PENDETEKSIAN KLONING KODE SECARA SEMANTIK DENGAN METODE *IOE-BEHAVIOR* PADA KODE SUMBER PHP

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Shofi Nastiti
NIM: 115090601111007

PROGRAM STUDI INFORMATIKA/ILMU KOMPUTER
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

PENDETEKSIAN KLONING KODE SECARA SEMANTIK DENGAN METODE *IOE-BEHAVIOR* PADA KODE SUMBER PHP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Shofi Nastiti
NIM: 115090601111007

Skripsi ini telah diuji dan dinyatakan lulus pada
22 Januari 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Fajar Pradana, S.ST., M.Eng.	Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 198711212015041004	NIP: 197105182003121001

Mengetahui
Ketua Program Informatika/Ilmu Komputer

Drs. Marji, M.T
NIP: 196708011992031001

PERNYATAAN ORISINALITAS

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

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

Malang, 22 Januari 2016

Shofi Nastiti

NIM: 115090601111007

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Kuasa, karena hanya atas limpahan dan rahmat Karunia-Nya lah, penulis dapat menyelesaikan skripsi dengan judul "Pendeteksian Kloning Kode Secara Semantik dengan Metode *IOE-Behavior* Pada Kode Sumber PHP" ini dengan lancar.

Penyusunan skripsi tak lepas dari bantuan secara moril yang berupa bimbingan, kritik, saran, dukungan, motivasi maupun doa banyak pihak. Oleh karena itu, ucapan terima kasih penulis sampaikan kepada :

1. Romadon dan Nurul Husniah selaku orang tua penulis dan seluruh keluarga yang telah memberikan semangat, do'a dan segala pengorbanan serta dukungan penuh dalam penyusunan skripsi ini.
2. Bapak Fajar Pradana, S.ST, M.Eng selaku dosen pembimbing 1 dan Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku dosen pembimbing 2, atas segala bimbingan dan waktu yang telah diluangkan serta kritik dan saran yang telah diberikan kepada penulis.
3. Bapak Ir. Sutrisno, M.T, Bapak Ir. Heru Nurwasito, M.Kom, Bapak Himawat Aryadita, S.T, M.Sc, dan Bapak Eddy Santoso, S.Kom selaku Ketua, Wakil Ketua 1, Wakil Ketua 2 dan Wakil Ketua 3 Fakultas Ilmu Komputer Universitas Brawijaya
4. Bapak Drs. Marji, MT dan Bapak Issa Arwani, S.Kom., M.Sc selaku Ketua dan Sekretaris Program Studi Informatika.
5. Seluruh Bapak dan Ibu dosen Fakultas Ilmu Komputer Universitas Brawijaya Malang atas segala bimbingan serta ilmu yang telah diajarkan kepada penulis.
6. Para pegawai dan staf Fakultas Ilmu Komputer Universitas Brawijaya Malang.
7. Thio Trihatmaja yang senantiasa memberikan dukungan dan banyak bantuan.
8. Seluruh teman-teman mahasiswa Fakultas Ilmu Komputer Universitas Brawijaya.
9. Seluruh teman-teman mahasiswa Program Studi Informatika / Ilmu Komputer khususnya angkatan 2011.
10. Seluruh pihak yang tidak bisa disebutkan satu per satu. Penulis mengucapkan terima kasih atas segala bantuan yang telah diberikan.

Semoga penulisan laporan skripsi ini bermanfaat bagi pembaca dan untuk pengembangan selanjutnya. Penulis menyadari bahwa skripsi ini masih jauh dari sempurna serta banyak kekurangan disebabkan oleh keterbatasan kemampuan

dan pengalaman, dengan kerendahan hati penulis mengharapkan kritik dan saran yang bersifat membangun dari pembaca.

Malang, 22 Januari 2016

Shofi Nastiti

shofi.nastiti@gmail.com

ABSTRAK

Proses pemeliharaan perangkat lunak (*software maintenance*) kini menjadi perhatian utama pihak industri karena perkembangan sistem yang semakin kompleks. Salah satu permasalahan yang terjadi pada proses pemeliharaan perangkat lunak adalah adanya *code clones* (kloning kode). Kloning kode dikelompokkan ke dalam dua jenis yaitu kloning kode secara sintaks dan kloning kode secara semantik. Kloning kode secara semantik merupakan fragmen program yang secara fungsional mirip namun secara sintaks tidak ada kesamaan. Banyak peneliti yang menganggap bahwa kloning kode merupakan sesuatu yang merugikan karena dapat menimbulkan cacat pada perangkat lunak. Saat ini, penelitian untuk mendeteksi kloning kode secara semantik masih relatif kurang. Sebelumnya telah dikembangkan sebuah metode pendeteksian kloning kode secara semantik yang diimplementasikan pada kode sumber Java, yaitu metode *IOE-Behavior*. Metode ini memberikan nilai presisi dan recall sebesar 100% untuk seluruh kasus uji yang diujikan. Oleh karena itu, pada penelitian ini penulis mengimplementasikan metode *IOE-Behavior* untuk mendeteksi kloning kode secara semantik pada kode sumber PHP. Mengingat saat ini PHP merupakan bahasa *server-side programming* yang paling populer. Pada implementasinya metode *IOE-Behavior* menghasilkan nilai akurasi sebesar 100% dalam mendeteksi kloning kode secara semantik pada kode sumber PHP sampel yang diperoleh dari *AJ PHP Software Source Code Library*.

Kata kunci: Kloning Kode Secara Semantik, *IOE-Behavior*, Bahasa Pemrograman PHP

ABSTRACT

The process of software maintenance (software maintenance) is now a major concern for the industry development because of the increasing number of complex systems. One of the problems that occur in the process of software maintenance is the code clones (cloning code). Code clones is grouped into two types of cloning code that is syntactically and semantically. Semantic clones; a program fragment that is functionally similar but syntactically nothing in common. Many researchers assume that cloning code is something harmful because it can lead in to software defects. Currently, research to detect the code semantic clones is relatively lacking. Previously, a semantic clones clone detection methods has developed. It was implemented in Java source code. The method called as method IOE-Behavior. This method provides precision and recall value 100% for all test cases tested. Therefore, in this study the authors implement IOE-Behavior method for detecting semantic clones in PHP source code, given that currently PHP is the most popular server-side programming language. In the implementation of the IOE-Behavior method produces a value of 100% accuracy in detecting semantic clones in PHP source code that we obtained from the AJ PHP Software Source Code Library.

Keyword: Semantic Clones, IOE-Behavior, PHP Language.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Proses pemeliharaan perangkat lunak (*software maintenance*) kini menjadi perhatian utama pihak industri karena perkembangan sistem yang semakin kompleks. Pemeliharaan perangkat lunak merupakan kegiatan yang dilakukan untuk menyediakan dukungan dengan biaya yang efektif pada sebuah perangkat lunak. Pemeliharaan perangkat lunak dilakukan sepanjang siklus hidup suatu produk perangkat lunak pengguna (Pigoski, 2001). Kegiatan dalam proses pemeliharaan meliputi mencatat setiap permintaan modifikasi, memutuskan setiap permintaan perubahan berdasarkan dampak perubahannya, memodifikasi kode sumber, melakukan pengujian, merilis versi baru dan memberikan pelatihan pada.

Salah satu permasalahan yang terjadi pada proses pemeliharaan perangkat lunak adalah adanya *code clones* (kloning kode) ketika dilakukan modifikasi pada kode sumber (Morshed,2012). Kloning kode dapat terjadi karena adanya proses penyalinan suatu bagian kode ke bagian kode yang lain. Sehingga hasil kloningnya memiliki kesamaan atau kemiripan baik secara sintaks maupun secara semantik. Oleh karena itu, kloning kode dikelompokkan ke dalam dua jenis yaitu kloning kode secara sintaks dan kloning kode secara semantik. Kloning kode secara sintaks dikelompokkan kembali ke dalam tiga tipe kloning yaitu *exact clones, renamed clones dan near miss clones. Exact clones* merupakan fragmen kode yang identik kecuali pada white space dan comment. *Renamed clones* merupakan fragmen kode yang mirip secara struktural tetapi dilakukan perubahan pada identifier, literal, type, layout dan comment. *Near-miss* clones merupakan fragmen program yang merupakan hasil copy namun telah dimodifikasi lebih jauh seperti penambahan atau pengurangan statements. Sedangkan *semantic clones* (kloning kode secara semantik) merupakan fragmen program yang secara fungsional mirip namun secara sintaks tidak ada kesamaan (Elva, 2012a).

Banyak peneliti yang menganggap bahwa kloning kode merupakan sesuatu yang merugikan. Hal ini terjadi karena adanya kesulitan dalam menjaga konsistensi ketika dilakukan modifikasi pada kode sumber, sehingga meningkatkan upaya dan biaya dalam proses pemeliharaan (Morshed, 2012). Modifikasi kode sumber yang tidak konsisten memungkinkan munculnya cacat pada perangkat lunak (Rattan, 2013).

Berbagai penelitian telah dilakukan dalam mendeteksi kloning kode pada berbagai jenis bahasa pemrograman seperti bahasa pemrograman Java, C dan HTML. Sebagian besar penelitian dilakukan pada kloning secara sintaks, sedangkan penelitian untuk kloning kode secara semantik relatif kurang, meskipun keberadaan kloning kode jenis ini telah dijelaskan dalam literatur kloning kode (Elva, 2012a).

Salah satu penelitian dalam pendeteksian kloning kode secara semantik menghasilkan sebuah perangkat lunak yang menerapkan metode *input*, *Output*,

Effect Behavior (*IOE-Behavior*) (Elva, 2012a). Metode tersebut digunakan untuk mendeteksi kloning kode secara semantik pada aplikasi bebasis desktop yang menggunakan kode sumber Java. Selain itu terdapat banyak penelitian lain yang telah dilakukan untuk mengidentifikasi berbagai tipe kloning kode pada Bahasa pemrograman Java (Elva, 2012a);(Elva, 2012b);(Jiang, 2009);(Elva, 2012c);(Juergens, 2010).

Berbeda dengan penelitian kloning kode pada kode sumber aplikasi berbasis Java, penelitian kloning kode secara semantik pada aplikasi berbasis web saat ini masih kurang (Bellon, 2007). Sebagian besar orang beranggapan bahwa prinsip-prinsip rekayasa perangkat lunak dan metode tertentu tidak berlaku untuk pengembangan web karena aplikasi web berevolusi dari situs web dengan menambahkan fungsi bisnis. Akibatnya, aplikasi web sering dikembangkan secara bertahap tetapi tanpa pendekatan yang sistematis, dan fenomena kloning kode menjadi lebih buruk karena aplikasi web lebih luas (Calefato,2004). Sebagian besar aplikasi web yang merupakan kombinasi antara Bahasa HTML dan bahasa pemrograman lain seperti seperti PHP, ASP.NET, Java, Coldfusion dan Perl.

Penelitian untuk mendeteksi kloning kode pada aplikasi berbasis web telah dilakukan sebelumnya dengan menggunakan metode pendekatan semi-otomatis (Calefato,2004). Penelitian tersebut menghasilkan aplikasi untuk mendeteksi kloning kode secara sintaks pada kode sumber HTML saja. Sedangkan saat ini bahasa pemrograman yang paling populer di kalangan server-side programming adalah Bahasa pemrograman PHP. Hal ini dibuktikan dengan hasil survey dimana Bahasa pemrograman PHP memiliki prosentase sebesar 82% dibandingkan dengan Bahasa pemrograman lain (Wtech, 2014). Meskipun Bahasa pemrograman PHP merupakan bahasa pemrogaman yang yang banyak digunakan pada aplikasi web, masih belum ada penelitian yang dilakukan untuk mendeteksi kloning kode secara semantik pada bahasa pemrograman ini (Bellon, 2007). Mengingat sebelumnya telah dikembangkan metode pendeteksian kloning kode secara semantik yang memberikan nilai presisi dan recall sebesar 100% untuk seluruh kasus uji yang diujikan (Elva, 2012a). Oleh karena itu perlu adanya penelitian yang lebih jauh untuk dapat mendeteksi kloning kode pada Bahasa pemrograman PHP.

Dari beberapa hal yang diuraikan di atas, melalui penelitian ini penulis akan mengembangkan sebuah perangkat lunak untuk mendeteksi kloning kode pada kode sumber PHP dengan menerapkan metode *IOE-Behavior*. Metode ini telah diterapkan untuk mendeteksi kloning kode secara semantik pada level fungsi suatu kode sumber Java dengan mempertimbangkan tiga parameter, yaitu parameter *input* (masukan), *output* (keluaran) dan *effect* (efek) dari sebuah method (Elva, 2012a). Hasil pengujian yang dilakukan pada implementasi metode ini menunjukkan bahwa pendeteksian kloning kode secara semantik pada kode sumber Java dapat dilakukan dengan tingkat kesalahan 0%.

Dari penelitian ini diharapkan dapat menghasilkan sebuah aplikasi deteksi kloning kode secara semantik pada kode sumber PHP dengan menerapkan metode *IOE-Behavior*. Aplikasi tersebut diharapkan dapat membantu pihak pengembang perangkat lunak dalam menjaga konsistensi dalam proses modifikasi

kode sumber dan meningkatkan optimasi dalam proses pemeliharaan perangkat lunak.

1.2 Rumusan Masalah

Berdasarkan pembahasan di atas maka rumusan masalahnya adalah:

1. Bagaimana membangun aplikasi untuk mendeteksi kloning kode secara semantik pada kode sumber PHP dengan menggunakan pendekatan berorientasi objek.
2. Bagaimana menerapkan metode *IOE-Behavior* untuk mendeteksi kloning kode secara semantik pada kode sumber PHP.
3. Bagaimana tingkat akurasi penerapan metode *IOE-Behavior* dalam mendeteksi kloning kode secara semantik pada kode sumber PHP.

1.3 Tujuan

1. Mengembangkan aplikasi yang menerapkan metode *IOE-Behavior* dengan menggunakan pendekatan berorientasi objek untuk mendeteksi kloning kode secara semantik pada kode sumber PHP.
2. Menguji tingkat akurasi penerapan metode *IOE-Behavior* dalam mendeteksi kloning kode secara semantik pada kode sumber PHP.

1.4 Manfaat

Hasil deteksi kloning kode secara semantik oleh perangkat lunak yang dikembangkan melalui penelitian ini diharapkan dapat membantu pihak pengembang dalam menjaga konsistensi dalam proses modifikasi kode sumber dan meningkatkan optimasi dalam proses pemeliharaan perangkat lunak.

1.5 Batasan Masalah

Dari latar belakang di atas, agar pembahasan tidak terlalu luas maka diperlukan pembatasan masalah sebagai berikut:

1. Pendeteksian dilakukan hanya pada kode sumber PHP.
2. Kode sumber yang dideteksi terlepas dari struktur framework yang digunakan dan struktur kode harus sesuai pemrograman berorientasi objek pada PHP.
3. Metode pendeteksian yang digunakan pada penelitian ini adalah metode *IOE-Behavior*.

1.6 Sistematika Pembahasan

Sistematika pembahasan berisi struktur penulisan skripsi dan deskripsi singkat dari masing-masing bagian penulisan yang dapat membantu pembaca dalam memahami sistematika pembahasan isi dalam penelitian ini. Adapun sistematika penulisan penelitian ini adalah sebagai berikut:

1. BAB I Pendahuluan

Bab ini berisi latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat serta sistematika penulisan yang digunakan dalam penyusunan makalah penelitian tentang pendeteksian kloning kode secara semantik pada kode sumber PHP menggunakan metode *IOE-Behavior*.

2. BAB II Landasan Kepustakaan

Bab ini berisi kajian pustaka dan dasar teori mengenai kloning kode, metode *IOE-Behavior*, kode sumber PHP, rekayasa perangkat lunak menggunakan pendekatan berorientasi obyek dan UML yang diperoleh dari berbagai macam sumber tentang pokok permasalahan dalam penelitian yang dilakukan penulis.

3. BAB III Metodologi Penelitian

Bab ini berisi tentang metodologi yang digunakan penulis dalam melakukan penelitian tentang pendeteksi kloning kode secara semantik menggunakan metode *IOE-Behavior* pada kode sumber PHP.

4. BAB IV Analisis dan Perancangan

Bab ini berisi proses analis dan perancangan perangkat lunak yang menerapkan metode *IOE-Behavior* untuk mendeteksi kloning kode secara semantik pada kode sumber PHP menggunakan pendekatan berorientasi objek.

5. BAB V Implementasi

Bab ini berisi tentang proses implementasi sistem sesuai dengan analisis dan perancangan sistem.

6. BAB VI Pengujian

Bab ini berisi tentang proses pengujian yang dilakukan pada hasil implementasi.

7. BAB VII Penutup

Bab ini berisi tentang kesimpulan dari penelitian yang dilakukan dan saran untuk penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Penelitian mengenai pendeteksian kloning kode telah dilakukan oleh beberapa peneliti sebelumnya. Beberapa penelitian menunjukkan masalah-masalah yang muncul akibat adanya kloning kode diantaranya yaitu tingginya biaya pemeliharaan, penyebaran bug, dampak buruk pada desain, dampak terhadap pemahaman atau perbaikan serta modifikasi sistem, selain itu beban kerja sistem semakin berat (Rattan, 2013);(Baxter, 1998). Penelitian yang lain menyimpulkan bahwa kloning kode merupakan salah satu faktor yang membuat proses perawatan perangkat lunak lebih sulit (Morshed, 2012). Sehingga perlu adanya pendeteksian kloning kode untuk meningkatkan hasil proses pemeliharaan perangkat lunak.

Berbagai metode pendeteksian telah dikembangkan dan diterapkan dalam beberapa penelitian, diantaranya adalah metode pendeteksian berbasiskan teks, pendeteksian berbasiskan *tree*, pendeteksian berbasiskan *graph*, pendeteksian berbasiskan matriks dan pendeteksian menggunakan teknik gabungan (Rattan, 2013). Suatu metode belum tentu dapat mendeteksi semua tipe kloning karena adanya jenis-jenis kloning kode yang berbeda, yaitu kloning kode secara sintaks dan semantik (Rattan, 2013);(Elva, 2012a);(Calefato,2004). Kloning kode secara sintaks dikelompokkan ke dalam tiga tipe kloning yaitu *exact clone, renamed clone* dan *near-miss clone*. Berbeda dengan jenis kloning kode secara sintaks, sebagian besar kloning kode secara semantik tidak memiliki kesamaan dari segi sintaks, melainkan kesamaan dari segi fungsionalitas, sehingga diperlukan metode yang berbeda dalam pendeteksiannya (Jiang, 2009).

Dalam implementasinya, kloning kode secara sintaks telah banyak diteliti. Sedangkan penelitian untuk kloning kode secara semantik relatif kurang, meskipun keberadaan kloning kode jenis ini telah dijelaskan dalam literatur kloning kode (Elva, 2012c). Berdasarkan literatur mengenai pendeteksian kloning kode secara semantik yang ada, pendeteksian secara otomatis telah dilakukan dalam penelitian sebelumnya dengan menerapkan metode random testing (Jiang, 2009);(Deissenboeck, 2012). Metode ini hanya menggunakan masukan dan keluaran untuk mendeteksi kloning kode secara semantik. Sedangkan kloning kode secara semantik dapat dideteksi dengan menganalisa perilaku dari kode yang ada dimana perilaku sebuah kode dapat dilihat dari informasi masukan, keluaran dan efek dari suatu kode (Priyambadha, 2014). Sehingga metode ini dianggap tidak lengkap.

