OPTIMALISASI KUALITAS INTEGRITY PADA SISTEM DINAS KESEHATAN KOTA CIMAHI MENGGUNAKAN TEKNIK REFACTORING

Intan Fitaloka, Yulison Herry Chrisnanto, Rezki Yuniarti
Informatika, Universitas Jenderal Achmad Yani
Jalan Terusan Jend. Sudirman, Cimahi, Kota Cimahi, Jawa Barat 40525
intanfitaloka20@unjani.ac.id

ABSTRAK

Proses pengembangan perangkat lunak melibatkan empat kegiatan utama yang penting: spesifikasi, perancangan, implementasi, dan pengujian, juga merupakan hal yang penting untuk meningkatkan kinerja sistem. Refaktorisasi dapat membantu meningkatkan kualitas sistem dengan memperbaiki kode yang ada dan memastikan bahwa kode dapat diuji secara otomatis dengan mudah.. Integritas pada sistem informasi sangat penting untuk memastikan bahwa data dan informasi yang ada dalam sistem akurat dan konsisten, serta untuk melindungi aset informasi dari potensi ancaman. Untuk melihat kekurangan dari sebuah program, metode code smell merupakan salah satu dari metode pendeteksi kesalahan source code. Pada jurnal ini melakukan optimalisasi pada sistem Dinas Kesehatan Kota Cimahi menggunakan metode refactoring. Metodologi yang digunakan dalam penelitian ini meliputi analisa sistem, analisa code smell, implementasi refactoring dan pengujian. Hasil pengujian pada sistem informasi yang telah direfactoring menunjukkan peningkatan. Refactoring dengan metode extract class dan extract method berhasil membuat metode dalam kelas menjadi lebih spesifik dan saling terkait. Kode yang sebelumnya belum terintegrasi keamanannya, menjadi lebih aman. Struktur kode yang lebih baik ini juga mempermudah proses deteksi dan perbaikan bug, yang mengarah pada pengurangan jumlah bug dan peningkatan performa serta stabilitas sistem.

Kata kunci : *refactoring, code smell, integrity, optimalisasi*

1. PENDAHULUAN

Proses pengembangan perangkat lunak melibatkan empat kegiatan utama yang penting: spesifikasi, perancangan, implementasi, dan pengujian [1], juga merupakan hal yang penting untuk meningkatkan kinerja sistem. Sistem akan menjadi usang tanpa dilakukan pengembangan [2]. Untuk menjaga kualitas perangkat lunak, sebanyak 73% organisasi telah melakukan pengujian test automation. Pengujian test automation penting untuk mempertahankan kualitas dalam pengembangan perangkat lunak [3].

Correctness, usability, reliability, integrity, dan *efficiency* adalah beberapa faktor kualitas yang dapat digunakan untuk mengevaluasi dan memastikan kualitas sistem informasi web. Faktor ini dikenal sebagai faktor kualitas software atau kualitas McCall [4], dan penentuan kualitas McCall sendiri dapat dilakukan berdasarkan faktor yang mempengaruhi sistem; faktor-faktor ini terkait dengan objek atau perangkat lunak yang digunakan. Integritas pada sistem informasi sangat penting untuk memastikan bahwa data dan informasi yang ada dalam sistem akurat dan konsisten, serta untuk melindungi aset informasi dari potensi ancaman [5]. Objek yang digunakan pada penelitian ini adalah Sistem Dinas Kesehatan Kota Cimahi.

Salah satu cara untuk meningkatkan kualitas dan kinerja sistem perangkat lunak adalah refaktorisasi [6]. Refaktorisasi adalah kegiatan yang mengambil tata letak internal perangkat lunak, untuk meningkatkan struktur desain tetapi tetap mempertahankan fungsionalitasnya [7].

Karena dengan refaktorisasi dapat meningkatkan kualitas sistem perangkat lunak dengan cara-cara seperti mengurangi smell code, memperbaiki kode yang sudah ada, memastikan kode dapat diuji secara otomatis, perbaikan bug, pengoptimalan kode, implementasi fitur atau modifikasi fitur yang sudah ada, dan mempermudah perawatan [5] [8].

Salah satu cara untuk mendeteksi kesalahan source code yang akan dilakukan refactoring adalah dengan menggunakan metode code smell untuk melihat kekurangan program. Aspek-aspek ini termasuk keterbacaan, konsistensi, kualitas, struktur, dan penggunaan bahasa pemrograman [6].

