OPTIMALISASI PERANGKAT LUNAK MENGGUNAKAN METODE *REFACTORING*

Heri Fredianto
Universitas Jenderal Achmad Yani (UNJANI) Jawa Barat, Indonesia
Email: herifred21@gmail.com

INFO ARTIKEL
Diterima
25 September 2021
Direvisi
05 Oktober 2021
Disetujui
15 Oktober 2021

Kata Kunci:
optimalisasi; *refactoring; smell code; extract method; benchmark*

ABSTRAK

Optimalisasi dengan *refactoring* merupakan aktivitas penting dalam meningkatkan struktur internal kode dan bertujuan untuk meningkatkan kualitas perangkat lunak. Dalam program modul pendaftaran sistem informasi klinik utama nur khadijah, terdapat *smell code* yang menjadi indikasi adanya permasalahan dari struktur kode program. Permasalahan yang terdapat pada program tersebut yaitu adanya *duplicate code* yang membuat pemeliharaan perangkat lunak menjadi lebih sulit. Ketika terjadi kesalahan akan sulit menemukan dan memperbaikinya, efek jangka panjang yang terjadi yaitu akan menimbulkan bug. Pada tugas akhir ini melakukan optimalisasi terhadap aplikasi klinik utama nur khadijah menggunakan metode refactoring dengan mengoptimalkan kode program yang terindikasi *smell code* yaitu *duplicated code* dan *query*. Dalam penyelesaian permasalahan *duplicate code* maka akan dilakukan dengan menggunakan Teknik *Extract Method*. Setelah itu pengujian *Benchmarking* untuk menguji *performance response time query* dengan memanfaatkan tools *Mysqlslap*. Penelitian ini akan menghasilkan sistem yang lebih optimal dibandingkan dengan system sebelumnya dengan menerapkan optimalisasi dengan metode *refactoring*.

ABSTRACT

Optimization of the refactoring is an important activity in improving st ruktur internal code and aims to improve the quality of software . In the registration module program for Nur Khadijah's main clinical information system, there is a smell code which is an indication of a problem with the structure of the program code. Problems with the programnamely the existence of duplicate code which makes software maintenance more difficult. When an error occurs it will be difficult to find and fix it, the long-term effect that occurs is that it will cause bugs. In this final project, optimizing Nur Khadijah's main clinical application using the refactoring method by optimizing the program code indicated by the smell code, namely duplicated

How to cite: Fredianto, H. (2021) Optimalisasi Perangkat Lunak Menggunakan Metode Refactoring. *Jurnal Syntax Admiration 2*(10). https://doi.org/10.46799/jsa.v2i10.326
E-ISSN: 2722-5356
Published by: Ridwan Institute

code and queries. In solving the duplicate code problem, it will be done using the Extract Method Technique. After that, Benchmarking is tested to test the query response time performance by using the Mysqlslap tools. This research will produce a system that is more optimal than the previous system.

Keywords: *optimalisasi; refactoring; smell code; extract method; benchmark*

Pendahuluan

Klinik Utama Nur Khadijah merupakan salah satu fasilitas pelayanan kesehatan berupa klinik yang menyelenggarakan atau menyediakan pelayanan bagi ibu yang hamil, persalinan, keluarga berencana, pemeriksaan umum, imunisasi, dsb. Proses pelayanan kesehatan di Klinik Utama Nur Khadijah sudah menggunakan aplikasi. Salah satu aplikasi klinik nur khadijah terdapat modul pendaftaran yang merupakan komponen penting dalam meningkatkan kualitas pelayanan klinik aplikasi dalam mengelola data pasien dan kunjungan pasien. Namun setelah dilakukan prariset yang dilakukan dengan melaksanakan inspeksi terhadap bagian *back end* modul pendaftaran dari aplikasi klinik. Ditemukan permasalahan berupa *smell code*. Salah satu teknik untuk memperbaiki *smell code* yang ditemukan pada stuktur kode program adalah dengan melakukan *refactoring* (Baqais & Alshayeb, 2020). *Refactoring* dilakukan untuk pengembangan dan pemeliharaan perangkat lunak (Ibrahim et al., 2020) dan meningkatkan kualitas kode program (Haque et al., 2018) yang sudah ada sebelumya *(legacy system)*. Program tersebut muncul permasalahan adanya bug pada program yaitu karena terindikasinya *smell code* atau *duplicate code* (Shrivastava, n.d.). Dikatakan bahwa keberadaan *duplicate code* memiliki dampak negatif pada pengembangan dan pemeliharaan perangkat lunak. Penyebab masalah tersebut ialah kurangnya pengetahuan seorang programmer terkait struktur kode yang baik atau solusi pengembang dirancang dengan tidak tepat, hal tersebut berdampak negatif pada kualitas desain kode. Sehingga dapat memperlambat proses pengembangan dan menimbulkan resiko *bug* atau kegagalan di masa mendatang (Singh & Kaur, 2018) (Fowler, 1999) (Cairo et al., 2018).