Sebuah penelitian untuk mendeteksi kloning kode secara semantik dengan menggunakan masukan, keluaran dan efek dari sebuah method telah dilakukan untuk kode sumber Java (Elva, 2012a). Penelitian tersebut mendefinisikan kloning kode secara semantik sebagai suatu bagian kode yang memiliki kesamaan pada masukan, keluaran dan efek. Metode pendeteksian yang dihasilkan dari penelitian ini disebut dengan metode *IOE-Behavior*. Aplikasi yang menerapkan metode

tersebut telah diujikan pada sebuah kode sumber sampel. Kode sumber yang diuji memiliki 22 *class*, dimana masing-masing *class* memiliki kisaran 900 *lines of code* (LOC) dengan 39 *method* hingga 70 LOC dengan 12 *method*. Hasil pengujian pada aplikasi tersebut memberikan nilai presisi dan recall sebesar 100% (Elva, 2012a). Sehingga dapat disimpulkan bahwa pengujian tersebut menghasilkan 0% kesalahan.

Metode *IOE-Behavior* belum pernah diterapkan untuk mendeteksi kloning kode scara semantik pada aplikasi berbasis web. Sebuah penelitian tentang pendeteksian kloning kode pada aplikasi berbasis web sebelumnya telah dilakukan dengan menggunakan pendekatan semi-otomatis. Dari hasil penelitian tersebut, tipe kloning yang dapat dideteksi terbatas pada kloning kode secara sintaks yang terdapat dalam kode HTML (Calefato,2004). HTML merupakan bahasa pemrograman web untuk sisi klien, sedangkan untuk sisi server saat ini PHP merupakan bahasa pemrograman yang paling populer (WTechs, 2014). Oleh karena itu perlu adanya pendeteksian kloning kode yang lebih dalam pada aplikasi berbasis web, khususnya pada kode sumber PHP.

2.2 Dasar Teori

Pada bagian ini menjelaskan tentang dasar teori yang digunakan untuk melakukan penelitian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior*. Adapun dasar teori yang diuraikan adalah dasar teori mengenai kloning kode, metode *IOE-Behavior*, pemrograman berorientasi objek pada PHP dan parsing kode sumber PHP.

2.2.1 Kloning Kode

Kloning kode dihasilkan dari proses penyalinan bagian-bagian kode dari satu bagian ke bagian lain baik dengan modifikasi atau tidak (Rattan, 2013). Sebuah kloning dapat didefinisikan pula sebagai sebuah bagian program yang identik atau similiar dengan fragmen lainnya (Higo, 2007).

Berdasarkan dengan kemiripannya, kloning kode dibagi ke dalam dua jenis yaitu kemiripan berdasarkan teks program dan kemiripan berdasarkan fungsionalitas program. Bagian kode yang memiliki kemiripan berdasarkan teks program disebut dengan kloning kode secara sintaks, dan bagian kode yang memiliki kemiripan berdasarkan fungsionalitasnya yang disebut dengan kloning kode secara semantik. Berdasarkan literatur yang ada, terdapat beberapa alasan dilakukannya kloning kode (Rattan, 2013);(Kim,2004), diantaranya adalah:

3. Keterbatasan programmer dan waktu. Perangkat lunak terkadang dibangun di bawah kondisi tertentu, dimana proses evolusi perangkat lunak terkendala akibat keterbatasan kemampuan programmer dan batas waktu yang ketat. Satu-satunya cara bagi *programmer* untuk menyelesaikan implementasi dengan tepat waktu adalah dengan melakukan *copying, pasting* atau *editing*.
4. Kompleksitas sistem. Kesulitan dalam memahami sistem yang besar dan kompleks mengakibatkan duplikasi terhadap fungsionalitas yang sudah

ada. Karena pihak pemelihara tidak memahami bahwa fungsi yang mereka tambahkan saat modifikasi sistem telah diimplementasikan sebelumnya.

5. Keterbatasan bahasa pemrograman. Studi etnografi yang dilakukan tentang mengapa *programmer* melakukan *copy* dan *paste* kode, menyimpulkan bahwa pada kondisi tertentu *programmer* diharuskan untuk melakukan *copy* dan *paste kode* karena adanya keterbatasan dalam bahasa pemrograman.
6. Fobia terhadap kode baru. Programmer sering takut untuk memasukkan kode baru dalam membangun perangkat lunak. Menggunakan kembali kode yang sudah ada dianggap lebih mudah daripada mengembangkan solusi baru, karena kode baru dapat mengakibatkan munculnya kesalahan baru.
7. Tidak adanya proses *restructuring. Restructuring* merupakan proses penyempurnaan sistem melalui proses modifikasi berupa perubahan, penambahan atau pembenahan pada bagian kode program yang masih terdapat kesalahan seperti kloning kode. *Programmer* menunda proses *restructuring* kode hingga dilakukannya *delivery* produk. Sehingga biaya pemeliharaan menjadi lebih tinggi.
8. *Forking* atau *templating*. *Forking* adalah menggunakan kembali (*reuse*) solusi yang sama, dengan tujuan bahwa evolusi kode akan terjadi secara independen setidaknya dalam jangka pendek.

Banyak peneliti yang menganggap adanya kloning kode merupakan sesuatu yang dapat menimbulkan permasalahan dalam evolusi perangkat lunak [MSD-12] (Rattan, 2013)(Elva, 2012a)(Jiang, 2009); (Juergens , 2010); (KIM, 2004). Permasalahan yang diakibatkan oleh adanya kloning kode adalah sebagai berikut (Rattan, 2013); (Baxter, 1998):

1. Tindakan *paste* melanggar prinsip enkapsulasi dalam rekayasa perangkat lunak.
2. Memperburuk masalah pemeliharaan karena adanya penyebaran bug. Jika sebuah fragmen kode mengandung *bug* di dalamnya tanpa diketahui oleh *programmer*, kemudian fragmen kode tersbut disisipkan ke bagian yang lain, maka *bug* tersebut dapat muncul di pada fragmen-fragmen lain.
3. Biaya perawatan yang lebih tinggi. Beberapa penelitian telah menunjukkan bahwa adanya kloning kode dalam perangkat lunak sangat berpengaruh dalam meningkatkan biaya pemeliharaan pada masa setelah implementasi.
4. Dampak buruk pada desain. Melakukan kloning kode mengakibatkan proses *refactoring, inheritance,* dll tidak dapat dilakukan. Hal ini menyebabkan praktik desain yang buruk.
5. Dampak terhadap pemahaman, perbaikan atau modifikasi sistem. Pihak yang melakukan pemeliharaan belum tentu merupakan pihak yang sama dengan pihak pengembang sistem. Adanya duplikasi kode mengakibatkan penurunan tingkat pemahaman mengenai system yang dipelihara, sehingga menghambat perbaikan dan modifikasi. Dalam jangka panjang,

perangkat lunak dapat menjadi begitu kompleks dimana perubahan kecil juga sulit untuk dilakukan.

6. Beban pada sumber daya. Kloning kode meningkatkan ukuran sistem perangkat lunak, sehingga beban pada sumber daya sistem juga meningkat. Hal ini mengakibatkan turunnya kinerja sistem secara keseluruhan dari segi kompilasi atau waktu eksekusi dan space requirement.

2.2.1.1 Kloning Kode Secara Sintaks

Kloning kode secara sintaks dikelompokkan ke dalam tiga tipe kloning (Rattan, 2013), yaitu:

1. Tipe-1 (*Exact Clones*): Pada kloning ini fragmen program benar-benar identik dengan fragmen lain kecuali adanya variasi pada white space dan komentar.
2. Tipe-2 (*Renamed* atau *Parameterized Clones*): Pada kloning ini fragmen program memiliki kesamaan secara sintaktik kecuali pada identifier, literal, tipe, layout dan komentar.
3. Type-3 (*Near Miss Clones*): Pada kloning ini fragmen program telah disalin dan dimodifikasi lebih jauh. Salah satu contoh modifikasi yang dilakukan bisa berupa penambahan perintah insert atau delete untuk merubah identifier, literal, tipe dan layout.

2.2.1.2 Kloning Kode Secara Semantik

Kloning kode secara semantik adalah bagian dari kode yang memiliki kesamaan perilaku [BTR-98]. Perilaku sebuah *method* merupakan hal yang dapat dilakukan oleh sebuah *method* ketika dijalankan. Perilaku ini dapat dilihat dari masukan, keluaran dan efek *method* tersebut (Priyambadha, 2014). Oleh karena itu, sebuah *method* dianggap sebagai kloning kode secara semantik jika masukan, keluaran dan efeknya sama (Elva, 2012a). Efek dari sebuah *method* merupakan suatu kemungkinan perubahan yang muncul pada bagian program seperti perubahan pada nilai properti *class* lain ketika *method* tersebut dijalankan. Oleh karena itu kloning kode secara semantik dapat dianggap sebagai redudansi dari fungsionalitas. Mengurangi jumlah kloning kode secara semantik dapat menghasilkan pengurangan biaya dan meningkatkan kemampuan pemeliharaan perangkat lunak (Raemekers, 2010). Gambar 2.1 menunjukkan contoh kloning kode secara semantik.

```php
<?php
function maxbal($a, $b,$c)
{
    $bal=array($a,$b,$c);
    $maxbal = max($bal);
    return $maxbal;
}
?>
```

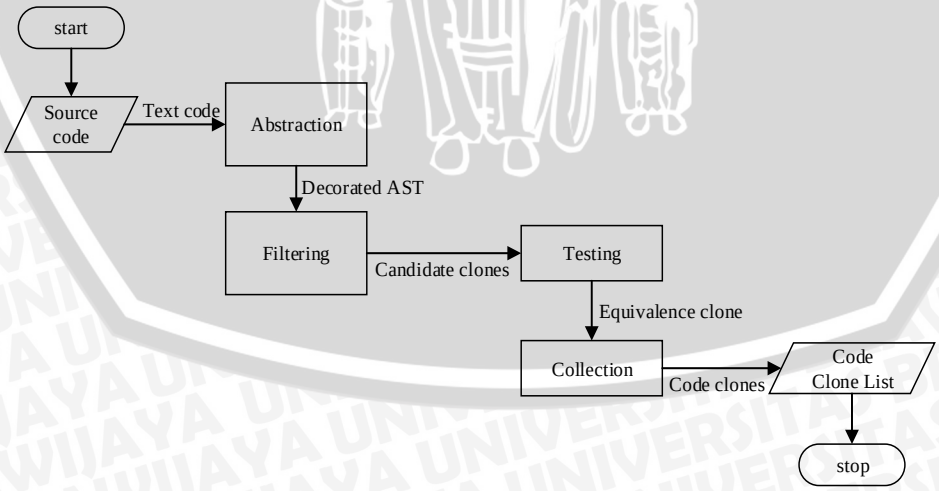
```php
<?php
function balrank($a,$b,$c)
{
   $data=array($a,$b,$c);
   $temp=0;
   for($x=0;$x<count($data);$x++)
   {
     if($data[$x]>$temp)
     {
       $temp=$data[$x];
     }
   return $temp;
   }
}
```

Gambar 2. 1 Contoh Kloning Kode Secara Semantik

Pada Gambar 2.1 menunjukkan bahwa pada kedua fungsi di atas secara sintaks berbeda, namun memiliki kesamaan pada tipe fungsi dan jumlah parameter masukan. Keduanya merupakan fungsi yang mengembalikan nilai dan memiliki tiga parameter masukan. Jika kedua fungsi tersebut dijalankan dengan menggunakan nilai masukan yang sama, maka akan menghasilkan nilai keluaran dan efek yang sama. Oleh karena itu, kedua fungsi pada Gambar 2.1 dapat dinyatakan sebagai kloning kode secara semantik.

2.2.2 Metode *IOE-Behavior*

Metode *IOE-Behavior* merupakan metode yang menggunakan masukan, keluaran dan efek dari sebuah fungsi untuk mendeteksi adanya kloning kode secara semantik (Elva, 2012a). Metode ini digunakan untuk mendeteksi *method* yang termasuk dalam kloning kode secara semantik pada kode sumber suatu perangkat lunak. Pendeteksi kloning kode secara semantik dengan menggunakan metode *IOE-Behavior* terbagi ke dalam 4 proses, yaitu abstraction, filtering, testing dan collection. Alur kerja metode *IOE-Behavior* dapat dilihat pada Gambar 2.2.



Gambar 2. 2 Alur Kerja Metode *IOE-Behavior*

Sumber: (Elva, 2012a)

Gambar 2.2 menunjukkan bahwa proses pendeteksian diawali dengan memasukkan kode sumber ke dalam sistem pendeteksian. Isi dari kode sumber selanjutnya digunakan dalam proses *abstraction* untuk membangun *Abstract Syntax Tree* (AST). AST yang diperoleh digunakan dalam proses *filtering* untuk menghasilkan kandidat kloning. Seluruh kandidat kloning yang telah diperoleh kemudian memasuki proses *testing*. Kandidat kloning yang memiliki hasil *testing* yang ekuivalen memasuki tahap *collection*. Sehingga diperoleh keluaran berupa daftar kloning kode secara semantik. Kemudian proses pendeteksian dihentikan setelah seluruh proses dilakukan.

2.2.2.1 Proses *Abstraction*

Pada proses *abstraction* dihasilkan AST yang berisikan daftar properti fungsi. AST ini digunakan sebagi informasi dalam proses pendeteksian selanjutnya. Properti fungsi merupakan informasi yang dimiliki dari sebuah fungsi berupa nama *class*, nama fungsi, tipe fungsi dan efek fungsi. Proses *abstraction* menghasilkan dua informasi. Informasi pertama adalah tipe fungsi yang merepresentasikan tipe parameter masukan dan keluaran dari fungsi. Sedangkan informasi yang kedua adalah efek fungsi. Pada Tabel 2.1 menunjukkan contoh properti dari tipe fungsi dan efek untuk 3 fungsi dari sebuah *class* dengan nama account.

Tabel 2. 1 Contoh Informasi Tipe dan Efek Fungsi

Code	Tipe Fungsi	Efek Fungsi

`function checkBal(){`
`echo $balance;`
`}`
	`void()`	`{}`

`function calculateBal(){`
`$balance=$balance*2`
`return $balance;`
`}`
	`return $balance`	`{balance}`

`function copyBal($Acct){`
`a.balance=balance;`
`echo "success";`
`}`
	`return $Acct`	`{acct,balance}`

2.2.2.2 Proses *Filtering*

Pada proses *filtering* dilakukan pemilihan kandidat kloning dengan menggunakan tipe fungsi yang ekuivalen, yaitu *return value* dan parameter yang sama. Kemudian dengan menggunakan informasi secara semantik dilakukan seleksi kandidat kloning yang kemungkinan memiliki efek yang berbeda. Dari proses *filtering* ini menghasilkan kelompok-kelompok fungsi dengan tipe fungsi dan efek yang sama, yang disebut dengan kandidat kloning. Proses *filtering* dilakukan dengan melalui dua tahapan, yaitu:

1. Filtering menggunakan tipe fungsi. Mengelompokkan fungsi-fungsi yang memiliki tipe fungsi yang sama dan mengeliminasi kelompok yang hanya terdiri dari satu fungsi.
2. Filtering menggunakan efek fungsi. Memisahkan kelompok dalam sub-kelompok berdasarkan fungsi yang memiliki efek sama. Kemudian mengeliminasi kelompok-kelompok yang hanya memiliki satu fungsi saja.

2.2.2.3 Proses *Testing*

Pada proses ini dilakukan pengujian pada seluruh kandidat kloning yang dihasilkan dari proses filtering. Proses pengujian kandidat kloning dilakukan dengan menggunakan *test-file. Test-file* merupakan suatu file yang berisi perintah untuk memanggil fungsi-fungsi yang merupakan kandidat kloning agar fungsi tersebut dapat berjalan, sehingga dapat diketahui nilai balik dan efek dari fungsi tersebut.

Seluruh kandidat kloning akan dieksekusi dengan menggunakan parameter masukan yang sama dan sesuai sebanyak lima kali. Parameter masukan yang digunakan diperoleh secara random. Kemudian hasil eksekusi masing-masing kandidat kloning akan dibandingkan, kandidat kloning dengan nilai balik dan efek yang berbeda akan dieliminasi.

2.2.2.4 Proses *Collection*

Proses collection dilakukan untuk memperoleh informasi mengenai *method* yang dinyatakan sebagai kloning kode secara semantik yang telah dideteksi. Suatu fungsi dikatakan kloning secara semantik apabila fungsi tersebut memiliki perilaku masukan, keluaran dan efek yang sama untuk setiap kasus uji yang dilakukan pada proses testing (Elva, 2012b). Proses *collection* menghasilkan informasi statistik berupa nomor fungsi, lokasi kloning, ukuran kloning dan nama *class* dimana kloning tersebut berada.

2.2.3 Pemrograman Beriorientasi Objek pada PHP

Pemrograman berorientasi objek merupakan metode pemrograman yang menggunakan *class* untuk mengatur data dan struktur aplikasi. Dimana dalam sebuah *class* PHP terdiri dari variabel untuk menyimpan data yang disebut dengan properti dan sekumpulan kode untuk menjalankan perintah yang disebut dengan *method* (Hayder, 2007). Oleh karena itu, memahami hubungan antara objek dengan *class* merupakan salah satu cara untuk memahami konsep pemrograman berorientasi objek (Zandstra, 2010).

2.2.3.1 Kode Sumber PHP

Kode sumber PHP merupakan file dengan ekstensi .php yang berisi sintaks program (script) PHP. Sintaks program PHP ditulis diantara tag PHP. Ada empat macam pasangan tag PHP yang digunakan untuk menandai bagian awal dan akhir dari sintaks program PHP, yaitu:

1. <?php … ?>
2. <script language = “PHP”> … </script>
3. <? … ?>
4. <% … %>

Cara a dan cara b merupakan cara yang umum digunakan, sekalipun cara c (short tag) tampak lebih praktis [PRG-06]. Untuk menggunakan cara c perlu

dilakukan konfigurasi terhaddap file php.ini untuk penggunaan short tag dengan mengubah nilai dari short_open_tag menjadi on. Pada PHP5, nilai default dari short_open_tag adalah off. Sedangkan untuk cara d merupakan PHP tag style ASP dimana tag ini dapat digunakan dengan mengubah nilai asp_tags dalam file php.ini menjadi on. Berikut contoh sintaks PHP sederhana.

```
<?php
     echo "Ini adalah contoh sintaks php";
?>
```

Gambar 2. 3 Contoh Sintaks PHP

Program PHP dapat dijalankan dengan menggunakan browser dengan kondisi dimana web server apache sedang aktif. Pada pendeteksian kloning kode secara semantik pada kode sumber PHP, dilakukan proses parsing kode sumber PHP. Proses parsing dilakukan dengan menggunakan library PHPParser. PHPParser melakukan proses *parsing* pada kode sumber PHP dan menghasilkan AST yang berisikan informasi secara sintaks dari suatu kode sumber seperti nama *class*, LOC, nama-nama *method* dan perintah-perintah yang ada di dalamnya.

2.2.3.2 *Class* & Objek PHP

Sebuah *class* merupakan sebuah template yang digunakan untuk menghasilkan objek [MAT-10]. Sebuah *class* dapat memiliki beberapa properti dan *method*. Properti sebuah *class* dapat berupa kontanta atau variabel, sedangkan *method* merupakan sebuah fungsi yang dideklarasikan di dalam sebuah *class* (Hayder, 2007). Pembahasan mengenai properti dan *method* akan dijelaskan pada sub bab selanjutnya.

Pendefinisian dasar sebuah *class* diawali dengan kata kunci *class*, kemudian diikuti dengan nama *class* dan kode yang berkaitan dengan *class* tersebut akan ditulis diantara kurung kurawal pembuka dan kurung kurawal penutup ({}). Nama sebuah *class* dapat berupa kata atau kalimat selain kata atau kalimat yang telah digunakan oleh PHP. Sebuah nama *class* yang dapat digunakan misalnya nama *class* yang diawali dengan garis bawah, kemudian diikuti dengan sejumlah huruf atau angka. Salah satu contoh pendefinisian *class* dapat dilihat pada gambar 2.4.

```
<?php
class SimpleClass
{
    //class body
}
?>
```

Gambar 2. 4 Mendefinisikan *Class* pada PHP

Pada Gambar 2.4 menunjukkan bagaimana sebuah *class* dengan nama Simple*Class* didefinisikan. Namun *class* tersebut masih belum memiliki bagian kode yang dapat mendefinisikan struktur data dari objek yang nantinya dapat dihasilkan menggunakan *class* ini.