Untuk melihat kelima aspek yang digunakan, salah satu jenis code smell yang terkait adalah long method. Metode ini terjadi ketika metode dalam sebuah program ditulis terlalu panjang atau memiliki banyak variabel, parameter, dan kondisi yang dijalankan dalam satu method. Kondisi ini dapat membuat program sulit dipahami, dipelihara, dan diubah. Oleh karena itu, deteksi long method sangat penting untuk meningkatkan kualitas perangkat lunak dan memudahkan pengembangan dan pemeliharaan program [9]. Selain long method, ada juga duplicate code, yang merujuk pada kesamaan penulisan kode yang dimulai dengan nama variabel atau kode yang sangat mirip dari segi penulisan, yang dapat membuat pemeliharaan menjadi sulit [10].

Penelitian sebelumnya membahas tentang masalah desain umum pada kelas besar yang memiliki banyak tanggung jawab, khususnya dengan teknik extract class. Beberapa penelitian terkait juga dibahas dalam penelitian tersebut, seperti penelitian tentang

pengaruh refactoring pada kualitas perangkat lunak dan penelitian tentang algoritma untuk mengidentifikasi urutan teknik refactoring yang optimal. Secara keseluruhan, penelitian ini memberikan kontribusi yang signifikan dalam pengembangan teknik refactoring dan peningkatan kualitas perangkat lunak [7].

Penelitian terdahulu mengenai kelas ekstraksi juga mempertimbangkan indikator dari Prinsip *Single Responsibility Principle* (SRP), yang menetapkan bahwa kelas ekstraksi akan dilakukan jika memiliki lebih dari satu tanggung jawab, yaitu hanya memenuhi kebutuhan fungsional. Penelitian ini menemukan bahwa metode SRP dapat melakukan ekstraksi metode dengan tingkat recall yang tinggi. Ekstraksi merupakan metode refactoring yang dapat digunakan untuk menangani beberapa kecacatan kode, yang memungkinkan metode dipecahkan berdasarkan relevansi fungsional. Metode ini didasarkan pada Prinsip *Single Responsibility Principle* (SRP), untuk menemukan kelompok instruksi yang paling konsisten dan merekomendasikan ekstraksi mereka ke dalam metode yang berbeda [11].

Berdasarkan uraian diatas, maka pada jurnal ini fokus pada pengoptimalan kualitas *integrity* pada Sistem Dinas Kesehatan Kota Cimahi dengan teknik refactoring menggunakan metode extract class dan extract method. Dengan tujuan untuk meningkatkan integritas dan kualitas sistem.

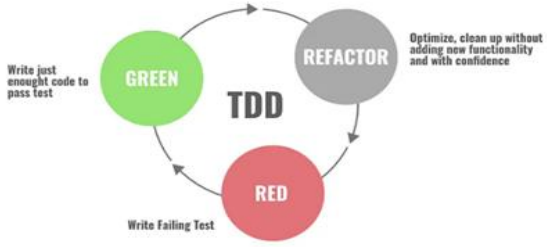
2. TINJAUAN PUSTAKA

2.1. Sistem Informasi

Sistem informasi menggabungkan aktivitas manusia dan teknologi untuk mendukung kegiatan manajemen dan operasional. Sistem informasi terdiri dari kumpulan perangkat keras, perangkat lunak, orang, database, dan prosedur yang bekerja sama untuk mengumpulkan, menyimpan, mengelola, dan menyebarkan informasi penting bagi suatu organisasi. Sistem informasi dapat membantu pengambilan keputusan, pengelolaan data, dan komunikasi di dalam organisasi [12].

2.2. Refactoring

Refactoring adalah proses menata kembali kode program, yang berfungsi untuk meningkatkan kinerja, memudahkan dalam penggunaan, memudahkan saat pemeliharaan, meningkatkan kemudahan pengguna. Refactoring adalah praktik penting dalam pengembangan perangkat lunak agar kode tetap bersih, efisien, dan mudah digunakan [13] [14].



Gambar 1. Proses Refactoring

Dari Gambar 1. Dapat dijelaskan bahwa:

- RED: Mulai dengan membuat test yang gagal / *the failing "red-test"*. Test berisi hal yang seharusnya dilakukan oleh fitur, tetapi karena belum ada implementasi fiturnya, test tersebut gagal.
- Green: Buat implementasi fitur yang dapat melewati test dengan baik.
- Refactor: Fokus dalam memperbaiki dan meningkatkan kode yang telah green tersebut.

