagai pihak. Untuk itu pada kesempatan ini penulis ingin mengucap terima kasih kepada :

1. Bapak Waskitho Wibisono, S.Kom, M.Eng, Ph. D, selaku Ketua Program Studi Pascasarjana Teknik Informatika Institut Teknologi Sepuluh Nopember.
2. Ibu Dr. Ir. Siti Rochimah, MT, selaku pembimbing dan penasehat akademik yang telah dengan sabar membimbing dan mengarahkan penulis dalam pengerjaan tesis ini sehingga dapat terselesaikan dengan baik.
3. Keluarga penulis, Ayah, Ibu, Istri dan Anak yang selalu memberikan dukungan mental dan spiritual selama penulis menyelesaikan studinya di Institut Teknologi Sepuluh Nopember.
4. Teman-teman dosen di Universitas Brawijaya pada umumnya dan dosen laboratorium Rekayasa Perangkat Lunak PTIIK UB, yang telah sangat banyak memberikan masukkan dalam penelitian ini.
5. Teman-teman Pascasarjana Teknik Informatika ITS, yang telah membatu dan berbagi informasi dengan penulis.

Serta semua pihak yang tidak dapat penulis sebutkan yang telah membantu penulis dalam penyelesaian tesis ini.

Penulis sangat menyadari bahwa tesis ini masih banyak kekurangan dan masih sangat mungkin untuk dikembangkan menjadi lebih baik, oleh karena itu penulis mengharapkan mesukan dari semua pihak.

Malang, 7 Juli 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Kegiatan penyalinan dan penyisipan fragmen kode dari suatu kode sumber ke kode sumber yang lain, dengan atau tidak disertai modifikasi, disebut sebagai kloning kode (Rattan et al, 2013). Proses tersebut sering dilakukan oleh pengembang perangkat lunak. Kloning kode sering dilakukan dengan berbagai alasan, antara lain keterbatasan waktu pengembangan perangkat lunak. Baker dalam Rattan et al, (2013) menyatakan bahwa dalam sebuah kode sumber setidaknya terdapat 20-30% kode yang terkloning. Keberadaan kloning pada kode sumber sebuah sistem membutuhkan perhatian lebih pada proses pengembangan perangkat lunak. Hal itu disebabkan karena dampak yang mungkin ditimbulkan dari keberadaan kloning.

Kloning kode dianggap dapat memangkas waktu pembuatan kode sumber pada proses pembangunan perangkat lunak. Kegiatan kloning kode meringankan penyusun kode sumber karena dengan menyalin tidak perlu lagi memikirkan bagaimana alur proses pada kode sumber yang sedang dibuat. Namun, proses kloning kode dapat menimbulkan permasalahan apabila sebuah fragmen kode yang disalin mengandung cacat. Hal ini menyebabkan seluruh fragmen kode yang sama juga akan mengandung kecacatan. Untuk mengatasi hal itu, maka seorang penyusun kode sumber harus melakukan pemeriksaan dan pembenaran pada seluruh fragmen kode yang sama. Kondisi ini akan meningkatkan usaha yang harus dilakukan oleh pengembang perangkat lunak dalam proses pemeliharaan kode baik berupa tenaga (Sjoberg & Yamashita, 2013; Lozano & Wermelinger, 2008), waktu dan biaya (Fowler, 1999; Jiang & Su, 2009). Kloning tidak hanya meningkatkan biaya pada proses pemeliharaan, tetapi juga dapat menimbulkan cacat pada perangkat lunak. Sjoberg dan Yamashita (2013), menyatakan bahwa bila proses pemeliharaan tidak dilakukan secara konsisten maka dapat menimbulkan cacat (*defect*) pada perangkat

lunak. Hal senada juga disampaikan oleh Bettenburg et al, (2009) yang menyatakan bahwa cacat dapat muncul apabila pengembang tidak melakukan pemeliharaan dengan baik pada kloning kode.

Beberapa metode telah digunakan dalam proses penemuan kloning kode. Metode-metode tersebut dikelompokkan dalam dua kelompok, yaitu metode yang dijalankan berdasarkan kesamaan sintak (sintaksis) dan kesamaan makna (semantik). Elva dan Leavens (2012), menyebutkan dengan istilah lain yaitu bentuk kesamaan representasional dan fungsional. Kesamaan sintak dilakukan dengan melihat bagaimana kode sumber itu disajikan atau dituliskan. Model pendekatan sintaksis hanya mempertimbangkan kesamaan tulisan dan minim memberikan informasi semantik (Keivanloo & Rilling, 2013). Sedangkan, pada kesamaan semantik, proses pencarian kloning kode dilakukan dengan melihat sisi maknanya. Makna dari sebuah fragmen kode dapat dilihat dari fungsinya (Jiang & Su, 2009; Elva & Leavens, 2012), dan hubungan relasi dari elemen-elemen kode tersebut (Gabel, Jiang, Su, 2008; Keivanloo & Rilling, 2013). Semantik kloning kode memiliki pengaruh terhadap peningkatan biaya pemeliharaan perangkat lunak. Oleh karena itu, kloning kode pada sebuah kode sumber perangkat lunak harus ditemukan dengan tujuan untuk membantu memfasilitasi proses manajemen kloning. Manajemen kloning dapat dilakukan dengan aktifitas refaktorisasi kode. Proses refaktorisasi kode tidak hanya dapat mengurangi biaya pemeliharaan perangkat lunak tetapi juga dapat mengurangi ketidak-konsistenan kode sumber (Keivanloo & Rilling, 2013). Ketidak-konsistenan kode sumber dapat menyebabkan cacat pada perangkat lunak.

Dalam proses pendeteksian kesamaan secara semantik, diperlukan adanya sebuah informasi tambahan untuk meningkatkan makna dari sebuah fragmen kode sumber. Proses pendeteksian kesamaan secara semantik tersebut telah diteliti oleh beberapa peneliti. Gabel et al (2008), mendefinisikan sebuah mekanisme pendeteksian kloning kode secara semantik. Pada penelitian tersebut terdapat informasi semantik yang didapatkan dengan menggunakan *Program Dependence Graphs* (PDGs). PDGs digunakan untuk merepresentasikan data dan kontrol dependensi antara statemen dan predikat dalam kode. Gobel et al (2008),

beranggapan bahwa fragmen kode isomorfis yang terlihat dari hasil representasi PDGs adalah kloning kode. Pendekatan berbasis PDGs dinyatakan sangat lambat dalam proses pembangkitan pasangan kloning. Pendekatan ini dinyatakan kurang akurat dalam mendefinisikan kode secara semantik. PDGs tidak mempertimbangkan urutan dari statemen kode, sehingga pada kasus kloning pada kode yang tidak berdekatan akan dinyatakan sebagai positif palsu (*false positive*) (Rattan et al, 2013). Gabel menyelesaikan permasalahan urutan statemen dengan melakukan transformasi dari PDGs menjadi sebuah *tree*.

Keivanloo dan Rilling (2013), mengusulkan sebuah pendekatan yang menekankan pada hubungan semantik pada setiap token. Mereka beranggapan bahwa, apabila hanya berdasarkan pada teks atau sintak akan sangat sulit untuk melakukan pengukuran kesamaan berdasarkan makna (semantik). Keivanloo dan Rilling melakukan perbaikan pada metode pendeteksian kloning berbasis sintak dengan cara menggunakan teknologi semantik web. Semantik web digunakan sebagai alat untuk menghimpun informasi relasi pewarisan pada kode sumber berbasis objek. Hubungan semantik pada token dilihat dari relasi warisan yang dipunyai.

Jiang dan Su (2009) menggunakan pendekatan yang berbeda. Dalam penelitiannya, Jiang dan Su mengartikan kesamaan semantik sebagai sebuah kesamaan fungsi (fungsional). Mereka berpendapat bahwa jika dua fragmen kode (pasangan kode) menghasilkan keluaran yang sama ketika mendapat masukan yang sama maka bisa disimpulkan bahwa kedua fragmen kode tersebut adalah sama secara fungsional. Dalam pendekatan ini, terdapat fase pembangkitan masukan yang berfungsi sebagai mendefinisikan masukan untuk pasangan fragmen kode secara otomatis.

Elva dan Leavens (2012) memiliki pandangan yang sejalan dengan Jiang dan Su (2009). Mereka memfokuskan penelitian pada metode atau fungsi pada bahasa pemrograman Java. Elva dan Leavens juga percaya bahwa fungsi dari sebuah metode dapat dilihat dari input dan outputnya. Elva dan Leavens melakukan penambahan parameter yang dipertimbangkan dalam mendeteksi kesamaan fungsi antara metode. Parameter tersebut adalah *effect* (efek) sehingga mereka menamai

metodenya dengan *IOE-Behaviour* (*Input, Output, Effect - Behaviour*). Input dilihat dari parameter masukan dari sebuah metode. Output dilihat dari tipe keluaran yang dihasilkan oleh sebuah metode. Sedangkan efek *(effect)*, dilihat dari *kelas* atau atribut apa yang mungkin dipengaruhi oleh metode tersebut ketika dijalankan.

Pada pendekatan Elva dan Leavens (2012), proses pembandingan kesamaan dilakukan dengan dua tahap filtrasi, tahap pertama proses filtrasi pada input dan output, dan tahap kedua proses filtrasi pada efek. Proses filtrasi pertama dilakukan dengan membandingkan tipe parameter input dan output. Proses filtrasi kedua adalah membandingkan informasi semantik yang sudah didefinisikan pada parameter efek. Proses filtrasi pertama menghasilkan kelompok kandidat klon berdasarkan kesamaan tipe input dan output. Input dan output diperlukan untuk melakukan uji coba pada tahap filtrasi kedua. Untuk metode yang tidak memiliki input dan/atau output, proses filtrasi berhenti pada tahap pertama. Metode-metode yang tidak memiliki input dan/atau output dinyatakan klon berdasarkan kesamaan informasi yang ada tanpa dilakukan uji coba. Metode void memungkinkan juga untuk mengandung kloning kode, oleh karena itu diperlukan juga untuk diteliti lebih dalam.

Dari beberapa kelemahan yang sudah dijelaskan sebelumnya, maka penulis mencoba untuk membuat sebuah langkah kerja dalam melakukan pendeteksian kloning kode. Langkah kerja tersebut bertujuan untuk melakukan deteksi kloning berdasarkan perilaku dari metode. Perilaku dilihat dari input, output dan efek. Pada input atau output, tipe-tipe parameter akan dilihat lebih detail, baik itu berupa kelas, tipe primitif, atau hubungan turunan antar kelas. Sedangkan pada efek, akan dianalisis lebih mendalam. Efek tidak hanya dilihat dari kelas yang memiliki metode saja, melainkan hierarki kelas tersebut juga dipertimbangkan. Pendefinisian input dan output pada metode *void* akan digali, sehingga metode *void* tidak diabaikan. Dengan melakukan pencarian kloning kode pada seluruh metode yang ada pada sistem, maka diharapkan kloning dapat ditemukan pada seluruh bagian dari kode sumber.

1.2 Perumusan Masalah

Perumusan masalah pada penelitian ini meliputi beberapa hal yang dijelaskan sebagai berikut.

a. Bagaimana mendefinisikan tipe data input, output dan efek pada seluruh metode baik *void* maupun *non-void*?
b. Bagaimana melakukan pencocokan perilaku antar metode?

1.3 Tujuan dan Manfaat Penulisan

Tujuan penelitian yang ingin dicapai adalah sebagai berikut.

a. mendefinisikan tipe data input, output dan efek pada seluruh metode baik *void* maupun *non-void*.
b. Melakukan pencocokan perilaku antar metode.

Manfaat yang diberikan adalah agar seorang pengembang perangkat lunak dapat dengan mudah menemukan kloning kode berdasarkan kesamaan semantik. Dimana, pada kloning kode tersebut akan dilakukan proses refaktorisasi kemudian. Kontribusi dalam penelitian ini adalah proses pencocokan kesamaan perilaku antar metode berdasarkan input, output dan efek dengan tidak mengabaikan metode void.

1.4 Batasan Masalah

Dalam penelitian ini, pembahasan yang dicakup adalah sebagai berikut.

a. Proses deteksi kloning kode dilakukan pada bahasa Java.
b. Proses deteksi kloning kode dilakukan pada seluruh metode yang ada pada kode sumber.

[Halaman ini sengaja dikosongkan]

BAB 2
KAJIAN PUSTAKA

2.1 Kloning Kode

Dalam kode sumber, sebuah fragmen kode yang disalin dan disisipkan kebagian lain dengan atau tanpa modifikasi adalah kloning kode (Rattan et al, 2013). Sepasang fragmen kode dinyatakan kloning apabila dinyatakan sama berdasarkan beberapa definisi similaritas. Kloning kode ada dalam kode sumber dalam dua bentuk kesamaan, yaitu kesamaan secara sintak (representasional) dan kesamaan secara semantik (fungsional) (Elva & Leavens, 2012). Kesamaan kode berdasarkan sintak dilihat dari bagaimana kode itu dituliskan. Proses pembandingan kode dilakukan melihat kesamaan tulisan atau sintak. Kode dinyatakan sama berdasarkan semantik atau fungsional apabila kode tersebut memiliki kesamaan secara makna atau fungsinya.

2.1.1 Tipe-Tipe Kloning

Dalam survei yang dilakukan oleh Rattan et al (2013), terdapat beberapa tipe kloning. Berikut ini adalah beberapa tipe kloning yang sudah diidentifikasi dari beberapa penelitian.

a. Kloning Tipe 1 (*exact clones*)
   Kloning tipe pertama memiliki sebutan lain yaitu *exact clones* atau kloning yang tepat sama. Pada kloning tipe ini, sepasang fragmen yang memiliki tulisan yang sama persis kecuali adanya penambahan spasi dan komentar.
b. Kloning Tipe 2 (*renamed/parameterized clones*)
   Kloning tipe 2 adalah kloning struktur, yang memiliki struktur yang sama kecuali ada perubahan pada penamaan penanda (*identifier*), literal, tipe data, layout dan komentar.
c. Kloning Tipe 3 (*near miss clones*)
   Kloning tipe 3 adalah kloning tipe 2 yang disertai dengan menambahan atau penghapusan pernyataan (*statement*) atau baris kode.

d. Kloning Tipe 4 (*semantic clones*)

   Kloning tipe 4 adalah kloning semantik. Kloning tipe ini menyatakan kesamaan secara fungsionalitas dengan tidak disertai kesamaan secara sintak atau tulisan.

e. Kloning Fungsi (*Function Clone*)

   Kloning yang terbatas pada level granularitas fungsi/metode atau prosedur.

f. Kloning Berbasis Model (*Model Based Clones*)

   Pada proses pengembangan perangkat lunak terdapat fase desain. Representasi sistem dalam sebuah model akan lebih mudah proses penyususnan kode. Kloning model melihat adanya duplikasi pemodelan yang dilakukan pada fase desain.

2.1.2 Alasan Kloning

Terdapat beberapa hal yang dapat menyebabkan terjadinya kloning pada fragmen kode. Kim et al (2004), telah melakukan studi etnografi terhadap pengembang perangkat lunak untuk menemukan alasan melakukan kloning kode. Alasan kloning kode antara lain adalah, terdapat keterbatasan dalam memahami bahasa program dan keterbatasan waktu memaksa seorang pengembang melakukan penyalinan dan penyisipan fragmen kode. Selain itu, menurut Kim et al (2004), pengembang sering menggunakan fragmen kode sebagai contoh (*template*) untuk disalin dan disisipkan di bagian kode sumber yang lain.

Tidak hanya Kim et al (2004), Kapser dan Godfrey (2008), melakukan investigasi pada system yang besar dan menemukan yang mereka sebut sebagai pola-pola kloning (*Patterns of Cloning*). Pola-pola kloning itu dijelaskan sebagai berikut ini.

a. *Forking* (Percabangan)

   Forking adalah proses kloning yang dilakukan dengan melakukan penyalinan kode yang diletakkan pada file sumber yang berbeda. Kemudian, file tersebut akan dikembangkan secara independen. Hal tersebut dilakukan dengan tujuan untuk mempercepat proses pengembangan perangkat lunak.

b. *Templating* (Contoh)

   Template digunakan sebagai metode untuk melakukan penyalinan perilaku kode secara langsung ketika mekanisme abstraksi tidak ada. Template dilakukan apabila terdapat beberapa hal yang harus dibagi (*share*) antar kloning seperti perilaku dan *library*.

c. *Customization* (Penyesuaian)

   Proses *customization* atau penyesuaian dilakukan ketika sebuah kebutuhan tidak dapat dipenuhi dengan cara kloning secara langsung. Hal tersebut akan terpenuhi apabila dilakukan penyesuaian terhadap fragmen kode.

2.1.3 Akibat Kloning

Kloning telah menjadi sebuah kebiasaan yang sering dilakukan oleh pengembang perangkat lunak. Aspek waktu dan kepraktisan adalah salah satu motivasi utama dalam melakukan kloning (Rattan et al, 2013). Namun, ternyata proses kloning memiliki dampak. Kloning dapat meningkatkan usaha dan biaya dalam proses pemeliharaan (Rattan et al, 2013; Mens & Demeyer, 2008). Kloning harus diperlakukan secara konsisten. Perubahan pada sebuah fragmen harus disertai dengan perubahan pada fragmen yang lain. Permasalahan muncul pada tahap ini, pencarian fragmen kloning pada sistem yang besar sangat sulit untuk dilakukan, terlebih lagi apabila fragmen tersebut telah mengalami modifikasi. Proses menganalisis pasangan fragmen kode bukanlah sebuah pekerjaan yang mudah. Proses tersebut membutuhkan ketelitian dan pemahaman terhadap alur program. Oleh karena itu analisis terhadap pasangan fragmen adalah proses yang sangat mahal.

Selain isu tentang meningkatnya usaha dan biaya pemeliharaan, Rattan et al (2013), dalam rangkumannya menjelaskan bahwa terdapat beberapa hal yang mungkin disebabkan oleh kloning. Akibat yang mungkin terjadi adalah sebagai berikut ini.

a. Perambatan kesalahan (*Bug Propagation*)

   Bug dapat dirambatkan melalui proses kloning. Apabila fragmen kode yang disalin dan disisipkan pada kode sumber mengandung *bug*, maka *bug*

tersebut juga akan terduplikasi pada fragmen yang lain.

b. Desain yang buruk

   Kloning dapat menjadi penghambat penerapan syarat-syarat desain yang baik seperti pewarisan dan refaktorisasi.

c. Sistem sulit untuk dipahami

   Kloning membuat sebuah sistem jauh dari terwujudnya sistem yang modular. Keadaan ini membuah sebuah sistem sulit untuk dipahami. Proses pemeliharaan juga akan sulit untuk diterapkan.

d. Kode sumber membengkak

   Kloning membuat kode sumber membengkak. Keadaan membengkaknya kode sumber dapat mengakibatkan menurunnya performa sistem dilihat dari waktu eksekusi dan kebutuhan ruang penyimpanan.

2.1.4 Manajemen Kloning

Manajemen kloning bertujuan untuk mengindentifikasi, mengorganisir, mengontrol pertumbuhan dan menghindari terjadinya kloning (Mens & Demeyer, 2008). Lague et al (1997), membedakan tiga hal utama dalam manajemen kloning adalah sebagai berikut ini.

a. Pencegahan (*preventive*), adalah proses untuk mencegah sebuah kloning dapat terjadi pada proses pengembangan perangkat lunak.
b. Penambangan masalah (*problem mining*), adalah aktivitas untuk melakukan pemeliharaan terhadap kloning yang ada pada kode sumber dari kemungkinan munculnya akibat yang negatif.
c. Pembenaran, adalah usaha untuk mencari dan menghapus kloning yang ada pada kode sumber.