Jika sebuah *class* merupakan template untuk menghasilkan objek, maka objek adalah data yang strukturnya sesuai dengan template yang didefinisikan dalam

class terkait. Sebuah objek dikatakan sebuah instance dari *class* yang digunakan untuk menghasilkan objek tersebut (Zandstra, 2010). Pada gambar 2.4 telah dilakukan pendeklarasian *class* dengan nama SimpleClass. Untuk menghasilkan sebuah objek dari *class* tersebut, maka diperlukan operator lain seperti contoh pada gambar 2.5.

```
$objsample1 = new SimpleClass();
$objsample2 = new SimpleClass();
```

Gambar 2. 5 Intansiasi Objek dari Sebuah *Class*

Pada Gambar 2.5 menunjukkan bahwa terdapat dua operator yang digunakan sebagai wadah objek yang dihasilkan dari instansiasi *class* dengan nama SimpleClass. Operator $objsample1 dan $objsample2 secara fungsional identik, namun keduanya merupakan objek berbeda yang dihasilkan dari satu kelas yang sama. Karena *class* yang digunakan untuk menghasilkan kedua objek tersebut tidak memiliki kode yang dapat mendefisikan struktur data didalamnya, maka perlu ditambahkan properti dalam *class* tersebut.

2.2.3.3 Properti pada *Class* PHP

Sebuah *class* dapat mendefinisikan properti yang disebut dengan variabel anggota. Variabel anggota dapat menampung data yang berbeda-beda dari setiap objek yang dihasilkan. Misalnya kita menambahkan variabel $prop1 dan variabel $prop2 dalam SimpleClass, maka kita dapat memanipulasi nilai dari kedua variabel tersebut pada objek yang dihasilkan dari *class* ini.

Cara mendeklarasikan sebuah properti *class* hampir sama dengan mendeklarasikan variabel pada umumnya. Perbedaannya terletak pada penambahan kata kunci visibilitas yaitu public, protected atau private di awal pendeklarasian. Kata kunci visibilitas ini digunakan sebagai batasan hak akses atas properti tersebut(Zandstra, 2010). Contoh pendeklarasian properti dari sebuah *class* dapat dilihat pada Gambar 2.6.

```
<?php
class SimpleClass
{
    public $prop1 = "first name";
    public $prop2 = "last name";
}
?>
```

Gambar 2. 6 Mendeklarasikan Properti pada *Class* PHP

Pada Gambar 2.6 menunjukkan bahwa SimpleClass memiliki dua properti *class* didalamnya yaitu $prop1 dan $prop2. Objek yang dihasilkan dari instansiasi *class* ini akan memiliki data default sesuai dengan properti yang telah didefinisikan. Selain itu, kata kunci public menyebabkan variabel properti tersebut untuk dapat diakses dari dalam maupun luar objek.

2.2.3.4 *Method* pada *Class* PHP

Seperti hal nya properti yang digunakan oleh objek untuk menyimpan data, *method* dibuat agar objek dapat menjalankan perintah (Zandstra, 2010). Sebuah *method* merupakan fungsi PHP yang dideklarasikan didalam sebuah *class*. Oleh karena itu, cara mendeklarasikan *method* sama dengan mendeklarasikan fungsi PHP. Pendeklarasian fungsi menggunakan kata kunci function. Contoh pendeklarasian sebuah fungsi ditunjukkan pada Gambar 2.7.

```
function nama_fungsi ($arg1, $arg2,…$argn)
{
        <Blok pernyataan fungsi>;
}
```

Gambar 2. 7 Mendeklarasikan Fungsi PHP

Bagian nama_fungsi merupakan nama yang akan digunakan untuk memanggil fungsi. Sedangkan $arg1, $arg2, …, $argn merupakan parameter yang akan diproses pada saat fungsi dipanggil. Bagian blok statements (pernyataan) fungsi adalah pernyataan-pernyataan yang diapit oleh tanda {} yang akan dijalankan saat fungsi dipanggil (Perangingangin, 2006).

Di dalam PHP terdapat dua tipe fungsi yaitu fungsi, yaitu fungsi yang tidak mengembalikan nilai yang biasa disebut void function, dan fungsi yang memberikan return value (nilai balik) (Perangingangin, 2006). Nilai balik adalah nilai yang dikembalikan oleh suatu fungsi yang dipanggil. Nilai balik dikembalikan melalui pernyataan return yang dapat berupa suatu nilai atau objek. Ditemukannya pernyataan return menyebabkan fungsi akan mengakhiri eksekusinya dengan seketika dan mengirim kembali kendali kepada baris dari mana fungsi tersebut dipanggil. Suatu fungsi tidak dapat mengembalikan banyak nilai [PRG-06], tetapi hasil serupa dapat diperoleh dengan mengembalikan suatu list (daftar).

```
function header()
{
   echo "<html>";
   echo "<html>";
   echo "<html>";
}
```

Gambar 2. 8 Fungsi PHP yang Tidak Mengembalikan Nilai

```
function math_calc($a, $b)
{
      $c=$a*$b;
      return $c;
}
```

Gambar 2. 9 Fungsi PHP yang Mengembalikan Nilai

Untuk memanggil fungsi yang telah dibuat, gunakan nama fungsi dan parameter yang menjadi argumennya jika ada. Pada gambar 2.9 di atas, nama fungsinya adalah calculation yang memiliki 2 argumen yaitu $a dan $b yang dipisahkan oleh tanda koma (,). Dan pernyataan return digunakan untuk memberikan nilai balik dari fungsi di atas. Berbeda dengan fungsi pada gambar

2.8, untuk fungsi yang mengembalikan nilai seperti pada gambar 2.9 sediakan sebuah variable bantu untuk menampung nilai baliknya atau dapat juga langsung dilakukan pemrosesan hasil (Raharja, 2012).

Dari hasil pembahasan diatas, berikut adalah contoh pendeklarasiam sebuah *method*.

```
class simpleClass
{   public $prop1 = "first name";
    public $prop2 = "last name";
    public function methodSample( $param1, $param2)
    {
        //method body
    }
}
```

Gambar 2. 10 Mendeklaraskan *Method* PHP

Gambar 2.10 menunjukkan pendeklarasian sebuah *method* dengan nama methodSample di dalam sebuah *class* dengan nama simpleClass. *Method* tersebut memiliki dua parameter yaitu $param1 dan $param2. Kata kunci visibilitas juga dapat ditambahkan pada sebuah *method*, sama seperti dengan penambahan kata kunci visibilitas pada properti *class* (Zandstra, 2010). Apabila sebuah *method* tidak didefinisikan kata kunci visibilitasnya, maka *method* tersebut bersifat public.

2.2.4 PHP *Parsing*

PHP Parsing merupakan proses pemecahan kode sumber PHP ke dalam bagian-bagian kecil yang menghasilkan suatu informasi kode sumber. Proses pemecahan kode sumber PHP dapat dilakukan dengan menggunakan library yang dapat disisipkan ke dalam sebuah projek (Nikic, 2015). PHP Parser menghasilkan AST dari kode sumber, sehingga dapat diperoleh seluruh informasi dari struktur kode sumber yang diparsing. AST kode sumber direpresentasikan ke dalam array multidimensi. Sebagai contoh, sebuah kode sumber PHP berisikan sintaks <?php echo 'Hi', 'World'. Maka hasil parsing terlihat seperti pada Gambar 2.11.

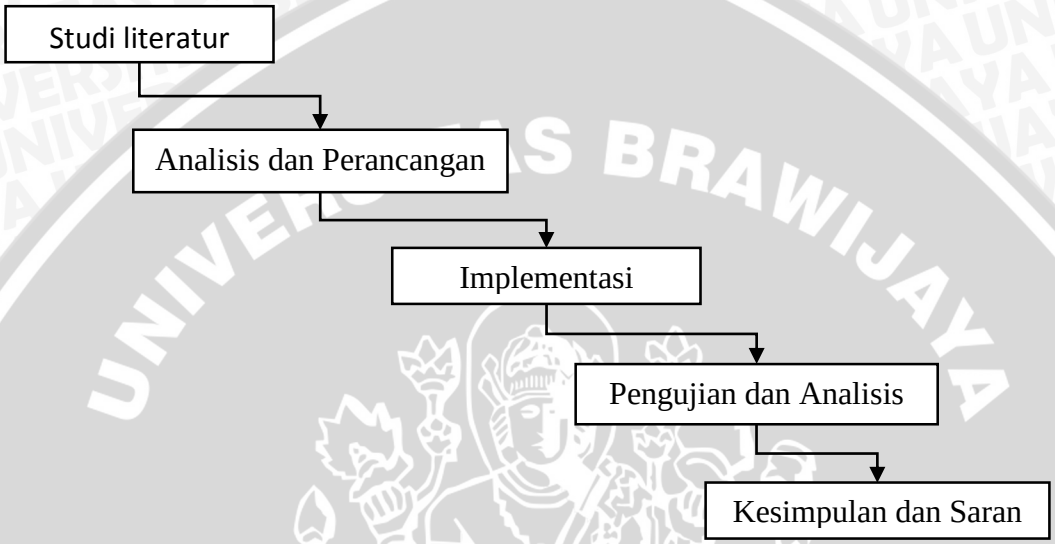
```
array(
    0: Stmt_Echo(
        exprs: array(
            0: Scalar_String(
                value: Hi
            )
            1: Scalar_String(
                value: World
            )
        )
    )
)
```

Gambar 2. 11 Hasil *Parsing* Kode Sumber PHP

Pada Gambar 2.11 dapat dilihat bahwa hasil parsing sesuai dengan struktur kode yaitu, sebuah pernyataan echo dengan dua string sebagai ekspresinya. Eskpresi pertama dengan nilai Hi dan yang kedua adalah World. AST mengabaikan bagian white space, tetapi dapat menunjukkan lokasi bagian kode tertentu yang ada pada kode sumber (Nikic, 2015). Misalnya baris awal dan baris akhir suatu fungsi dan *class*.

BAB 3 METODOLOGI

Metodologi penelitian menjelaskan mengenai langkah-langkah yang dilakukan dalam penelitian tentang pendeteksian kloning kode pada kode sumber PHP dengan metode *IOE-Behavior*. Langkah-langkah yang dilakukan dalam penelitian ini adalah studi literatur, analisis kebutuhan, perancangan, implementasi, pengujian dan analisis, kemudian ditutup dengan kesimpulan dan saran. Gambar 3.1 menunjukkan tahapan-tahapan penelitian yang akan dilakukan.



Gambar 3. 1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Pada tahap ini dilakukan studi literatur yang dari berbagai sumber yaitu jurnal, buku dan artikel terkait dengan masalah yang dibahas dalam penelitian ini. Dari beberapa sumber yang ada, dapat diperoleh berbagai permasalahan yang muncul pada proses pemeliharaan perangkat lunak yang diakibatkan oleh adanya kloning kode, serta penelitian yang telah dilakukan untuk menyikapi permasalahan tersebut. Selain itu, dari literature yang ada peneliti mendapatkan referensi mengenai objek-objek penelitian dimana metode-metode penelitian kloning kode telah diterapkan.

Penelitian ini membahas tentang bagaimana menerapkan metode *IOE-Behavior* untuk mendeteksi kloning kode secara semantik pada kode sumber PHP dan berapa akurasi dari hasil penerapan metode tersebut. Untuk mendukung tahapan penelitian selanjutnya, maka disusun pula dasar teori mengenai kloning kode, metode IOE Behavior dan PHP dari berbagai sumber yang ada.

3.2 Analisis dan Perancangan

Tahap analisis dilakukan untuk menganalisis permasalahan dan memodelkan kasus pendeteksian kloning kode secara semantik guna mendukung proses pengembangan aplikasi. Proses analisis kebutuhan memberikan gambaran umum

sistem yang dibangun dengan mengidentifikasi aktor, kebutuhan fungsional dan kebutuhan non-fungsionalitas sistem. Hasil dari analisis kebutuhan sistem dapat menunjukkan sejauh mana sistem dikembangkan dan dapat digunakan sebagai standar untuk proses pengujian validasi sistem setelah dilakukan implementasi.

Metode analisis yang digunakan dalam penelitian ini adalah *Object-Oriented Analysis (OOA)* dengan menggunakan bahasa pemodelan *Unified Modelling Language (UML)*. Diagram UML yang digunakan dalam proses analisis kebutuhan adalah diagram dan skenario *Use Case*. Diagram *Use Case* digunakan untuk memodelkan kebutuhan fungsionalitas sistem berdasarkan sudut pandang pengguna. Sedangkan skenario *Use Case* digunakan untuk mendeskripsikan bagaimana suatu *Use Case* berjalan.

Tahap perancangan dilakukan setelah tahap analisis kebutuhan. Perancangan merupakan proses untuk memperoleh desain sistem yang menunjukkan alur kerja sistem dalam menerapkan metode *IOE-Behavior* dengan menggunakan *Object Oriented Design (OOD)*. Proses yang dilakukan dalam tahap perancangan adalah pembuatan desain basis data, desain antarmuka sistem, memodelkan objek-objek yang terlibat dengan sistem dalam penyelesaian masalah beserta relasinya ke dalam *class diagram* dan menggambarkan bagaimana objek-objek tersebut saling berinteraksi menggunakan *sequence diagram*.

3.3 Implementasi

Proses implementasi atau pembuatan aplikasi meliputi pemilihan bahasa pemrograman yang digunakan dalam pengembangan sistem pendeteksian kloning kode dengan menggunakan metode *IOE-Behavior*. Hasil dari tahap perancangan diimplementasikan ke dalam bahasa pemrograman sehingga menghasilkan suatu sistem yang dapat digunakan oleh pengguna. Pada penelitian ini sistem dibangun dengan menggunakan bahasa pemrograman PHP. Pemilihan bahasa pemrograman karena adanya proses testing, dimana sistem menjalankan kandidat kloning agar menghasilkan informasi berupa hasil keluaran kode. Informasi keluaran ini digunakan pada salah satu proses pendeteksian. Selain itu sistem juga menggunakan bahasa HTML yang didukung dengan CSS agar antar muka sistem lebih untuk membantu pengguna dalam berinteraksi dengan sistem yang dibangung. Pada tahap akhir dilakukan simulasi sistem menggunakan browser

3.4 Pengujian dan Analisis

Pengujian dilakukan untuk mengetahui apakah sistem yang dibangun telah sesuai dengan kebutuhan dan berapa akurasi sistem yang menerapkan metode *IOE-Behavior* dalam mendeteksi kloning kode secara semantik pada kode sumber PHP. Strategi pengujian yang dilakukan terdiri pengujian unit, pengujian integrasi dan pengujian validasi.

Pengujian unit dilakukan untuk memastikan bahwa unit-unit yang memiliki prioritas tinggi memiliki hasil implementasi yang sesuai dengan harapan di awal analisis kebutuhan. Teknik yang pengujian yang digunakan adalah teknik

pengujian white box dengan jenis pengujian basis path. Proses pengujian basis path yaitu dengan menentukan tiga unit yang akan diujikan sesuai dengan prioritas, kemudian mengeksekusi proses pengujian basis path.

Pengujian integrasi dilakukan dengan menguji *class* yang saling berinteraksi. Diawali dengan membuat kasus uji yang terdiri dari rincian masukan dan hasil yang diharapkan. Kemudian menjalankan sistem sesuai kasus uji dan membandingkan antara hasil sebenarnya dengan hasil yang diharapkan.

Pengujian validasi dilakukan dengan menggunakan teknik pengujian black-box. Pendekatan yang dilakukan adalah dengan menguji sistem hasil implementasi berdasarkan spesifikasi kebutuhan sistem. Proses pengujian validasi adalah dengan membuat kasus uji untuk setiap kebutuhan, kemudian mengeksekusi masing-masing kasus uji. Pada pengujian validasi ini, sistem dianggap sudah berjalan dengan benar apabila tidak terdapat kesalahan ketika kasus uji dijalankan

3.5 Penutup

Penutup terdiri dari kesimpulan dan saran. Proses pembuatan kesimpulan dilakukan setelah tahap analisis hingga pengujian sistem selesai. Sehingga diperoleh inti dari keseluruhan proses penelitian. Kesimpulan dibuat dengan tujuan untuk menjawab rumusan masalah. Berdasarkan kesimpulan yang diperoleh, penulis dapat menuliskan saran untuk memperbaiki kekurangan atau kesalahan yang ada pada penelitian ini guna memberikan pertimbangan untuk penelitian selanjutnya.

BAB 4 REKAYASA KEBUTUHAN

4.1 Gambaran Umum Sistem

Gambaran umum sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan menerapkan metode *IOE-Behavior* ini terdiri dari dua bagian, yaitu deskripsi umum sistem dan lingkungan sistem.

4.1.1 Deskripsi Umum Sistem

Sistem yang dibangun dalam penelitian ini adalah sebuah perangkat lunak dengan judul "Pendeteksian Kloning Kode Secara Semantik Pada Kode SUmber PHP dengan Metode *IOE-Behavior*". Tujuan utama dari sistem ini adalah untuk mengetahui mendeteksi keberadaan kloning kode secara semantik pada kode sumber PHP dengan menggunakan informasi masukan, keluaran dan efek dari suatu *method* PHP. Pengguna diharapkan dapat memperoleh informasi mengenai *method* mana saja yang merupakan kloning kode secara semantik serta lokasi kloning kode tersebut dalam kode sumber yang dideteksi.

Proses pendeteksian diawali dengan pengguna memberikan masukan berupa kode sumber PHP ke dalam sistem. Kemudian sistem melakukan proses pendeteksian kloning kode secara semantik pada kode sumber PHP dan menghasilkan keluaran berupa daftar informasi kloning kode dari hasil pendeteksian kode sumber yang telah dimasukkan pengguna.

4.1.2 Lingkungan Sistem

Sistem pendeteksian kloning kode secara semantik ini dibangun dengan menggunakan bahasa pemrograman PHP dan *faremwork* CodeIgniter, sehinga membutuhkan web server untuk berjalan. Sistem ini dapat berjalan di seluruh sistem operasi yang sudah terpasang XAMPP minimal versi V3.2.1.

4.2 Analisis Kebutuhan

Analisis kebutuhan ini bertujuan untuk mendefinisikan seluruh kebutuhan yang harus disediakan oleh sistem untuk memenuhi kebutuhan dari pengguna. Proses analisis kebutuhan diawali dengan melakukan identifikasi terhadap aktor yang terlibat dengan sistem. Berikutnya ialah menjabarkan kebutuhan sistem, baik kebutuhan fungsional maupun non-fungsional dan memodelkannya ke dalam bentuk diagram dan skenario *Use Case*.

4.2.1 Identifikasi Aktor

Pada tahap ini dilakukan identifikasi aktor yang dapat berinteraksi dengan sistem pendeteksian kloning kode secara semantik pada kode sumber PHP. Sistem ini tidak memiliki spesifikasi khusus untuk aktor, sehingga semua pengguna dapat mengakses dan menjalankan sistem

4.2.2 Identifikasi Kebutuhan Perangkat Lunak

Kebutuhan sistem dibagi ke dalam dua kelompok kebutuhan, yaitu kebutuhan fungsional dan kebutuhan non-fungsional. Kebutuhan fungsional sistem merupakan pernyataan tentang sekumpulan layanan atau fitur yang harus tersedia dalam perangkat lunak. Sedangkan kebutuhan non-fungsional sistem didefinisikan untuk mengetahui seberapa baik standar yang dapat dihasilkan oleh sistem. Daftar kebutuhan fungsional sistem dapat dilihat pada Tabel 4.1 dan daftar kebutuhan non-fungsional sistem ditunjukkan pada Tabel 4.2.