2.3. Extract Class

Salah satu metode refactoring dalam pengembangan perangkat lunak adalah extract class, yang bertujuan untuk memisahkan sekelompok metode dari sebuah kelas menjadi kelas baru. Teknik ini digunakan ketika sekelompok metode berinteraksi erat dan membentuk fitur atau tanggung jawab yang berbeda dari kelas utama. Dengan memisahkan sekelompok metode ini ke dalam kelas baru, kita dapat meningkatkan modularitas dan keterbacaan kode, serta memudahkan penggunaan kembali kode tersebut di tempat lain [7].

2.4. Extract Method

Metode ekstraksi adalah sebuah metode refactoring yang digunakan untuk memindahkan sebagian kode dari satu metode ke metode lainnya. Metode ini membantu dalam memisahkan tanggung jawab yang berbeda-beda ke dalam metode yang lebih kecil dan lebih terfokus, yang memudahkan pemahaman dan pemeliharaan kode program [11].

2.5. Code Smell

Code smell adalah istilah yang digunakan dalam rekayasa perangkat lunak untuk merujuk pada tanda-tanda potensial dari masalah dalam desain atau implementasi kode program. Istilah ini pertama kali diperkenalkan oleh *Kent Beck*. *Code smell* tidak secara langsung mengindikasikan adanya bug atau kesalahan dalam kode, namun merupakan petunjuk bahwa ada potensi untuk memperbaiki struktur atau desain kode agar lebih baik. Dengan mendeteksi code smell, pengembang dapat melakukan perbaikan proaktif dalam kode program untuk mencegah masalah yang lebih serius di kemudian hari [6] [9].

2.6. Duplicate Code

Kode yang sama yang muncul di berbagai bagian perangkat lunak disebut kode duplikat. Ini dapat disebabkan oleh banyak hal, seperti kurangnya perencanaan yang baik, kurangnya pemahaman tentang cara menerapkan fungsionalitas yang sama di berbagai bagian program, atau kurangnya pemahaman tentang menerapkan pola desain yang tepat. Selain itu, kode duplikat menunjukkan kemungkinan untuk mengabstraksi logika umum ke dalam metode atau kelas yang berbeda yang dapat digunakan kembali. Refactoring dengan metode *extract* memungkinkan pengembang untuk mengidentifikasi dan mengisolasi

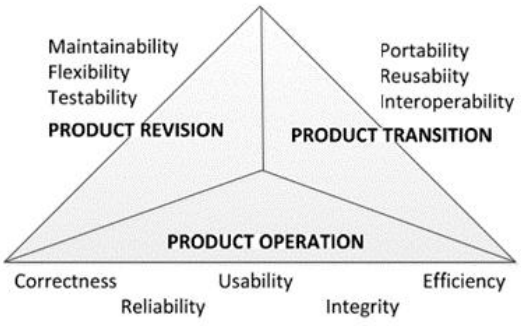
logika berulang ini ke dalam metode tunggal yang dapat dipanggil di mana saja diperlukan, yang mengurangi redundansi dan mempermudah pemeliharaan [10].

2.7. Long Method

Long Method adalah code smell yang terjadi ketika metodenya sangat panjang. Metode ini biasanya mengkonsolidasikan fungsionalitas kelas. Mereka kompleks, rumit, dan memproses banyak data dari kelas lain. Metode yang panjang biasanya menunjukkan bahwa metode tersebut melakukan terlalu banyak pekerjaan. Ini melanggar prinsip pemisahan tanggung jawab. Sebuah metode menjadi lebih sulit untuk didebug, diuji, atau diubah tanpa menimbulkan bug baru karena memiliki banyak logika dan operasi [9].

2.8. *McCall Quality*

Model kualitas perangkat lunak McCall adalah suatu kerangka kerja yang menggambarkan sebelas kriteria yang terbagi dalam tiga visi utama, yaitu operasi, revisi, dan transisi produk. Dalam model ini, kriteria tersebut dikelompokkan ke dalam faktor-faktor tertentu, yang kemudian dibuat alokasi untuk setiap faktor dengan menetapkan kriteria dan metrik yang sesuai.