Menurut G. Lacerda pada penelitiannya yang berjudul "*Code smells and refactoring*: *A tertiary systematic review of challenges and observations*" *Refactoring* terkait dengan *smell code*, karena *refactoring* merupakan strategi utama untuk meminimalkan peluang munculnya *software defaect (bug)* (Cairo et al., 2018) pada implementasi yang dibuat, serta meningkatkan kualitas perangkat lunak (Lacerda et al., 2020). Sehingga masalah kode atau desain yang dapat membuat perangkat lunak sulit dipahami, dikembangkan, dan dipelihara (Fontana & Spinelli, 2011). Hal itu terjadi karena *development* sebelumnya malas membuat kode program baru. Kemudian *development* dengan sengaja melakukan copy paste kode-kode program lain. Dalam mengatasi *smell code* itu sendiri mengusulkan dengan mengunakan *Extract Method*

dalam membangun struktur kode agar lebih terstruktur dan mudah dipahami serta memangkas *duplicated code* pada program (Fowler, 1999).

Dengan seiringnya waktu aplikasi klinik selalu bertambah data pasien dan kunjungan setiap harinya. Jumlah data pasien dan kunjungan pasien dengan model *query* belum optimal, tentu dapat mengakibatkan *performance* akses data menjadi lambat (Gunawan et al., 2019). Proses pengaksesan dan pengolahan data yang berskala besar serta melibatkan banyak operasi join akan membutuhkan waktu yang cukup lama sehingga menyebabkan kinerja menjadi tidak efektif dan efisien (Studi et al., n.d.) (Safitri et al., 2018). Apabila ini terjadi maka dapat memperlambat proses menampilkan transaksi data pada sistem informasi klinik. Oleh karena itu penelitian ini dilakukan *refactoring* kode *query* menggunakan VIEW.

Penelitian ini bertujuan untuk Optimalisasi dengan dilakukan *refactoring* pada program yang terindikasi *smell code* pada modul pendaftaran aplikasi klinik dimana dalam modul pendaftaran tersebut terdapat berisi function yang kompleks yang mengindikasikan permasalahan yaitu terdeteksi *duplicate code* dan lambatnya *performance* eksekusi *query* saat menampilkan data. Hasil optimalisasi dengan menggunakan metode *refactoring* pada perangkat lunak menjadi optimal baik dari desain struktur kode dan *performance response time querynya*.

Metode Penelitian

Tahapan penelitian yang akan dilakukan dalam penelitian ini terdiri dari lima tahapan yaitu; 1) Identifikasi Masalah. 2) Analisa *Source Code*. 3) Pemetaan *Source Code*. 4) Implementasi *Refactoring*. 5) Pengujian (Fowler, 1999).

Berikut merupakan tahapan penelitian ini:

a) Identifikasi Masalah

Pada tahapan ini dilakukan Identifikasi terhadap perangkat lunak berupa analisis sistem secara umum meliputi analisis masalah struktur internal kode yang ditemukan pada perangkat lunak.

b) Analisis *Source Code*

Pada tahap ini dilakukan analisis *source code* yang sudah identidikasi masalah pada perangkat lunak setelah itu dilakukan identifikasi *smells code* dan *query* lalu mulai mengkaji *source code* tersebut.

c) Pemetaan *Source Code*

Pada tahap ini peneliti mengambil beberapa indikasi yang sudah di analisis sebelumnya untuk dijadikan sebagai acuan untuk mengubah program sistem informasi ini agar sesuai.

d) Implementasi *Refactoring*

Pada tahap ini dilakukan *refactoring* terhadap modul pendaftaran program lama (*legacy system*), dilakukan *refactoring* pada kompleksitas *code* yang didalamnya mengandung *smell code* dan *refactoring query* lama.

e) Pengujian

Pada tahap ini dilakukan pengujian di mana perangkat lunak sudah melewati tahap implementasi. Pada tahap ini, proses tersebut meliputi implementasi *refactoring* dan pengukuran *response time query*.

Hasil dan Pembahasan

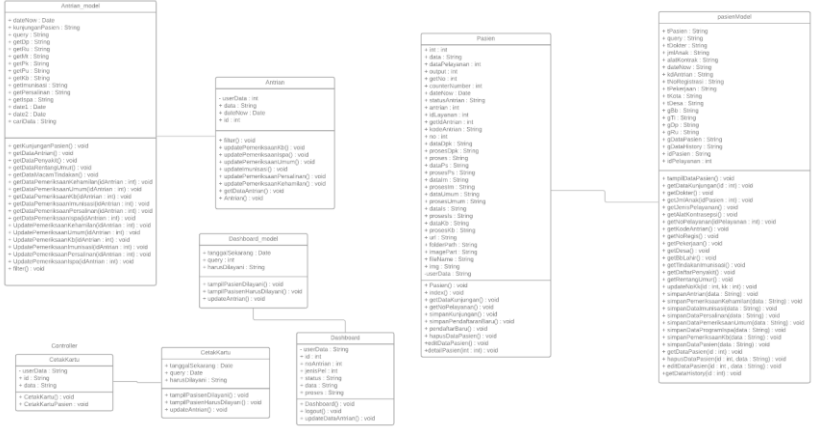
Perancangan sistem optimalisasi merupakan tahap selanjutnya dalam proses optimalisasi sistem yang memberikan gambaran dan penjelasan bagaimana perencanaan untuk mengoptimalkan kompleksitas *code* yang didalamnya terindikasi *duplicate code*.