Tabel 4. 1 Kebutuhan Fungsional Perangkat Lunak

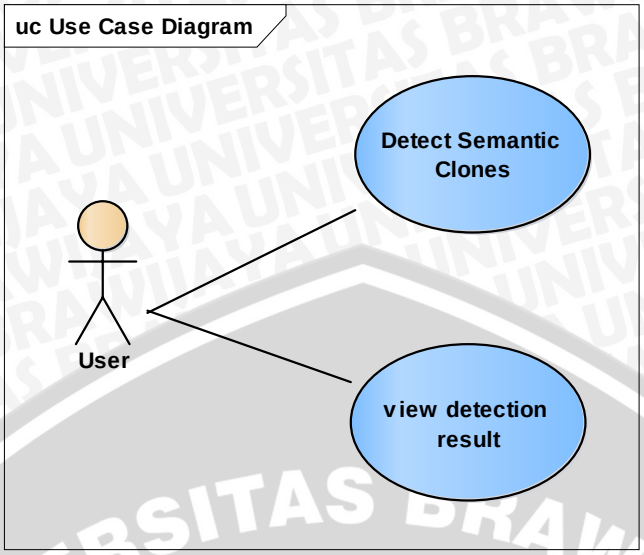
Kode Kebutuhan	Kebutuhan	Nama *Use Case*
SRS_F_CD_10	Sistem harus mampu menyediakan fasilitas untuk mendeteksi kloning kode secara semantik dengan metode *IOE-Behavior* pada kode sumber PHP yang telah ditentukan oleh pengguna.	*Detect Semantic Clones*
SRS_F_CD_20	Sistem harus mampu menyediakan fasilitas untuk menampilkan seluruh hasil pendeteksian yang dilakukan oleh sistem dan rincian setiap *method* yang dinyatakan sebagai kloning kode secara semantik	*View Detection Result*

Tabel 4. 2 Kebutuhan Non-Fungsional Perangkat Lunak

Kode Kebutuhan	Kebutuhan
SRS_NF_CD_1	Sistem pendeteksian kloning kode secara semantik yang menerapkan metode *IOE-Behavior* harus memiliki tingkat akurasi lebih dari 95%.

4.2.3 Diagram *Use-Case*

Diagram *Use Case* memberikan gambaran mengenai fungsionalitas yang diharapkan dari sistem dengan aktor yang terlibat. Diagram *Use Case* dari sistem ini dapat dilihat pada Gambar 4.1.



Gambar 4. 1 Diagram *Use-Case*

Berikut penjelasan dari diagram *use-case* pada Gambar 4.1 :

1. *Detect Semantic Clones*
   Pada *use-case Detect Semantic Clones* aktor dapat melakukan pendeteksian kloning kode secara semantik pada kode sumber PHP. Proses pendeteksian dilakukan oleh sistem menggunakan metode *IOE-Behavior*. Sebelum melakukan pendeteksian, aktor harus memilih kode sumber yang akan dideteksi. Aktor dapat menambahkan kode sumber yang akan dideteksi apabila kode sumber tersebut belum terdapat di dalam direktori sistem.
2. *View Detection Result*
   Pada *use-case View Detection Result* aktor dapat melihat riwayat pendeteksian, rincian hasil pendeteksian untuk masing-masing proses pendeteksian yang pernah dilakukan oleh sistem, rincian kloning kode secara semantik yang pernah dideteksi oleh sistem berdasarkan berkas kode sumber atau berdasarkan kelompok kloning serta perbandingan kode *method* yang dinyatakan sebagai kloning kode pada kelompok yang sama.

4.2.4 Skenario *Use-Case*

Skenario *Use Case* mendeskripsikan tentang cara kerja masing-masing *Use Case* yang digambarkan dalam diagram *Use Case* pada Gambar 4.1. Setiap *Use Case* memiliki nama *Use Case*, aktor yang terlibat, tujuan, langkah-langkah utama *Use Case* (*main flow*), kondisi awal yang harus dipenuhi (*pre-condition*), langkah alternatif *Use Case* (*alternative flow*) dan kondisi akhir yang diharapkan (post-condition) pada skenarionya.

Tabel 4. 3 Skenario *Use-Case Detect Semantic Clones*

Nomor *Use Case*	SRS_F_CD_10
Nama	*Detect Semantic Clones*
Aktor	User
Tujuan	Sistem mampu mendeteksi kloning kode secara semantik dengan metode *IOE-Behavior* pada semua kode sumber yang telah ditentukan oleh pengguna.
Pre-condition	Terdapat kode sumber yang telah diunggah oleh aktor ke dalam sistem.
Main flow	1. Aktor masuk ke halaman "*Detection"* 2. Sistem menampilkan halaman "*Detection*" 3. Aktor memilih kode sumber yang akan dideteksi. 4. Aktor menekan tombol *"Detect"*. 5. Sistem menjalankan proses pendeteksian kemudian menampilkan tabel hasil pendeteksian pada halaman "*Detection Result*".
Alternative Flow 1	1. Jika kode sumber belum terdapat pada direktori sistem 2. Aktor menekan tombol "choose file" 3. Sistem menampilkan jendela baru untuk mencari dan memilih berkas. 4. Aktor menekan tombol "*Upload*". 5. Sistem mengunggah berkas kode sumber PHP ke direktori sistem dan menyimpan informasi berkas pada basis data sistem 6. Aktor menjalankan alur utama.
Post-condition	Sistem dapat menampilkan hasil pendeteksian kloning kode secara semantik.

Tabel 4. 4 Skenario *Use-Case View Detection Result*

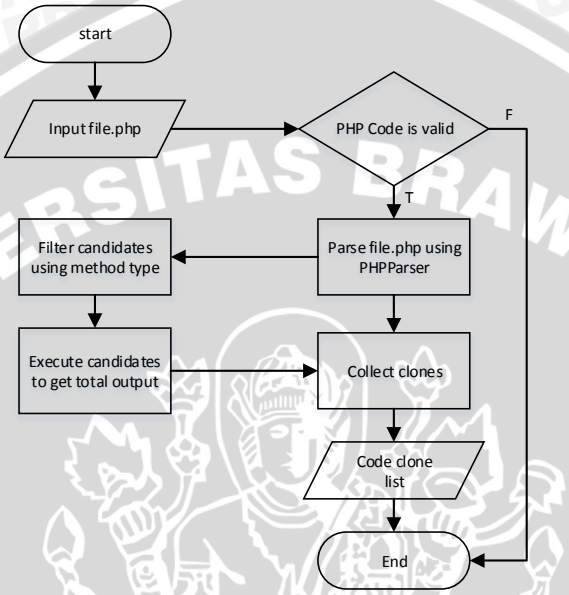
Nomor *Use Case*	SRS_F_CD_30
Nama	*View Detection Result*
Aktor	User
Tujuan	Sistem mampu menampilkan seluruh data hasil pendeteksian yang pernah dilakukan oleh sistem.
Pre-condition	Terdapat hasil pendeteksian kloning kode secara semantik di dalam sistem.
Main flow	1. Aktor masuk ke halaman "*Detection Result"*

	2. Sistem menampilkan tabel berisi informasi proses pendeteksian yang pernah dilakukan sistem pada halaman "*Detection Result*". 3. Aktor menekan salah satu *link* pada salah satu proses pendeteksian. 4. Sistem menampilkan tabel statistik hasil pendeteksian
Alternative flows 1	1. Aktor menekan *link* pada salah satu proses pendeteksian. 2. Aktor menekan salah satu *link* yang terdapat pada kolom "*number of clones*". 3. Sistem menampilkan informasi *method* yang dinyatakan sebagai kloning kode secara semantik.
Alternave Flows 2	1. Aktor menjalankan alur alternatif 1 2. Aktor menekan salah satu *link* pada kolom "*Clone Group*" 3. Sistem menampilkan rincian hasil pendeteksian berupa daftar method yang termasuk dalam kelompok kloning kode yang sama.
Alternative Flows 3	1. Aktor menjalankan alur alternatif 1 2. Aktor menjalankan alur alternatiif 2 3. Aktor menekan salah satu *link* pada kolom "*method code"* 4. Sistem menampilkan bagian kode sumber *method.*
Post-condition	Aktor dapat melihat seluruh informasi hasil pendeteksian kloning kode secara semantik yang telah dilakukan oleh sistem.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Algoritma Pendeteksian

Alur pendeteksian kloning kode secara semantik pada sistem yang menerapkan metode IOE Behavior digambarkan kedalam diagram alir untuk mempermudah dalam proses implementasi. Diagram alir dari algoritma pendeteksian sistem dapat dilihat pada Gambar 5.1.



Gambar 5. 1 Algoritma Pendeteksian Sistem

Pada Gambar 5.1 terdapat 4 proses dalam pendeteksian yang memiliki alur masing-masing yaitu proses. Sebelum melakukan keempat proses pendeteksian tersebut, dilakukan pengecekan kondisi kode sumber PHP yang akan dideteksi apakah kode sumber tersebut valid. Kode sumber dinyatakan valid untuk dilakukan proses pendeteksian apabila struktur kode di dalamnya memenuhi prinsip dasar pemrograman berorientasi objek pada bahasa pemrograman PHP, terlepas dari *framework* dan bahasa pemrograman lainnya. Pengecekan kondisi ini dilakukan secara manual atau diluar sistem. Keluaran dari proses pendeteksian adalah informasi *method* yang dinyatakan sebagai kloning kode secara semantik pada kode sumber PHP yang telah dideteksi oleh sistem dengan menerapkan metode *IOE-Behavior*.

Proses selanjutnya adalah *parsing* kode sumber dengan menggunakan *library* PHPParser. Proses *parsing* yang dilakukan merupakan implementasi proses *abstraction* pada metode *IOE-Behavior*. Hasil dari proses ini adalah AST dari kode sumber. AST yang dihasilkan oleh PHPParser direpresentasikan dalam bentuk *array*. Informasi hasil *parsing yang* digunakan untuk proses pendeteksian diantaranya adalah infromasi nama *class*, nama *method*, jumlah parameter, tipe *method* dan lokasi *method* yang kemudian disimpan ke dalam basis data sistem untuk proses selanjutnya.

Proses *filtering* secara otomatis dilakukan oleh sistem setelah proses *parsing* selesai. Pada proses ini dilakukan pengelompokan kandidat kloning berdasarkan data *method* yang disimpan dalam basis data sistem. Masing-masing *method* disebu dengan kandidat kloning. Pengelompokan kandidat kloning dilakukan berdasarkan jumlah parameter *method* dan tipe *method*. Sehingga diperoleh kelompok kandidat kloning dimana anggotanya memiliki kesamaan pada jumlah parameter dan tipe *method*. Kemudian kelompok kloning yang hanya memiliki satu kandidat sebagai anggota akan dieliminasi dari proses pendeteksian.

Proses *testing* secara otomatis dilakukan oleh sistem setelah proses sebelumnya selesai. Pada proses *testing* dilakukan pengeksekusian kandidat kloning yang masih belum tereliminasi pada proses *filtering*. Kandidat kloning dieksekusi sebanyak lima kali dimana pada masing-masing proses eksekusi diberikan argumen yang berbeda. Proses *testing* menghasilkan informasi *total output* seluruh kandidat kloning yang kemudian disimpan ke dalam basis data sistem untuk proses selanjutnya.

Proses *collection* secara otomatis dilakukan oleh sistem setelah proses *testing* selesai. Pada proses *collection* dibuat sub-kelompok dari kelompok kloning hasil proses filtering. Pengelompokan dilakukan berdasarkan *total output* masing-masing kandidat koning. Sub-kelompok yang hanya memiliki satu kandidat kloning sebagai anggota akan dieliminasi. Kandidat kloning yang masih bertahan dalam sub-kelompok ini lah yang dinyatakan sebagai kloning kode secara semantik pada kode sumber PHP.

Hasil akhir dari proses pendeteksian yang dilakukan oleh sistem adalah informasi *method* yang dinyatakan sebagai kloning kode secara semantik pada kode sumber PHP. Informasi kloning kode secara semantik yang terdeteksi ditampilkan dalam bentuk tabel.

5.2 Perancangan Perangkat Lunak

Pada tahap perancangan dilakukan beberapa hal yang digunakan sebagai acuan pada proses implementasi, yaitu pembahasan algoritma pendeteksian yang menerapkan metode *IOE-Behavior* serta pembuatan diagram-diagram perancangan sistem meliputi sequence diagram, *class* diagram, relational data model dan rancangan antarmuka pengguna..

5.2.1 Perancangan Arsitektur

Perancangan arsitektur adalah tahap pertama dalam proses perancangan perangkat lunak. Perancangan arsitektur berkaitan dengan bagaimana sistem harus terorganisir dengan cara merancang struktur keseluruhan sistem. Output dari perancangan arsitektur adalah *class* diagram berdasarkan dari identifikasi objek menggunakan sequence diagram.

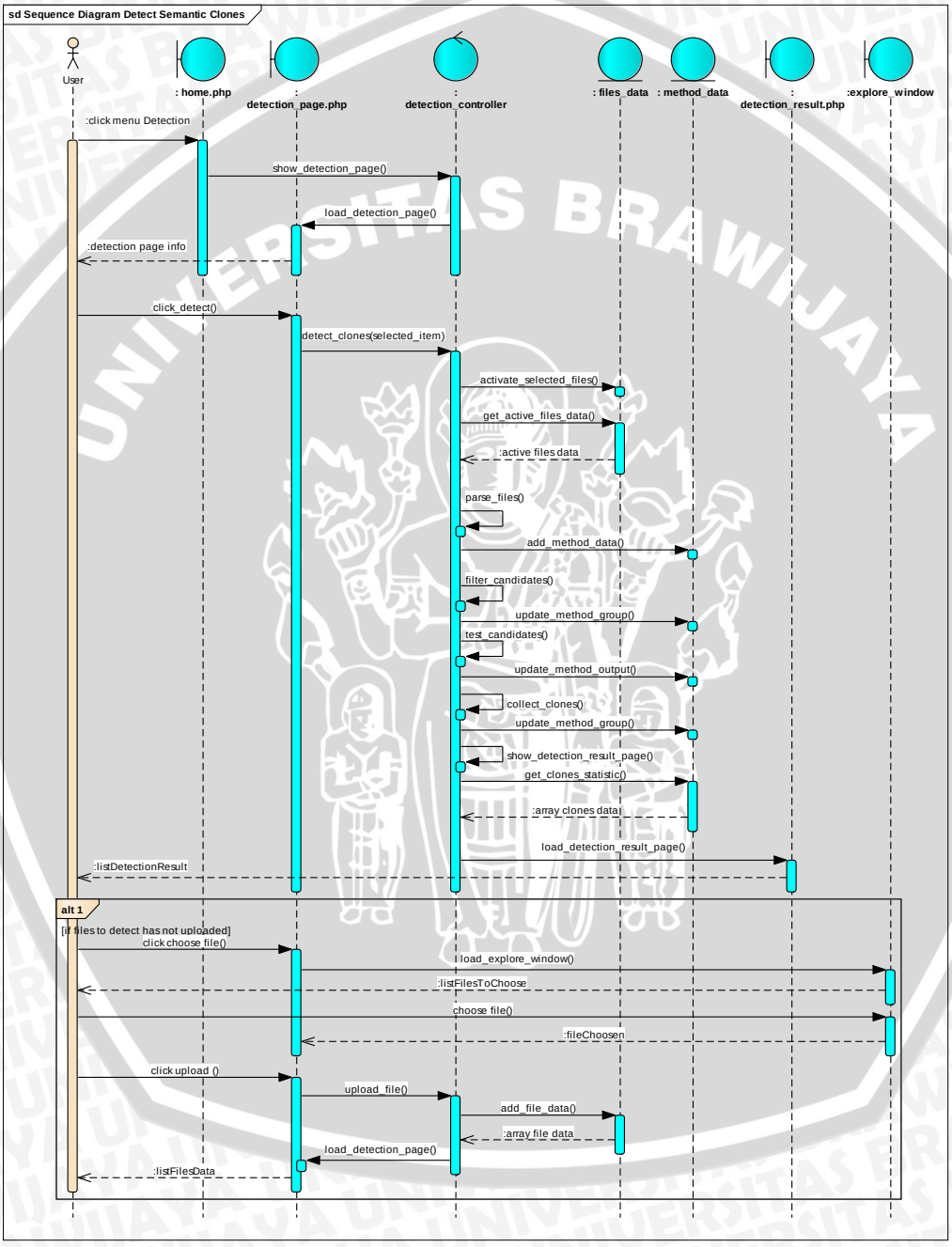
5.2.1.1 Perancangan Diagram *Sequence*

Sequence diagram dibuat berdasarkan pada masing-masing scenario *use-case yang* telah didefinisikan pada saat proses analisis kebutuhan fungsional perangkat

lunak. Berikut merupakan *sequence diagram* sistem pendeteksian kloning kode secara semantik pada kode sumber PHP.

1. Diagram *Sequence Use-Case Detect Semantic Clones*

Sequence diagram untuk detect semantic clones merupakan model interaksi antar objek yang berada dalam proses mendeteksi kloning kode secara semantik pada berkas yang telah diunggah oleh pengguna.



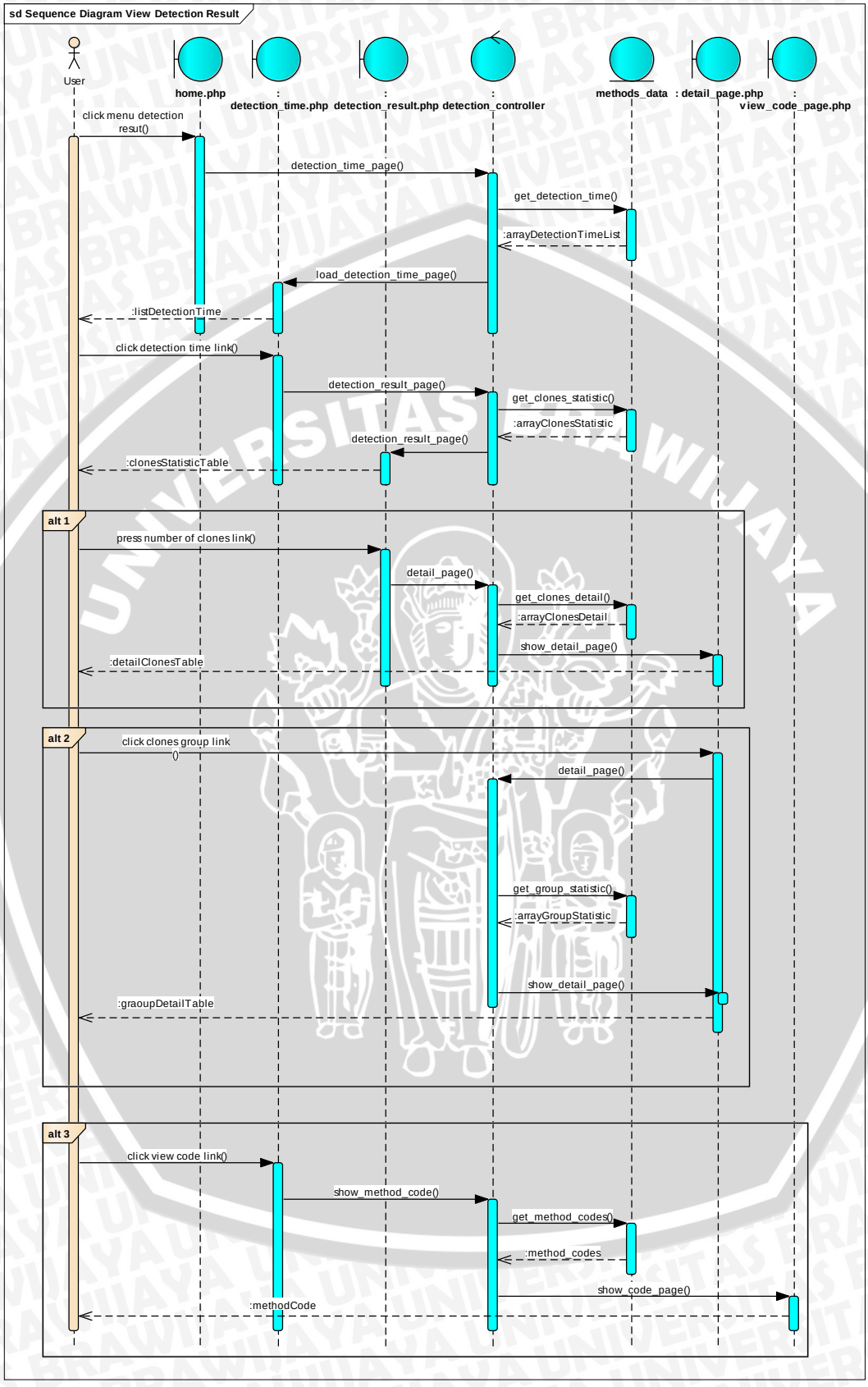
Gambar 5. 2 Diagram *Sequence Use-Case Detect Semantic Clones*

Gambar 5.2 menjelaskan bahwa proses cari berkas berawal dari pengguna memberi perintah dengan menekan menu *detection*. Kemudian sitem menampilkan antarmuka halaman *detection*. Pengguna kemudian memilih kode sumber yang akan dideteksi dengan menggunakan *form check box* yang tersedia. Setelah memilih kode sumber yang akan dideteksi pengguna menekan tombol "detect". Ketika tombol tersebut ditekan, sistem menjalankan fungsi untuk melakukan pendeteksian pada kode sumber yang telah dipilih oleh pengguna. Hasil pendeteksian ditampilkan pada antarmuka pengguna ketika sistem telah selesai melakukan proses pendeteksian.