2.2 Semantik

Dalam pembahasan tentang semantik, akan dititikberatkan pada sumber-sumber yang menjelaskan tentang definisi semantik. Pembahasan tentang semantik dimulai dengan pembahasan tentang arti semantik, informasi dan makna dan semantik pada kode.

2.2.1 Arti Semantik

Istilah semantik lebih sering dikaitkan dengan bahasa. Dalam ilmu bahasa, bahasa tergantung dari kata dan kalimat yang memiliki makna (Kempson, 1977). Makna sebuah kalimat tergantung dari makna dari masing-masing kata dalam kalimat tersebut. Hubungan antara makna setiap kata dalam kalimat akan memperkaya arti sebuah kalimat. Semakin banyak kata yang menyusun sebuah kalimat, maka makna dari sebuah kalimat semakin beragam.

Semantik dapat diartikan sebagai makna. Sesuatu dilihat secara semanik berarti bahwa sesuatu tersebut dilihat dengan menggali maknanya. Banyak metode yang digunakan untuk menggali makna sesuatu. Seperti dengan menangkap seluruh informasi yang ada pada sesuatu tersebut.

2.2.2 Informasi dan Makna

Informasi dan makna adalah dua hal yang sangat penting dalam kehidupan kita. Sesuatu dapat bermakna apabila kita dapat menginterpretasikan informasi. Informasi adalah bentukan data yang dikemas sedemikian rupa. Dengan demikian, data, informasi dan makna adalah hal-hal yang saling berkaitan.

Seekor kucing akan selalu tertarik pada aroma daging walaupun daging tersebut ditempatkan pada tempat yang tertutup. Informasi aroma daging sangat bermakna bagi seekor kucing. Berbeda dengan sepotong roti, aroma roti tersebut tidak akan bermakna bagi kucing walaupun kita tempatkan roti tersebut di depan seekor kucing. Informasi akan bermakna tergantung pada siapa yang menginterpretasikan informasi.

Konsep Piercean yang dituliskan oleh Menant (2010), membahas tentang sebuah konsep tanda atau informasi. Teori tersebut menyatakan bahwa terdapat tiga komponen dalam konsep Piercean, yaitu objek pemikiran, tanda/ informasi yang merepresentasikan objek, dan penafsir (*interpretant*). Makna akan muncul tergantung pada penafsir. Dengan kata lain bahwa, data yang terbentuk menjadi informasi akan memiliki makna tergantung bagaimana penafsir menafsirkannya (Menant, 2010). Penilaian secara semantik berhubungan dengan data, informasi

dan penafsiran, karena makna sangat bergantung pada ketiga hal tersebut.

2.2.3 Semantik pada Kode

Semantik juga digunakan dalam pengembangan perangkat lunak. Salah satu yang menerapkan prinsip semantik adalah proses deteksi kloning kode dalam kode sumber. Dalam deteksi kloning kode, kesamaan kode dapat dilihat dengan menerapkan prinsip dasar semantik. Untuk dapat melihat kesamaan kode dengan cara semantik, maka seluruh informasi yang ada pada kode tersebut harus digali. Beberapa peneliti telah melakukan penelitian tentang deteksi kloning kode dengan pendekatan semantik (Gabel et al ,2008; Jiang dan Su, 2009; Elva dan Leavens, 2012; Keivanloo dan Rilling, 2013).

Banyak cara yang digunakan untuk menggali informasi semantik dari dalam kode sumber. Pada tabel 2.1 dijelaskan rangkuman penelitian terdahulu oleh beberapa peneliti tentang semantik kloning.

Tabel 2. 1 Penelitian Terdahulu Tentang Semantik Kloning

Peneliti	**Tahun**	**Metode**	**Kelebihan & Kelemahan**
Gabel et al	2008	PDG, *Abstract Syntac Tree* (AST)	Kelebihan : Definisi semantik dilihat dari makna generalisasi dari sintak kode dengan PDG. Pemecahan bagian grap dari PDG untuk memperkecil objek teliti. Hal ini bertujuan untuk mengurangi kompleksitas proses deteksi. Proses deteksi dilakukan dengan translasi dari bagian grap menjadi AST. Kekurangan :

			Proses generalisasi kode tidak terlalu memunculkan makna secara semantik, karena dalam representasi PDG masih tergambar sintak kode. Proses deteksi dilakukan dalam waktu yang lama, karena proses transformasi dari PDG ke AST.
Jiang & Su	2009	Input, output behavior	Keuntungan : Kesamaan antara dua fragmen kode dilihat dari input dan output. Kekurangan : Proses pemotongan fragmen kode yang sama masih didasarkan pada kesamaan urutan kode.
Elva & Leavens	2012	Input, output, effect behavior	Keuntungan : Kesamaan antara dua fragmen kode tidak dipengaruhi oleh bentuk struktur kode atau sintak. Kesamaan fungsional tidak hanya dilihat dari input dan output saja, juga dilihat dari efek. Fokus analisis dilakukan pada metode dari sebuah kelas. Kekurangan : Terdapat kemungkinan kesalahan dalam memprediksi kesamaan. Secara sintaktik, sepasang tipe input atau output dapat diidentifikasi berbeda secara sintak

			namun pada aplikasinya mungkin tidak memiliki perbedaan fungsi, misalnya *Double* dan *double*. Proses analisis efek yang terlalu konservatif, sehingga membuat dua buah metode dibedakan padahal metode-metode tersebut memiliki efek yang tidak terlalu berbeda. Tipe output berupa *void* akan diabaikan
Kong et al	2012	AST, PDG, K-nearest neighbor clustering	Keuntungan : Kesamaan kode dilihat dari nilai input dan output yang dibangkitkan secara otomatis. Kekurangan : Salah satu faktor dalam penentuan kelompok fragmen kode adalah tingkat kohesi dari fragmen-fragmen kode. Dua fragmen akan memiliki koefisien kemiripan yang tinggi apabila saling berbagi atribut. Kemiripan fungsional tidak selalu dilihat dari tingkat kohesi dari sebuah kode.

2.2.4 Klon Semantik

Terdapat banyak definisi yang menjelaskan tentang klon secara semantik. Beberapa peneliti telah mendefinisikan pengertian tentang klon semantik. Penelitian ini menggunakan definisi klon semantik yang dijelaskan oleh Elva dan Leavens.

Elva dan Leavens menjelaskan bahwa dua buah metode dapat dinyatakan sama secara semantik jika keduanya mempunyai perilaku yang sama dan dapat diobservasi. Perilaku dari sebuah metode adalah nilai input, output dan deskripsi efek. Oleh karena itu, Elva dan Leavens menamai pendekatan yang diusulkan menjadi *IOE-Behavior* (Elva dan Leavens, 2012). Input merujuk pada nilai dari setiap parameter masukan ketika metode dipanggil. Output merujuk pada nilai balik dari sebuah metode atau hasil dari proses pemanggilan metode. Dan, Efek merujuk pada perubahan yang mungkin terjadi pada atribut class yang terkait pada metode yang sedang berjalan. Seluruh nilai dari input, output dan efek diamati dengan seksama. Berdasarkan penelitian Elva dan Leavens, kesamaan secara semantik dari dua buah metode dapat dijelaskan sebagai berikut ini.

a. $\forall A \, \forall B \mid A, B \in Methods$
$IOE - Behavior(A) = IOE - Behavior(B) \leftrightarrow$
$SemanticEquivalent(A, B)$ (1)

b. Dua buah metode sama secara semantik apabila berada pada kontek yang sama. A dan B dapat dinyatakan sama dalam kontek jika diberikan nilai input dengan kontek yang sama.

```
public int multiply1(int a, int b)
{
  int result = a * b;
  return result;
}
```

```
public int multiply2(int a, int b)
{
   int result = 0;
   for (int i = 0; i < b; i++)
   {
      result += a;
   }
   return result;
}
```

Gambar 2. 1 Contoh Kandidat Klon

Gambar 2.1 menjelaskan dua buah metode yang menjadi kandidat klon (multiply1 and multiply2). Kedua metode memiliki kode yang berbeda. Tetapi, kedua metode tersebut memiliki kesamaan dalam hal tipe data input dan output. Kedua metode tersebut dapat diklasifikasikan sebagai kandidat klon. Kedua

metode tersebut akan sama secara semantik apabila memiliki perilaku yang sama. Pemanggilan semua metode kandidat klon adalah cara untuk melihat perilaku dari metode. Hasil pencatatan nilai input dan output pada proses pemanggilan metode dijelaskan pada tabel 2.2.

Tabel 2. 2 Pencatatan Nilai Input dan Output

No.	Method's Name	Input		Output (int)
		A (int)	B (int)	
1.	multiply1	2	4	8
	multiply2	2	4	8
2.	multiply1	3	6	18
	multiply2	3	6	18
3.	multiply1	5	2	10
	multiply2	5	2	10

Dari tabel 2.2, terlihat telah dilakukan tiga kali percobaan pemanggilan metode. Setiap kali percobaan, kedua metode tersebut diberikan data input yang sama untuk menjaga agar kedua metode tersebut berada pada kontek yang sama. Tabel 2.2 menjelaskan bahwa pada setiap percobaan, data terlihat sama pada input dan output. Kondisi ini dapat diartikan bahwa kedua metode tersebut (multiply1 dan multiply2) memiliki kesamaan perilaku sesuai dengan definisi yang disampaikan oleh Elva dan Leavens. Dan pada akhirnya, kedua metode tersebut dapat dinyatakan sebagai secara semantik sama.

2.3 *Object Oriented Programming (OOP)*

Object Orented Programming (OOP) merupakan sebuah paradigm pemrograman yang mengenalkan tentang konsep objek di dalam sebuah sistem. Dalam pembangunan sebuah sistem perangkat lunak dengan menggunakan pendekatan objek, objek merupakan satuan terkecil di dalam sistem (Miller & Kasparian, 2006). Objek merupakan cerminan dari sesuatu yang ada di kehidupan nyata yang digunakan di dalam sistem perangkat lunak. Dalam satu objek mengandung data dan perilaku yang kemudian dinamakan metode.

Metode OOP menyuguhkan fleksibilitas dalam proses pengembangan perangkat lunak. Representasi objek dalam OOP membuat metode ini lebih modular dan mudah untuk dimodifikasi. Modifikasi implementasi pada sebuah objek atau penambahan fungsi servis dari sebuah objek tidak akan mempengaruhi sistem objek yang lain. Karena objek merupakan sesuatu, dimana merepresentasikan bentuk nyata dari kehidupan nyata maka hal ini dapat meningkatkan kemudahan dalam hal pemahaman dan perawatan (Sommerville, 2011). Beberapa hal yang harus diperhatikan dalam melakukan pengembangan sistem dari konsep hingga terperinci dengan pendekatan OOP adalah sebagai berikut (Sommerville, 2011).

1. Memahami dan mendefinisikan kontek dan interaksi eksternal dengan sistem;
2. Merancang arsitektur sistem;
3. Mengidentifikasi objek penting dalam sistem;
4. Membangun model sistem; dan
5. Menentukan antarmuka objek yang berfungsi sebagai media komunikasi antar objek.

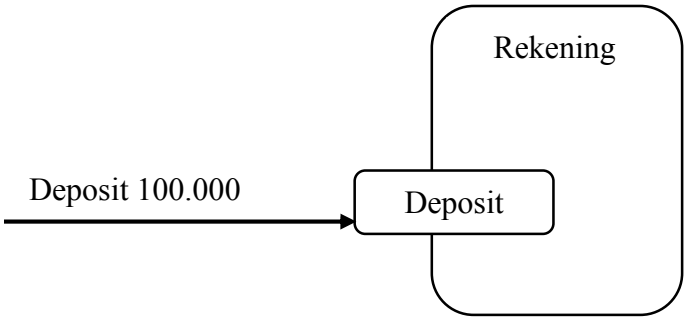
2.3.1 Kelas dan Objek

Dalam pendekatan OOP, istilah kelas dan objek merupakan hal yang mutlak harus diketahui oleh pengembang sistem. Perbedaan antara kelas dan objek seringkali menjadi pertanyaan diantara beberapa pengembang.

Objek adalah sesuatu yang berwujud atau tidak berwujud yang dapat dibayangkan (Wu, 2009). Sebuah sistem yang dikembangkan dengan berorientasi objek akan mengandung banyak objek yang saling berintaraksi. Objek tidak akan terwujud tanpa adanya kelas. Kelas adalah definisi objek yang telah dijelaskan sebelum objek tersebut dibentuk (Wu, 2009). Dengan kata lain, kelas adalah cetakan dari objek. Proses pembentukan objek disebut sebagai proses instansiasi objek. Instansiasi dari sebuah kelas menghasilkan instan objek. Seluruh instan objek adalah milik dari kelas.

2.3.2 Pesan dan Metode

Objek adalah hasil dari proses instansiasi sebuah kelas. Di dalam kelas terdapat data dan perilaku yang diwujudkan dalam bentuk metode. Untuk dapat menginstruksikan sebuah objek melakukan sesuatu sesuai dengan perilaku yang sudah didefinisikan, maka pesan (*message*s) harus dikirimkan kepada objek tersebut (Wu, 2009). Pesan yang dikirimkan ke sebuah objek adalah pemicu untuk sebuah objek untuk melakukan sesuatu. Pesan tidak dapat hanya sekedar dikirimkan kepada objek. Objek harus memahami apa pesan yang sampai pada dirinya. Pesan yang dikirimkan dituliskan sesuai dengan nama metode yang dipunyai oleh objek dan disertai dengan argument (disesuaikan dengan metode). Objek yang mendapatkan pesan harus menjalankan metode yang cocok dengan pesan yang diterima. Pada gambar 2.2 adalah contoh dari proses pengiriman pesan ke sebuah objek.



Gambar 2. 2 Contoh Pengiriman Pesan ke Sebuah Objek

Pada contoh tersebut Rekening merupakan nama kelas, dan deposit adalah metode yang dipunyai oleh Rekening. Untuk memerintahkan objek dari kelas Rekening melakukan proses deposit maka harus dikirimkan pesan ke objek tersebut. Pesan dikirimkan dengan menuliskan nama metode yang akan dijalankan yang disertai dengan argumen. Argumen adalah data yang dikirimkan ke sebuah objek untuk menjadi data masukan ke metode yang dipanggil. Dalam contoh pada gambar 2.2, pengiriman pesan dituliskan dengan "Deposit 100.000", yang dapat diartikan sebagai perintah ke objek Rekening untuk menjalankan metode Deposit dengan parameter masukan adalah 100.000.

Argumen yang dikirimkan menunjukkan data yang akan diproses di dalam metode Deposit.

2.3.3 Pewarisan

Pewarisan atau pewarisan adalah salah satu fitur dalam OOP. Perilaku yang ada pada satu kelas dapat diadopsi atau diperluas oleh kelas lain. Menurut Miller dan Kasparian (2006), ada tiga cara untuk melakukan pewarisan, yaitu :

1. Memperluas fungsionalitas dari kelas yang ada,
2. Mengimplementasikan satu atau lebih *interface* (kelas antarmuka), atau
3. Mengimplementasikan keduanya dengan memperluas fungsi dari kelas yang ada dan mengimplementasikan satu atau lebih *interface*.

Pewarisan memiliki tiga tujuan penting dalam proses pengembangan perangkat lunak dengan OOP. Tiga tujuan tersebut adalah sebagai berikut :

1. Memungkinkan seorang pengembang untuk berfikir tentang generalisasi dan spesialisasi sebuah objek yang dirancang,
2. Menyediakan cara untuk melakukan pengukuran penggunaan kembali kode (*reuse*) dalam struktur objek,
3. Memungkinkan seorang pengembang dapat mengembangkan kode secara bertahap.

Generalisasi dan spesialisasi telah digunakan secara luas pada beberapa disiplin. Generalisasi dan spesialisasi kadang-kadang memiliki kontek atau makna khusus akan sesuatu hal. Sebagai contoh, terdapat kosep A yang merupakan generalisasi dari B, dan sebaliknya, B adalah spesialisasi dari A. Hal tersebut terjadi jika, (1) seluruh konsep instan B adalah konsep instan A, dan (2) tidak semua konsep instan A adalah B. Contoh pada kasus nyata, misalnya terdapat hewan dan burung, setiap konsep pada burung adalah hewan, tetapi tidak semua hewan adalah burung. Namun, secara makna, konsep burung adalah hewan, dan hewan tidak selalu burung. Hubungan generalisasi dan spesialisasi ini akan menambah makna dari sebuah objek.

Makna generalisasi dan spesialisasi tergambar pada hubungan hypernim dan hyponim pada kata. Hypernim adalah bentuk generik dari hyponim. Hewan

merupakan memiliki makna hypernim dari burung, begitu sebaliknya. Hal ini memberikan gambaran bahwa generalisasi dan spesialisasi mengandung makna hubungan dari dua konsep.

2.4 *Program Dependence Graphs* (PDG)

Program Dependence Graphs (PDG) adalah representasi program tingkat menengah yang secara eksplisit menggambarkan hubungan ketergantungan baik data maupun kontrol untuk setiap operasi dalam program (Ferrante et al, 1987). Grafik ketergantungan data menyuguhkan secara eksplisit hubungan ketergunaan antar data yang secara implisit terdapat di dalam kode sumber. Sedangkan grafik ketergantungan kontrol menggambarkan alur kontrol dalam kode sumber. PDG mewakili kedua hubungan data dan kontrol tanpa adanya penggambaran urutan proses yang dilakukan oleh kode sumber.