1. *Class Diagram*

Class Diagram adalah diagram yang menggambarkan kelas-kelas yang bekerja pada sistem yang saling terhubung dan berkaitan pada modul pendaftaran. Class diagram yang digunakan untuk sistem ini mengikuti konsep MVC (Model, *View*, *Controller*). Berdasarkan analisis terdahap class diagram tersebut, dapat dilihat pada Gambar 1 dan Gambar 2 dibawah ini.

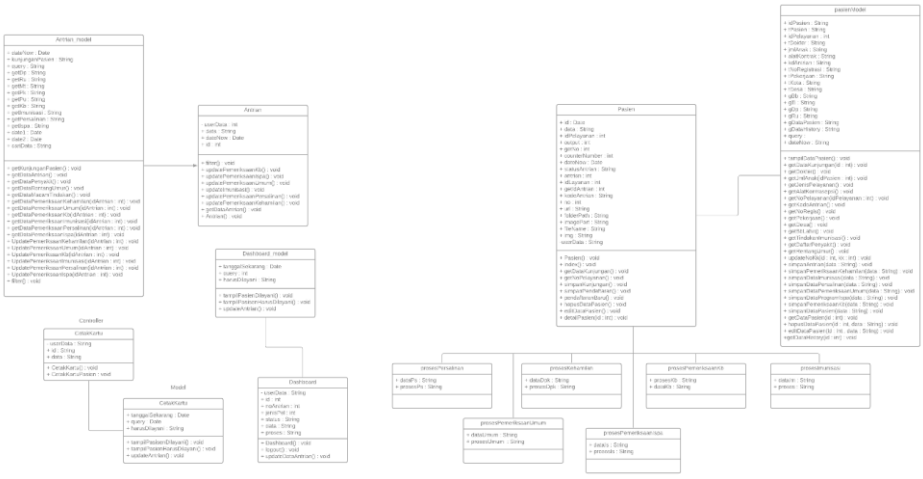
a. Class Diagram

Pada class diagram controller ini merupakan penghubung antara model dan view. Didalam *controller* inilah terdapat class dan fungsi-fungsi yang memproses permintaan dari view kedalam struktur data didalam model. Tugas *controller* ini adalah menyediakan berbagai variabel yang akan ditampilkan di view, memanggil model untuk melakukan akses ke database, menyediakan validasi atau pengecekan terhadap input. Dan *Class* Diagram Model, didalam digram model ini berisi class dan fungsi untuk melakukan manipulasi data seperti *insert*, *update*, *delete* dan *search*, namun tidak dapat berhubungan dengan bagian view secara langsung, model ini berhubungan dengan perintah-perintah *query SQL*.



Gambar 1
Class Diagram Sebelum Di *Refactoring*

Dan setelah di *refactoring* pada bagian *class pasien*, maka gambar tersebut seperti dibawah ini :



Gambar 2
Class Diagram Setelah Di *Refactoring*

b. Refactoring Duplicate Code

Tahap implementasi pemetaan *smell code* yaitu *duplicate code* pada program klinik menggunakan *Extensions* yaitu *SonarLint* pada *software Visual Studio Code*. Yang dimana *duplicate code* terjadi pengulangan baris kode yang sama. Untuk memperbaikinya dengan melakukan refatoring menggunakan teknik *extract method* yaitu membuat method baru untuk menampung code yang duplikat.

Untuk memperbaiki struktur kode pada program yaitu kompleksitas kode dan *duplicated code*. Terdapat pada kondisi if ($idLayanan == 1), maka dibuat sebuah 2 *function* baru yaitu proses Persalinan () dan kondisi Cetak (). *Function* proses Persalinan () akan diimplementasikan pada kode yang kompleks dan *function* kondisi Cetak akan di implementasikan pada kode yang terindikasi *duplicate code*.