Selanjutnya apabila kode sumber yang akan dideteksi belum pernah ditambahkan ke dalam direktori sistem, maka pengguna dapat menambahkannya dengan menggunakan *form upload* yang tersedia pada halaman *detection*. Pengguna memilih kode sumber yang akan ditambahkan, kemudian menekan tombol "upload". Ketika pengguna menekan tombol tersebut maka sistem menjalankan fungsi untuk menambahkan berkas kode sumber ke dalam direktori sistem dan menyimpan informasi berkas tersebut ke dalam basis data sistem.

2. Diagram *Sequence Use-Case View Detection Result*

Sequence diagram untuk *View Detection Result* merupakan model interaksi antar objek yang berada dalam proses menampilkan seluruh hasil pendeteksian kloning kode secara semantik yang telah dilakukan oleh sistem.



Gambar 5. 3 Diagram *Sequence Use-Case View Detection Result*

Gambar 5.3 menjelaskan bahwa proses cari berkas berawal dari pengguna memberi perintah dengan menekan menu *detection result*. Sistem kemudian menampilkan informasi riwayat pendeteksian yang pernah dilakukan oleh sistem. Pengguna memilih salah satu proses pendeteksian dengan menekan *link* yang tersedia, kemudian sistem menampilkan tabel statistik hasil dari salah satu proses pendeteksian.

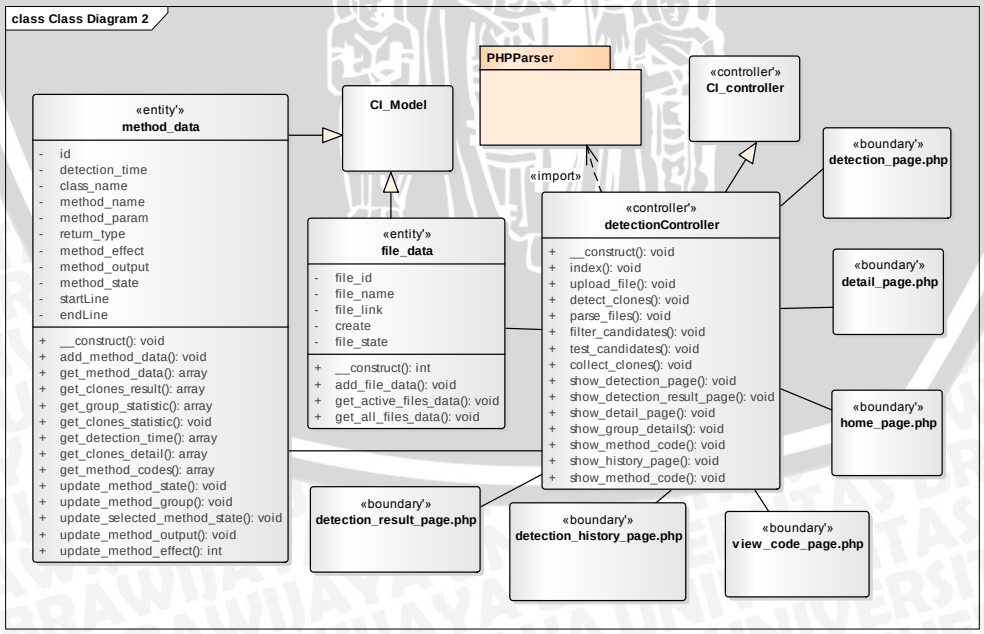
Apabila pengguna ingin melihat rincian *method* dari salah satu proses pendeteksian, maka pengguna dapat menekan *link* yang menunjukkan jumlah kloning kode yang terdeteksi. Kemudian sistem akan menampilkan rincian *method* yang dinyatakan sebagai kloning kode.

Apabila pengguna ingin melihat rincian *method* yang dinyatakan sebagai kloning kode secara semantik berdasarkan kelompok kloningnya, maka pengguna dapat menekan *link* yang menunjukkan kelompok kloning dari *method* tersebut. Kemudian sistem akan menampilkan rincian *method* yang dinyatakan sebagai kloning kode pada kelompok kloning yang saman.

Apabila pengguna ingin melihat potongan kode sumber dari *method* yang dinyatakan sebagai kloning kode secara semantik dan membandingkan dengan *method* pada kelompok kloning yang sama, maka pengguna dapat menekan *link* "*view code*" yang tersedia. Kemudian sistem akan menampilkan potongan kode sumber dari *method* yang berada dalam kelompok kloning yang sama.

5.2.1.2 Perancangan Diagram *Class*

Diagram *class* merupakan diagram yang digunakan untuk merepresentasikan struktur dari *class* yang diimplementasikan ke dalam sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior*.



Gambar 5. 4 Diagram *Class*

Pada Gambar 5.4 menunjukkan *class diagram* yang akan diimplementasikan ke dalam sistem pendeteksian. Hasil perancangan *class diagram* terdiri dari sebelas *class* dan satu *package*. Berikut penjelasan untuk masing-masing *class*.

1. CI_Controller

   Class yang digunakan sebagai class induk controller pada *framework* CI.

2. CI_Model

   Class yang digunakan sebagai class induk model pada *framework* CI.

3. detection_controller

   Class yang digunakan sebagai kontroller seluruh proses yang ada pada sistem pendeteksian pada kode sumber PHP dengan metode *IOE-Behavior*.

4. files_data

   Class yang digunakan sebagai entitas data berkas kode sumber yang diggunakan pada proses pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior*.

5. methods_data

   Class yang digunakan sebagau entitas data *method* yang digunakan pada proses pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior.*

6. detection_page.php

   Class boundary yang digunakan sebagai antarmuka pengguna untuk berinterasksi dengan sistem untuk melakukan pendeteksian kloning kode secara semantik pada kode sumber PHP.

7. detail_page.php

   Class yang digunakan sebagai antarmuka penguna untuk menampilkan rincian *method* yang dinyatakan sebagai kloning kode secara semantik dari hasil pendeteksian.

8. home_page.php

   Class yang digunakan sebagai antarmuka pengguna untuk menampilkan halaman utama sistem.

9. view_code_page.php

   Class boundary yang digunakan sebagai antarmuka pengguna untuk menampilkan halaman rincian potongan kode dari method yang dinyatakan sebagai kloning kode secara semantik.

10. detection_history_page.php

Class yang digunakan sebagai antarmuka pengguna untuk berinteraksi dengan sistem untuk menampilkan halaman riwayat proses pendeteksian yang pernah dilakukan oleh sistem.

11. detection_result_page.php

    Class yang digunakan sebagai antarmuka pengguna untuk berinteraksi dengan sistem untuk menampilkan halaman hasil pendeteksian.

5.2.2 Perancangan Komponen

Perancangan komponen merupakan dekomposisi sub-sistem menjadi komponen detail. Perancangan komponen menjelaskan atribut dan algoritma method dalam sebuah *class* yang telah dimodelkan sebelumnya pada pemodelan *class* diagram. Dalam perancangan komponen pada sistem ini mengambil sample 3 *method* utama pada *class* kontroller detection_controller.

5.2.2.1 Algoritma Method filter_candidates

Nama *Class* : detection_controller

Nama *Method* : filter_candidates()

Algoritma : (ALGO_CD_01)

```
FUNCTION filter_candidates;
DB SELECT method data, grouped return type, grouped effect
FOREACH method data
    IF(method type match return type group)
      UPDATE DB method group match to method type group;
    ENDIF
    IF(method effect match effect group)
      UPDATE DB method group match to effect group;
    ENDIF
END FOR
Eliminate candidates from group that has <=1 member;
END filtering_ast
```

5.2.2.2 Algoritma Method test_candidates

Nama *Class* : detection_controller

Nama *Method* : test_candidates()

Algoritma : (ALGO_CD_02)

```
FUNCTION testing_method;
DB SELECT filtered method data, all file data;
SET array argument 1 to 5;
FOREACH file data
 FOREACH method data
  IF(file_id in method data == file_id in current file)
    INCLUDE current file;
    FOR i=1 to 5
      CALL method_name using arguments array index i;
    ENDFOR
  END IF
 END FOR
END FOR
END testing_method
```

5.2.2.3 Algoritma Method collect_clones

Nama *Class* : detection_controller

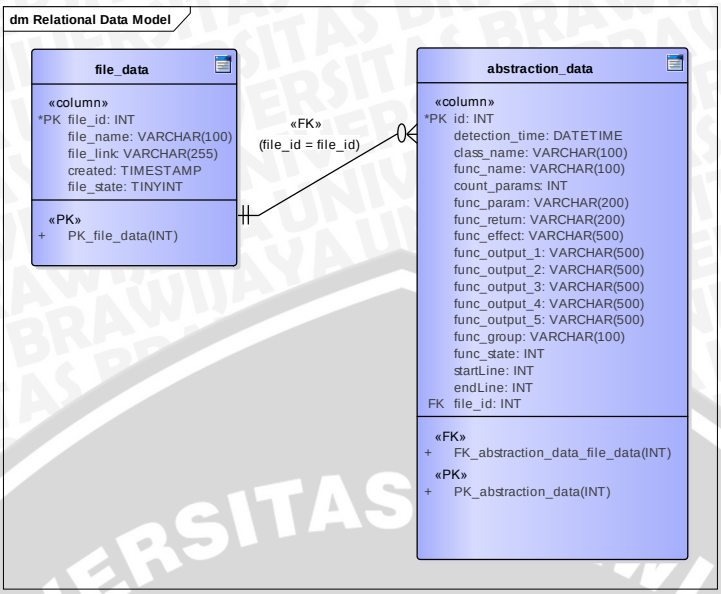
Nama Method : collect_clones()

Algoritma : (ALGO_CD_03)

```
FUNCTION collect_clones;
DB SELECT grouped return value, tested method data
FOREACH return value
   FOREACH tested method data
    IF(return value in tested method match current return value)
       UPDATE tested method group;
    END IF
   END FOR
  END IF
END FOR
END collect_clones
```

5.2.3 Perancangan Basis Data

Perancangan digunakan untuk membuat rancangan bagaimana data direkam dalam sistem untuk mendukung pross pendeteksian. Berdasarkan diagram kelas level analisis yang telah dimodelkan sebelumnya, maka diperoleh rancangan basis data yang direpresentasikan dalam bentuk *relational data model* seperti yang ditunjukkan pada Gambar 4.4.



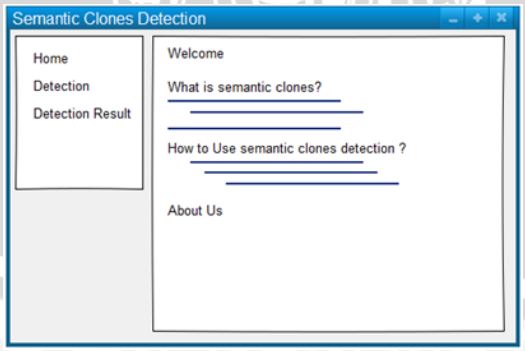
Gambar 5. 5 *Relational Data Model*

Gambar 5.4 Menunjukkan rancangan basis data yang digunakan untuk proses pendeteksian kloning kode secara semantik pada kode sumber PHP. Basis data yang dirancang terdiri dari dua tabel yaitu tabel files_data dan methods_data. Tabel files_data berisi data berkas kode sumber yang dideteksi, sedangkan tabel abstraction_data digunakan sebagai tabel untuk menyimpan data *method* dari hasil proses *abstraction*.

5.2.4 Perancangan Antarmuka Pengguna

Perancangan antarmuka pengguna merupakan proses pembuatan rancangan tampilan sistem yang akan dibangun agar pengguna dapat berinteraksi dengan sistem pendeteksian. Berikut adalah rancangan antarmuka untuk halaman-halaman yang ada pada sistem.

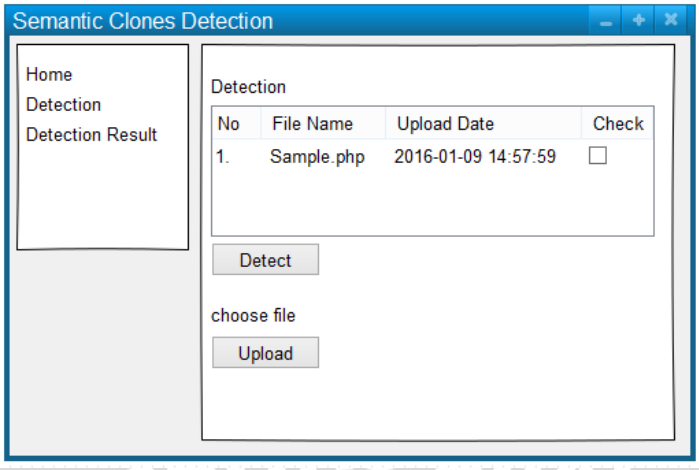
1. Perancangan Antarmuka Halaman Utama



Gambar 5. 6 Rancangan Antarmuka Halaman Utama

Halaman home page akan menampilkan halaman utama dengan pilihan menu pada header sebelah kiri dan deskripsi singkat mengenai proses pendeteksian kloning kode secara semantik pada kode sumber PHP menggunakan sistem ini.

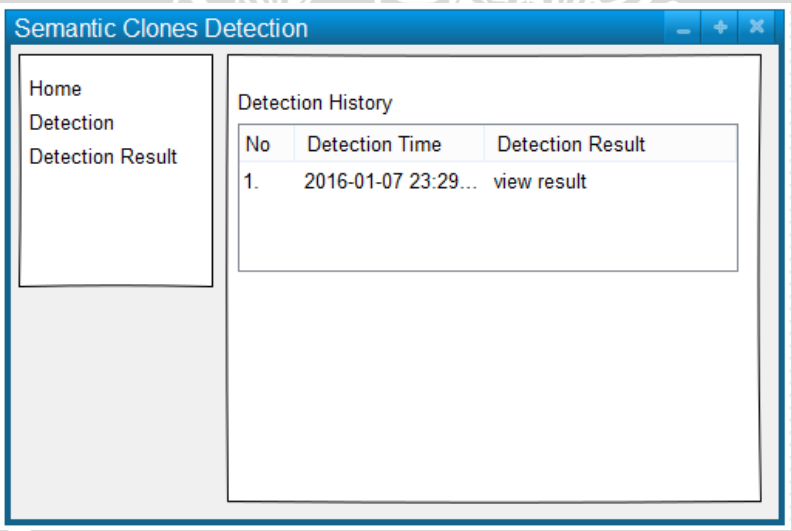
2. Perancangan Antarmuka Halaman *Detection*



Gambar 5. 7 Rancangan Antarmuka Halaman Detection

Halaman detection akan menampilkan tabel berisikan kode sumber PHP yang siap dideteksi oleh sistem. Tersedia *check box* yang digunakan untuk memilih kode sumber yang akan dideteksi, tombol "Detect" yang digunakan untuk memulai proses pendeteksian dengan nama dan tombol "Upload" yang digunakan untuk menambahkan kode sumber yang akan dideteksi.

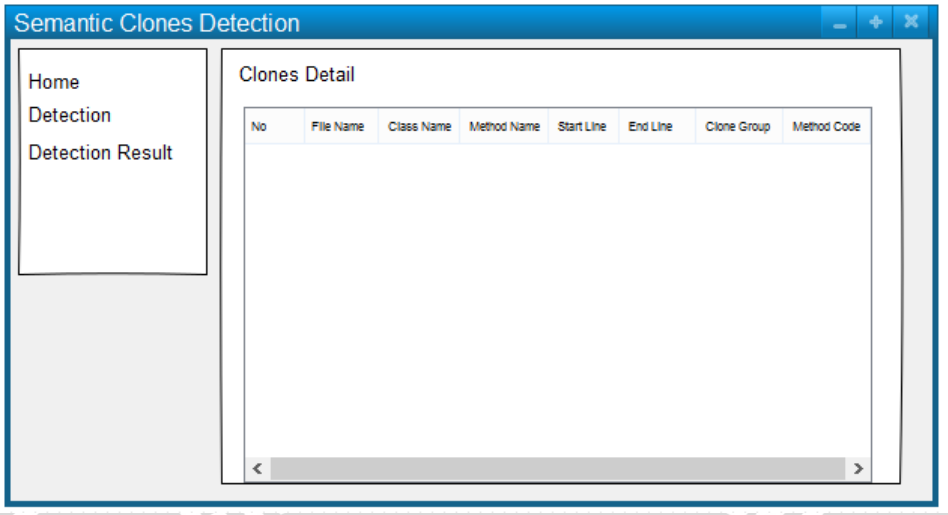
3. Perancangan Antarmuka Halaman *Detection Result*



Gambar 5. 8 Rancangan Antarmuka Halaman Detection Result

Halaman detection result akan menampilkan informasi seluruh hasil pendeteksian yang pernah dilakukan oleh sistem. Kolom detection time nantinya akan berisi link yang dapat digunakan untuk melihat rincian hasil pendeteksian baik berupa statistik hasil pendeteksian, rincian kloning, dan rincian kelompok kloning.

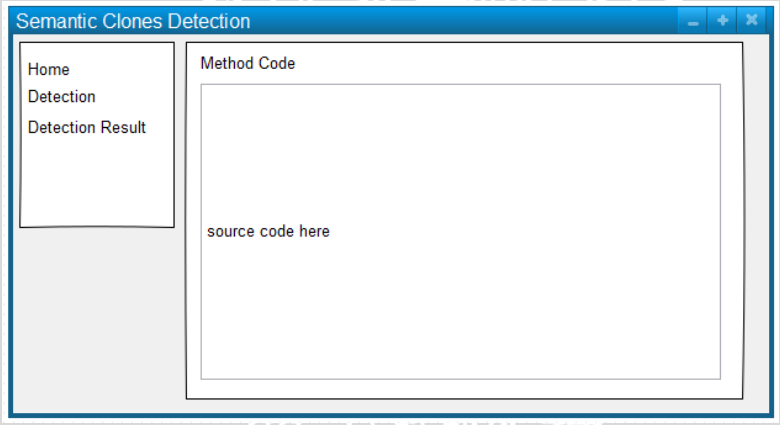
4. Perancangan Antarmuka Halaman *Clones Detail*



Gambar 5. 9 Rancangan Antarmuka Halaman Clones Detail

Halaman Clones Details menunjukkan hasil pendeteksian secara rinci berupa method mana saja yang merupakan klong kode, dimana letak method tersebut dan kelompok kloningnya.

5. Perancangan Antarmuka Halaman *Method Code*



Gambar 5. 10 Rancangan Antarmuka Halaman Method Code

Halaman *method code* merupakan halaman untuk menampilkan bagian kode berupa method yang dinyatakan sebagai kloning kode secara semantik.