Hubungan ketergantungan dilihat dari dua hal. Pertama, ketergantungan yang ada antara dua pernyataan (*statement*) kode. Sebuah variabel di dalam pernyataan akan bernilai salah apabila urutan pernyataan yang memuat variabel tersebut disusun terbalik. Ferrante et al (1987), mengilustrasikan model ketergantungan ini sebagai berikut ini :

```
A = B * C              (S1)
D = A * E + 1          (S2)
```

`S1` adalah pernyataan pertama dan `S2` adalah pernyataan kedua. Dalam contoh tersebut, apabila `S2` diposisikan sebelum `S1` maka variabel `A` akan bernilai salah atau tidak sesuai dengan apa yang diharapkan. Hubungan ketergantungan tipe ini adalah hubungan ketergantngan data. Kedua, hubungan ketergantungan yang terdapat antara pernyataan dan predikat. Ilustrasi Ferrante et al (1987) adalah sebagai berikut :

```
If (A) then            (S1)
 B = C * D             (S2)
```

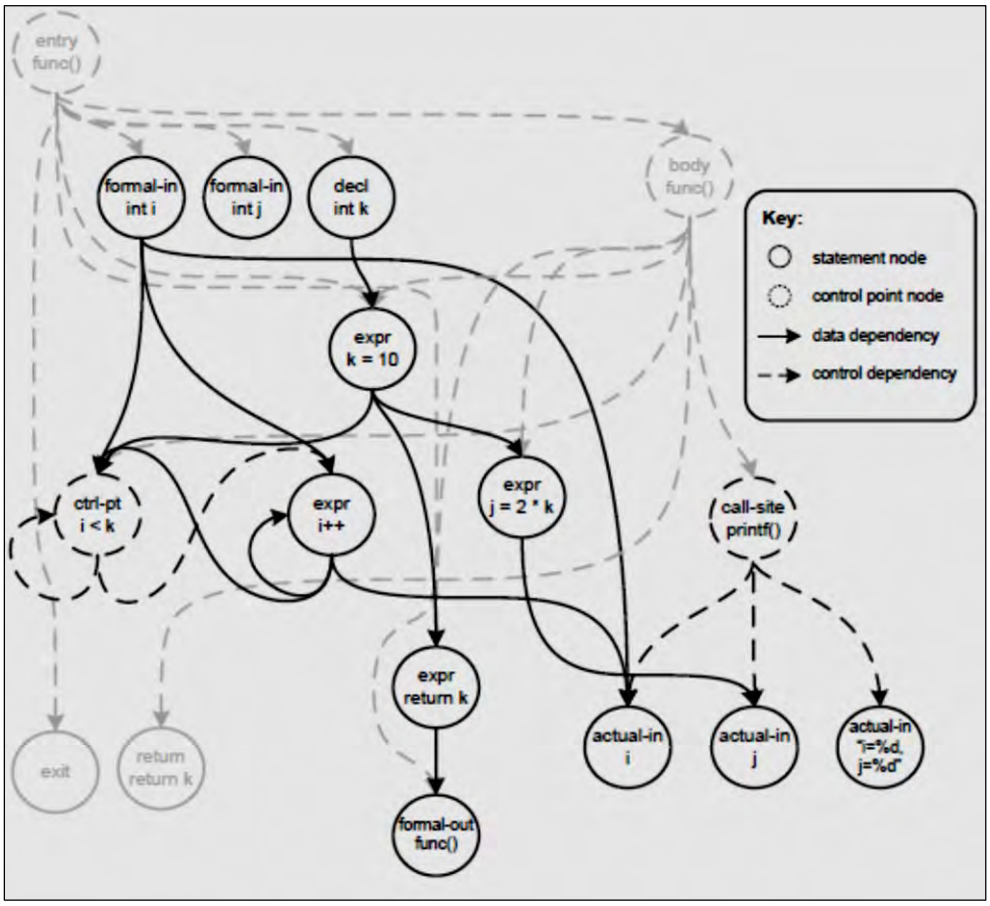
```
Endif
```

S2 tergantung pada variabel A pada S1, karena nila yang terkandung dalam A akan menentukan apakah S2 akan dijalankan atau tidak. Hubungan ketergantungan model ini dinamakan hubungan ketergantungan kontrol. Contoh transformasi dari sebuah potongan kode menjadi PDG digambarkan dalam gambar 2.3 dan gambar 2.4 berikut ini.

```
1  int func(int i, int j) {
2    int k = 10;
4    while (i < k) {
5      i++;
6    }
8    j = 2 * k;
10   printf("i=%d,  j=%d\n", i, j);
11   return k;
12 }
```

Gambar 2. 3 Potongan kode (Gebel et al, 2008)

Gambar 2.3 menjelaskan sebuah fungsi yang bernama `func` dan memiliki dua parameter bertipe integer `i` dan `j`. Dalam fungsi tersebut terdapat deklarasi dan inisialisasi variabel `k`. Nilai pada variabel `k` digunakan pada beberapa pernyataan dalam kode tersebut. k dan I menjadi penentu dalam proses perulangan (*loop*) pada baris ke 4 dan menjadi salah satu faktor penentu nilai dari variabel `j` pada baris 8. Dalam beberapa penjelasan tersebut terdapat hubungan ketergantungan antara variabel. Hubungan ketergantungan tersebut dijelaskan dalam bentuk grafik pada gambar 2.4. Tidak hanya itu saja, gambar 2.4 juga menunjukkan alur kontrol yang digambarkan dengan garis hubung putus-putus.



Gambar 2. 4 Hasil transformasi ke PDG dari Gambar 2.1 (Gebel et al, 2008)

2.5 Input, Output dan Efek pada Metode

Terdapat beberapa data yang terdapat pada metode yang dapat dikemas menjadi sebuah informasi yang bermakna. Informasi-informasi itu antara lain adalah parameter input, output dan efek yang terjadi ketika metode dijalankan. Ketiga hal tersebut adalah informasi yang dapat digunakan untuk memperkaya makna dari metode sebuah kelas.

2.5.1 Input dan Output pada Metode

Dalam sebuah metode, tipe data input dan output dapat diketahui dari sintak pendefinisian metode. Argumen yang dipunyai metode dapat digunakan menjadi input dan nilai kembali dari sebuah metode dapat digunakan sebagai

output (Jiang & Su, 2009). Input juga dapat diketahui dari status dari *heap* ketika metode tersebut dipanggil (Elva & Leavens, 2012). Pada metode *void*, metode yang tidak mempunyai parameter input atau nilai kembalian, proses identifikasi input dan output dapat dilakukan dengan melihat aliran data pada fragmen kode (Jiang & Su, 2009).

2.5.2 Efek pada Metode

Efek adalah dampak yang dihasilkan dalam menjalankan sesuatu. Sebagai contoh adalah, seseorang melakukan pembayaran terhadap pembelian online melalui internet banking. Hasil dari kegiatan tersebut adalah pembelian mempunyai status lunas, sedangkan dampak atau efek dari kegiatan tersebut adalah saldo simpanan pembeli dalam bank berkurang. Tapi pada suatu kasus tertentu, efek dapat didefinisikan sebagai output (Elva & Leavens, 2012). Pada penelitian Elva & Leavens (2012), efek diartikan sebagai gabungan antara atribut dan nama kelas yang dikelola oleh sebuah metode. Contoh pada gambar 2.5 menggambarkan tentang sebuah efek dari sebuah metode.

```
class acct
{
      double balance;
      double rate;

      double calcInt()
      {
            return balance * rate;
      }
}
```

Gambar 2. 5 Contoh Metode

Dari gambar 2.5, terdapat kelas `acct` dengan atribut `balance` dan `rate` yang masing-masing bertipe `double`. Kelas `acct` mempunyai satu metode bernama `calcInt` yang memiliki nilai kembalian bertipe `double`. Dengan definisi yang dijelaskan Elva dan Leavens (2012), maka efek dari metode `calcInt` adalah `acct.balance` dan `acct.rate`.

Pada pendekatan yang lain, Rountev (2004), menyatakan bahwa efek

metode adalah perubahan status yang dapat diamati dari kode yang memanggil metode tersebut. Rountev (2004), menggunakan analisa kelas untuk melakukan identifikasi hubungan memanggil antar metode. Terdapat beberapa pandangan tentang bagaimana mendapatkan efek dari sebuah metode, yaitu dengan berorientasi di dalam kelas (Elva & Leavens, 2012), dan berorientasi dari luar kelas yang sedang diamati (Rountev, 2004). Kedua peneliti tersebut (Elva & Leavens, 2012; Rountev, 2004), tidak mendefinisikan jenis-jenis atau kategori berdasarkan suatu kriteria apapun.

2.6 Koefisien *Cohen's Kappa*

Koefisien *Cohen's Kappa* diusulkan oleh Jacob Cohen pada tahun 1960 adalah koefisien untuk mengevaluasi kesepakatan antara beberapa penilai atau dua metode penilaian. *Cohen's Kappa* adalah sebuah metode pengukuran kebenaran dari data (Sim & Wright, 2005). Koefisien ini dapat dirumuskan pada (Sim & Wright, 2005) :

$$K = \frac{P_o - P_c}{1 - P_c} \qquad (2)$$

dimana, P_o adalah proporsi yang kesamaan pengamatan dan P_c adalah proporsi yang diharapkan secara kebetulan. Kemudian, data yang didapat dari hasil pengamatan dari dua pengamat digambarkan dalam bentuk tabel relevansi seperti pada tabel 2.2.

Tabel 2. 3 Tabel Relevansi

Pengamat		**II**		**Total**
		Relevan	**Tidak Relevan**	
I	**Relevan**	a	b	g_1
	Tidak Relevan	c	d	g_2
Total		f_1	f_2	n

Dari gambaran pada tabel 2.2, P_o didapatkan dengan menjumlahkan nilai pada a dan d, dan dibagi dengan n. P_o dirumuskan sebagai berikut (Sim & Wright, 2005):

$$P_o = \frac{a+d}{n} \quad (2)$$

dimana a dan d masing-masing adalah kesamaan relevan dan tidak relevan, dan n adalah total data. Sedangkan P_c didapatkan dengan menggunakan rumus berikut ini (Sim & Wright, 2005):

$$P_c = \frac{\left(\frac{f_1 x g_1}{n}\right)+\left(\frac{f_2 x g_2}{n}\right)}{n} \quad (3)$$

sehingga dengan mendapatkan nilai P_o dan P_c, maka nilai K (Cohen's Kappa) dapat dihitung. Nilai kappa dapat menentukan tingkat kesepakatan antar pakar dengan sistem. Tabel 2.4 menjelaskan interpretasi nilai Kappa.

Tabel 2. 4 Interprestasi Nilai Kappa (Landis & Koch, 1977)

Indeks Kappa	Proporsi Kesepakatan
< 0	Rendah (*less than chance agreement*)
0.01 – 0.20	Sedikit (*slight agreement*)
0.21 – 0.40	Cukup (*fair agreement*)
0.41 – 0.60	Sedang (*moderate agreement*)
0.61 – 0.80	Substansial (*substansial agreement*)
0.81 – 1	Hampir Sempurna (*almost perfect agrement*)

2.7 Analisis *ByteCode*

Analisis terhadap kode sumber dapat dilakukan dengan beberapa cara. Analisis kode dapat dilakukan langsung pada kode sumber dan *byte-code*. Analisis kode terhadap kode sumber biasanya dilakukan dengan melakukan proses penguraian terhadap string kode. Analisis kode pada *byte-code* dilakukan terhadap

kode program yang sudah terkompilasi.

Satu baris kode dalam kode sumber terdiri dari beberapa instruksi *byte-code*. Masing-masing instruksi tersebut dipetakan berdasarkan baris yang sesuai dengan posisi baris pada kode sumber. Pemetaan instruksi byte-code dicatat dalam sebuah tabel yang bernama LineNumberTable. Keuntungan dalam analisis kode pada byte-code adalah hasil yang lebih akurat, terutama pada proses pemotongan (*slicing*) kode. (Walkinshaw et. al., 2003).

2.8 *Java System Dependence Graph API*

Java System Dependence Graph API (SDGAPI) adalah sebuah *Application Programming Interface* yang berfungsi untuk membangkitkan sebuah grafik ketergantungan antara elemen-elemen pada kode sumber. Grafik ini biasanya digunakan untuk proses analisis pada elemen-elemen kode sumber.

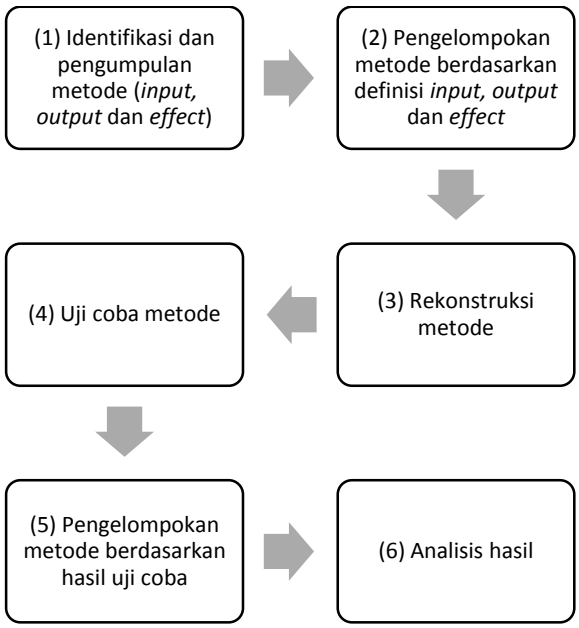
SDGAPI dikembangkan dengan dua jenis. Pertama adalah *Procedural Dependence Graph* (PDG). PDG merupakan grafik yang merepresentasikan prosedur atau metode dalam sebuah program. Setiap garis penghubung menunjukkan proses penetapan atau penugasan (*assignment*) sebuah statemen kode dan alur kontrol yang terjadi pada program. PDG terdiri dari *Data Dependence Edges* dan *Control Dependence Edges*. PDG merupakan komponen penting dalam pembentukan *System Dependence Graph* (SDG). Kedua adalah SDG, SDG merupakan representasi ketergantungan pada level granularitas yang lebih tinggi. Pada SDG tidak lagi dilihat bagaimana data berjalan tetapi melihat pada pemanggilan metode, ketergantungan kelas, hubungan pewarisan kelas, ketergantungan interface dan polimorfisme (Tong, 2009).

BAB 3
METODOLOGI PENELITIAN

3.1 Tahapan Penelitian

Bagian ini menjelaskan tentang metodologi penelitian yang dilakukan dalam penelitian ini, yang terdiri dari (1) Studi literatur, (2) Identifikasi dan pengumpulan metode (input, output dan efek), (3) Pengelompokan metode berdasarkan definisi input, output dan efek, (4) Uji coba, (5) Pengelompokan metode berdasarkan hasil uji coba, (6) Pengujian dan analisis hasil.

Ilustrasi alur metodologi penelitian dijelaskan pada gambar 3.1.



Gambar 3. 1 Tahapan Penelitian

3.1.1 Studi Literatur

Studi literatur dilakukan dengan mengumpulkan informasi-informasi yang terkait dengan topik penelitian yang dilakukan. Sumber informasi pada tahap ini didapatkan dari berbagai macam artikel prosiding, jurnal maupun buku.

Informasi-informasi tersebut difokuskan pada metode-metode deteksi kloning kode yang sudah dilakukan sebelumnya.

Proses studi literatur dimulai dari pencarian informasi tentang metode yang telah berkembang yang berkaitan dengan kloning kode. Hal tersebut bertujuan untuk melihat seberapa jauh perkembangan metode yang telah dilakukan untuk melakukan proses deteksi kloning.

Proses pencarian informasi fokus pada metode-metode yang pernah ada dalam proses deteksi kloning. Deteksi kloning secara semantik merupakan metode yang paling baru. Deteksi secara semantik mengerucut pada metode deteksi secara kesamaan fungsi yang dilihat dari perilaku sistem. Perilaku sistem didapatkan dari kode sumber, oleh karena itu informasi tentang kode sumber sangat dibutuhkan pada penelitian ini. Proses, formalisasi dan perilaku pada kode adalah informasi yang sangat penting dalam melakukan perhitungan kesamaan secara semantik. Perhitungan kesamaan secara semantik membutuhkan rumusan, dimana beberapa cara telah dicoba oleh beberapa peneliti pada berbagai kasus. Pencarian informasi diakhiri dengan beberapa cara untuk melakukan pengelompokan data berdasarkan karakteristik yang sama. Pengelompokan ini bertujuan untuk melakukan pengelompokan pada metode-metode yang ada pada sistem.

3.1.2 Identifikasi dan Pengumpulan Metode

Fase teknis pertama dalam penelitian ini adalah pengumpulan metode-metode yang ada dalam sebuah kelas. Dalam proses pengumpulan metode, diperlukan teknis dalam menemukan metode dalam sebuah kode sumber berbasis objek. Proses ini dilakukan pada setiap file yang ada pada sistem. Teknis yang diambil dalam pengumpulan metode adalah dengan menggunakan SDGAPI. SDGAPI ini berfungsi untuk melakukan analisis *byte-code* dari sebuah file .class.

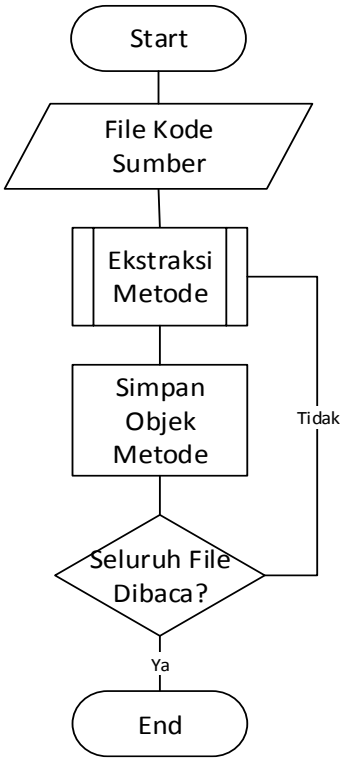
Dalam pengumpulan metode pada kelas terdapat beberapa informasi yang akan diambil. Informasi-informasi itu adalah sebagai berikut,

1. Nama metode, adalah sebuah nama pengenal untuk setiap metode yang ada

dalam kelas,

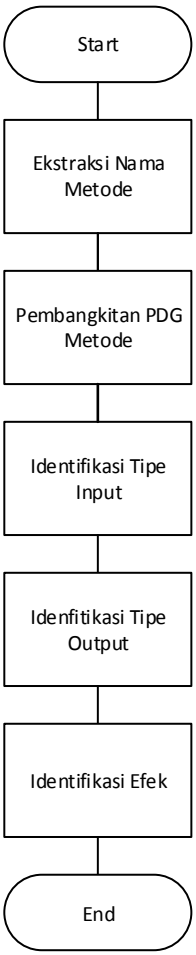
2. Tipe input, adalah tipe data input yang dimasukkan dalam metode, dimana data ini bisa lebih dari satu data dan masing-masing memiliki tipe data tertentu,
3. Tipe output, adalah tipe keluaran dari sebuah metode,
4. Efek metode, adalah objek yang dimanipulasi oleh metode,
5. Grafik ketergantungan data dari setiap metode.

Data dari metode tersebut disimpan menjadi satu kesatuan objek. Pada akhirnya data metode akan dikumpulkan menjadi satu himpunan metode yang disimpan dalam tipe data *list*. Proses pengumpulan kelas digambarkan pada gambar 3.2.



Gambar 3. 2 Proses Pengumpulan Metode

Dalam gambar 3.2 terdapat sub proses Ekstraksi Metode. Proses ini adalah proses bagaimana mendapatkan informasi yang berkaitan dengan metode. Urutan proses dalam sub proses Ekstraksi Metode digambarkan pada gambar 3.3.



Gambar 3. 3 Sub Proses Ekstraksi Metode

Pada gambar 3.3 menjelaskan detail proses yang dilakukan dalam proses ekstraksi metode. Proses-proses yang dilakukan dalam alur proses tersebut bertujuan untuk menangkap seluruh informasi yang ada pada sebuah metode. Proses ekstraksi dibedakan pada dua jenis metode, yaitu ekstraksi pada metode void dan metode bukan void (memiliki parameter input dan output).

Proses ekstraksi metode yang tidak memiliki parameter input dan / atau tidak memiliki nilai kembalian, dilakukan dengan analisis menggunakan PDG. PDG terdiri dari dua jenis grafik, *Control Dependence Graph* dan *Data Dependence Graph*. Dalam penelitian ini, hanya *Data Dependence Graph* yang digunakan untuk analisa PDG. *Data Dependence Graph* merupakan gambaran

hubungan data atau variabel lokal dalam sebuah metode. Dengan melihat hubungan ketergantungan antar variabel ini proses analisis input dan output akan dilakukan. Proses analisis input dan output dilakukan dengan melihat dari mana data itu bermula dan sampai dimana data itu akan bermuara. *Control Dependence Graph* tidak digunakan dalam proses analisis ini, karena grafik tersebut menggambarkan alur dari suatu fragmen kode. Penelitian ini tidak mempertimbangkan alur dari suatu fragmen kode.

3.1.2.1 Indentifikasi Input

Input dilihat dari struktur metode yang berhasil dikumpulkan dari proses sebelumnya. Pada identifikasi input, yang digunakan sebagai acuan adalah tipe parameter input.

Definisi 1. Input adalah seluruh tipe parameter input yang ada pada metode.

Pada contoh nyata dapat digambarkan pada gambar 3.4.

```
int tambah (int x, int y)
{
    int hasil;
    hasil = x + y;
    return hasil;
}
```

Gambar 3. 4 Contoh Metode Ber-Parameter Input

Dari contoh diatas, merujuk pada definisi pertama, maka Input dari metode `tambah` adalah `int` dan `int`.