1. Langkah pertama ini melakukan identifikasi potongan kode yang akan di ekstrak. Berikut dibawah terdapat kompleksitas *code* dan *duplicate code* yang akan di *extract method*.

```
if ($idLayanan == 1) {
if (empty($antrian)) {
//untuk simpan ke table antrian
$no = "1";
$data = array('created_at'=>$dateNow,
'id_dokter'=>$this->input->post('namaDokter'),
'id_pasien'=>$this->input->post('namaPasien'),
'no_antrian'=>$no,
'status_antrian'=>$statusAntrian,
'id_jenis_pelayanan'=>$this->input->post('jenisPelayanan'),
'tgl_antrian'=>$dateNow,
'kode_antrian'=>$this->input->post('kode_antrian'));
//untuk simpan ke table detail_pemeriksaan kehamilan
```

```
$dataDpk = array('created_at'=>$dateNow,
'id_antrian'=>$kodeAntrian,
'id_pasien'=>$this->input->post('namaPasien'),
'tgl_lahir'=>$this->input->post('tglLahir'),
'nik'=>$this->input->post('nik'),
'umur'=>$this->input->post('umur'),
'nama_suami'=>$this->input->post('namaPj'),
'no_kk'=>$this->input->post('noKk'),
'buku_kia'=>$this->input->post('bukuKia'),
'alamat'=>$this->input->post('alamat'),
'hpht'=>$this->input->post('hpht'),
'tp'=>$this->input->post('tp'),
'bb'=>$this->input->post('bb'),
'tb'=>$this->input->post('tb'),
'usia_kehamilan'=>$this->input->post('usiaKehamilan'),
'gpa'=>$this->input->post('gpa'),
'k1'=>$this->input->post('k1'),
'k4'=>$this->input->post('k4'),
'tt'=>$this->input->post('tt'),
'lila'=>$this->input->post('lila'),
'hb'=>$this->input->post('hb'),
'resiko'=>$this->input->post('resiko'),
'keterangan'=>$this->input->post('keterangan'),
'baru_lama'=>$this->input->post('baruLama'),
'catatan'=>$this->input->post('catatan'));
$prosesDpk = $this >Pasien_model >simpanPemeriksaanKehamilan($dataDpk);
//simpan data ke table detail_pemeriksaan kehamilan ke table detail_pemeriksaan_kehamilan
$proses = $this->Pasien_model->simpanAntrian($data); //simpan data antrian ke table antrian
if (!$proses & !$prosesDpk) {
//script pake print nomor antrian
$url = base_url('index.php/cetakKartu');
echo "<script>window.open('".$url."','_blank');</script>";
echo "<script>history.go(-2);</script>";

//script ga pake print
// echo "<script>alert('Data Berhasil Disimpan');window.location='index'</script>";
} else {
echo "<script>alert('Data Gagal Di Simpan');history.go(-2)</script>";
}

} else {
$data = array('created_at'=>$dateNow,
'id_dokter'=>$this->input->post('namaDokter'),
'id_pasien'=>$this->input->post('namaPasien'),
'no_antrian'=>$this->input->post('noAntrian'),
'status_antrian'=>$statusAntrian,
'id_jenis_pelayanan'=>$this->input->post('jenisPelayanan'),
'tgl_antrian'=>$dateNow,
'kode_antrian'=>$this->input->post('kode_antrian'));
```

```
//untuk simpan ke db detail_pemeriksaan kehamilan
$dataDpk = array('created_at'=>$dateNow,
'id_antrian'=>$kodeAntrian,
'id_pasien'=>$this->input->post('namaPasien'),
'tgl_lahir'=>$this->input->post('tglLahir'),
'nik'=>$this->input->post('nik'),
'umur'=>$this->input->post('umur'),
'nama_suami'=>$this->input->post('namaPj'),
'no_kk'=>$this->input->post('noKk'),
'buku_kia'=>$this->input->post('bukuKia'),
'alamat'=>$this->input->post('alamat'),
'hpht'=>$this->input->post('hpht'),
'tp'=>$this->input->post('tp'),
'bb'=>$this->input->post('bb'),
'tb'=>$this->input->post('tb'),
'usia_kehamilan'=>$this->input->post('usiaKehamilan'),
'gpa'=>$this->input->post('gpa'),
'k1'=>$this->input->post('k1'),
'k4'=>$this->input->post('k4'),
'tt'=>$this->input->post('tt'),
'lila'=>$this->input->post('lila'),
'hb'=>$this->input->post('hb'),
'resiko'=>$this->input->post('resiko'),
'keterangan'=>$this->input->post('keterangan'),
'baru_lama'=>$this->input->post('baruLama'),
'catatan'=>$this->input->post('catatan'));
$prosesDpk = $this->Pasien_model->simpanPemeriksaanKehamilan($dataDpk);
$proses = $this->Pasien_model->simpanAntrian($data);
if (!$proses & !$prosesDpk) {
// header('Location: antrian.php');
//script pake print nomot antrian
$url = base_url('index.php/cetakKartu');
echo "<script>window.open('".$url."','_blank');</script>";
echo "<script>history.go(-2);</script>";
//script ga pake print nomor antrian
// echo "<script>alert('Data Berhasil Disimpan');window.location='index'</script>";
} else {
echo "<script>alert('Data Gagal Di Simpan');history.go(-2)</script>";
}
}
// end fungsi simpan data ke table detail pemeriksaan kehamilan
```