5.3 Implementasi

Tahapan implementasi yang dilakukan terdiri dari penjelasan tentang spesifikasi lingkungan pengembangan sistem, batasan-batasan implementasi, implementasi algoritma, implementasi basis data dan implementasi antarmuka pengguna.

5.3.1 Spesifikasi Sistem

Spesifikasi sistem merupakan lingkungan implementasi dalam pembuatan sistem ini yang terdiri dari spesifikasi perangkat keras dan perangkat lunak.

5.3.1.1 Spesifikasi Perangkat Keras

Dalam penelitian ini penulis menggunakan sebuah komputer karena sistem yang dibangun tidak berkomunikasi dengan perangkat keras yang lain. Spesifikasi perangkat keras yang digunakan dalam pembuatan sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dapat dilihat pada tabel 5.1.

Tabel 5. 1 Spesifikasi Perangkat Keras Sistem

Nama Komponen	Spesifikasi
System Model	Asus X200M Notebook PC
Processor	Intel® Coelero®CPU N2920 @1.86GHz
Memory	500GB HDD
Memory RAM	4GB

5.3.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam pembuatan sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dapat dilihat pada table 5.2.

Tabel 5. 2 Sesifikasi Perangkat Lunak Sistem

Nama Komponen	Spesifikasi
Operating System	Windows 8.1 Pro 64-bit
Programming Language	PHP
Framework	Code Igniter
Text Editor	Sublime Text
Web Server	XAMPP v3.2.1
Database	MySQL
Browser	Google Chrome

5.3.2 Batasan Implementasi

Berikut ini merupakan batasan implementasi yang digunakan dalam pembuatan sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior*.

1. Untuk mengakses sistem ini pengguna harus memiliki apache web-server yang sedang aktif.
2. Basis data yang digunakan adalah MySQL.

5.3.3 Implementasi *Class*

Setiap kelas yang telah dirancang pada tahap perancangan sistem direalisasikan pada sebuah file sistem dengan ekstensi *.php. Penjelasan terkait kelas dan file sistem yang digunakan pada sistem akan dijelaskan pada Tabel 5.1.

Tabel 5. 3 Implementasi *Class*

No	***Package***	**Nama Kelas**	**Nama *File* Sistem**
1	*Controllers*	detection_controller	detection_controller.php
2	*Models*	methods_data	fethod_data.php
3	*Models*	files_data	files_data.php
4	*Views*	File HTML	detail_page.php
5	*Views*	File HTML	detection_page.php
6	*Views*	File HTML	detection_history_page.php
7	*Views*	File HTML	footer_view.php
8	*Views*	File HTML	header_view.php
9	*Views*	File HTML	detection_result_page.php
10	*Views*	File HTML	home_page.php
11	*Views*	File HTML	view_code_page.php
12	*Library*	PHPParser *Library*	PHPParser *Library*

5.3.4 Implementasi Algoritma

Berikut adalah hasil implementasi algoritma dari *class* controller cloneDetect dalam bahasa pemrograman PHP pada *class* kontokker dengan menggunakan *framework* CI.

5.3.4.1 Implementasi ALG_CD_01

```
function filtering_ast($detection_time)
{
      $funcparam                =                $this->detect_model-
>get_grouped_funcparam($detection_time);
      $funcreturn               =                $this->detect_model-
>get_grouped_funcreturn($detection_time);
      $astdata                  =                $this->detect_model-
>get_abstraction_data($detection_time);
      $p=1;  $r=1;
      foreach($funcparam as $rowparam)
      {
            foreach($funcreturn as $rowreturn)
            {
                  foreach($astdata as $rowdata)
                  {
                        if($rowdata->func_return=='VALUE')
                        {
                              if(($rowdata-
>count_params==$rowparam->count_params)&&($rowdata-
>func_return==$rowreturn->func_return))
                              {
                                    $data['id']=$rowdata->id;
                                    $data['group']=$p.$r;
      $this->detect_model->group_update($data,$detection_time);
                              }
                        }
                        else
                        {
                              $paramreturn = $rowdata->id;
                              $this->detect_model-
>update_funcid_state($paramreturn,$detection_time);
                        }
                  }
            $r=$r+1;}
      $p=$p+1; }
      $this->eliminate($detection_time); }
```

Gambar 5. 11 Implementasi Algoritma ALG_CD_01

5.3.4.2 *Implementasi Algoritma ALG_CD_02*

```
function testing_method($detection_time)
        {
                $data                   =               $this->detect_model-
>get_filtered_data($detection_time);
                $classx                 =               $this->detect_model-
>get_files_data($detection_time);
                $countdata=count($data); // jumlah method
                $countclassx=count($classx); // jumlah class

                for($i=0;$i<5;$i++)
                {
                        $param1[$i]=rand(70,100);
                        $param2[$i]=rand(20,30);
                        $param3[$i]=rand(30,40);
                        $param4[$i]=rand(40,50);
                        $param5[$i]=rand(50,60);
                }

                for($a=0;$a<$countclassx;$a++)
                {
                        $fileurl='assets/files/'.$classx[$a]-
>file_link;
                        require $fileurl;
                        $class_file_id = $classx[$a]->file_id;

                        for($b=0;$b<$countdata;$b++)
                        {
                                $func_file_id = $data[$b]->fk_file_id;
                                if($func_file_id==$class_file_id)
                                {
                                        for($t=0;$t<5;$t++)
                                        {
                                                $no=$t+1;
                                $param_testing1=$param1[$t];
                                $param_testing2=$param2[$t];
                                $param_testing3=$param3[$t];
                                $param_testing4=$param4[$t];
                                $param_testing5=$param5[$t];
                                $obj = new $data[$b]->class_name();
                                $test_func_name = $data[$b]->func_name;
                                $func_output            =           $obj-
>$test_func_name($param_testing1,$param_testing2,$param_testing3,

        $param_testing4,$param_testing5,$param_testing1,$param_testi
ng2,$param_testing3,$param_testing4,$param_testing5);

        $param_update['id']=$data[$b]->id;
        $param_update['output']=$func_output.'';
                                                $this->detect_model-
>update_func_output($param_update,$no,$detection_time);

                                        }}}}}
```

Gambar 5. 12 Implementasi Algoritma ALG_CD_02

5.3.4.3 *Implementasi Algoritma ALG_CD_03*

```
1   function collect_clones($o,$detection_time)
2         {
3                   $returnvalue                =           $this->detect_model-
4   >get_grouped_output($o,$detection_time);
5                   $astdata1                   =           $this->detect_model-
6   >get_filtered_data($detection_time);
7                   $v=1;
8                   $output='func_output_'.$o;
9                   foreach($returnvalue  as $rowvalue)
10                  {
11                        foreach($astdata1 as $rowdata1)
12                        {
13                        if($rowvalue->$output==$rowdata1->$output)
14                              {
15                              $data['id']=$rowdata1->id;
16                              $data['group']=$rowdata1->func_group.$v;
17                              $this->detect_model-
18  >group_update($data,$detection_time);
19                              }
20                        }
21                        $v=$v+1;
22                  }
23        }
```

Gambar 5. 13 Implementasi Algoritma ALG_CD_03

5.3.5 Implementasi Basis Data

Implementasi basis data dilakukan berdasarkan perancangan sistem yang diterapkan ke dalam basis data MySQL. Skema basis data yang telah diimplementasikan dapat dilihat pada Gambar 5.13.

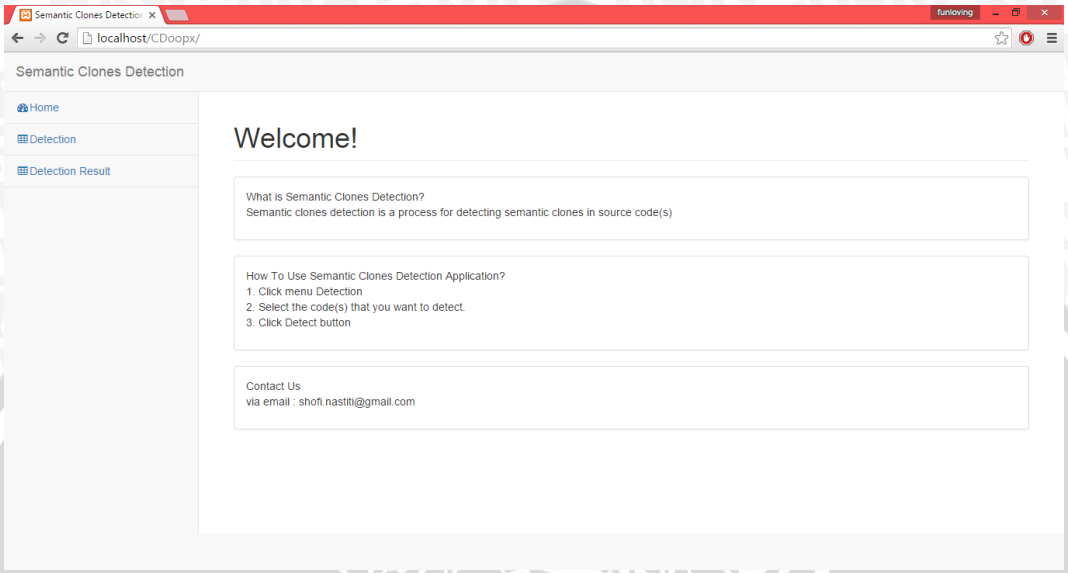
Gambar 5. 14 Implementasi Basis Data

Gambar 5.13 menunjukkan hasil implementasi perancangan basis data yang telah dibuat ke dalam basis data MySQL.

5.3.6 Implementasi Antarmuka Pengguna

Implementasi antarmuka pengguna dibuat berdasarkan perancangan yang diterapkan ke dalam bahasa HTML sebagai *view* pada *framework* CI. Berikut adalah hasil implementasi antarmuka pengguna.

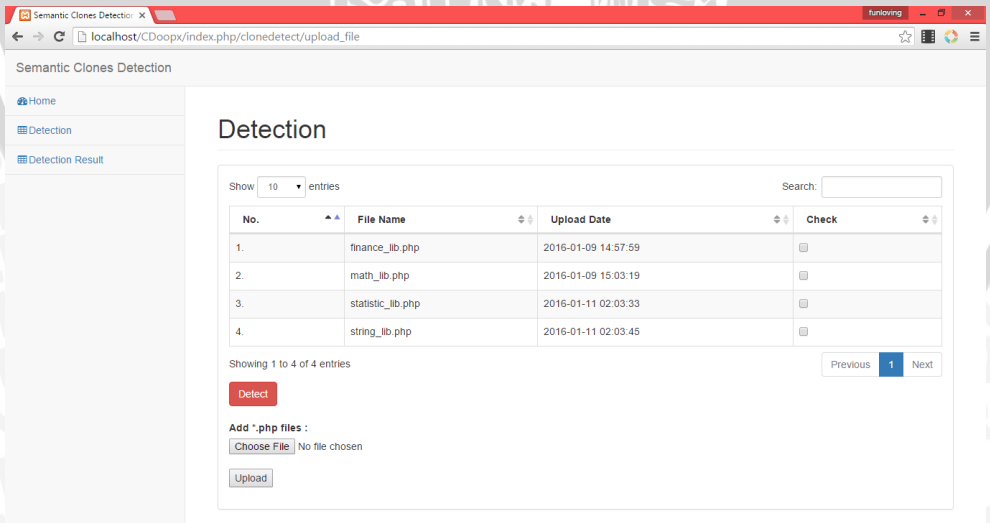
5.3.6.1 Implementasi Antarmuka Halaman Utama



Gambar 5. 15 Implementasi Antarmuka Halaman Utama

Gambar 5.16 menunjukkan hasil implementasi rancangan antarmuka Halaman Utama yang menunjukkan halaman pembuka saat sistem diakses atau dijalankan.

5.3.6.2 Implementasi Antarmuka Halaman *Detection*

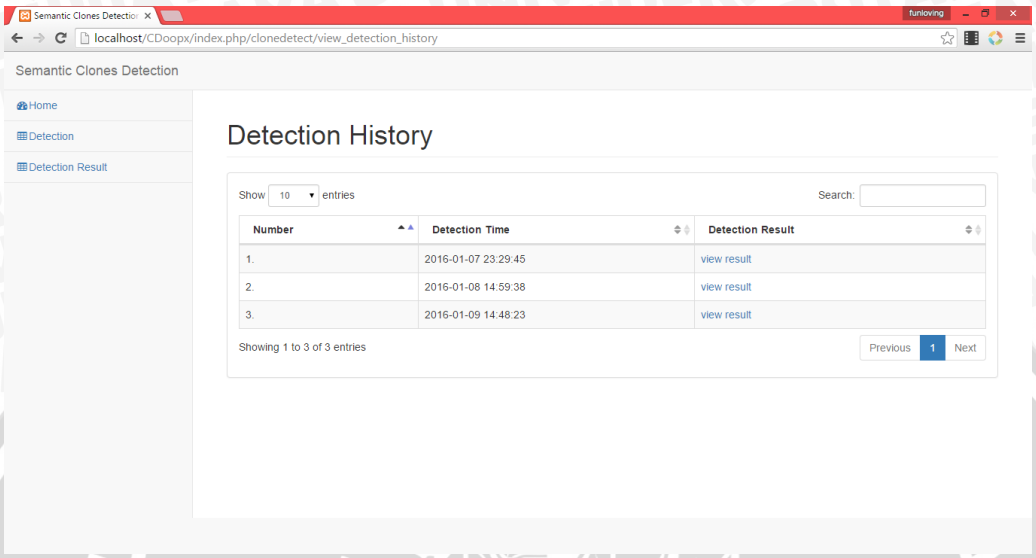


Gambar 5. 16 Implementasi Antarmuka Halaman Detection

Gambar 5.17 menunjukkan hasil implementasi rancangan antarmuka halaman detection. Berisi tabel informasi berkas yang dapat dipilih untuk

dideteksi, tombol “Detect” untuk memulai pendeteksian dan form “Upload” untuk menambahkan kode sumber yang belum pernah ditambahkan atau memperbarui kode sumber yang telah ditambahkan dengan versi baru.

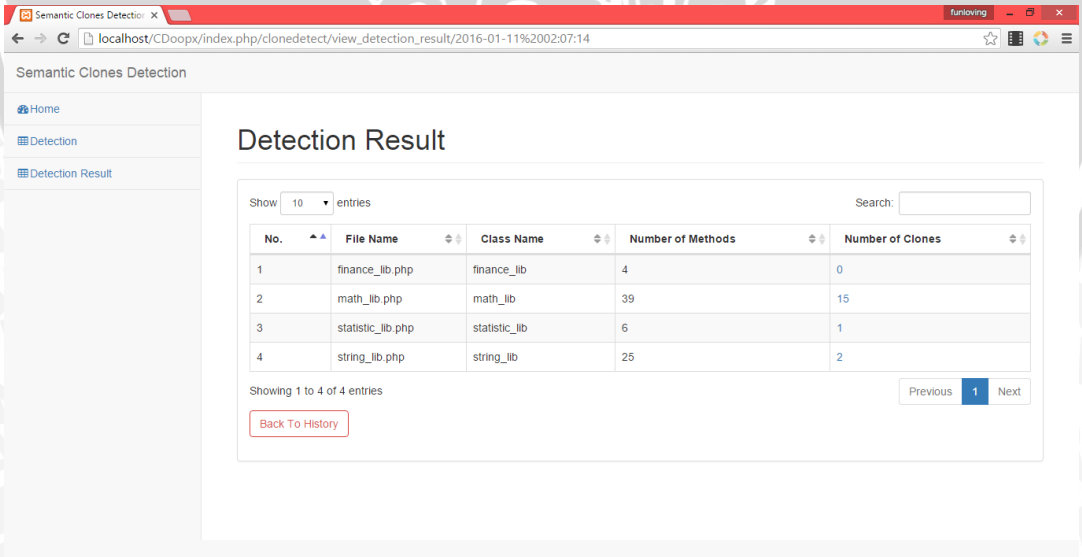
5.3.6.3 Implementasi Antarmuka Halaman *Detection History*



Gambar 5. 17 Implementasi Antarmuka Halaman Detection *History*

Gambar 5.18 menunjukkan hasil implementasi rancangan antarmuka halaman *Detection History.* Halaman ini terdiri dari tabel yang berisi informasi riwayat pendeteksian yang pernah dilakukan oleh sistem.

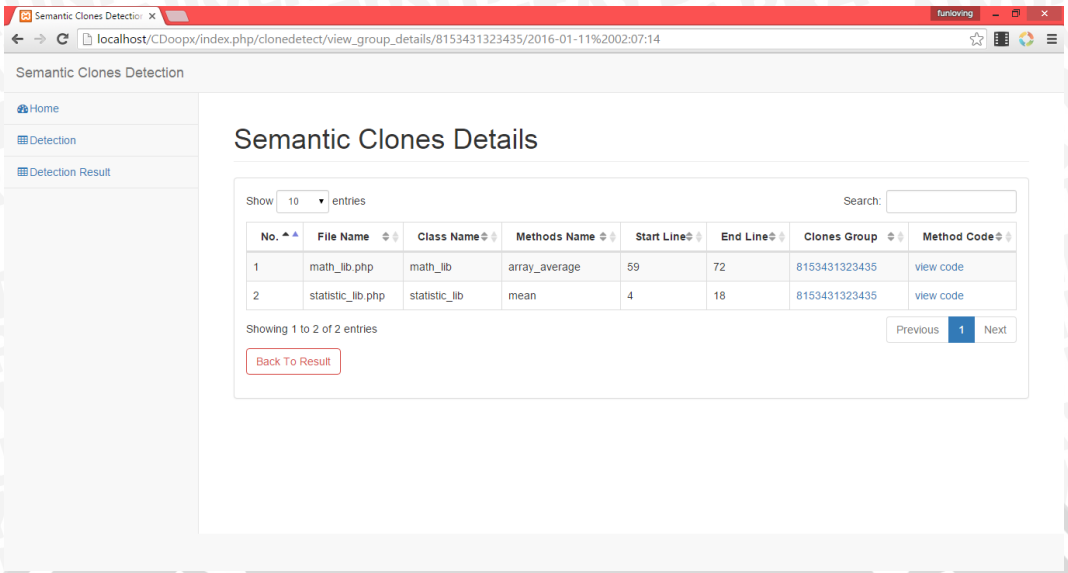
5.3.6.4 Implementasi Antarmuka Halaman *Detection Result*



Gambar 5. 18 Implementasi Antarmuka Halaman Detection Resut

Gambar 5.19 menunjukkan hasil implementasi rancangan antarmuka halaman *Detection Result.* Halaman ini menampilkan tabel yang berisi informasi hasil dari salah satu proses pendeteksian.

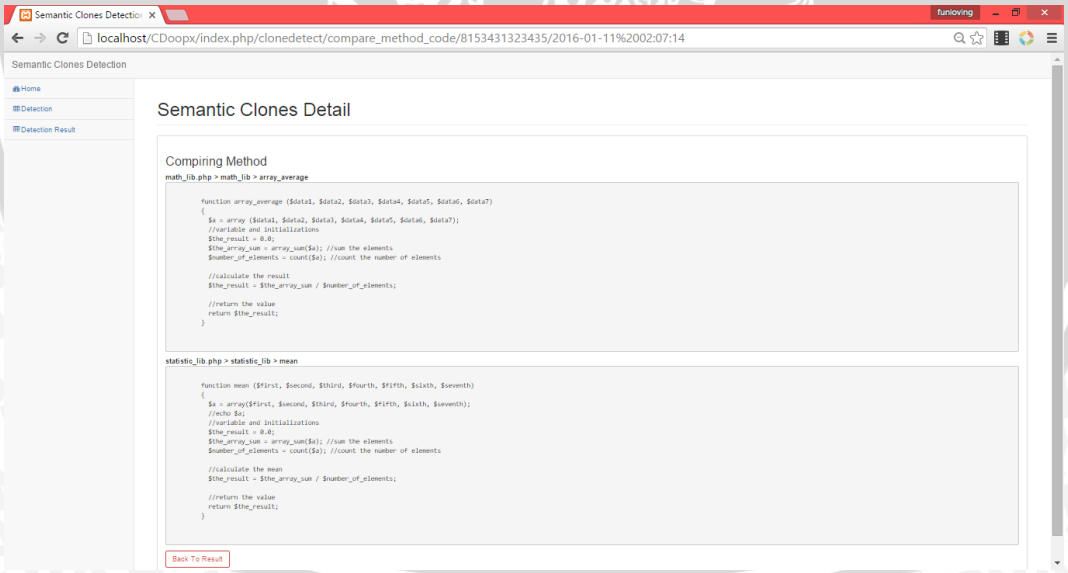
5.3.6.5 Implementasi Antarmuka Halaman *Clones Detail*



Gambar 5. 19 Implementasi Antarmuka Halaman *Clones Detail*

Gambar 5.20 menampilkan hasil implementasi rancangan antarmuka halaman *Clones Detail.* Halaman ini menampilkan tabel yang berisi informasi *method* yang dinyatakan sebagai kloning kode secara semantik dari hasil pendeteksian yang dilakukan oleh sistem.