Permasalahan timbul saat metode yang sedang dianalisis adalah metode yang tidak mempunyai parameter input. Dalam penelitian ini akan digunakan sebuah cara untuk mendapatkan parameter input pada metode yang tidak mempunyai parameter input. Parameter input pada metode tersebut didapatkan dengan menggunakan PDG (*Program Dependence Graph*).

PDG berfungsi untuk menemukan relasi ketergantungan antara variabel dalam sebuah metode. Hubungan relasi ketergantungan ini akan mengarahkan dari mana sumber variabel itu berasal. Apabila sebuah variabel dalam posisi

sebagai sumber atau awalan dari sebuah relasi hubungan, maka variabel tersebut tidak dipengaruhi oleh variabel lain, dan hanya mempengaruhi variabel lain.

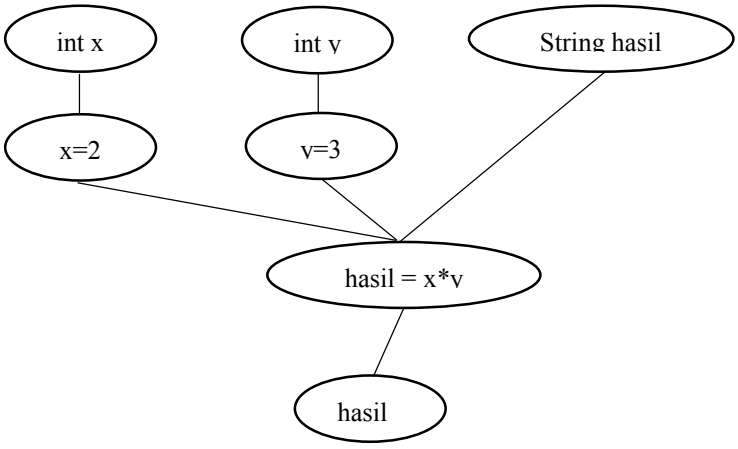
Definisi 2. Input adalah tipe variabel yang hanya memiliki relasi mempengaruhi ke variabel lain, dan tidak dipengaruhi oleh variabel input lainnya.

Pada contoh nyata, implementasi PDG untuk mendeteksi input pada metode yang tidak memiliki parameter input dijelaskan pada gambar 3.5 dan 3.6.

```
String cetak()
{
    int x;
    int y;
    x = 2;
    y = 3;
    String hasil;
    hasil = "Hasil perkalian : " + x * y;
    return hasil;
}
```

Gambar 3. 5 Contoh Metode Tanpa Parameter Input

Contoh metode diatas adalah metode yang berfungsi untuk mencetak. Data yang dihasilkan adalah data hasil pemrosesan di dalam metode tersebut. PDG digambarkan untuk dapat mendapatkan sebuah variabel yang memiliki kriteria seperti yang didefinisikan pada definisi 2. PDG dari metode tersebut digambarkan dalam gambar 3.6.



Gambar 3. 6 PDG dari metode pada Gambar 3.5

Dari PDG pada gambar 3.6, sesuai dengan definisi 2, maka input dari metode `Cetak` adalah `int` dan `int` dari variabel `x` dan `y`. Dari hasil tersebut, selanjutnya didefinisikan menggunakan metode `getName` untuk mencari nama kelas dari variabel yang dihasilkan, sehingga untuk int akan menjadi `java.lang.Integer`.

3.1.2.2 Indentifikasi Output

Seperti halnya input, output dilihat dari informasi tipe output dari metode yang sudah dikumpulkan pada tahap sebelumnya. Output pada metode dengan nilai kembalian adalah tipe nilai kembalian yang terdapat pada definisi metode. Gambar 3.6 adalah metode dengan nilai kembalian. Dari gambar 3.6, metode tersebut memiliki output adalah `int`.

Definisi 3. Output adalah tipe dari nilai kembalian pada sebuah metode.

Pada metode void atau metode yang tidak memiliki output, maka tipe dari output dibutuhkan untuk ditemukan. Metode yang digunakan sama dengan apa yang sudah dijelaskan pada proses pencarian tipe input pada metode yang tidak mempunyai parameter input. Gambar 3.7 adalah sebuah contoh dari metode void yang kemudian dijelaskan bagaimana cara mendapatkan tipe output dari metode tersebut.

```
void Cetak(int x, int y)
{
    int hasil;
    hasil = x * y;
    System.out.println("Hasil perkalian dari "+x+" dan "+y+"
adalah "+hasil);
}
```

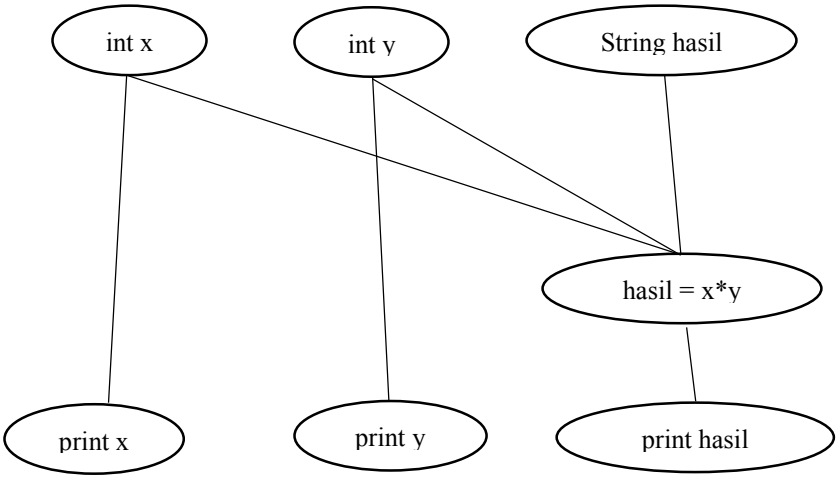
Gambar 3. 7 Contoh Kode Metode Void

Gambar 3.7 adalah cuplikan kode yang mencontohkan sebuah metode void. Metode tersebut tidak memiliki nilai kembalian atau nilai kembalian. Metode

seperti ini adalah metode yang biasanya hanya menlakukan aksi tanpa ada nilai kembalian yang tergambar jelas dari hasil eksekusi metode. PDG akan digunakan dalam proses pencarian hubungan ketergantungan antara variabel dalam metode tersebut.

Definisi 4. Apabila metode tidak mempunyai tipe kembalian maka, output adalah tipe dari variabel yang menjadi muara dari seluruh relasi ketergantunagn antar variabel dalam satu metode, atau variabel yang hanya memiliki relasi dipengaruhi oleh variabel lain.

Penggambaran PDG untuk contoh metode pada gambar 3.7 digambarkan pada gambar 3.8.



Gambar 3. 8 PDG Untuk Metode Void

Gambar 3.8 menggambarkan PDG dari metode pada gambar 3.7. Dari hasil tersebut, dapat diketahui bahwa relasi ketergantungan bermuara pada tiga variabel, yaitu `x`, `y` dan `hasil`. Dengan gambaran ini, maka output dari metode pada gambar 3.7 adalah tipe data dari variabel `x`, `y` dan `hasil` yaitu `int`, `int` dan `int`. Dari hasil tersebut, selanjutnya didefinisikan menggunakan metode `getName` untuk mencari nama kelas dari variabel yang dihasilkan, sehingga untuk int akan menjadi `java.lang.Integer`.

3.1.2.3 Identifikasi Efek

Efek berbeda makna dengan output. Efek merupakan hal yang terjadi akibat proses untuk mencapai output. Dalam kasus metode kelas, efek adalah sesuatu yang dimanipulasi dalam proses menghasilkan output.

Pengidentifikan efek dari sebuah metode dilakukan dengan cara melihat atribut dari kelas apa saja yang ada dalam sebuah metode. Atribut dari sebuah kelas yang didefinisikan dalam sebuah metode memiliki kemungkinan akan terkena efek dari metode.

Definisi 5. Efek dari sebuah metode adalah atribut dan kelas yang memiliki kemungkinan untuk dimanipulasi yang didefinisikan di dalam metode.

Contoh dari sebuah kelas beserta metode-metodenya digambarkan pada cuplikan kode berikut ini.

```
public class Bicycle {
    public int cadence;
    public int gear;
    public int speed;

    public Bicycle(int startCadence, int startSpeed, int
startGear)
    {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
      gear = newValue;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }
}

public class MountainBike extends Bicycle {
```

```
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear) {
      super(startCadence, startSpeed, startGear);
              seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }

    public void IncSpeed(int spd) {
        speed += 2 * spd;
    }

}
```

Gambar 3. 9 Contoh Cuplikan Kode

Dari cuplikan kode pada gambar 3.9 diatas, terdapat dua kelas yang didefinisikan, yaitu kelas `Bicycle` dan `MountainBike`. Kelas `MountainBike` mewarisi sifat dari `Bicycle`. Metode `IncSpeed` adalah metode milik kelas `MountainBike`. Dari definisi 5, maka efek dari metode `IncSpeed` adalah atribut yang dimanipulasi dan kelas dimana metode tersebut didefinisikan beserta informasi hubungan pewarisan kelas, sehingga efek dari metode `IncSpeed` adalah `speed.Bicycle` yang memiliki tipe integer.

3.1.3 Pengelompokan Metode Berdasarkan Definisi Input, Output dan Efek

Proses pengelompokan metode tahap I dilakukan untuk mengelompokkan metode-metode yang memiliki kemungkinan berfungsi sama. Identifikasi pada tahap pengelompokan pertama ini dilakukan terhadap data yang dimiliki metode baik itu input, output dan efek. Proses pengelompokan metode tahap I ini dilakukan untuk menghilangkan metode-metode yang tidak memiliki kemungkinan kesamaan dalam hal input, output dan efek.

Definisi 6. (Kesamaan metode) Dua buah metode dinyatakan sama apabila memiliki tipe input dan output, dan efek yang sama.

Hasil pengelompokan metode tahap I ini akan disimpan dalam beberapa

himpunan metode yang memiliki kesamaan input, output dan efek. Suatu himpunan metode yang berisi hanya satu metode akan dihilangkan. Metode dalam himpunan tersebut dianggap tidak memiliki kemungkinan kloning secara perilaku (input, output dan efek).

Input, output dan efek didefinisikan berbentuk *String*. Proses pendeteksian kesamaan dilakukan dengan pendekatan *exact match* (sama persis), dengan cara pembandingan langsung. Proses pembandingan dilakukan pada setiap elemen input, output dan efek.

3.1.4 Rekonstruksi Metode

Rekonstruksi metode dilakukan untuk mempersiapkan metode yang akan dijalankan pada tahap uji coba. Proses rekonstruksi dilakukan untuk merubah struktur parameter input dan menyisipkan baris kode pada badan kode pada sebuah metode. Proses ini dilakukan pada metode-metode yang tidak memiliki input dan output atau salah satunya. Dasar perubahan struktur input dan penyisipan kode adalah hasil dari analisa input dan output pada proses sebelumnya.

Metode-metode yang tidak memiliki parameter input dan output atau salah satunya, memerlukan perubahan struktur input dan penangkapan nilai output. Perubahan struktur input adalah mendefinisikan parameter input yang dimiliki oleh sebuah metode. Parameter ini akan menjadi sarana untuk memasukkan nilai random pada proses uji coba. Penangkapan nilai output dilakukan dengan menyisipkan sebuah kode untuk menangkap hasil sebuah variabel yang dinyatakan sebagai output pada proses analisa input, output dan efek. Proses pendefinisian input dan output dilakukan untuk mempermudah proses uji coba metode.

Beberapa hal yang perlu diperhatikan dalam proses rekonstruksi metode didefinisikan sebagai berikut ini.

1. Penamaan parameter input disesuaikan dengan nama variabel masukan kandidat.
2. Baris kode yang mendefinisikan dan/atau memberikan nilai ke variabel input

yang akan dihapus, karena variabel ini akan menjadi parameter masukan untuk metode ini.

3. Baris kode yang menyatakan inisialisasi dari input variabel dihapus (*dead code*).
4. Node input yang berbentuk *method call* dihilangkan, dan diganti dengan variabel yang memiliki tipe yang sama.

3.1.5 Uji Coba

Proses uji coba metode dilakukan pada kelompok metode yang memiliki kesamaan input, output dan efek. Proses pengujian dilakukan dengan membuat kode pemanggil dari metode-metode tersebut. Kode pemanggil dibangkitkan secara otomatis seiring dengan proses pengelompokan berdasarkan definisi input, output dan efek. Uji coba metode dilakukan sebanyak 5 (lima) kali dengan memasukkan data random sesuai dengan tipe data pada parameter input (Elva & Leavens, 2012).

Metode akan dipanggil dengan memasukkan parameter input yang sama. Domain input disesuaikan dengan tipe yang terdefinisi pada metode dan dimasukkan secara random kepada metode. Kemudian, dilakukan pencatatan terhadap hasil pemrosesan. Hasil pemrosesan dilihat dari output. Hasil output dilihat dari nilai yang ada pada variabel output. Seluruh hasil eksekusi uji coba metode, baik itu input dan output dicatat untuk kemudian akan dibandingkan. Data log eksekusi metode akan disimpan untuk menjadi masukan pada tahap selanjutnya.

Definisi 7. (Kesamaan metode) Dua buah metode dinyatakan sama apabila memiliki nilai input, output dan efek yang sama.

3.1.6 Pengelompokan Metode Berdasarkan Hasil Uji Coba

Pengelompokan metode pada tahap ini dilakukan pada hasil proses uji coba metode. Uji coba metode menghasilkan berupa log pencatatan input dan output dari sebuah metode. Dari data tersebut, proses pengelompokan dilakukan. Metode-metode yang memiliki input dan output sama maka akan dijadikan satu

kelompok.

Kelompok metode dengan karakteristik yang sama (input, output, efek dan log uji coba) adalah hasil akhir dari proses inti dalam penelitian deteksi kloning kode. Kelompok metode dengan karakteristik yang sama dinyatakan sebagai kelompok kloning.

3.1.7 Pengujian dan Analisis Hasil

Pengujian dan analisa hasil dilakukan dengan tujuan untuk melakukan evaluasi dari hasil penelitian. Proses evaluasi dilakukan dengan melakukan perbandingan hasil analisa dengan seorang ahli dengan hasil dari kerangka kerja yang diusulkan. Proses pengujian dari penelitian ini menggunakan rumusan perhitungan koefisien *Cohen's Kappa*.

Cohen's Kappa digunakan untuk membandingkan hasil yang dihasilkan oleh proses deteksi kloning pada penelitian ini dengan seorang pakar. Seorang pakar adalah seorang yang sangat paham dan berpengalaman dalam merancang kode sumber sebuah perangkat lunak. Seluruh metode hasil dari proses pengumpulan metode akan diberikan kepada pakar untuk dianalisa secara manual. Disisi lain, proses deteksi kloning secara sistematis sesuai dengan kerangka kerja yang diusulkan dijalankan. Pada akhir proses hasil dari pakar dan hasil dari sistem akan dibandingkan dan kemudian dihitung dengan menggunakan koefisien *Cohen's Kappa*.

3.2 Perbandingan Metode

Penelitian ini mengacu pada penelitian yang dilakukan oleh Elva dan Leavens (2012). Beberapa kontribusi telah diusulkan untuk menyelesaikan permasalahan pada metode yang diulkan oleh Elva dan Leavens (2012).

Berikut ini adalah tabulasi perbandingan metode yang dilakukan oleh Elva dan Leavens (2012), dengan metode yang diusulkan di penelitian ini. Perbedaan tersebut dijelaskan pada tabel 3.1.

Tabel 3. 1 Perbedaan Metode Elva dan Leavens (2012) dengan Metode yang Diusulkan

No	Proses	Elva & Leavens (2012)	Metode yang Diusulkan
1	Pengumpulan metode	Pengumpulan metode dengan pendekatan parsing string.	Pengumpulan metode dengan pendekaatan parsing string
2	Identifikasi input	• Input didapatkan dari tipe parameter input suatu metode.	• Input didapatkan dari tipe parameter input untuk metode berparameter input. • Input didapatkan dengan analisa ketergantungan data pada metode yang tidak memiliki parameter input. • Informasi semantik dari tipe input disertakan.
3	Identifikasi output	• Output didapatkan dari tipe nilai kembalian suatu metode. • Metode *void* diabaikan.	• Output didapatkan dari tipe nilai kembalian. • Pada metode void, output didapatkan dengan analisa ketergantungan data dalam metode. • Informasi smantik dari tipe output disertakan.
4	Identifikasi efek	Efek adalah pasangan atribut dan nama kelas yang dikelola dalam metode (analisa ke dalam metode)	• Efek merupakan pasangan atribut dan nama kelas yang dikelola dalam metode (analisa ke dalam). • Efek merupakan kelas dimana metode tersebut dipanggil (analisa ke luar).
5	Pencocokan	Proses pencocokan dilakukan	Proses pencocokan dilakukan dengan

		perbandingan string input, output dan efek.	melihat makna semantik dari tipe input, output dan efek.
6	Uji coba	Proses uji coba dilakukan dengan memasukkan data input secara random.	Proses uji coba dilakukan dengan memasukkan data input secara random.
7	Pengelompokan	Pengelompokan dilakukan berdasarkan kesamaan input, output, efek dan log uji coba.	Pengelompokan dilakukan berdasarkan kesamaan input, output, efek dan log uji coba.

3.3 Percobaan Awal

Percobaan awal adalah proses dimana dilakukan sebuah percobaan menjalankan metode yang diusulkan. Percobaan ini dilakukan secara sederhana dengan data yang kecil. Proses percobaan ini dilakukan untuk memberikan gambaran singkat akan jalannya metode yang diusulkan. Gambaran singkat ini akan digunakan sebagai penelitian pendahuluan sebelum penelitian dilakukan pada data yang sebenarnya.

Studi kasus yang digunakan pada proses percobaan awal ini adalah sebuah kode sumber yang dibuat sedemikian rupa sehingga dapat menggambarkan proses deteksi kloning yang diusulkan. Studi kasus dibangun menggunakan bahasa pemrograman Java. Proses percobaan awal secara keseluruhan disesuaikan dengan kerangka kerja yang sudah dirancang. Percobaan awal secara penuh dilakukan secara manual.

Dalam studi kasus ini, analisa kloning dilakukan pada sebuah kasus tentang perbankan. Dalam dunia perbankan, kita mengenal ada sebutan tentang rekening bank yang bermacam-macam jenisnya. Beberapa jenis rekening adalah rekening tabungan dan rekening tabungan pelajar. Kedua jenis rekening tersebut merupakan paket rekening yang masing-masing memiliki fasilitas sendiri-sendiri. Kelas rekening dinamakan Saving, kelas tabungan dinamakan RegularSaving dan kelas tabungan pelajar dinamakan `CollegeStudent`. Gambaran kode dari ketiga kelas tersebut digambarkan pada gambar 3.10.

```
public class saving {
    public double balance;

    public void withdraw(double amt)
    {
        this.balance = this.balance - amt;
    }

    public void deposit(double amt)
    {
        this.balance = this.balance + amt;
    }

    public void setBalance(double amt)
    {
        this.balance = amt;
    }

    public double getBalance()
    {
        return this.balance;
    }
}

public class RegularSaving extends saving
{
    public RegularSaving(double amt)
    {
        this.balance = amt;
    }

    public void PayBill(Double amt)
    {
        double totalBalance;
        totalBalance = this.balance - amt;
        this.balance = totalBalance;
    }

    public void printBalance()
    {
        DecimalFormat formatter = new
DecimalFormat("###,###,###.##");
        double blnc = this.balance;
        String s = formatter.format(blnc);;
        System.out.println(s);
    }

}

public class CollegeStudent extends saving
{

    public CollegeStudent(double amt) {
        this.balance = amt;
```

```
    }

    public void Pay(double numBill)
    {
        this.balance = this.balance - numBill;
    }

    public void showBalance()
    {
        double amount;
        amount = this.balance;
        NumberFormat numberFormatter;
        String amountOut;
        numberFormatter = NumberFormat.getNumberInstance();
        amountOut = numberFormatter.format(amount);
        System.out.println(amountOut);
    }

}
```

Gambar 3. 10 Kode Studi Kasus Rekening Bank

Dari gambar 3.10, dilakukan pengumpulan metode dan pendefinisian input, output dan efek dari masing-masing metode. Definisi tersebut dijelaskan pada tabel 3.2.