2. Langkah selanjutnya buat function kosong dan salin kode.
   Untuk memperbaiki kompleksitas kode dan *duplicate code* maka dilakukan refactoring seperti berikut ini. Dengan membuat function baru yaitu proses Kehamilan () setelah itu menyalin kode ke dalam function proses Kehamilan (), sehingga kode seperti dibawah ini :

```
public function prosesKehamilan(){
$dateNow = $waktuSekarang = gmdate("Y-m-d H:i:s",
time()+60*60*7);
$statusAntrian = "Proses";
$antrian = $this->input->post('noAntrian');
$idLayanan = $this->input->post('jenisPelayanan');
$getIdAntrian = $this->input->post('idAntrian');
$kodeAntrian = $getIdAntrian + 1;

if (empty($antrian)) {
//untuk simpan ke table antrian
$no = "1";
$data = array('created_at'=>$dateNow,
'id_dokter'=>$this->input->post('namaDokter'),
'id_pasien'=>$this->input->post('namaPasien'),
'no_antrian'=>$no,
'status_antrian'=>$statusAntrian,
'id_jenis_pelayanan'=>$this->input->post('jenisPelayanan'),
'tgl_antrian'=>$dateNow,
'kode_antrian'=>$this->input->post('kode_antrian'));

//untuk simpan ke table detail_pemeriksaan kehamilan
$dataDpk = array('created_at'=>$dateNow,
'id_antrian'=>$kodeAntrian,
'id_pasien'=>$this->input->post('namaPasien'),
'tgl_lahir'=>$this->input->post('tglLahir'),
'nik'=>$this->input->post('nik'),
'umur'=>$this->input->post('umur'),
'nama_suami'=>$this->input->post('namaPj'),
'no_kk'=>$this->input->post('noKk'),
'buku_kia'=>$this->input->post('bukuKia'),
'alamat'=>$this->input->post('alamat'),
'hpht'=>$this->input->post('hpht'),
'tp'=>$this->input->post('tp'),
'bb'=>$this->input->post('bb'),
'tb'=>$this->input->post('tb'),
'usia_kehamilan'=>$this->input->post('usiaKehamilan'),
'gpa'=>$this->input->post('gpa'),
'k1'=>$this->input->post('k1'),
'k4'=>$this->input->post('k4'),
'tt'=>$this->input->post('tt'),
'lila'=>$this->input->post('lila'),
'hb'=>$this->input->post('hb'),
'resiko'=>$this->input->post('resiko'),
'keterangan'=>$this->input->post('keterangan'),
'baru_lama'=>$this->input->post('baruLama'),
'catatan'=>$this->input->post('catatan'));
$prosesDpk = $this->Pasien_model-
>simpanPemeriksaanKehamilan($dataDpk); //simpan data ke table
detail_pemeriksaan kehamilan ke table
detail_pemeriksaan_kehamilan
```

```
$proses = $this->Pasien_model->simpanAntrian($data); //simpan
data antrian ke table antrian

if (!$proses & !$prosesDpk) {
//script pake print nomor antrian
$url = base_url('index.php/cetakKartu');
echo "<script>window.open('".$url."','_blank');</script>";
echo "<script>history.go(-2);</script>";

//script ga pake print
// echo "<script>alert('Data Berhasil
Disimpan');window.location='index'</script>";
} else {
echo "<script>alert('Data Gagal Di Simpan');history.go (-2) </script>";
}

} else {
$data = array('created_at'=>$dateNow,
'id_dokter'=>$this->input->post('namaDokter'),
'id_pasien'=>$this->input->post('namaPasien'),
'no_antrian'=>$this->input->post('noAntrian'),
'status_antrian'=>$statusAntrian,
'id_jenis_pelayanan'=>$this->input->post('jenisPelayanan'),
'tgl_antrian'=>$dateNow,
'kode_antrian'=>$this->input->post('kode_antrian'));
//untuk simpan ke db detail_pemeriksaan kehamilan
$dataDpk = array('created_at'=>$dateNow,
'id_antrian'=>$kodeAntrian,
'id_pasien'=>$this->input->post('namaPasien'),
'tgl_lahir'=>$this->input->post('tglLahir'),
'nik'=>$this->input->post('nik'),
'umur'=>$this->input->post('umur'),
'nama_suami'=>$this->input->post('namaPj'),
'no_kk'=>$this->input->post('noKk'),
'buku_kia'=>$this->input->post('bukuKia'),
'alamat'=>$this->input->post('alamat'),
'hpht'=>$this->input->post('hpht'),
'tp'=>$this->input->post('tp'),
'bb'=>$this->input->post('bb'),
'tb'=>$this->input->post('tb'),
'usia_kehamilan'=>$this->input->post('usiaKehamilan'),
'gpa'=>$this->input->post('gpa'),
'k1'=>$this->input->post('k1'),
'k4'=>$this->input->post('k4'),
'tt'=>$this->input->post('tt'),
'lila'=>$this->input->post('lila'),
'hb'=>$this->input->post('hb'),
'resiko'=>$this->input->post('resiko'),
'keterangan'=>$this->input->post('keterangan')
'baru_lama'=>$this->input->post('baruLama'),
'catatan'=>$this->input->post('catatan'));
```

```
$prosesDpk = $this->Pasien_model-
>simpanPemeriksaanKehamilan($dataDpk);
$proses = $this->Pasien_model->simpanAntrian($data);
if (!$proses & !$prosesDpk) {
// header('Location: antrian.php');
//script pake print nomot antrian
$result = $this->kondisiCetak();
//script ga pake print nomor antrian
// echo "<script>alert('Data Berhasil
Disimpan');window.location='index'</script>";

} else {
echo "<script>alert('Data Gagal Di Simpan');history.go(-2)</script>";
}
}
// end fungsi simpan data ke table detail pemeriksaan kehamilan
}
```

Dan setelah itu pada bagian duplicate code dilakukan *refactoring* dengan menyalin kode ke *function* baru yaitu bernama kondisi Cetak () sehingga kode seperti dibawah ini :

```
public function kondisiCetak($proses) {
//script pake print nomor antrian
$url = base_url('index.php/cetakKartu');
echo "<script>window.open('".$url."','_blank');</script>";
echo "<script>history.go(-2);</script>";
}
```