5.3.6.6 Implementasi Antarmuka Halaman *Method Code*



Gambar 5. 20 Implementasi Antarmuka Halaman *Method Code*

Gambar 5.21 menunjukkan hasil impelementasi rancangan antarmuka halaman *Method Code.* Halaman ini menampilkan potongan kode dari *method* yang merupakan kloning kode secara semantik pada kode sumber PHP berdasarkan kelompok kloningnya.

BAB 6 PENGUJIAN

6.1 Pengujian Unit

Proses pengujian basis path merupakan proses pengujian unit yang dilakukan dengan menggunakan pseudocode yang menerapkan metode atau logika yang digunakan oleh sistem dalam mendeteksi kloning kode secara semantik. Pseudocode tersebut akan dimodelkan ke dalam suatu flow graph. Proses ini dilakukan untuk menentukan jumlah *cyclomatic complexity* (kompleksitas siklomatis) dan menentukan jalur independen. Jumlah kompleksitas siklomatis diperoleh melalui tiga persamaan berikut:

$$V(G) = E-N+2 \quad (6.1)$$

$$V(G) = P+1 \quad (6.2)$$

$$V(G) = R \quad (6.3)$$

Dimana,

V(G) : Jumlah kompleksitas siklomatis.

E : Sisi atau edge (garis penguhubung antar node).

N : Jumlah simpul (node).

P : Predicate node pada grafik alir.

R : Jumlah region pada flow graph.

Tujuan dilakukannya pengujian unit adalah untuk memastikan beberapa algoritma yang memiliki prioritas tinggi telah diimplementasikan sesuai dengan yang diharapkan.

Algoritma yang diujikan pada pengujian ini merupakan algoritma proses filtering dan collection. Pengujian unit dilakukan pada ketiga algoritma tersebut sebab merupakan domain permasalahan dari sistem yang telah dibangun.

6.1.1 Pengujian Algoritma ALG_CD_01

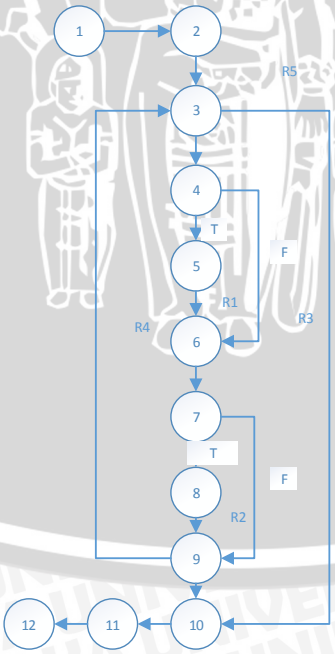
Berikut ini merupakan algoritma untuk melakukan proses filtering yang dijelaskan pada Tabel 6.1.

Tabel 6. 1 Algoritma ALG_CD_01

Nama Algoritma: ALG_CD_01 **Deskripsi: Algoritma untuk melakukan proses *filtering*** **Masukan: grouped_func_param, grouped_func_return, abstraction_data** **Proses:**

FUNCTION filtering_ast; (1)
DB SELECT method data, grouped return type, grouped effect (2)
FOREACH method data (3)
IF(method type match return type group) (4)
UPDATE DB method group match to method type group; (5)
ENDIF (6)
IF(method effect match effect group) (7)
UPDATE DB method group match to effect group; (8)
ENDIF (9)
END FOR (10)
Eliminate candidates from group that has <=1 member; (11)
END filtering_ast (12)

Berdasarkan algoritma yang telah diperoleh seperti pada Tabel 6.1 maka diperoleh *flow graph* seperti pada Gambar 6.1.



Gambar 6. 1 *Flow Graph* Algoritma ALG_CD_01

Berdasarkan *flow graph* yang diperoleh seperti pada gambar 6.1 maka diperoleh jumlah kompleksitas siklomatis seperti berikut:

V(G) = E – N + 2

= 15 – 12 + 2

= 5

V(G) = P + 1

= 4 + 1

= 5

V(G) = R

= 5

Berdasarkan jumlah kompleksitas siklomatis yang telah didapatkan, maka dihasilkan ditentukan 5 jalur independen, yaitu:

Jalur 1: 1 – 2 – 3 – 10 – 11 – 12

Jalur 2: 1 – 2 – 3 – 4 – 6 – 7 – 9 –10 –11 – 12

Jalur 3: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 – 12

Jalur 4: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 11 – 12

Jalur 5: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 3 – 10 – 11 – 12

Berdasarkan 5 jalur independen yang telah didefinisikan tersebut maka dapat dibentuk kasus ujinya. Tabel 5.1 memaparkan kasus uji dari algoritma *Filtering*.

Tabel 6. 2 Kasus Uji Algoritma ALG_CD_01

Jalur	***Class***	***Method***	**Data *Input***	**Hasil yang Diharapkan**	**Hasil yang Diperoleh**	**Status**
1	Detection_ controller	Filter_can didates	method_data, grouped_type, grouped_effect	Sistem tidak memperbarui group	Sistem tidak memperbarui group	Valid
2	Detection_ controller	Filter_can didates	method_data, grouped_type, grouped_effect	Sistem tidak memperbarui group	Sistem tidak memperbarui group	Valid
3	Detection_ controller	Filter_can didates	method_data, grouped_type, grouped_effect	Sistem memperbarui group	Sistem memperbarui group	Valid
4	Detection_ controller	Filter_can didates	method_data, grouped_type, grouped_effect	Sistem memperbarui group	Sistem memperbarui group	Valid
5	Detection_ controller	Filter_can didates	method_data, grouped_type, grouped_effect	Sistem memperbarui group	Sistem memperbarui group	Valid

6.1.2 Pengujian Algoritma ALG_CD_03

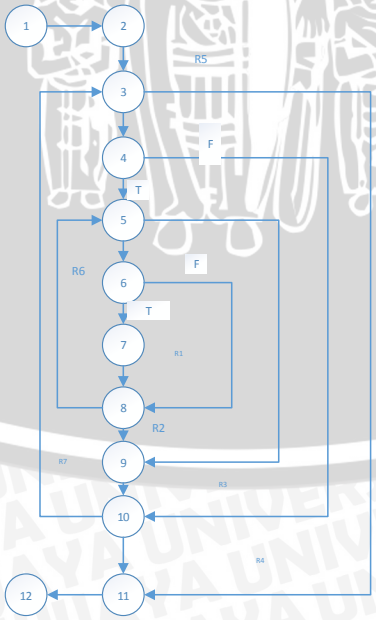
Berikut ini merupakan algoritma untuk melakukan proses *collect clones* yang dijelaskan pada Tabel 6.3.

Tabel 6. 3 Algoritma ALG_CD_03

Nama Algoritma: ALG_CD_03 **Deskripsi: Algoritma untuk melakukan proses *collection*** **Masukan: method_data** **Proses:**

FUNCTION collect_clones; (1)
DB SELECT tested method data, counted group method; (2)
FOREACH groupmethod (3)
IF(current group has 1 member) (4)
FOREACH tested method data (5)
IF(method group in tested method match current group) (6)
Eliminate tested method; (7)
END IF (8)
END FOR (9)
END IF (10)
END FOR (11)
END collect_clones (12)

Berdasarkan algoritma yang telah diperoleh seperti pada Tabel 6.1 maka diperoleh flow graph seperti pada Gambar 6.3.



Gambar 6. 2 Flow Graph Algoritma ALG_CD_03

Berdasarkan flow graph yang diperoleh seperti pada gambar 6.2 maka diperoleh jumlah kompleksitas siklomatis (cyclomatic complexity) seperti berikut:

V(G) = E – N + 2

= 17 – 12 + 2

= 7

V(G) = P + 1

= 6 + 1

= 7

V(G) = R

= 7

Berdasarkan jumlah kompleksitas siklomatis yang telah didapatkan, maka akan ditentukan 7 jalur independen, yaitu:

Jalur 1: 1 – 2 – 3 – 11 – 12

Jalur 2: 1 – 2 – 3 – 4 – 10 – 11 – 12

Jalur 3: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9

Jalur 4: 1 – 2 – 3 – 4 – 5 – 9 – 10 – 11 – 12

Jalur 5: 1 – 2 – 3 – 4 – 5 – 6 – 8 – 9 – 10 – 11 – 12

Jalur 6: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 5 – 9 – 10 – 11 – 12

Jalur 7: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 5 – 9 – 10 – 3 – 11 –12

Berdasarkan 7 jalur independen yang telah didefinisikan tersebut maka dapat dibentuk kasus ujinya. Tabel 6.4 memaparkan kasus uji dari algoritma Collection.

Tabel 6. 4 Kasus Uji Algoritma ALG_CD_03

Jalur	***Class***	***Method***	**Data *Input***	**Hasil yang Diharapkan**	**Hasil yang Diperoleh**	**Status**
1	detection_ controller	Collect_ clones	Method_ data	Sistem tidak melakukan eliminasi method kandidat kloning	Sistem tidak melakukan eliminasi method kandidat kloning	Valid
2	detection_ controller	Collect_ clones	Method_ data	Sistem tidak melakukan eliminasi method kandidat kloning	Sistem tidak melakukan eliminasi method kandidat kloning	Valid
3	detection_ controller	Collect_ clones	Method_ data	Sistem melakukan eliminasi method kandidat kloning	Sistem melakukan eliminasi method kandidat kloning	Valid
4	detection_ controller	Collect_ clones	Method_ data	Sistem tidak melakukan	Sistem tidak melakukan	Valid

				eliminasi method kandidat kloning	eliminasi method kandidat kloning	
5	detection_ controller	Collect_ clones	Method_ data	Sistem tidak melakukan eliminasi method kandidat kloning	Sistem tidak melakukan eliminasi method kandidat kloning	Valid
6	detection_ controller	Collect_ clones	Method_ data	Sistem melakukan eliminasi method kandidat kloning	Sistem melakukan eliminasi method kandidat kloning	Valid
7	detection_ controller	Collect_ clones	Method_ data	Sistem melakukan eliminasi method kandidat kloning	Sistem melakukan eliminasi method kandidat kloning	Valid

6.1.3 Analisis Hasil Pengujian Unit

Berdasarkan hasil pengujian yang telah dilakukan pada algoritma filtering, testing dan collection bahwa seluruh kasus uji memiliki hasil yang telah sesuai dengan yang diharapkan atau bernilai valid.

6.2 Pengujian Integrasi

Pengujian integrasi dilakukan pada proses yang melibatkan interaksi antar kelas. Sistem memenuhi aspek pengujian integrasi apabila seluruh hasil yang diharapkan (*expected result*) sesuai dengan hasil yang diperoleh (*actual result*). Namun jika terdapat hasil yang tidak sesuai yang diharapkan maka proses integrasi antar kelas harus diperbaiki.

6.2.1 Kasus Uji Pengujian Integrasi

Berikut merupakan kasus uji yang digunakan untuk melakukan pengujian integrasi.

Tabel 6. 5 Kasus Uji Integrasi INT_01

Nomor Kasus Uji	INT_01
Nama Kasus Uji	Kasus Uji Integrasi unit ALG_CD_01 dan fungsi update_method_group
Unit yang Terlibat	Unit filter_candidates dan update_method_group
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa proses yang melibatkan interaksi antara unit filter_candidates pada *class* detection_controller dan update_method_group pada *class* methods_data telah terintegrasi dengan benar.
Prosedur Uji	1. Penguji menyediakan kelas Tester untuk mengaktifkan unit filter_candidates. 2. Penguji menyediakan masukan yang diperlukan unit filter_candidates dan update_method_group. 3. Penguji mengeksekusi kelas Tester.

	4. Penguji mendokumentasikan hasil eksekusi bernilai *true* atau *false*.
Hasil yang Diharapkan	Nilai kembalian bernilai *true* jika data data func_group pada basis data berhasil diperbarui dan nilai kembalian bernilai *false* jika data func_group tidak berhasil diperbarui.

6.2.2 Hasil Pengujian Integrasi

Tabel 6.6 menjelaskan hasil pengujian integrasi yang telah dilakukan.

Tabel 6. 6 Hasil Pengujian Integrasi

No. Kasus Uji	Nama Kasus Uji	Hasil yang Diharapkan	Hasil yang Diperoleh	Status
INT_01	Kasus Uji Integrasi unit ast_filtering dan group_update	Nilai kembalian bernilai *true* jika data func_group berhasil diperbarui dan nilai kembalian bernilai *false* jika data func_group tidak berhasil diperbarui.	Nilai kembalian bernilai *true* jika data func_group berhasil diperbarui dan nilai kembalian bernilai *false* jika data func_group tidak berhasil diperbarui.	Valid

6.2.3 Analisis Hasil Pengujian Integrasi

Berdasarkan hasil pengujian integrasi, proses inti yang melibatkan interaksi antar kelas pada sistem telah terintegrasi dengan baik. Sehingga dapat dikatakan sistem pendeteksian kloning kode secara semantik lolos uji integrasi. Maka dari itu tidak diperlukan adanya perbaikan proses integrasi antar kelas sebab seluruh kasus uji integrasi telah valid.

6.3 Pengujian Validasi

Pengujian validasi dilakukan untuk memastikan setiap spesifikasi kebutuhan perangkat lunak yang didefinisikan telah sesuai dengan yang diharapkan. Setiap spesifikasi kebutuhan perangkat lunak yang didefinisikan pada tahap analisis kebutuhan akan diujikan pada pengujian ini dengan cara mendefinisikan kasus uji terhadap setiap kebutuhan tersebut lalu membandingkannya dengan hasil yang diperoleh.

6.3.1 Kasus Uji SRS_F_CD_10

Berikut ini merupakan kasus uji untuk melakukan pengujian validasi pada sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior*.

Tabel 6. 7 Kasus Uji Validasi Alur Utama SRS_F_CD_10

Nomor Kasus Uji	VAL_101
Nama Kasus Uji	Kasus Uji Validasi *Detect Semantic Clones*
Objek Uji	SRS_F_CD_10
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses "*detect semantic clones*"

Prosedur Uji	1. Membuka sistem dan masuk ke menu "*Detection*". 2. Menekan tombol "*detect*".
Hasil yang Diharapkan	Daftar statistik hasil pendeteksian berupa nama *class,* jumlah *method* pada *class* tersebut dan jumlah *method* yang dinyatakan sebagai kloning kode secara semantik.

Tabel 6. 8 Kasus Uji Validasi Alur Alternatif 1 SRS_F_CD_10

Nomor Kasus Uji	VAL_102
Nama Kasus Uji	Kasus Uji Validasi *Detect Semantic Clones*
Objek Uji	SRS_F_CD_10
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses "*detect semantic clones*"
Prosedur Uji	1. Membuka sistem dan masuk ke menu "*Detection*". 2. Menekan tombol "*choose file*" 3. Memilih file pada jendela pop-up 4. Menekan tombol "Upload" 5. Memilih file yang akan dideteksi. 6. Menekan tombol "*detect*".
Hasil yang Diharapkan	Daftar statistik hasil pendeteksian berupa nama *class,* jumlah *method* pada *class* tersebut dan jumlah *method* yang dinyatakan sebagai kloning kode secara semantik.

6.3.2 Kasus Uji SRS_F_CD_20

Tabel 6. 9 Kasus Uji Validasi Alur Utama SRS_F_CD_20

Nomor Kasus Uji	VAL_201
Nama Kasus Uji	Kasus uji Validasi *View Detection Result*
Objek Uji	SRS_F_CD_20
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses "*View detection result*"
Data Masukan	-
Prosedur Uji	1. Membuka sistem dan masuk ke menu "*detection result*". 2. Menekan *link* pada salah satu hasil pendeteksian
Hasil yang Diharapkan	Sistem hasil salah satu proses pendeteksian yang telah dilakukan sistem berupa tabel statistik.

Tabel 6. 10 Kasus Uji Validasi Alur Alternatif 1 SRS_F_CD_20

Nomor Kasus Uji	VAL_202
Nama Kasus Uji	Kasus uji Validasi *View Detection Result*
Objek Uji	SRS_F_CD_20
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses "*View detection result*"
Data Masukan	-
Prosedur Uji	1. Membuka sistem dan masuk ke menu "*detection result"*. 2. Melihat daftar hasil pendeteksian. 3. Menekan salah satu *link* pada kolom detection time 4. Menekan salah satu *link* pada kolom "*number of clones"*
Hasil yang Diharapkan	Sistem menampilkan hasil pendeteksian berupa tabel rincian *method* yang merupakan kloning kode secara semantik berdasarkan file yang sama.

Tabel 6. 11 Kasus Uji Validasi Alur Alternatif 2 SRS_F_CD_20

Nomor Kasus Uji	VAL_203
Nama Kasus Uji	Kasus uji Validasi *View Detection Result*
Objek Uji	SRS_F_CD_20
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses "*View detection result*"
Data Masukan	-
Prosedur Uji	1. Membuka sistem dan masuk ke menu "*detection result"*. 2. Melihat daftar hasil pendeteksian. 3. Menekan salah satu *link* pada kolom *"detection time"*. 4. Menekan salah satu *link* pada pada kolom *"number of clones"* 5. Menekan salah satu *link* pada kolom "*clones group".*
Hasil yang Diharapkan	Sistem menampilkan hasil pendeteksian berupa tabel rincian *method* yang merupakan kloning kode secara semantik berdasarkan kelompok kloning yang sama.

Tabel 6. 12 Kasus Uji Validasi Alur Alternatif 3 SRS_F_CD_20

Nomor Kasus Uji	VAL_204
Nama Kasus Uji	Kasus uji Validasi *View Detection Result*
Objek Uji	SRS_F_CD_20
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses "*View detection result*"
Data Masukan	-

Prosedur Uji	1. Membuka sistem dan masuk ke menu "*detection result*". 2. Melihat daftar hasil pendeteksian. 3. Menekan salah satu *link* pada kolom *"detection time"*. 4. Menekan salah satu *link* pada pada kolom *"number of clones"* 5. Menekan salah satu *link* pada kolom "*clones group"*. 6. Menekan salah satu *link* pada kolom *"method code"*
Hasil yang Diharapkan	Sistem menampilkan hasil pendeteksian berupa potongan kode sumber dari *method* yang berada pada kelompok kloning yang sama.

6.3.3 Hasil Pengujian Validasi

Berikut ini merupakan hasil pengujian validasi pada sistem pendeteksian klonig kode secara semantik pada akode sumber PHP dengan menerapkan metode *IOE-Behavior*.