Tabel 3. 2 Definisi Metode Kasus Rekening Bank

No.	Metode	Input	Output	Efek
1	withdraw	double, double	double	balance.saving
2	deposit	double, double	double	balance.saving
3	setBalance	double, double	double	balance.saving
4	getBalance	double	double	balance.saving
5	PayBill	Double, double	double	balance.RegularSaving balance.saving
6	PrintBalance	double	string	balance.RegularSaving balance.saving
7	Pay	double, double	double	balance.CollegeStudent balance.saving
8	showBalance	double	string	balance.CollegeStudent

				balance.saving

Dari hasil pengumpulan metode yang tergambar pada tabel 3.2, metode akan dikelompokkan berdasarkan input, output, dan efeknya. Metode yang memiliki input, output dan efek yang sama dijadikan satu kelompok. Metode yang tidak memiliki kelompok, maka akan dihilangkan. Pada kasus diatas, terdapat beberapa metode yang memiliki efek yang berbeda tetapi pada hakikatnya sama, yaitu `PayBill` dan `Pay`, dan `PrintBalance` dan `showBalance`. Kedua pasang metode tersebut memiliki efek yang sama, karena atribut kelas yang dimanipulasi adalah sama. `RegularSaving` dan `CollegeStudent` adalah sama dengan saving. `PayBill` dan `Pay` memiliki tipe input yang serupa tapi tidak sama. `Double` dan `double` adalah tipe yang secara fungsi adalah sama, hal ini dilihat dari nama kelas dari masing-masing tipe yaitu `java.lang.Double`. Hasil pengelompokan pertama ini digambarkan pada tabel 3.3.

Tabel 3. 3 Hasil Pengelompokan Metode Pertama

No.	Metode	Input	Output	Efek
Kelompok I				
1	withdraw	double, double	double	balance.saving
2	deposit	double, double	double	balance.saving
3	setBalance	double, double	double	balance.saving
Kelompok II				
4	getBalance	double	double	balance.saving
Kelompok III				
5	PayBill	Double, double	double	balance.saving
7	Pay	double, double	double	balance.saving

Kelompok IV				
6	PrintBalance	double	string	balance.saving
8	showBalance	double	string	balance.saving

Pada data tabel 3.3 metode kelompok II berjumlah hanya satu metode, oleh karena itu kelompok ini dihapus dari daftar kandidat kloning.

Tahapan selanjutnya adalah melakukan uji coba pada kelompok-kelompok metode dari proses sebelumnya. Sebelum dilakukan uji coba pada setiap metode, metode-metode yang tidak memiliki parameter input akan direkonstruksi sehingga memiliki parameter input, sesuai dengan analisa input menggunakan PDG. Hasil rekonstruksi metode tergambar pada gambar 3.11.

```
public void withdraw(double amt, double balance)
    {
        balance = balance - amt;
        System.out.println(balance);
    }

    public void deposit(double amt, double balance)
    {
        balance = balance + amt;
        System.out.println(balance);
    }

    public void setBalance(double amt, double balance)
    {
        balance = amt;
        System.out.println(balance);
    }

    public void Pay(double numBill, double balance)
    {
        balance = balance - numBill;
        System.out.println(balance);
    }

    public void PayBill(Double amt, double balance)
    {
        double totalBalance;
        totalBalance = balance - amt;
        balance = totalBalance;
        System.out.println(balance);
    }

    public void showBalance(double balance)
```

```
    {
        double amount;
        amount = balance;
        NumberFormat numberFormatter;
        String amountOut;
        numberFormatter = NumberFormat.getNumberInstance();
        amountOut = numberFormatter.format(amount);
        System.out.println(amountOut);
    }

    public void printBalance(double balance)
    {
        DecimalFormat          formatter          =          new
DecimalFormat("###,###,###.##");
        double blnc = balance;
        String s = formatter.format(blnc);;
        System.out.println(s);
    }
```

Gambar 3. 11 Hasil Rekonstruksi Metode

Mekanisme pengujian dilakukan dengan memberikan data input pada masing-masing metode. Data input diberikan sama untuk seluruh metode pada kelompok yang sama. Lalu output dan efek akan dilihat untuk masing-masing metode. Pencatatan hasil uji coba setiap metode pada contoh diatas dijelaskan pada tabel 3.4.

Tabel 3. 4 Hasil Uji Coba Metode

No.	Metode	Input	Output	Efek
Kelompok I				
1	withdraw	500;1200	700	balance berkurang
2	deposit	500;1200	1700	balance bertambah
3	setBalance	500;1200	500	balance berubah
Kelompok II				
5	PayBill	750;1000	250	balance berkurang
7	Pay	750;1000	250	balance berkurang
Kelompok III				
6	PrintBalance	1000	"1,000"	balance tidak berubah

8	showBalance	1000	"1,000"	balance tidak berubah

Proses pengelompokan kedua, akan dikelompokkan kembali untuk metode-metode yang memiliki nilai input, output dan efek yang sama. Metode yang tidak memiliki kelompok akan dihilangkan. Hasil dari proses pengelompokan kedua setelah dilakukan uji coba ditampilkan pada tabel 3.5.

Tabel 3. 5 Hasil Pengelompokan Kedua

No.	Metode	Input	Output	Efek
Kelompok I				
5	PayBill	Double, double	double	balance.saving
7	Pay	double, double	double	balance.saving
Kelompok II				
6	PrintBalance	double	string	balance.saving
8	showBalance	double	string	balance.saving

Dari data tabel 3.4, metode withdraw, deposit, dan setBalance dihapus dari tabel kandidat kloning karena metode-metode tersebut memiliki nilai output dan nilai efek yang berbeda. Withdraw memiliki efek mengurangi, deposit memiliki efek menambahkan, dan setBalance memiliki efek memberikan nilai kepada balance. Oleh karena itu, ketiga metode tersebut dianggap memiliki fungsi yang berbeda walaupun memiliki tipe input, output dan efek yang sama. Dengan proses pengelompokan berdasarkan uji coba tersebut, maka dapat disimpulkan bahwa beberapa metode yang dinyatakan sama atau kloning secara semantik dilihat dari perilakunya adalah pasangan PayBill dan Pay, dan pasangan PrintBalance dan showBalance.

3.4 Rencana dan Jadwal Kegiatan Penelitian

Pada bagian ini dijelaskan jadwal kegiatan yang akan dilakukan selama penelitian. Kegiatan-kegiatan yang dilakukan selama penelitian disesuaikan dengan tahapan penelitian yang sudah didefinisikan sebelumnya.

Jadwal kegiatan tersebut dijelaskan pada tabel 3.6. Kegiatan penelitian dilakukan selama lima bulan dan melaksanakan delapan kegiatan. Setiap kegiatan yang tercantum dalam tabel jadwal kegiatan akan dilaksanakan sesuai alokasi waktu yang tergambar pada tabel 3.6.

Tabel 3. 6 Jadwal Kegiatan

Kegiatan	**Bulan 1**				**Bulan 2**				**Bulan 3**				**Bulan 4**				**Bulan 5**			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Studi Literatur																				
Pengumpulan Kelas Metode																				
Identifikasi Input, Output dan Efek Metode																				
Pengelompokan Metode Tahap I																				
Uji coba Metode																				
Pengelompokan Metode Tahap II																				
Pengujian dan Analisis Hasil																				
Pembuatan Laporan																				

BAB 4
HASIL DAN PEMBAHASAN

4.1 Pengumpulan Dataset

Penelitian ini membutuhkan dataset untuk melakukan uji coba metode yang diusulkan. Dataset terdiri dari beberapa kode sumber yang dituliskan dengan bahasa Java. Kode sumber tersebut didapatkan pada beberapa situs repositori kode. Dari beberapa kode sumber yang didapatkan, kode dikelompokkan menjadi kode ukuran kecil, sedang dan besar berdasarkan jumlah metode yang terkandung didalam masing-masing kode sumber. Kode dengan ukuran kecil adalah kode yang memiliki metode kurang dari 1500 metode. Kode dengan ukuran sedang adalah kode dengan metode lebih dari 1500 hingga 3000. Dan, kode dengan ukuran besar adalah kode yang memiliki metode lebih dari 3000. Ketentuan pengelompokan kode sumber berdasarkan jumlah metode tersebut adalah permasalahan terbuka *(open issue)*. Belum ada referensi yang mengatakan secara pasti pengelompokan ukuran kode berdasarkan jumlah metode.

Kode sumber yang digunakan pada penelitian ini adalah kode sumber yang menenuhi klasifikasi ukuran kecil, medium dan besar. Masing-masing diberikan contoh satu sistem aplikasi. Aplikasi yang digunakan untuk uji coba dalam penelitian ini dijelaskan pada tabel 4.1.

Tabel 4. 1 Daftar Kode Sumber

No.	Nama Aplikasi	Ukuran	Jumlah Metode	Repositori
1.	DNSJava 2.1.6	Kecil	806	http://www.dnsjava.org/
2.	jDraw 1.1.5	Medium	1243	http://jdraw.sourceforge.net/
3.	jEdit 5.2	Besar	4004	http://www.jedit.org/

4.2 Skenario Pengujian

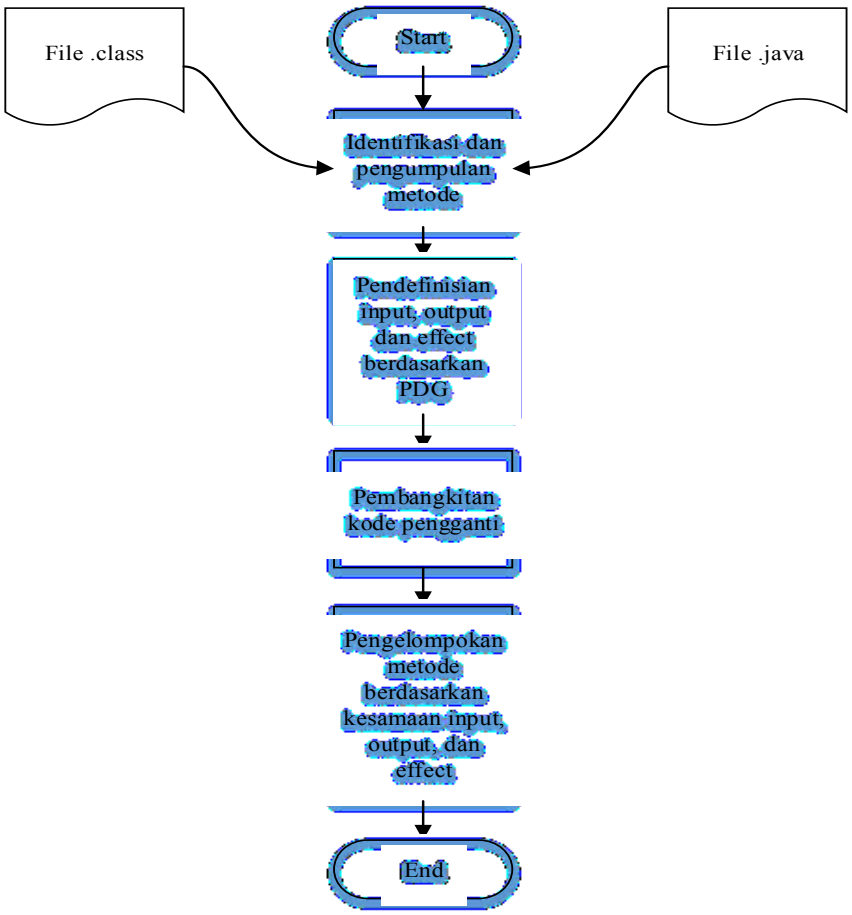
Pengujian terhadap metode yang ditawarkan pada penelitian ini dilakukan dengan mengimplementasikan alur kerja yang sudah diusulkan dan dijelaskan pada metodologi. Alur kerja tersebut diimplementasikan pada kasus data yang diangkat untuk uji coba. Data yang digunakan untuk uji coba dibagi menjadi tiga kategori yaitu ukuran kecil, sedang dan besar.

Pengujian dilakukan dengan membandingkan hasil identifikasi sistem dengan hasil dari pakar. Pakar yang dipilih untuk melakukan identifikasi adalah seorang yang paham dengan alur kode sumber dan memahami tipe-tipe klon yang ada pada kode sumber. Terdapat tiga pakar, masing-masing pakar melakukan identifikasi pada satu kode sumber sebuah sistem aplikasi.

Proses identifikasi oleh sistem dilakukan dengan dua cara, yaitu secara otomatis (dilakukan oleh kakas bantu) dan secara manual. Hasil identifikasi oleh sistem dicatat pada sebuah database. Kemudian, data log tersebut akan dianalisis dengan cara membandingkan dengan hasil identifikasi oleh pakar. Pada tahap akhir, koefisien Kappa dihitung untuk melihat tingkat kecocokan antara hasil identifikasi sistem dengan pakar.

4.3 Implementasi

Proses pendeteksian klon pada kode sumber dilakukan dengan dua tahap, proses otomatis dan proses manual. Proses otomatis dilakukan dengan menggunakan kakas bantu. Kakas bantu dibangun dengan bahasa Java, dan menerapkan teori yang sudah dipaparkan pada bab sebelumnya. Proses otomatis menyelesaikan tiga proses pada metodologi yang diusulkan dalam penelitian ini, yaitu proses pengumpulan metode, identifikasi metode dan pengelompokan metode berdasarkan definisi *input*, *output* dan *effect*. Proses manual menyelesaikan proses uji coba metode dan pengelompokan metode berdasarkan hasil uji coba.



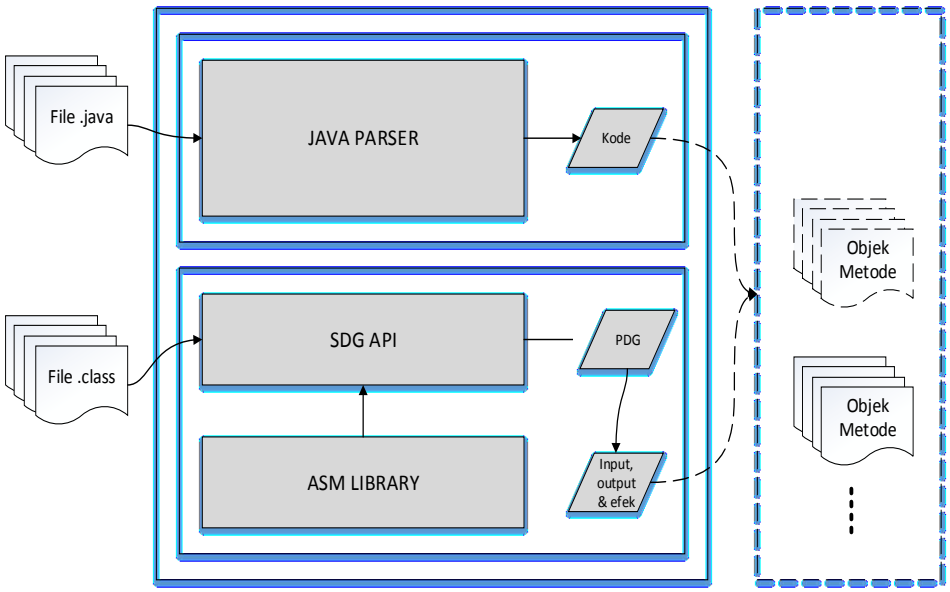
Gambar 4. 1 Alur Proses Otomatis

Gambar 4.1 menunjukkan proses otomatis yang dilakukan oleh kakas bantu. Pada gambar tersebut dijelaskan bahwa data masukkan yang akan diproses dalam proses otomatis ini berbentuk file `.java` dan `.class`. Proses ini memerlukan dua jenis file tersebut karena analisa pada proses otomatis ini menggunakan proses analisis bytecode. Sedangkan file `.java` digunakan untuk memberikan referensi pada proses pembangkitan kode pengganti.

4.3.1 Identifikasi dan Pengumpulan Metode (*input, output* dan *effect*)

Proses identifikasi dan pengumpulan metode, dan pendefinisian input, output dan efek berdasarkan PDG dilakukan pada metode-metode yang tidak memiliki parameter input dan output. PDG dibangkitkan dari librari SDGAPI yang bekerja berdasarkan analisis *bytecode*. PDG yang dihasilkan oleh SDGAPI akan

menjadi dasar dalam penentuan input, output dan efek. Penentuan input, output dan efek sesuai dengan definisi input, output dan efek yang dijelaskan pada bab tiga. Gambar 4.2 menjelaskan tentang bagaimana librari SDGAPI dan librari lainnya berkolaborasi dalam proses otomatis.



Gambar 4. 2 Arsitektur Aplikasi

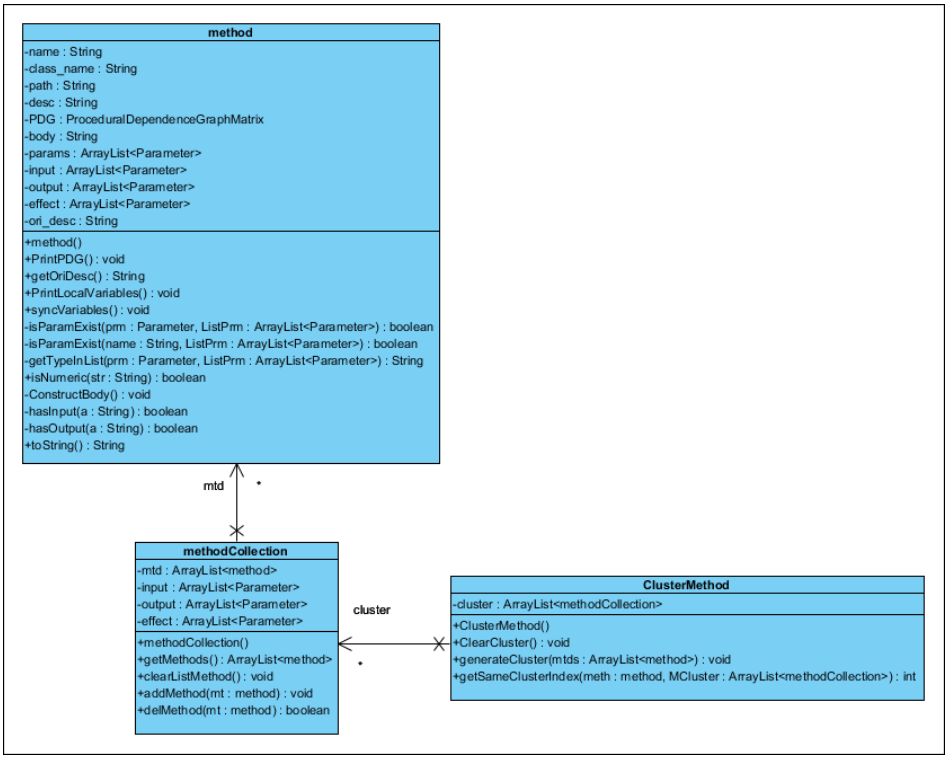
File .java dan .class menjadi masukan dari sistem yang nanti akan diproses masing-masing sesuai dengan tujuannya. Terdapat dua librari utama yang digunakan dalam sistem otomatis, yaitu SDGAPI dan Java Parser. SDGAPI memproses file .class dan Java Parser memproses file .java. Java Parser menghasilkan kode bodi dari metode, sedangkan SDGAPI menghasilkan PDG dan diskripsi input, output dan efek dari setiap metode. Pada akhir proses, hasil dari kedua librari disimpan dalam sebuah obyek metode dengan skema yang dijelaskan dalam tabel 4.2 berikut ini.