3. Langkah terakhir, program tinggal memanggil *method* tadi kedalam *method/function* simpan Kunjungan ( ) . Setelah di *extract method* maka hasil *refactoring* untuk memperbaiki kompleksitas *code* dan *Duplicated Code* akan menjadi seperti ini:

```
public function simpanKunjungan(){
$dateNow = $waktuSekarang = gmdate("Y-m-d H:i:s",
time()+60*60*7);
$statusAntrian = "Proses";
$antrian = $this->input->post('noAntrian');
$idLayanan = $this->input->post('jenisPelayanan');
$getIdAntrian = $this->input->post('idAntrian');
$kodeAntrian = $getIdAntrian + 1;

if ($idLayanan == 1) {
// start fungsi simpan data ke table pemeriksaan kehamilan
$result = $this->prosesKehamilan();

} else if ($idLayanan == 3){
```

```
// start fungsi simpan data ke table persalinan
$result = $this->prosesPersalinan();

} else if ($idLayanan == 8){
// start fungsi simpan data ke table Imunisasi
$result = $this->prosesImunisasi();

} else if ($idLayanan == 9){
// start fungsi simpan data ke table Pemeriksaan Umum
$result = $this->prosesPemeriksaanUmum();

} else if ($idLayanan == 34){
// start fungsi simpan data ke table pemeriksaan Ispa
$result = $this->prosesPemeriksaanIspa();

} else if($idLayanan == 37){
// start fungsi simpan data ke table pemeriksaan Kb
$result = $this->prosesPemeriksaanKb();

}
}
```

Hasil *extract method* menggunakan pengujian *white box* :

1. Perhitungan CC

   Berikut ini tahap menentukan perhitungan *Cyclomatic Complexity*:

   a. Jumlah Region = 6

   b. V (G) = E – N + 2
      = 19 – 15 + 2
      = 4 + 2
      = 6

   c. V (G) = P + 1
      = 5 + 1 = 6

Jadi *cyclomatic complexity* untuk *flowgraph* perhitungan nilai akhir adalah 6. Nilai V(G) atau CC diperlukan untuk mengetahui jumlah independent path yang dapat dibuat pada tahap selanjutnya.

2. Jalur Independent Path

   Jalur Independent Path adalah jalur pada program yang menghubungkan node awal dengan node akhir. Independent Path minimal melewati sebuah *edge* baru dengan alur yang belum pernah dilaluinya. Berdasarkan *cyclomatic complexity* tersebut, maka terdapat 6 jalur (path) sesuai dengan nilai V (G) atau CC. Independent path dapat dilihat pada dibawah ini :

   Jalur 1 : 1-2-3-4-15
   Jalur 2 : 1-2-3-5-6-15
   Jalur 3 : 1-2-3-5-7-8-15
   Jalur 4 : 1-2-3-5-7-9-10-15

Jalur 5 : 1-2-3-5-7-9-11-12-15
Jalur 6 : 1-2-3-5-7-9-11-13-14-15

Berdasarkan independent path dapat dilihat bahwa node awal dan node akhir sudah terhubung dan semua edge sudah dilalui. Setelah diketahui jumlah jalur independennya, maka akan dilakukan perbandingan menggunakan tabel antara *cyclomatic complexity* dan resiko pada tabel dibawah ini :

Tabel 1
Hubungan Cyclocmatic Complexity Dengan Resiko

Nilai CC	Tipe Prosedur	Tingkat Resiko
1-4	Prosedur sederhana	Rendah
5-10	Prosedur yang terstruktur dengan baik dan stabil	Rendah
11-20	Prosedur yang lebih kompleks	Menengah
21-50	Prosedur yang kompleks dan kritis	Tinggi
>50	Rentan kesalahan, sangat menganggu, prosedur tidak dapat di uji	Sangat Tinggi

Jadi menurut tabel di atas untuk fungsi simpan Kunjungan ini memiliki tingkat resiko yang rendah dengan Prosedur yang terstruktur dengan baik dan stabil karena memiliki jalur independen berjumlah 6. Setelah diketahui jalur independennya maka langkah selanjutnya adalah test case.

3. *Test Case*

Test case atau kasus uji dibuat untuk mengeksekusi semua alur logika yang telah dibuat. Setelah dijalankan, maka akan dapat diketahui apakah hasil pengujian sesuai atau tidak dengan yang direncanakan.
Berikut ini adalah test case yang telah dibuat:

Tabel 2
Hasil Pengujian *Test Case*

Path	Proses pengujian	Hasil
1	Mulai inputan data pasien, Mendapat data pasien, Melakukan pengecekan data berdasarkan jenis layanan, input data pasien jika benar maka data pasien tersebut berhasil di inputkan berdasarkan jenis layanan, selesai.	sesuai
2	Mulai inputan data pasien, Mendapat data pasien, Melakukan pengecekan data berdasarkan jenis layanan, jika tidak sesuai maka melakukan pengecekan selanjutnya, jika data pasien sesuai dengan jenis layanan maka data pasien tersebut berhasil di inputkan berdasarkan jenis layanan, muncul	Sesuai