Tabel 6. 13 Hasil Pengujian Validasi

No.	Kebutuhan	Kasus Uji	Hasil yang Didapatkan	Status
1	SRS_F_CD_10	VAL_101	Sistem melakukan pendeteksian kloning kode secara semantik pada kode sumber yang telah dipilih.	Valid
2	SRS_F_CD_10	VAL_102	Sistem menambahkan kode sumber baru ke dalam sistem	Valid
3	SRS_F_CD_20	VAL_201	Sistem hasil salah satu proses pendeteksian yang telah dilakukan sistem berupa tabel statistik.	Valid
4	SRS_F_CD_20	VAL_202	Sistem menampilkan hasil pendeteksian berupa tabel rincian *method* yang merupakan kloning kode secara semantik berdasarkan file yang sama.	Valid
5	SRS_F_CD_20	VAL_203	Sistem menampilkan hasil pendeteksian berupa tabel rincian *method* yang merupakan kloning kode secara semantik berdasarkan kelompok kloning yang sama.	Valid
6	SRS_F_CD_20	VAL_204	Sistem menampilkan hasil pendeteksian berupa potongan kode sumber dari *method* yang berada pada kelompok kloning yang sama.	Valid

6.3.4 Analisis Hasil Pengujian Validasi

Dari hasil pengujian validasi diperoleh hasil fungsi dalam sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior* sudah valid 100% sesuai dengan hasil yang diharapkan.

6.4 Pengujian Akurasi

Pengujian akurasi dilakukan dengan membandingkan hasil pendeteksian secara manual yang dilakukan oleh penulis dengan menggunakan definisi kloning kode secara semantik, yaitu *method* yang memiliki masukan, keluaran dan efek yang sama (Elva, 2012a). Kasus uji yang digunakan untuk pengujian akurasi adalah 4 kode sumber yang didalamnya terdapat 4-39 method. Kode sumber tersebut merupakan hasil modifikasi dari kode sumber AJ PHP Software Source Code Library dengan kode sumber sampel yang ada pada web PHP Manual. Berikut adalah hasil pendeteksian secara manual pada salah satu kode sumber sampel string_lib.php ditunjukkan pada Tabel 6.14.

Tabel 6. 14 Pendeteksian Manual

No	Method Name	Paremeter	Return	Total Output	Clones Group
1	print_5x	1	VALUE	argument printed 5x	211
2	print_data	1	VALUE	argument printed 5x	211
3	writeMsg	0	VOID	eliminated	eliminated
4	compare_str	2	VALUE	string compared	312
5	str_padding	2	VALUE	padding string	313
6	txt_length	1	VALUE	text lengt	214
7	count_words	1	VALUE	words counted	215
8	str_reverse	1	VALUE	string reversed	216
9	str_shuffle_unicode	1	VALUE	string shuffled	217
10	scramble_word	1	VOID	eliminated	eliminated
11	str_repeat_extended	3	VALUE	string repeated	418
12	get_width	1	VOID	eliminated	eliminated
13	matchlen	2	VALUE	length of matching string	319
14	text_similar	2	VALUE	text similiarity prosentage	3110
15	ucase_percent	1	VALUE	uppercase prosentage	2111
16	get_substr	2	VALUE	n-index character	3112
17	replace_substr	2	VALUE	substring replaced	3113
18	replace_str	3	VALUE	string replaced	4114
19	setmaxHeight	2	VOID	eliminated	eliminated
20	get_string_between	3	VALUE	n-index string	4115
21	random_string	0	VALUE	random string	1116
22	setHeight	1	VOID	eliminated	eliminated
23	uppercase_first	1	VOID	eliminated	eliminated
24	print_res	1	VOID	eliminated	eliminated
25	cut_string_using_last	4	VALUE	eliminated	eliminated

Hasil pendeteksian diatas dilakukan secara manual dengan cara mencari *method* yang memiliki jumlah parameter dan tipe *method* yang sama. Kemudian mengelompokkannya dalam satu kelompok. Selanjutnya dari hasil pengelompokan tersebut dibuat kembali sub-kelompok *method* yang memiliki *total output* sama. Sehingga dari kelompok-kelompok yang ada, diperoleh sub-kelompok dengan *method* yang memiliki kesamaan jumlah parameter, tipe *method* dan *total output* yang sama. Sub-kelompok yang hanya memiliki lebih dari satu anggota yang dinyatakans sebagai kloning kode secara semantik pada kode sumber PHP.

Dari hasil proses pendeteksian secara manual yang telah dilakukan penulis, menunjukkan terdapat dua method yang memiliki kesamaan secara fungsional yaitu method print_5x dan print_data karena merupakan anggota pada kelompok yang sama. Hasil pendeteksian secara manual pada kode sumber sampel string_lib.php ditunjukkan pada Tabel 6.15.

Tabel 6. 15 Hasil Pendeteksian Manual

no	method	clones group	start line	end line	class name
1	print_5x	211	5	13	string_lib
2	print_data	211	15	24	string_lib

Perbandingan kode sumber kedua method yang dinyatakan sebagai kloning kode secara semantik pada kode sumber string_lib.php ditunjukkan pada gambar 6.3.

```
function print_5x($string_data)
    {
       $print='';
       for($a=0;$a<5;$a++)
       {

       $print=$print.$string_data;
       }
      return $print;
    }
```

```
function print_data($words)
    {
       $print='';
       $print=$print.$words;
       $print=$print.$words;
       $print=$print.$words;
       $print=$print.$words;
       $print=$print.$words;
       return $print;
    }
```

Gambar 6. 3 Perbandingan *Method* Hasil Pendeteksian Manual

Pada Gambar 6.3 menunjukkan bahwa kedua method memiliki kode yang berbeda. Jika dilihat sesuai prinsip kloning kode secara semantik (Elva, 2012a), kedua method tersebut memiliki kesamaan pada tipe *method* dan jumlah parameter, serta memiliki *total output* yang sama ketika dieksekusi. Oleh karena itu, kedua *method* tersebut dinyatakan sebagai kloning kode secara semantik.

6.4.1 Kasus Uji Akurasi Kode Sumber math_lib.php

Berikut ini merupakan kasus uji akurasi dan hasil untuk kode sumber math_lib.php yang ditunjukkan pada Tabel 6.14.

Tabel 6. 16 Kasus Uji Akurasi Kode Sumber math_lib.php

Nomor	Nama *Class*	Nama *Method*	Kloning	
			Sistem	Manual
1	math_lib	factorial_integer	Tidak	Tidak
2	math_lib	log_base	Tidak	Tidak
3	math_lib	addition	Ya	Ya
4	math_lib	array_average	Ya	Ya
5	math_lib	get_sum	Ya	Ya
6	math_lib	get_absolute	Ya	Ya
7	math_lib	get_absolute2	Ya	Ya
8	math_lib	square_numb	Ya	Ya
9	math_lib	square_numb2	Ya	Ya
10	math_lib	maxbal	Ya	Ya
11	math_lib	balrank	Ya	Ya
12	math_lib	add_and_sub	Tidak	Tidak
13	math_lib	minbal	Tidak	Tidak
14	math_lib	num2alpha	Tidak	Tidak
15	math_lib	alpha2num	Tidak	Tidak
16	math_lib	binarycodedstring2dec	Tidak	Tidak
17	math_lib	expm1	Tidak	Tidak
18	math_lib	random_password	Tidak	Tidak
19	math_lib	convert2bin	Ya	Ya
20	math_lib	dec2bin	Ya	Ya
21	math_lib	doublemax	Tidak	Tidak
22	math_lib	mt_randf	Tidak	Tidak
23	math_lib	lcg_randf	Tidak	Tidak
24	math_lib	randf	Tidak	Tidak
25	math_lib	intdiv_1	Ya	Ya
26	math_lib	intdiv_2	Ya	Ya
27	math_lib	floordec	Tidak	Tidak
28	math_lib	modulo	Ya	Ya
29	math_lib	fmodulo	Ya	Ya
30	math_lib	dec_to_hex	Tidak	Tidak
31	math_lib	minus_data	Tidak	Tidak
32	math_lib	hex2dec	Tidak	Tidak

33	math_lib	ln	Tidak	Tidak
34	math_lib	toAlphaColor	Tidak	Tidak
35	math_lib	random_hex_color	Tidak	Tidak
36	math_lib	rgb_to_color	Tidak	Tidak
37	math_lib	color_mkwebsafe	Tidak	Tidak
38	math_lib	BinString2BitSequence	Tidak	Tidak
39	math_lib	BCDec2Bin	Tidak	Tidak

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.x maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi kode sumber math_lib.php:

$$Akurasi = \frac{39}{39} \times 100\% = 100\%$$

6.4.2 Kasus Uji Akurasi Kode Sumber statistic_lib.php

Berikut ini merupakan kasus uji untuk kode sumber statistic_lib.php yang ditunjukkan pada Tabel 6.17.

Tabel 6. 17 Kasus Uji Akurasi Kode Sumber statistic_lib.php

Nomor	**Nama *Class***	**Nama *Method***	**Kloning**	
			Sistem	**Manual**
1	statistic_lib	mean	Ya	Ya
2	statistic_lib	median	Tidak	Tidak
3	statistic_lib	standard_deviation_sample	Tidak	Tidak
4	statistic_lib	standard_deviation_population	Tidak	Tidak
5	statistic_lib	variance_population	Tidak	Tidak
6	statistic_lib	variance_sample	Tidak	Tidak

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.x maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi kode sumber math_lib.php:

$$akurasi = \frac{6}{6} \times 100\% = 100\%$$

6.4.3 Kasus Uji Akurasi Kode Sumber finance_lib.php

Berikut ini merupakan kasus uji untuk kode sumber finance_lib.php yang ditunjukkan pada tabel 6.18.

Tabel 6. 18 Kasus Uji Akurasi Kode Sumber finance_lib.php

Nomor	Nama *Class*	Nama *Method*	Kloning	
			Sistem	Manual
1	finance_lib	future_worth_SPCAF	Tidak	Tidak
2	finance_lib	simple_interest	Tidak	Tidak
3	finance_lib	present_worth_SPPWF	Tidak	Tidak
4	finance_lib	percent_interest_rate	Tidak	Tidak

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.x maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi kode sumber math_lib.php:

$$Akurasi = \frac{4}{4} \times 100\% = 100\%$$

6.4.4 Kasus Uji Akurasi Kode Sumber string_lib.php

Berikut ini merupakan kasus uji untuk kode sumber string_lib.php yang ditunjukkan pada tabel 6.19.

Tabel 6. 19 Kasus Uji Akurasi Kode Sumber string_lib.php

Nomor	Nama *Class*	Nama *Method*	Kloning	
			Sistem	Manual
1	string_lib	print_5x	Ya	Ya
2	string_lib	print_data	Ya	Ya
3	string_lib	writeMsg	Tidak	Tidak
4	string_lib	compare_str	Tidak	Tidak
5	string_lib	str_padding	Tidak	Tidak
6	string_lib	txt_length	Tidak	Tidak
7	string_lib	count_words	Tidak	Tidak
8	string_lib	str_reverse	Tidak	Tidak
9	string_lib	str_shuffle_unicode	Tidak	Tidak
10	string_lib	scramble_word	Tidak	Tidak
11	string_lib	str_repeat_extended	Tidak	Tidak
12	string_lib	get_width	Tidak	Tidak
13	string_lib	matchlen	Tidak	Tidak
14	string_lib	text_similar	Tidak	Tidak
15	string_lib	ucase_percent	Tidak	Tidak
16	string_lib	get_substr	Tidak	Tidak
17	string_lib	replace_substr	Tidak	Tidak

18	string_lib	replace_str	Tidak	Tidak
19	string_lib	setmaxHeight	Tidak	Tidak
20	string_lib	get_string_between	Tidak	Tidak
21	string_lib	random_string	Tidak	Tidak
22	string_lib	setHeight	Tidak	Tidak
23	string_lib	uppercase_first	Tidak	Tidak
24	string_lib	print_res	Tidak	Tidak
25	string_lib	cut_string_using_last	Tidak	Tidak

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.x maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi kode sumber math_lib.php:

$$Akurasi = \frac{25}{25} \times 100\% = 100\%$$

6.4.5 Analisis Hasil Pengujian Akurasi

Berikut ini merupakan hasil pengujian akurasi pada sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan metode *IOE-Behavior*.

Tabel 6. 20 Hasil Pengujian Akurasi

Nomor	Kode Sumber	Akurasi
1	math_lib.php	100 %
2	finance_lib.php	100 %
3	statistic_lib.php	100 %
4	string_lib.php	100 %
Rata-Rata Akurasi		100 %

Dari hasil pengujian akurasi yang diperoleh pada Tabel 6.20 menunjukkan bahwa rata-rata akurasi yang dimiliki oleh sistem ini sebesar 100%. Akurasi sebesar 100% dapat diperoleh karena tidak ada kesalahan dalam proses pendeteksian pada kode program yang dimasukkan oleh pengguna. Sebagai salah satu contoh kloning kode secara semantik yang berhasil dideteksi adalah fungsi maxbal() dan balrank() pada class math_lib yang ditunjukkan pada Gambar 6.4 Berikut.

```
function balrank($a,$b,$c)
{
   $data=array($a,$b,$c);
   $temp=-10000000;
   for ($x = 0; $x <3 ; $x++) {
        if($data[$x]>$temp){
                $temp=$data[$x];
              }
          return $temp;
          }
}
```

```
function maxbal($a, $b,$c)
{
    $bal=array($a,$b,$c);
    $maxbal = max($bal);
    return $maxbal;
}
```

Gambar 6. 4 Perbandingan *Method* Hasil Pendeteksian Sistem

Kedua fungsi yang ditunjukkan pada Gambar 6.4 merupakan fungsi yang mengembalikan nilai dan memiliki jumlah parameter masukan sama, sehingga dapat dikatakan bahwa fungsi-fungsi tersebut memiliki tipe yang sama. Selain itu fungsi maxbal() dan balrank() tidak memiliki efek pada fungsi lain, sehingga dapat dikatakan bahwa kedua fungsi tersebut memiliki efek yang sama. Jika dijalankan menggunakan nilai parameter yang sama, maka kedua fungsi di atas akan menghasilkan nilai yang sama yaitu nilai maksimum dari parameter-parameter yang dimasukkan, sehingga dapat dikatakan fungsi-fungsi tersebut memiliki total output yang sama. Dari hasil analisa tersebut maka fungsi balrank() dan maxbal() dikategorikan ke dalam kloning kode secara semantik karena memiliki tipe dan efek fungsi yang sama.

BAB 7 PENUTUP

7.1 Kesimpulan

Kesimpulan yang dapat diambil dari hasil seluruh tahapan dalam penelitian ini adalah:

1. Berdasarkan proses rekayasa perangkat lunak yang telah dilakukan, sistem pendeteksian kloning kode secara semantik pada kode sumber PHP dengan menerapkan metode *IOE-Behavior* dapat dikembangkan dengan menggunakan pendekatan berorientasi objek.
2. Pendeteksian kloning kode secara semantik pada kode sumber PHP dapat dilakukan dengan menerapkan metode *IOE-Behavior* melalui 4 proses yaitu proses *abstraction, filtering, testing dan collection*.
3. Pengujian sistem yang dilakukan dalam penelitian ini terdiri dari pengujian unit, pengujian integrasi dan pengujian validasi. Berdasarkan setiap hasil pengujian yang telah dilakukan, diperoleh nilai validitas 100% untuk seluruh kasus uji sesuai dengan hasil yang diharapkan.

7.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, penulis mendapatkan saran untuk penelitian selanjutnya dari hasil penelitian ini adalah:

1. Menambahkan beberapa fitur seperti menampilkan kode sumber yang dideteksi.
2. Memberikan highlight pada bagian kode sumber yang merupakan kloning kode secara semantik dengan warna yang berbeda-beda sesuai dengan kelompok kloningnya.

DAFTAR PUSTAKA

Morshed, Monzur et.al.2012.”*A Literarure Review of Code Clone Analysis to Improve Software Mantenance Process". Departent of Computer Science, American International University-Bangladesh.*

Rattan, Dhavleesh et.al.2013.*"Software Clone Detection: A Systematic Review". Department of Computer Science and Engineering and Information Technology, Baba Banda Singh Bahadur Engineering College, Punjab India.*

Elva, Rochelle et.al.2012a.*"JSTracker: A Semantic Clone Detection Tool for Java Code". Department of EECS, University of Central Florida.*

Elva, Rochelle et.al.2012b.*"Semantic Clone Detection Method IOE-Behavior". Department of EECS,* University of Central Florida.

Jiang, Lingxiao dan Su.2009.*"Automatic Mining of Functionally Equivalent Code Fragments via Random Testing".* University of California, Davis.

Calefato, Fabio et.al.2004.*”Function Clone Detection in Web Applications: A Semiautomated Approach”.* Dipartimento di Informatica, University of Bari, Italy.

WTechs. 2014. *Most Popular Server-Side Programming Languages* 1 October 2014. Tersedia di: <http://w3techs.com/> [Diakses 26 Oktober 2014]

Elva, Rochelle et.al.2012c.*" JSCTracker A Tool and Algorithm for Semantic Method Clone Detection Using Method IOE-Behavior*". Department of EECS, University of Central Florida.

Y. Higo et al. 2007.*”Information and Software Technology 49”*. Hal 985–998

Pigoski, Thomas M. 2001. *“Swebok chapter 6: Software Maintenance”.* Technical Software Services (TECHSOFT), Inc. Pensacola, Florida 32501 USA.

Juergens, E. dan Deissenboeck, F. dan Hummel, B.2010.*”Code similarities beyond copy & paste”.* Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering”. CSMR '10, Washington, DC, USA, IEEE Computer Society.

Kawrykow, D. dan Robillard, M.P.2009.*”Improving API usage through automatic detection of redundant code”*. Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. ASE '09, Washington, DC, USA, IEEE Computer Society

Deissenboeck, F. dan Heinemann, L. dan Hummel, B. dan Wagner, S. 2012. *“Challenges of the dynamic detection of functionally similar code fragments”.*Software Maintenance and Reengineering, European Conference on 0

Perangingangin, K.2006.”Aplikasi WEB dengan PHP dan MySQL”. ANDI.Yogyakarta.ISBN 979-763-52-68

Raharjo, B. dan Heryanto, I.dan RK, Enjang.2012."Modul Pemrograman WEB HTML, PHP & MySQL".Modul.Bandung.ISBN 978-602-8759-18-2

Raemekers, Steven.et.al.2010.*"Testing Semantic Clones Detection Candidates"*. University of Amsterdam

Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E. 2007. *"Comparison and evaluation of clone detection tools".* IEEE Transactions on Software Engineering

Kim, M., Bergman, L., Lau, T., Notkin, D. 2004.*"An Ethnographic Study of Copy and Paste Programming Practices in OOPL"*. Proceedings of 3rd International ACM-IEEE Symposium on Empirical Software Engineering (ISESE'04), Redondo Beach, CA, USA

Baxter, Ira D., Yahin, A., Moura, L., Sant'Anna, M., Bier, L. 1998. *"Clone Detection Using Abstract Syntax Trees".* Proceedings of the 14th International Conference on Software Maintenance (ICSM '98), Bethesda, Maryland, USA

Priyambadha, B., Rochimah, S. 2014. *"Case Study on Semantic Clone Detection Based On Code Behavior".* International Conference on Data and Software Engineering 978-1-4799-7996-7/14 2014 IEEE.

Zandstra, Matt. 2010. *"PHP Objects, Patterns, and Practice"*. Appres. USA. ISBN 978-1-4302-2926-1

Hayder, Hasin. 2007.*"Object Oriented Programming with PHP5"*. PACK Publishing Ltd. Birmingham, UK. ISBN 978-1-847192-56-1

PHP Manual.*"PHP Class & Object Documentations: The Basic"*. Tersedia di: <http://php.net/manual/en/language.oop5.basic.php> [Diakses 15 April 2015]

Plotz, Marc. 2011.*" Principles of Object Oriented Programming in PHP".*Tersedia di: <http://www.htmlgoodies.com/beyond/php/article.php/3909681> [Diakses 15 Mei 2015]

Popov, Nikira. 2015.*"PHP Parser: A PHP Parser Written in PHP"*.Tersedia di: <https://github.com/nikic/PHP-Parser> [Diakses 07 April 2015]