Tabel 4. 2 Data Pada Obyek Metode

No.	Data	Tipe	Keterangan
1.	name	String	Nama metode

2.	class_name	String	Nama class
3.	path	String	Direktori file
4.	desc	String	Deskripsi metode
5.	PDG	ProceduralDependenceGraphMatrix	Obyek PDG
6.	body	String	Kode metode
7.	params	ArrayList<Parameter>	Daftar parameter
8.	input	ArrayList<Parameter>	Daftar input
9.	output	ArrayList<Parameter>	Daftar output
10.	effect	ArrayList<Parameter>	Daftar efek
11.	ori_desc	String	Deskripsi asli dari metode

Kelompok obyek dengan deskripsi seperti dalam tabel 4.2, yang memiliki kesamaan dalam input, output dan efek, disimpan dalam obyek *methodCollection*. Obyek *methodCollection* merepresentasikan satu *cluster* (kelompok). Terdapat banyak obyek *methodCollection* yang dihasilkan dalam proses pengumpulan metode dari kode sumber. Obyek-obyek *methodCollection* kemudian disimpan dalam sebuah obyek dengan nama *ClusterMethod*. Obyek-obyek *method* dalam obyek *methodCollection* merupakan obyek yang merepresentasikan metode kandidat klon.



Gambar 4. 3 Class Diagram dari *method*, *methodCollection*, dan *ClusterMethod*

Gambar 4.3 menjelaskan tentang bagaimana hubungan antara *method*, *methodCollection* dan *ClusterMethod* dalam menyusun sistem otomatis pada penelitian ini. Ketiga class tersebut memiliki hubungan asosiasi satu sama lain. Hubungan tersebut ditunjukkan dengan *ClusterMethod* memiliki atribut koleksi bertipe *methodCollection*, dan *methodCollection* memiliki atribut koleksi bertipe *method*.

4.3.2 Pengelompokan Metode Berdasarkan definisi *input*, *output*, dan *effect*

Proses pengelompokan dilakukan secara otomatis. Pengelompokan menggunakan alat bantu yang sama dengan proses identifikasi dan pengumpulan metode. Gambar 4.4 menggambarkan potongan kode proses pengelompokan metode berdasarkan kesamaan input, output dan effect.

```
1  for (method meth : mtds)
2  {
```

```
3         if (getSameClusterIndex(meth, cluster) > 0)
4       {
5               this.cluster.get(getSameClusterIndex(meth,
6   cluster)).addMethod(meth);
7       } else
8       {
9               methodCollection       newMCl       =       new
10  methodCollection();
11          newMCl.setInput(meth.getInput());
12          newMCl.setOutput(meth.getOutput());
13          newMCl.setEffect(meth.getEffect());
14          newMCl.addMethod(meth);
15          cluster.add(newMCl);
16      }
17  }
18
```

Gambar 4. 4 Potongan Kode Proses Pengelompokan Metode

4.3.3 Rekonstruksi Metode

Rekonstruksi metode dilakukan dengan dua langkah, yaitu membangkitkan string parameter input dan menyisipkan kode pada bodi metode. Proses pembangkitan string parameter input dilakukan dengan operasi string. String disusun berdasarkan variabel-variabel input yang ditemukan pada proses analisa input. Sebagai contoh proses rekonstruksi metode, berikut ini terdapat satu contoh metode yang diambil dari kode program aplikasi JDraw. Contoh ini mengambil metode `getTextHeight` yang terdapat pada class `TextCalculator`. Kode asli dari metode `getTextHeight` dijelaskan pada gambar 4.5.

```
1   public int getTextHeight()
2   {
3   final int lineHeight = getLineHeight();
4         final int lineCount = textLines.length;
5         return lineHeight * lineCount;
6   }
```

Gambar 4. 5 Kode metode getTextHeight

Hasil analisa dengan menggunakan aplikasi (proses otomatis) menghasilkan PDG sebagai berikut ini.

```
*********     Printing     Data     Dependency     of
Owner:jdomain/util/gui/TextCalculator
```

```
Method:getTextHeight Desc: ()I**********

getLineHeight$Result_1 --> lineHeight
lineHeight --> getTextHeight$return
lineCount --> getTextHeight$return
getTextHeight$return --> getTextHeight$return

*********  End  of  Printing  Data  Dependency  of
Owner:jdomain/util/gui/TextCalculator
Method:getTextHeight Desc: ()I**********
```

Gambar 4. 6 PDG dari Metode getTextHeight

Dari PDG pada gambar 4.6 di atas, definisi input dan output didapatkan terdapat dua input variabel dan satu output. Metode `getTextHeight` memiliki dua input bertipe `int` dan satu output bertipe `int`. Proses otomatis juga menghasilkan kode sementara sebelum kode mengalami penyisipan kode untuk mendapatkan nilai output dari metode tersebut. Berikut ini adalah kode sementara yang sudah mengalami perubahan pada struktur parameter input. Sesuai dengan aturan rekonstruksi yang pertama, penamaan variabel pada parameter input disesuaikan dengan nama asli variabel pada bodi metode. Sampai pada tahap ini, proses dilakukan secara otomatis. Proses selanjutnya dilakukan secara manual berdasarkan ketentuan rekonstruksi metode yang sudah didefinisikan.

```
1  public int getTextHeight(int lineCount,int
2  getLineHeight$Result_1)
3  {
4      final int lineHeight = getLineHeight();
5      final int lineCount = textLines.length;
6      return lineHeight * lineCount;
7  }
```

Gambar 4. 7 Metode getTextHeight Setelah Mengalami Rekonstruksi Input

Dari hasil gambar 4.7 tersebut, terdapat beberapa kejanggalan pada kode yang dapat membuat kode tidak akan berjalan seperti apa yang diinginkan. Pada kasus diatas, contohnya adalah baris kode `final int lineCount = textLines.length;` dan variabel dengan nama

`getLineHeight$Result_1` pada parameter input. Baris 5 pada kode diatas merupakan proses pendefinisian dan proses pengisian nilai ke variabel `lineCount`. Sesuai dengan aturan nomor dua dan tiga, terdapat sebuah kode yang mendefinisikan dan menginisialisasi sebuah variabel yang menjadi input, maka baris kode ini dihapus. Berikutnya, `getLineHeight$Result_1` merupakan variabel yang terbentuk dari analisis PDG. Nama variabel tersebut diambil dari node yang merupakan pemanggilan sebuah metode bernama `getLineHeight`. Dengan ketentuan nomor empat, bahwa pemanggilan sebuah metode (*method call*) yang menjadi input node pada analisis PDG, diganti dengan variabel yang bertipe sama dengan tipe keluaran metode tersebut. Proses ini dilakukan secara manual. Gambar 4.8 berikut ini adalah hasil kode setelah proses perubahan.

```
1  public int getTextHeight(int lineCount,int
2  getLineHeight)
3  {
4      final int lineHeight = getLineHeight;
5      return lineHeight * lineCount;
6  }
```

Gambar 4. 8 Metode getTextHeight Setelah Mengalami Perubahan

Selanjutnya, dan merupakan proses terakhir pada proses rekonstruksi, adalah proses penyisipan kode. Kode yang disisipkan berfungsi sebagai perekam nilai variabel yang menjadi variabel output. Sebelum proses penyisipan kode, berikut dijelaskan sebuah class yang berfungsi untuk mencatat log dari nilai input dan output suatu metode. Kode pemanggilan metode pada class ini yang disisipkan pada bodi metode. Class ini akan membangkitkan sintak SQL sebagai cara untuk menyimpan hasil log ke database. Gambar 4.9 berikut ini adalah definisi class `testing_sql`.

```
1  public class testing_sql {
2      public static String method;
3      public static String class_nm;
4      public static String input;
5      public static String output;
6      public static String cluster;
7      public static String app;
8
```

```
9       public static String sql()
10      {
11          return "insert into log_testing values
12  (null, '"+method+"','"+class_nm+
13  "','"+input+"','"+output+"','"+
14  cluster+"','"+app+"');\n";
15      }
16
17      public static void reset()
18      {
19          method = "";
20          class_nm = "";
21          input = "";
22          output = "";
23          app = "";
24      }
25  }
```

Gambar 4. 9 Definisi Class `testing_sql`

Data log akan disimpan dalam database dengan skema tabel seperti pada tabel 4.3 berikut ini.

Tabel 4. 3 Tabel Daftar Field Tabel Penysmpanan Log

Field	**Type**
id	int(12)
nama_method	varchar(200)
nama_class	varchar(200)
input	text
output	text
cluster	varchar(25)
app	varchar(50)

Proses penyisipan kode pada bodi metode, dilakukan dengan mengisikan nilai dari variabel output pada atribut static `output` milik class testing_sql. Untuk kasus diatas, hasil dari `lineHeight * lineCount` merupakan node output. Hasil akhir proses rekonstruksi ini tergambar seperti pada gambar 4.10 berikut ini.

```
1   public int getTextHeight(int lineCount,int getLineHeight)
2   {
3       final int lineHeight = getLineHeight;
```

```
4       testing_sql.output   =   Integer.toString(lineHeight   *
5   lineCount);
6       return lineHeight * lineCount;
7   }
```

Gambar 4. 10 Hasil Akhir Proses Rekonstruksi Metode

4.3.4 Uji Coba Metode

Uji coba dilakukan dengan cara memanggil metode-metode yang berada pada kelompok kandidat klon. Pemanggilan dilakukan dengan membuat sebuah kode pemanggil untuk setiap metode. Untuk menjaga kontek dari metode tersebut, maka pada saat proses pemanggilan metode, nilai input yang sama diberikan untuk setiap satu kelompok kandidat klon. Nilai input dibangkitkan secara acak (*random*).

Pada tahap uji coba, pembuatan kode pemanggil dilakukan dengan cara otomatis. Aplikasi pembantu akan membangkitkan kode pemanggil secara otomatis yang kemudian bisa dijalankan. Proses menjalankan kode pemanggil menghasilkan keluaran berupa sintak SQL. Kemudian, dengan menggunakan sintak tersebut, maka data dapat dimasukkan atau disimpan dalam database. Gambar 4.11 adalah contoh kode pemanggil yang dihasilkan oleh aplikasi pembantu.

```
testing_sql.reset();
testing_sql.method = "getTextHeight ";
testing_sql.class_nm = "TextCalculator";
testing_sql.cluster = "Cluster 10 (Clone)";
testing_sql.app = "jDraw";
TextCalculator TextC = new TextCalculator ();
TextC.getTextHeight();
System.out.println(testing_sql.sql());
```

Gambar 4. 11 Kode Pemanggil

Pada kode diatas, metode yang akan dijalankan adalah getTextHeight dari class TextCalculator. Pada kode tersebut belum ada proses penentuan nilai input untuk metode yang akan dijalankan. Oleh karena itu, proses selanjutnya adalah menentukan nilai input secara acak. Untuk menghasilkan nilai acak, dalam kasus ini dibuat sebuah class yang berfungsi untuk membangkitkan data acak dengan beberapa tipe data. Data acak yang dihasilkan adalah string, integer, float dan double. Untuk data selain keempat data tersebut akan dibuatkan sendiri secara

manual. Class pembangkit data tersebut diberi nama class `RandomData` yang didefinisikan seperti pada gambar 4.12.

```
public final class RandomData {

  public String RandomString(boolean caps)
  {
    char[] chars = "abcdefghijklmnopqrstuvwxyz".toCharArray();
    StringBuilder sb = new StringBuilder();
    Random random = new Random();
    for (int i = 0; i < 20; i++) {
        char c = chars[random.nextInt(chars.length)];
        sb.append(c);
    }

    String output = sb.toString();
    if (caps)
        output.toUpperCase();
    return output;
  }

  public int RandomInteger(int univ)
  {
    Random randomGenerator = new Random();
    int randomInt = randomGenerator.nextInt(univ);
    return randomInt;
  }

  public float RandomFloat()
  {
    Random randomGenerator = new Random();
    float randomFloat = randomGenerator.nextFloat();
    return randomFloat;
  }

  public double RandomDouble()
  {
    Random randomGenerator = new Random();
    double randomDouble = randomGenerator.nextDouble();
    return randomDouble;
  }

  private static void log(String aMessage){
    System.out.println(aMessage);
  }
}
```

Gambar 4. 12 Definisi Class `RandomData`

Implementasi class RandomData tersebut pada kode pemanggil ditunjukkan pada kode pada gambar 4.13.

```
1   int a = new RandomData().RandomInteger(100);
2   int b = new RandomData().RandomInteger(100);
3   testing_sql.reset();
4   testing_sql.input        =        Integer.toString(a)+”|”+
5   Integer.toString(b);
6   testing_sql.method = "getTextHeight ";
7   testing_sql.class_nm = "TextCalculator";
8   testing_sql.cluster = "Cluster 10 (Clone)";
9   testing_sql.app = "jDraw";
10  TextCalculator TextC = new TextCalculator ();
11  TextC.getTextHeight(a,b);
12  System.out.println(testing_sql.sql());
13
```

Gambar 4. 13 Implementasi Class RandomData

Kode yang didasari warna kuning adalah kode tambahan. Kode pada baris 1 dan 2, menghasilkan nila acak bertipe integer sebanyak dua nilai. Jumlah dua disesuaikan dengan jumlah parameter input yang dipunyai oleh metode `getTextHeight` setelah mengalami rekonstruksi. Baris 3 berfungsi sebagai pencatat nilai input dari proses menjalankan metode `getTextHeight`. Baris 11 adalah dimana proses memasukkan nilai input ke metode untuk diproses. Sehingga, kode pemanggil tersebut menghasilkan sintak SQL seperti dituliskan pada gambar 4.14.

```
insert into log_testing values (null, 'getTextHeight
','TextCalculator','39|57','2223','Cluster 10 (Clone)','jDraw');
```

Gambar 4. 14 Sintak SQL Log Uji Coba

4.3.5 Pengelompokan Metode Berdasarkan Hasil Uji Coba

Uji coba metode menghasilkan sintak log SQL. Sebelum proses pengelompokan, log SQL tersebut dieksekusi sebagai pada sebuah database. Sehingga data log uji coba dapat disimpan di dalam database. Menyimpan log di dalam database bertujuan agar proses analisa dapat dilakukan lebih mudah.

Salah satu yang dapat kita lakukan setelah data berhasil disimpan di dalam database adalah pengelompokan metode berdasarkan hasil uji coba. Dalam proses pengelompokan ini, dibuat sebuah sintak SQL untuk melakukan pengelompokan metode berdasarkan kesamaan nilai input dan output. Gambar 4.15 adalah contoh sintak SQL untuk melakukan pengelompokan.

```
SELECT      COUNT(*),      methods.`cluster`,      methods.`app`,
methods.`status_clone`  FROM log_testing JOIN methods
      ON log_testing.`nama_method` = methods.`nama_method`
      AND log_testing.`nama_class` = methods.`nama_class`
      AND log_testing.`app` = methods.`app`
      WHERE log_testing.`app` = "jDraw"
      GROUP                BY                log_testing.`input`,
log_testing.`output`,methods.`cluster`, methods.`status_clone`
```

Gambar 4. 15 Sintak SQL Pengelompokan Metode

4.4 Analisis Hasil

Pengujian dilakukan pada kode sumber dari tiga sistem aplikasi yaitu DNSJava, jDraw dan jEdit. Masing-masing memiliki jumlah metode yang beragam. Total seluruh metode yang dimiliki oleh ketiga sistem aplikasi tersebut adalah 6053 metode. 6053 metode akan mendapatkan perilaku yang sama dari awal proses hingga akhir.

4.4.1 Analisis Koefisien Kappa Metode Deteksi

Hasil pendeteksian yang dicatat dari proses deteksi menggunakan kerangka kerja yang diusulkan dalam penelitian ini disebut sebagai hasil deteksi sistem. Dalam deteksi sistem dilakukan dengan dua tahap, secara otomatis dan secara manual. Tabel 4.4 menjelaskan detail hasil dari proses deteksi oleh sistem.

Tabel 4. 4 Hasil Deteksi Sistem

DNSJava		**jDraw**		**jEdit**	
Otomatis		**Otomatis**		**Otomatis**	
Method	806	Method	1243	Method	4004
Kandidat klon	552	Kandidat klon	755	Kandidat klon	2381
Kluster	325	Kluster	610	Kluster	1999
Manual		**Manual**		**Manual**	
Klon	299	Klon	486	Klon	886
Tidak Klon	69	Tidak Klon	62	Tidak Klon	445
Total	368	Total	548	Total	1331
Gagal Deteksi	**184**	**Gagal Deteksi**	**207**	**Gagal Deteksi**	**1050**

Dari tabel diatas, dapat dijelaskan bahwa pada proses otomatis menghasilkan metode-metode yang diindentifikasi menjadi kandidat klon. Metode kandidat klon dikelompokkan dalam sebuah kelompok dengan persamaan tipe input, output dan efek. jEdit memiliki 2381 metode, jDraw memiliki 755 dan DNSJava 552 metode yang dinyatakan menjadi kandidat klon. Hasil dari proses manual dibagi menjadi klon, tidak klon dan gagal deteksi. jEdit memiliki 886 klon, 445 tidak klon dan 1050 gagal deteksi. jDraw memiliki 486 klon, 62 tidak klon dan 207 gagal deteksi. Dan, DNSJava memiliki 299 klon, 69 tidak klon dan 184 gagal deteksi. Terdapat sejumlah data yang tidak bisa dideteksi oleh sistem yang masuk ke dalam kategori gagal deteksi. Prosentase gagal deteksi dibandingkan dengan jumlah total metode adalah, DNSJava 22.8%, jDraw 16.65%, dan jEdit 26.22%. Rata-rata prosentase kegagalan adalah 21.89%.

Hasil yang ada pada table 4.3 tersebut dibandingkan dengan hasil yang diperoleh dari identifikasi oleh pakar. Hal itu bertujuan untuk mendapatkan nilai koefisien Kappa untuk masing-masing aplikasi. Tabel 4.5 menampilkan hasil deteksi oleh pakar.

Tabel 4. 5 Hasil Deteksi Pakar

DNSJava		**jDraw**		**jEdit**	
Klon	304	Klon	502	Klon	1261
Tidak Klon	248	Tidak Klon	253	Tidak Klon	1120

Dari kedua hasil pada table 4.3 dan 4.4, maka koefisien Kappa bisa dihitung. Perhitungan diawali dengan menghitung Po dan Pc. Po adalah prosentase pengukuran yang konsisten antara sistem dan pakar. Pc adalah prosentase perubahan pengukuran antara sistem dan pakar. Tabel 4.6, 4.7 dan 4.8 menjelaskan perhitungan koefisien Kappa pada masing-masing aplikasi (DNSJava, jDraw, jEdit).

Tabel 4. 6 Perhitungan Koefisien Kappa DNSJava

DNSJava			
	Pakar		
Sistem	**Clone**	**Tidak**	**Total**
Clone	299	179	478
Tidak	5	69	74
Total	304	248	552
Po	0.666667		
Pc	0.537125		
Kappa	0.279864		

DNSJava memiliki nilai Po sebesar 0.666667 dan Pc sebasar 0.537125. Koefisien Kappa dari hasil perbandingan antara sistem dan pakar adalah 0.279864. Dari hasil koefisien Kappa tersebut maka dapat diinterpretasikan bahwa nilai tersebut adalah cukup (*fair agreement*).