	data pasien di halaman kunjungan, selesai.	
3	Mulai inputan data pasien, Mendapat data pasien, Melakukan pengecekan data berdasarkan jenis layanan, jika tidak sesuai maka melakukan pengecekan selanjutnya, masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, jika data pasien sesuai dengan jenis layanan maka data pasien tersebut berhasil di inputkan berdasarkan jenis layanan, muncul data pasien di halaman kunjungan, selesai.	Sesuai
4	Mulai inputan data pasien, Mendapat data pasien, Melakukan pengecekan data berdasarkan jenis layanan, jika tidak sesuai maka melakukan pengecekan selanjutnya, masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya , jika data pasien sesuai dengan jenis layanan maka data pasien tersebut berhasil di inputkan berdasarkan jenis layanan, muncul data pasien di halaman kunjungan, selesai.	Sesuai
5	Mulai inputan data pasien, Mendapat data pasien, Melakukan pengecekan data berdasarkan jenis layanan, jika tidak sesuai maka melakukan pengecekan selanjutnya, masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, Maka masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, Maka masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, jika data pasien sesuai dengan jenis layanan maka data pasien tersebut berhasil di inputkan berdasarkan jenis layanan, muncul data pasien di halaman kunjungan, selesai.	Sesuai
6	Mulai inputan data pasien, Mendapat data pasien, Melakukan pengecekan data berdasarkan jenis layanan, jika tidak sesuai maka melakukan pengecekan selanjutnya, masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, Maka masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan selanjutnya, Maka masuk ke kondisi selanjutnya jika data tidak sesuai maka melakukan pengecekan	Sesuai

	selanjutnya, Maka masuk ke kondisi selanjutnya jika data pasien tidak sesuai dengan jenis layanan maka melakukan pengecekan selanjutnya, jika data pasien sesuai dengan jenis layanan maka data pasien tersebut berhasil di inputkan berdasarkan jenis layanan, muncul data pasien di halaman kunjungan dan tersimpan pada database, selesai.	

Pengujian *white box* telah selesai dan dapat diperoleh hasil bahwa aplikasi dapat berjalan sesuai dengan yang diharapkan.

4. *Refactoring Query*

Desain *query* memainkan peran penting dalam menentukan kinerja sistem (Satoto et al., 2017). Pada tahap ini dilakukan dengan cara melakukan strestest dengan mengunakan software *MySQLSlap*. Tahap *refactoring query* ini akan melakukan pengujian dari hasil optimalisasi menggunakan view (Safitri et al., 2018). Hasil yang akan dicatat dalam pengujian ini adalah *performance* kecepatan serta efisien penggunaan sintak dalam sistem. Pengujian dilakukan pada data Kunjungan Pasien yang berelasi dengan beberapa tabel kemudian akan di ambil hasil perbandingan dari penggunaan join dan *view*.

Refactoring Query

No	Pemetaan Query lama	Pemetaan query baru

1	public function getKunjunganPasien(){
	$dateNow = date('Y-m-d');
	//script untuk menampilkan semua kunjungan
	$kunjunganPasien = $this->db->query('SELECT a.id,a.id_pasien,a.kode_antrian,a.no_antrian,a.status_antrian,
	a.tgl_antrian,d.nama_dokter,
	p.nama_pasien, j.nama_pelayanan
	FROM antrians AS a
	JOIN dokters AS d ON a.id_dokter = d.id
	JOIN pasiens AS p ON a.id_pasien = p.id
	JOIN jenis_pelayanans AS j
	ON a.id_jenis_pelayanan = j.id ORDER BY a.tgl_antrian DESC ');
	
	return $kunjunganPasien;}
		public function getKunjunganPasien(){
		
		$this->db->from(‘kunjunganPasien’);
		
		$query = $this->db->get();
		}.

Tahap selanjutnya melakukan banchmarking query untuk mengetahui *performance response time* saat melakukan proses penggunaan *view* dan join

(Safitri et al., 2018). Benchmarking ini menggunakan *tools mysqslap* (Informasi, 2017). Berikut hasil proses benchmarking menggunakan *mysqlslap* (*Mysqlslap_ Benchmark MySQL - OnnoCenterWiki*, n.d.).

Tabel 3
Hasil *Performance Response Time*
***Query* Menggunakan Join**

JOIN	
Running ke	**Hasil benchmarking**
1	0.125
2	0.023
3	0.025
4	0.125
5	0.025
6	0.125
7	0.024
8	0.050
9	0.129
10	0.025
Rata-rata	0.0676

Pada tabel 3 merupakan hasil sebelum dilakukan refactoring. Pada pengujian menggunakan JOIN, setiap jumlah baris data diuji sebanyak 10 kali kemudian dihitung nilai rata-rata setiap baris data. Dilihat dari hasil pengujian menggunakan join, hasil eksperimen 1000 data mengalami perubahan setiap proses benchmarking namun peningkatan response time relatif tidak stabil dan hanya mengalami kenaikan response *time query* tidak terlalu signifikan dengan hasil rata-rata 0,0676 detik.

Tabel 4
Hasil Performance Response Time
Query Menggunakan VIEW

VIEW	
Running ke	**Hasil benchmarking**
1	0.023
2	0.025
3	0.025
4	0.023
5	0.025
6	0.025
7	0.025
8	0.023
9	0.025
10	0.025
Rata-rata	**0.0244**

Pada tabel 4 merupakan hasil sesudah dilakukan refactored. Pada pengujian menggunakan *View*, setiap jumlah baris data diuji sebanyak 10 kali

kemudian dihitung nilai rata-rata setiap baris data. Dilihat dari hasil pengujian menggunakan view, hasil eksperimen 1000 data mengalami perubahan setiap proses benchmarking namun relatif stabil *response time query* dengan hasil rata-rata 0.0244 detik.

Kesimpulan

Dari optimalisasi perangkat lunak hasil refactoring di atas, maka bisa didapatkan kesimpulan sebagai berikut ini: 1) Bug atau kesalahan yang ditemukan pada *function* simpan kunjungan ditemukan disebabkan karena *function* tersebut terindikasi *duplicate code*. Untuk mengoptimalkan *duplicate code* dengan melakukan *refactoring* kompleksitas kode yang didalamnya mengandung *bad smell code* dalam sebuah struktur kode di *function* simpan Kunjungan. Pada *function* tersebut dibuat *flowchart* meliputi *flowchart* simpan kunjungan pasien. Setelah dibuat *flowgraph*, dihitung *cyclomatic complexity* dan menghasilkan 24 jalur independen dan setelah dilakukan *refactoring* pada bagian *function* tersebut menggunakan teknik *extract method* maka fungsi simpan Kunjungan ini menghasilkan 6 jalur independen sehingga memiliki tingkat resiko yang rendah dengan prosedur yang terstruktur dengan baik dan stabil. Setelah itu dibuat tes case dapat diperoleh hasil 6 tes case yang valid. Sehingga struktur internal sebuah sistem perangkat lunak dengan tetap mempertahankan fungsionalitas dari sebuah *system* yang sudah di *refactoring*. 2) Penggunaan view untuk menampilkan semua data kunjungan pasien memiliki penulisan sintak yang lebih sedikit daripada menggunakan join. Penggunaan join memiliki waktu rata-rata 0.0676 perdetik dan penggunaan view untuk menampilkan semua data kunjungan pasien memiliki waktu yang lebih cepat daripada menggunakan join yaitu 0.0244 perdetik mengalami penurunan hingga 63.91%.

BIBLIOGRAFI

Baqais, A. A. B., & Alshayeb, M. (2020). *Automatic software refactoring: a systematic literature review. Software Quality Journal*, 28 (2), 459–502. https://doi.org/10.1007/s11219-019-09477-y. Google Scholar

Cairo, A. S., Carneiro, G. de F., & Monteiro, M. P. (2018). *The impact of code smells on software bugs: A systematic literature review. Information (Switzerland)*, 9 (11), 1–21. https://doi.org/10.3390/info9110273. Google Scholar

Fontana, F. A., & Spinelli, S. (2011). *Impact of refactoring on quality code evaluation. Proceedings - International Conference on Software Engineering*, 37–40. https://doi.org/10.1145/1984732.1984741. Google Scholar

Fowler, M. (1999). *Refactoring: Refactoring: Improving the Design of Existing Programs.* Google Scholar

Gunawan, R., Rahmatulloh, A., & Darmawan, I. (2019). *Performance evaluation of query response time in the document stored nosql database*. *2019 16th International Conference on Quality in Research, QIR 2019 - International Symposium on Electrical and Computer Engineering*, 1–6. https://doi.org/10.1109/QIR.2019.8898035. Google Scholar

Haque, M. S., Carver, J., & Atkison, T. (2018). C*auses, impacts, and detection approaches of code smell: A survey. Proceedings of the ACMSE 2018 Conference*, *2018-Janua* (1). https://doi.org/10.1145/3190645.3190697. Google Scholar

Ibrahim, R., Ahmed, M., Nayak, R., & Jamel, S. (2020). *Reducing redundancy of test cases generation using code smell detection and refactoring. Journal of King Saud University - Computer and Information Sciences*, 32 (3), 367–374. https://doi.org/10.1016/j.jksuci.2018.06.005. Google Scholar

Informasi, F. T. (2017). Database *Performance Benchmark on.* Google Scholar

Lacerda, G., Petrillo, F., Pimenta, M., & Guéhéneuc, Y. G. (2020). *Code smells and refactoring: A tertiary systematic review of challenges and observations*. *Journal of Systems and Software*, *167*. https://doi.org/10.1016/j.jss.2020.110610. Google Scholar

Mysqlslap_ benchmark MySQL - OnnoCenterWiki. (n.d.). Google Scholar

Safitri, P. K., Winarno, W. W., & Pramono, E. (2018). *Optimasi Query* untuk Sistem Informasi Penjadwalan Mata Pelajaran Sekolah Menggunakan View (Studi Kasus : SMK VIP Purworejo). XIII, 46–51. Google Scholar

Satoto, K. I., Isnanto, R. R., Kridalukmana, R., & Martono, K. T. (2017). *Optimizing MySQL database system on information systems research, publications and community service. Proceedings - 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2016*, 1–5. https://doi.org/10.1109/ICITACEE.2016.7892476. Google Scholar

Shrivastava, A. (n.d.). Martin Fowler - *Refactoring Improving the Design of Existing Code*. Google Scholar

Singh, S., & Kaur, S. (2018). *A systematic literature review*: *Refactoring for disclosing code smells in object oriented software*. Ain Shams Engineering Journal, 9 (4), 2129–2151. https://doi.org/10.1016/j.asej.2017.03.002. Google Scholar

Studi, P., Informasi, S., & Tarumanagara, U. (n.d.). Perbandingan Optimasi Query. 113–117. Google Scholar