Tabel 4. 7 Perhitungan Koefisien Kappa jDraw

jDraw			
	Pakar		
Sistem	**Clone**	**Tidak**	**Total**
Clone	486	191	677
Tidak	16	62	78
Total	502	253	755
Po	0.725828		
Pc	0.630828		
Kappa	0.257331		

jDraw memiliki nilai Po sebesar 0.725828 dan Pc sebasar 0.630828. Koefisien Kappa dari hasil perbandingan antara sistem dan pakar adalah 0.257331. Dari hasil koefisien Kappa tersebut maka dapat diinterpretasikan bahwa hasil tersebut adalah cukup (*fair agreement*).

Tabel 4. 8 Perhitungan Koefisien Kappa jEdit

jEdit			
	Pakar		
Sistem	**Clone**	**Tidak**	**Total**
Clone	886	675	1561

Tidak	375	445	820
Total	1261	1120	2381
Po	0.559009		
Pc	0.509215		
Kappa	0.101458		

jEdit memiliki nilai Po sebesar 0.559009 dan Pc sebasar 0.509215. Koefisien Kappa dari hasil perbandingan antara sistem dan pakar adalah 0.101458. Dari hasil koefisien Kappa tersebut maka dapat diinterpretasikan bahwa hasil tersebut adalah sedikit (*slight agreement*).

Rata-rata koefisien Kappa yang didapat dari perhitungan pada masing-masing aplikasi adalah 0,2128. Dari hasil rata-rata tersebut, dapat diinterpretasikan bahwa hasil tingkat kecocokan antara sistem dan pakar pada penelitian ini adalah cukup (*fair agreement*).

4.4.2 Perbandingan dengan Penelitian Terdahulu

Proses perbandingan dilakukan dengan membandingkan hasil analisis metode yang diusulkan dengan metode pada penelitian terdahulu. Perhitungan metode terdahulu dilakukan dengan menghilangkan proses analisis PDG pada metode yang diusulkan. Hasil perhitungan koefisien kappa pada metode terdahulu (tanpa analisis PDG) dijelaskan pada table 4.9, 4.10 dan 4.11 berikut ini.

Tabel 4. 9 Perhitungan Koefisien Kappa pada DNSJava (Tanpa PDG)

Akurasi			
	Pakar		
Sistem	**Clone**	**Tidak**	**Total**
Clone	304	299	603
Tidak	0	87	87
Total	304	386	690
Po	0.566667		
Pc	0.455564		
Kappa	0.204069		

Dengan menghilangkan analisis PDG, DNSJava memiliki nilai Po sebesar 0.566667 dan Pc sebesar 0.455564. Koefisien Kappa dari yang dihasilkan adalah 0.204069. Hasil koefisien Kappa tersebut dapat diinterpretasikan bahwa tingkat kecocokan antara sistem dan pakar adalah sedikit (*slight agreement*).

Tabel 4. 10 Perhitungan Koefisien Kappa pada jDraw (Tanpa PDG)

Akurasi			
	Pakar		
Sistem	**Clone**	**Tidak**	**Total**
Clone	208	828	1036
Tidak	0	66	66
Total	208	894	1102
Po	0.248639		
Pc	0.22603		
Kappa	0.029211		

jDraw memiliki nilai Po sebesar 0.248639 dan Pc sebesar 0.22603. Koefisien Kappa dari yang dihasilkan adalah 0.029211. Hasil koefisien Kappa tersebut dapat diinterpretasikan bahwa tingkat kecocokan antara sistem dan pakar adalah sedikit (*slight agreement*).

Tabel 4. 11 Perhitungan Koefisien Kappa pada jEdit (Tanpa PDG)

Akurasi			
	Pakar		
Sistem	**Clone**	**Tidak**	**Total**
Clone	1229	1749	2978
Tidak	0	459	459
Total	1229	2208	3437
Po	0.491126		
Pc	0.395619		
Kappa	0.158025		

jEdit memiliki nilai Po sebesar 0.491126 dan Pc sebesar 0.395619. Koefisien Kappa dari yang dihasilkan adalah 0.158025. Hasil koefisien Kappa tersebut dapat diinterpretasikan bahwa tingkat kecocokan antara sistem dan pakar adalah sedikit

(*slight agreement*).

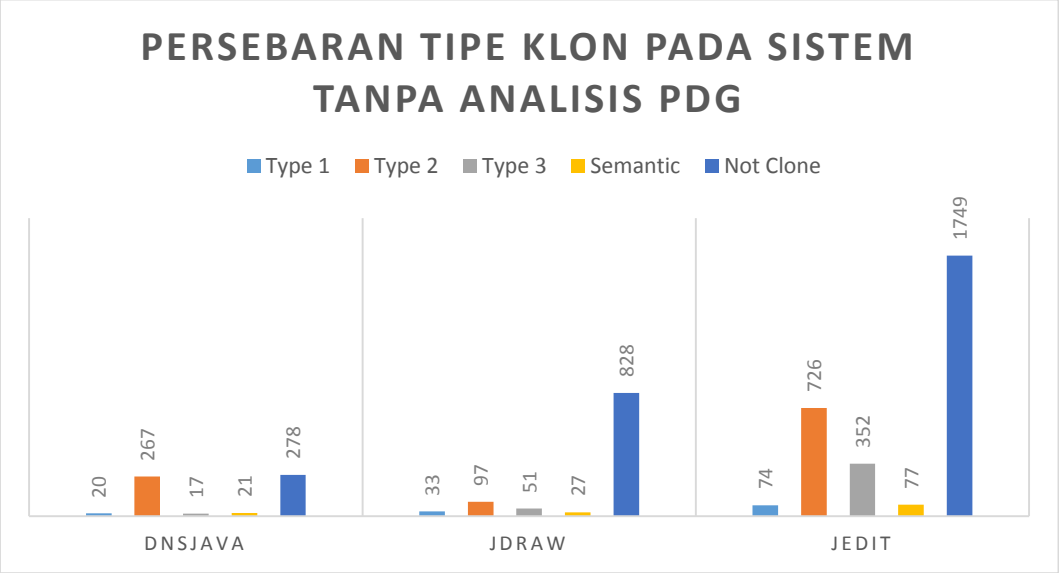
Rata-rata nilai koefisien Kappa pada pendeteksian klon dengan menghilangkan analisis PDG adalah 0,13043. Nilai tersebut dapat diinterpretasikan bahwa tingkat kecocokan rata-rata dari proses pendeteksian klon yang dilakukan oleh pakar dan sistem tanpa analisis PDG adalah sedikit (*slight agreement*).

Deteksi klon dengan menghilangkan analisis PDG seperti yang dilakukan pada penelitian sebelumnya, manghasilkan jumlah metode klon yang banyak dibandingkan dengan metode yang diusulkan dalam penelitian ini. Tabel 4.12 menjelaskan tentang perbandingan hasil klon dari metode terdahulu (tanpa PDG) dengan metode yang dilakukan pada penelitian ini. Dari table tersebut menunjukkan jumlah yang berbeda secara signifikan antara kedua metode. Metode terdahulu (tanpa PDG) terlihat sangat baik dalam mendeteksi klon.

Tabel 4. 12 Perbandingan Jumlah Klon

	DNSJava		jDraw		jEdit	
	Terdahulu	Diusulkan	Terdahulu	Diusulkan	Terdahulu	Diusulkan
Klon	603	299	1036	486	2978	886
Tidak Klon	87	69	66	62	459	445

Proses analisis dilakukan lebih mendalam khususnya pada hasil yang diperoleh dari metode terdahulu. Hasil tersebut dibandingkan dengan hasil deteksi oleh pakar. Gambar 4.16 menggambarkan persebaran tipe klon yang ada pada hasil deteksi metode terdahulu.

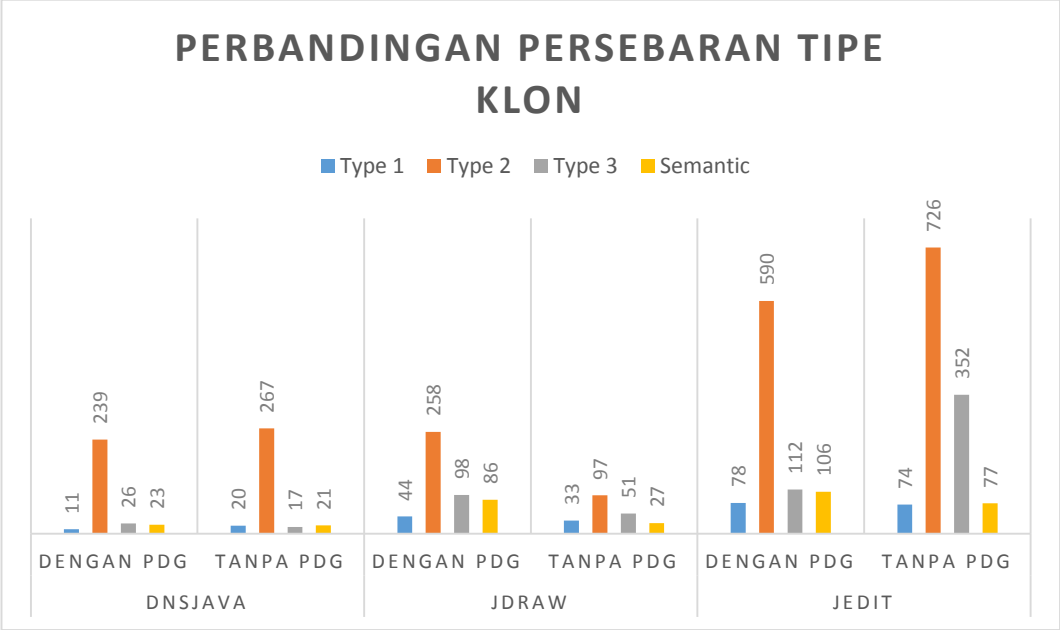


Gambar 4. 16 Persebaran Tipe Klon Tanpa Analisis PDG

Dari persebaran tipe klon pada gambar 4.16, terdapat nilai yang cukup besar yang diperoleh untuk jenis *Not Clone* (tidak klon). Jumlah tidak klon pada masing-masing kode sumber (DNSJava, jDraw, jEdit) menunjukkan jumlah yang paling besar disbandingkan dengan tipe yang lain yang benar-benar klon. Hasil tersebut mengisyaratkan bahwa, pada metode terdahulu (tanpa PDG), terdapat nilai *false positif* yang banyak. Dimana, *false positif* adalah kesalahan identifikasi sebuah metode oleh sistem sebagai klon namun pada kenyataannya tidak klon. Hal ini dapat menyebabkan turunnya nilai akurasi atau tingkat kecocokan antara metode dengan pakar. Perbandingan rata-rata nilai koefisien Kappa antara metode terdahulu dan metode yang diusulkan adalah 0,13043 dan 0,2128. Terdapat selisih 0,08237 nilai antara kedua metode.

Perbandingan tidak hanya dilihat dari nilai koefisien Kappa, namun juga pada persebaran tipe klon pada kedua metode. Gambar 4.17 menggambarkan perbandingan persebaran tipe klon pada kedua metode. Kedua metode memiliki kemampuan deteksi tipe klon semantic yang hampir sama. Dilihat dari gambar 4.17, klon semantik lebih banyak didapatkan pada metode dengan analisis PDG, walaupun selisihnya tidak terlalu banyak. Dari ketiga kode sumber, terdapat rata-rata 13,9% dari jumlah klon semantik meningkat dengan menggunakan analisis

PDG. Dari total metode klon yang dapat dideteksi, metode tanpa analisis PDG dapat mendeteksi lebih banyak klon, namun banyak juga data yang seharusnya tidak klon terdeteksi klon (*false positif*). Pada fase filtrasi pertama, metode tanpa analisis PDG menganggap sepasang metode yang tidak memiliki parameter input dan / atau tidak memiliki nilai kembalian sebagai klon hanya dengan kesamaan diskripsi metode saja (input, output, efek). Hal tersebut yang menyebabkan munculnya banyak data *false positif* pada hasil deteksi. Dan, metode tanpa analisis PDG dirasa kurang detail dalam menganalisa metode yang tidak memiliki input parameter dan / atau tidak memiliki nilai kembalian.



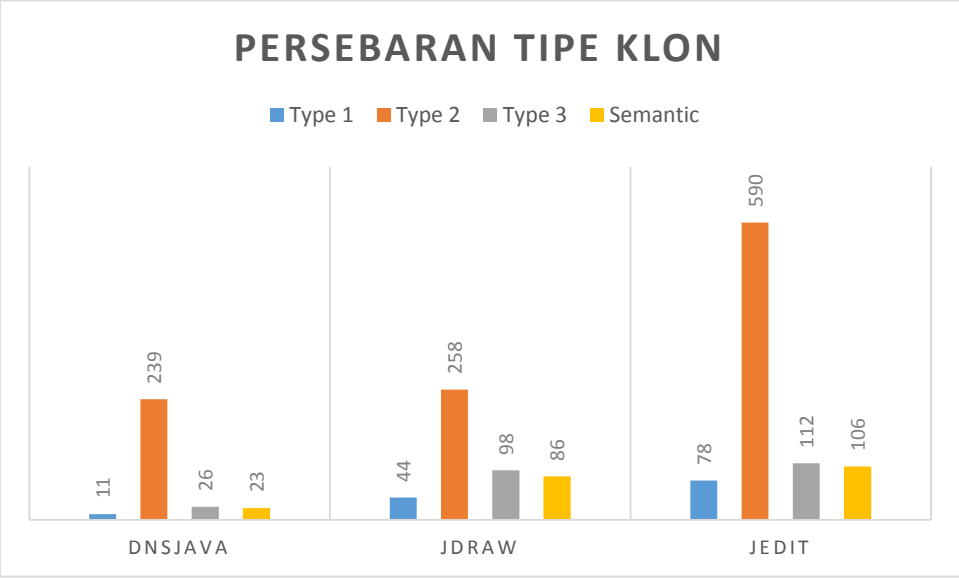
Gambar 4. 17 Perbandingan Persebaran Tipe Klon

4.4.3 Pembahasan Hasil Uji Coba

Pembahasan dilakukan untuk analisa lebih lanjut berkaitan dengan hasil uji coba yang sudah dilakukan. Pembahasan antara lain dilakukan untuk melihat persebaran tipe klon dan permasalahan yang dihadapi sistem ketika gagal melakukan deteksi metode.

4.4.2.1 Persebaran Tipe Klon

Tujuan penelitian ini adalah untuk menemukan keberadaan klon semantik. Persamaan semantik atau makna dilihat dari kesamaan perilaku dari metode. Tidak menutup kemungkinan klon perilaku metode ini juga termasuk klon tipe lain. Tipe-tipe lain itu adalah tipe 1, tipe 2 dan tipe 3. Dari pembahasan ini, hasil uji coba penelitian ini menunjukkan bahwa terdapat tipe lain selain tipe semantik dalam rekapitulasi data uji coba. Gambar 4.18 menggambarkan persebaran tipe klon yang dapat diidentifikasi dari hasil proses identifikasi oleh sistem.



Gambar 4. 18 Grafik Persebaran Tipe Klon

Dalam rekapitulasi data temuan klon metode pada penelitian ini, berikut dipaparkan contoh kode dari masing-masing tipe klon. Tipe 1 adalah tipe klon identik. Kode dari dua metode sama persis hanya berbeda nama metodenya. Contoh dari klon tipe 1 adalah sebagai berikut ini.

```
public RRset findExactMatch(Name name, int type) {
      Object types = exactName(name);
      if (types == null)
            return null;
      return oneRRset(types, type);
```

```
}
private synchronized RRset findRRset(Name name, int type) {
	Object types = exactName(name);
	if (types == null)
		return null;
	return oneRRset(types, type);
}
```

Klon tipe 2 adalah klon dengan perubahan penamaan variabel. Klon tipe 2 merupakan jenis klon yang sering ditemukan pada masing-masing aplikasi. Terdapat banyak sekali implementasi metode getter dan metode setter yang hanya berisi satu baris saja seperti contoh dibawah ini.

```
public String getFileName(String input)
{
    return fileName;
}
public String getPath(String input)
{
    return prefix;
}
public String getSaveDir(String input)
{
    return saveDir;
}
```

Klon tipe 3 adalah klon yang sudah mengalami perubahan penamaan variabel dan penambahan baris kode. Klon tipe 3 lebih sulit untuk ditemukan karena terdapat penyisipan kode diantara baris kode yang klon. Contoh dari klon tipe 3 adalah sebagai berikut ini.

```
public final File savePalette() {
	mode = SAVING_PALETTE;
	BrowserFilter[] filters = new BrowserFilter[] { PAL_FILTER
};
	File f = open( MainFrame.INSTANCE, "Save Palette...",
filters, null, true );
	mode = UNDEFINED_MODE;
```

```
        return f;
    }
```

```
public final File saveImage() {
        mode = SAVING_IMAGE;
        BrowserFilter[] filters;
        filters = new BrowserFilter[] { DRAW_FILTER, GIF_FILTER,
GIF_INTERLACED_FILTER, ICO_FILTER, JPG_FILTER,
                PNG_FILTER, PNG_INTERLACED_FILTER };

        File f = save( MainFrame.INSTANCE, "Save Image...",
filters, ALL_IMAGES_WRITE_FILTER, true );
        mode = UNDEFINED_MODE;
        return f;
    }
```

Tipe terakhir yang dapat dideteksi dan menjadi tujuan utama dalam penelitian ini adalah klon semantik. Dalam klon jenis ini, struktur kode di dalam metode diabaikan. Pendeteksian klon semantik dilakukan berdasarkan kesamaan perilaku dari metode ketika dijalankan. Contoh klon semantik adalah sebagai berikut ini.

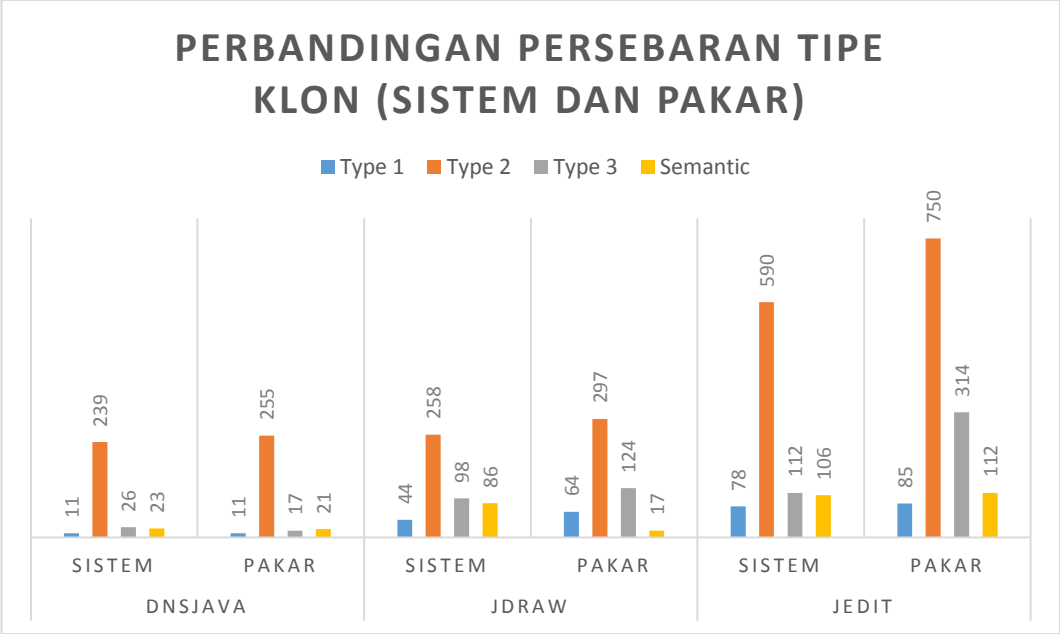
```
public Mode getMode(String name)
{
        return modes.get(name);
}
```

```
protected Mode instantiateMode(String modeName)
{
        return new Mode(modeName);
}
```

Dalam contoh klon semantik diatas, metode terdiri dari satu baris kode saja. Namun, bila dilihat kedalam kode pada masing-masing metode, kode tersebut menjalankan proses yang berbeda. Kedua metode tersebut memiliki kesamaan pada tipe parameter input dan output. Dan, setelah dilakukan proses uji coba dengan memberikan nilai input yang sama, nilai output yang didapatkan dari proses uji coba adalah sama. Metode `getMode()` berfungsi mengeluarkan/ mengembalikan

object `Mode` dengan nama sesuai dengan parameter input. Metode `instatiateMode()` berfungsi menginstansiasi object `Mode` dengan nama sesuai denang parameter input. Kedua metode akan menghasilkan object Mode dengan nama yang sama sesuai dengan nilai input yang dimasukkan. Proses uji coba dijalankan dengan memberikan lingkungan yang memadai dalam proses menjalankan metode.

Jumlah klon tipe 2 menduduki peringkat paling banyak. DNSJava memiliki klon tipe 2 sebanyak 239, jDraw memiliki 258 dan jEdit memiliki 590. Rata-rata prosentase jumlah klon tipe 2 pada ketiga aplikasi adalah 21,71%. Hal serupa juga terlihat pada persebaran tipe klon pada hasil deteksi oleh pakar. Tipe 2 klon merupakan tipe paling sering dijumpai oleh pakar pada setiap aplikasi. Masing-masing aplikasi memiliki, DNSJava 255 tipe 2 klon, jDraw 239 tipe 2 klon, dan jEdit 750 tipe 2 klon. Dimana, apabila diprosentase rata-rata keberadaan tipe 2 klon pada ketiga aplikasi adalah 24,75% dari seluruh metode. Selisih prosentase antara hasil deteksi yang dilakukan oleh sistem dan pakar untuk kasus keberadaan tipe 2 klon adalah 3,04%. Gambar 4.19 menujukkan jumlah secara lengkap persebaran tipe klon dari deteksi yang dilakukan oleh sistem dan pakar.

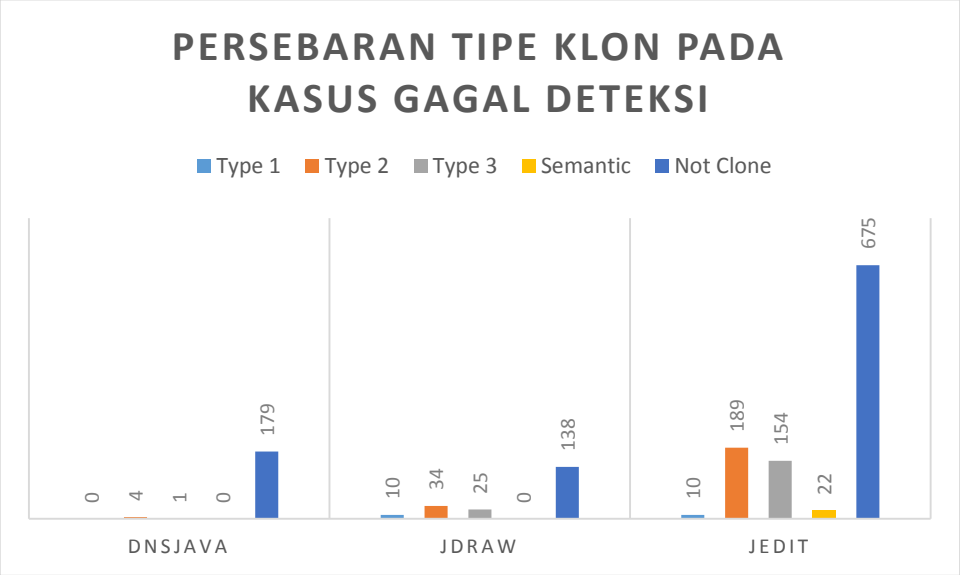


Gambar 4. 19 Grafik Perbandingan Persebaran Tipe Klon (Sistem dan Pakar)

Terdapat empat tipe klon dapat dideteksi dalam penelitian ini. Mulai dari tipe 1, tipe 2, tipe 3 dan klon semantik. Tujuan utama dari penelitian ini adalah pendeteksian klon semantik berdasarkan kesamaan perilaku metode. Kriteria kesamaan perilaku kode telah didefinisikan. Definisi tersebut memberikan sebuah aturan tentang bagaimana memahami atau menafsirkan makna (semantik) yang sama dari sepasang metode yang dibandingkan. Hasil total perolehan kelompok klon metode merupakan hasil dari proses pendeteksian semantic berdasarkan definisi yang sudah ditetapkan. Tipe 1, 2 dan 3 yang juga dapat ditemukan dalam penelitan ini adalah keuntungan dari metode ini. Definisi kesamaan semantik berdasarkan perilaku kode menduduki posisi yang lebih abstrak, sehingga tipe 1, 2 dan 3 dapat tercakup himpunan metode dengan perilaku yang sama (klon). Dengan kata lain, tipe 1,2 dan 3 klon yang ditemukan dalam studi kasus di penelitian ini juga dapat disimpulkan memiliki kesamaan secara makna (semantik) sesuai dengan definisi.

4.4.2.2 Kegagalan Deteksi

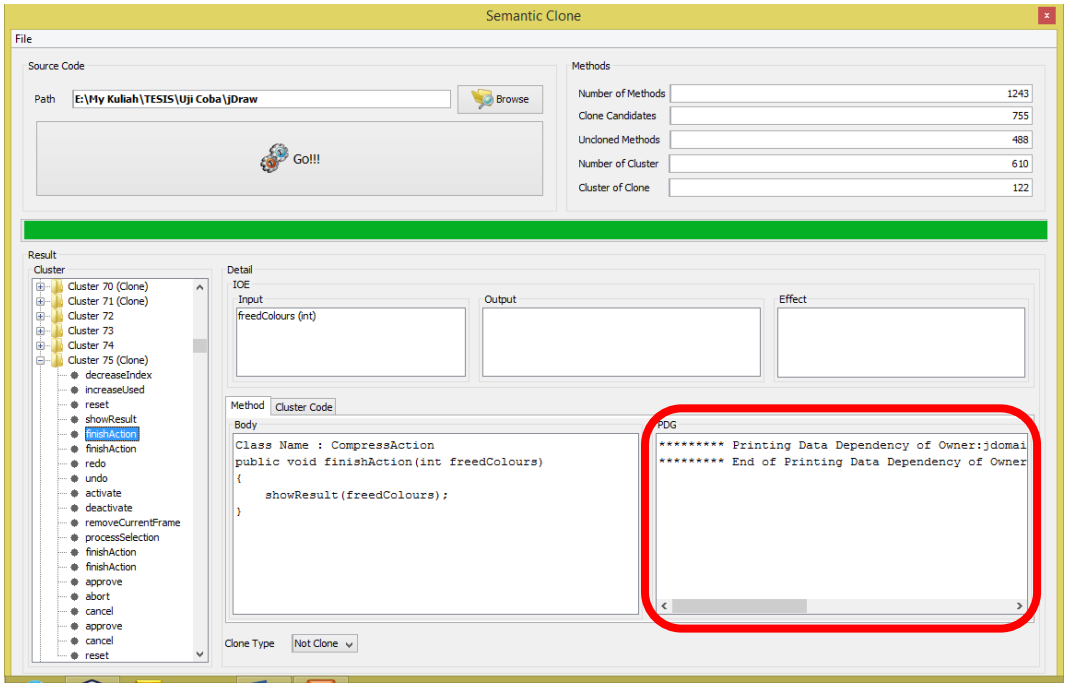
Metode yang digunakan dalam penelitian ini masih memiliki kelemahan. Terdapat sejumlah metode gagal terdeteksi yang menyebabkan kecilnya nilai koefisien Kappa yang didapat. Dari 21.89% data yang gagal, dimungkinkan terdapat berbagai tipe klon yang dapat terdeteksi. Gambar 4.20 menjelaskan persebaran tipe klon pada data yang tidak terdeteksi.



Gambar 4. 20 Grafik Persebaran Tipe Klon Pada Kasus Gagal Deteksi

Dari grafik tersebut dapat disimpulkan bahwa, mayoritas data yang tidak dapat atau gagal terdeteksi adalah metode yang dinyatakan tidak klon. Status klon pada data yang gagal terdeteksi dinyatakan oleh pakar.

Kegagalan pendeteksian beberapa metode dikarenakan sistem tidak dapat menghasilkan bentukkan PDG sehingga pada analisis input, output dan efek tidak dapat dilakukan dengan benar. Mengingat, proses analisis input, output dan efek berdasarkan pada PDG yang dihasilkan oleh sistem. Gambar 4.21 menampilkan tidak munculnya PDG pada sistem aplikasi pembantu deteksi klon.



Gambar 4. 21 Kondisi Sistem Gagal Menampilkan PDG

Kondisi ini memunculkan sebuah asumsi bahwa kegagalan ini disebabkan karena salah satu *library* yang digunakan untuk membangkitkan PDG telah gagal. Library yang digunakan untuk membangkitkan PDG berjalan dengan metode *bytecode* analisis. Pembacaan .class file dengan metode bytecode diasumsikan telah gagal dalam proses pembacaan sehingga PDG tidak bisa dibangkitkan.

Dalam analisa lebih dalam, terdapat beberapa hal yang tidak dapat dideteksi oleh *library* SDGAPI. Beberapa hal yang tidak dapat dideteksi oleh *library* tersebut adalah sebagai berikut :

- variabel lokal yang memiliki modifier static,
- deklarasi array, dan
- pemanggilan method lain secara langsung (tidak ada variabel yang menangkap hasilnya) seperti contoh kode pada gambar 4.12.

Hal ini membutuhkan perhatian lebih dalam pada pengembangan metode di penelitian mendatang.

BAB 5
KESIMPULAN DAN SARAN

Bab ini menjelaskan tentang apa saja yang dapat disimpulkan dari penelitian yang sudah dilakukan. Kemudian, peneliti menjelaskan bagaimana saran atau kelanjutan penelitian pada masa mendatang.

5.1 Kesimpulan

Beberapa kesimpulan yang dapat ditarik dari hasil pengerjaan penelitian ini adalah sebagai berikut ini.

1. Tipe input, output dan efek dari sebuah metode dapat ditentukan pada semua metode baik metode *void* atau *non-void*. Penentuan input, output dan efek dari metode dilakukan dengan melakukan analisa pada PDG yang terbentuk berdasarkan *bytecode* dari sebuah metode. Pembangkitan PDG menggunakan sebuah library yang berjalan menggunakan metode *bytecode* analisis.
2. Proses pencocokan metode dilakukan dengan dua tahap. Tahap pertama proses filtrasi dilakukan berdasarkan kesamaan tipe input, output dan efek. Tahap kedua, pencocokan dilakukan dengan mengelompokkan metode berdasarkan kesamaan log nilai input dan output. Proses pada tahap kedua dilakukan pemanggilan seluruh metode yang dikategorikan sebagai metode kandidat klon. Sebelum dilakukan pemanggilan setiap metode kandidat klon, metode direkonstruksi berdasarkan definisi input dan output berdasarkan analisa PDG.
3. Hasil akhir uji coba oleh sistem dibandingkan dengan uji coba oleh pakar. Pendekatan ini bertujuan untuk mendapatkan koefisien Kappa dari metode yang diusulkan dalam penelitian ini. Hasil perhitungan koefisien Kappa adalah 0,2128 yang kemudian dapat diinterpretasikan bahwa hasil tingkat kecocokan antara sistem dan pakar pada penelitian ini adalah cukup (*fair agreement*).

5.2 Saran

Saran untuk penelitian mendatang adalah sebagai berikut ini.

1. Proses deteksi klon pada metode dalam penelitian tidak semua dilakukan otomatis. Tahap uji coba metode dan sebagian pada proses rekonstruksi metode masih dilakukan secara manual. Proses tersebut sangat membutuhkan usaha yang besar baik itu tenaga dan waktu. Oleh karena itu proses tersebut hendaknya lebih dioptimalkan.
2. Hasil rekapitulasi uji coba dalam penelitian ini menunjukkan masih ada metode-metode yang tidak dapat diidentifikasi. Hal tersebut terjadi diasumsikan karena gagalnya *library* SDGAPI dalam menghasilkan PDG. Oleh karena itu, beberapa metode yang tidak memiliki definisi PDG tidak bisa dilakukan analisa input, output dan efek. Pada penelitian mendatang, perlu dilakukan proses optimalisasi pembangkitan PDG. Sistem pembangkit PDG berbasis analisis kode sumber (*source code*) perlu dipertimbangkan untuk penelitian mendatang.

BAB 5
KESIMPULAN DAN SARAN

Bab ini menjelaskan tentang apa saja yang dapat disimpulkan dari penelitian yang sudah dilakukan. Kemudian, peneliti menjelaskan bagaimana saran atau kelanjutan penelitian pada masa mendatang.

5.1 Kesimpulan

Beberapa kesimpulan yang dapat ditarik dari hasil pengerjaan penelitian ini adalah sebagai berikut ini.

1. Tipe input, output dan efek dari sebuah metode dapat ditentukan pada semua metode baik metode *void* atau *non-void*. Penentuan input, output dan efek dari metode dilakukan dengan melakukan analisa pada PDG yang terbentuk berdasarkan *bytecode* dari sebuah metode. Pembangkitan PDG menggunakan sebuah library yang berjalan menggunakan metode *bytecode* analisis.
2. Proses pencocokan metode dilakukan dengan dua tahap. Tahap pertama proses filtrasi dilakukan berdasarkan kesamaan tipe input, output dan efek. Tahap kedua, pencocokan dilakukan dengan mengelompokkan metode berdasarkan kesamaan log nilai input dan output. Proses pada tahap kedua dilakukan pemanggilan seluruh metode yang dikategorikan sebagai metode kandidat klon. Sebelum dilakukan pemanggilan setiap metode kandidat klon, metode direkonstruksi berdasarkan definisi input dan output berdasarkan analisa PDG.
3. Hasil akhir uji coba oleh sistem dibandingkan dengan uji coba oleh pakar. Pendekatan ini bertujuan untuk mendapatkan koefisien Kappa dari metode yang diusulkan dalam penelitian ini. Hasil perhitungan koefisien Kappa adalah 0,2128 yang kemudian dapat diinterpretasikan bahwa hasil tingkat kecocokan antara sistem dan pakar pada penelitian ini adalah cukup (*fair agreement*).

DAFTAR PUSTAKA

Bettenburg, N., Shang, W. S. W., Ibrahim, W., Adams, B., Zou, Y. Z. Y., & Hassan, a. E., (2009), "An Empirical Study on Inconsistent Changes to Code Clones at Release Level", *2009 16th Working Conference on Reverse Engineering*, Vol. 77(6), hal. 760–776.

Elva, R., Leavens, G., (2012). "Jsctracker: A Semantic Clone Detection Tool for Java Code". *Orlando, FL: University of Central Florida*, (March).

Ferrante, J., Ottenstein, K., & Warren, J., (1987), "The program dependence graph and its use in optimization". Journal ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 9(3), hal. 319–349.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Gabel, M., Jiang, L., & Su, Z., (2008), "Scalable Detection of Semantic Clones", *Proceedings of the 13th International Conference on Software Engineering - ICSE '08*, hal. 321.

Jiang, L., Su, Z., (2009), "Automatic Mining of Functionally Equivalent Code Fragments Via Random Testing", *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis - ISSTA '09*, hal. 81.

Kapser, C. J., & Godfrey, M. W., (2008), ""Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software", Empirical Software Engineering, 13(6), hal. 645–692.

Keivanloo, I., & Rilling, J., (2013), "Semantic-Enabled Clone Detection", *2013 IEEE 37th Annual Computer Software and Applications Conference*, hal. 393–398.

Kempson, R.M., (1977), "Semantic Theory". Cambridge: Cambridge University Pres.

Kim, M., Bergman, L., Lau, T., & Notkin, D., (2004), "An ethnographic study of copy and paste programming practices in OOPL", *Proceedings. 2004*

International Symposium on Empirical Software Engineering, 2004. ISESE '04., hal. 83–92.

Kong, D., Su, X., Wu, S., & Wang, T. ,(2012), "Detect Functionally Equivalent Code Fragments via K-nearest". *Advanced Computational Intelligence (ICACI), 2012 IEEE Fifth International Conference*, hal. 3–7.

Landis, J., & Koch, G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 159-174.

Lozano, A., & Wermelinger, M., (2008), "Assessing the Effect of Clones on Changeability", *2008 IEEE International Conference on Software Maintenance*, hal. 227–236.

Menant, C., (2010), "Introduction to a Systemic Theory of Meaning". *5th EFFC*, hal. 1–9.

Mens, T., & Demeyer, S., (2008), "Software Evolution", Berlin, Heidelberg: Springer Berlin Heidelberg.

Miller, R., & Kasparian, R., (2006), "Java For Artists: The Art, Philosophy, and Science of Object-Oriented Programming", Virginia: Pulp Free Press.

Rattan, D., Bhatia, R., & Singh, M., (2013), "Software Clone Detection : A Systematic Review", Information and Software Technology, Vol. 55, hal. 1165–1199.

Rountev, A., (2004), "Precise Identification of Side-Effect-Free Methods in Java", *20th International Conference on Software Maintenance*.

Roy, C. K., Cordy, J. R., (2007), A survey on software clone detection research, Technical Report 541, Queen's University at Kingston, Canada.

Sim, J., & Wright, C. C., (2005), "The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements", *Physical Therapy*, Vol. 85(3), hal. 257–68.

Sjoberg, D., & Yamashita, A., (2013), "Quantifying the effect of code smells on maintenance effort", *IEEE Transactions On Software Engineering*, Vol. 39, hal. 1144–1156.

Sommerville, Ian, (2011), "Software Engineering", (9th Edition), Boston : Pearson Education Inc.

Tong, C. Y., (2009), "A System Dependence Graph API", http://www4.comp.polyu.edu.hk/~cscllo/teaching/SDGAPI/

Walkinshaw, N., Roper, M., & Wood, M., (2003), "The Java system dependence graph". *Proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'03)*.

Wu, C., (2009), "An Introduction to object-oriented programming with Java TM". (Fifth Edit.). New York: McGraw-Hill Science/Engineering/Math.

BIOGRAFI PENULIS

Penulis dilahirkan di Tulungagung pada tanggal 9 September 1982, yang merupakan anak pertama dari dua bersaudara. Penulis telah menempuh pendidikan dasar di SDN Bareng VIII Malang, SLTP Negeri 1 Malang, dan SMA Negeri 1 Malang. Pada tahun 2001 penulis melanjutkan pendidikan ke jenjang pendidikan tinggi di DIII Manajemen Informatika Universitas Brawijaya dan lulus tahun 2005. Setelah itu, pada tahun 2005 penulis mengambil jenjang sarjana di Institut Teknologi Sepuluh Nopember melalui program Lintas Jalur di Program Studi Teknik Informatika dan lulus tahun 2007. Tahun 2008 penulis mendapatkan amanah untuk menjadi tenaga pengajar di Universitas Brawijaya di Program Studi Teknik Informatika. Kemudian, tahun 2012 penulis melanjutkan studi S2 di Institut Teknologi Sepuluh Nopemeber di Program Studi Teknik Informatika dan lulus sebagai Magister Komputer pada tahun 2015. Penulis mengambil bidang minat Rekayasa Perangkat Lunak baik di jenjang S1 maupun S2. Untuk korespondensi, penulis dapat dihubungi melalui email bayu_priyambadha@ub.ac.id.