Gambar 2. McCall Quality

Dari Gambar 2. dapat dijelaskan bahwa McCall terdiri atas 3 aspek [4]:

a. Sifat Operasional perangkat lunak (*Product Operation*)
   Saat membuat sebuah aplikasi, developer harus mempertimbangkan fitur yang berkaitan dengan operasional perangkat lunak. Yang diukur terkait dengan metode analisis, perancangan, dan pembuatan perangkat lunak.
b. Kemampuan perangkat lunak menjalani perubahan (*Product Revision*)
   Jika perangkat lunak yang dirancang dan dikembangkan dengan baik, akan mudah untuk direvisi apabila dibutuhkan. Tingkatan dimana perangkat lunak dapat diperbaiki adalah hal penting yang harus diperhatikan.
c. Penyesuaian perangkat lunak terhadap lingkungan baru (*Product Transition*)
   Faktor transisi membahas kemampuan perangkat lunak untuk beroperasi pada berbagai platform atau kerangka sistem.

2.9. *Class Complexity*

Class complexity dalam pemrograman merujuk pada tingkat kesulitan atau kerumitan dari sebuah kelas (*class*) dalam kode.

Weighted Method Per Class (WMC) adalah jumlah Cyclomatic Complexity dari semua metode dalam suatu kelas. WMC: [15]

$$WMC = \sum_{i=1}^{n} Ci \quad (1)$$

Dimana:
Ci: kompleksitas metode ke -i dalam kelas

Sedangkan *Cyclomatic Complexity Number* (CCN) adalah ukuran kompleksitas struktur kontrol dari suatu fungsi atau prosedur [16].

$$(CCN) = E - N + 2P \quad (2)$$

Dimana:
P: jumlah bagian yang tidak terhubung dalam flow graph (seperti pemanggilan program)
E: jumlah edges (sisi dari graf)
N: jumlah nodes (simpul dari graf)

2.10. White Box

Dalam menguji suatu sistem, bagan alir program (*flowchart*) yang didesain sebelumnya dipetakan ke dalam bentuk bagan alir control (*flowgraph*). Hal ini memudahkan untuk penentuan jumlah region, *Cyclomatic Complexity* (CC) dan independent path. Jika jumlah region, *Cyclomatic Complexity* (CC) dan independent path sama besar maka sistem dinyatakan benar, tetapi jika sebaliknya maka sistem masih memiliki kesalahan, mungkin dari segi logika maupun dari sisi lainnya.

Cyclomatic Complexity (CC) dapat dihitung dengan menggunakan rumus:

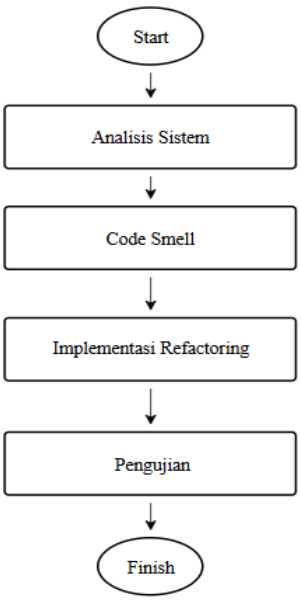
$$V(G) = E - N + 2 \quad (3)$$

Dimana:
E= jumlah edge pada flowgraph
N= jumlah node pada flowgraph

3. METODE PENELITIAN

Dari gambar 3. dapat dijelaskan bahwa pada penelitian ini terdapat beberapa tahapan yang dilakukan, yaitu:

a. Analisis Sistem
   Pada tahap awal penelitian, dilakukan identifikasi masalah dan kekurangan yang muncul dalam sistem Dinas Kesehatan Kota Cimahi. Temuan ini didokumentasikan secara rinci, memfokuskan perhatian pada kelemahan fungsionalitas, tingkat kompleksitas kode dalam perangkat lunak. Pada tahap ini dilakukan analisis terhadap perangkat lunak dengan membuat *class diagram* sistem.



Gambar 3. Metode Penelitian

b. Code Smell

Pada tahap ini melibatkan pemeriksaan mendalam terhadap struktur dan logika source code. Identifikasi source code berdasarkan factor SRP (*Single Responsibility Principle*) dan *code smells* adalah menjadi focus utama, memungkinkan penentuan potensi perbaikan dan optimasi yang dibutuhkan untuk meningkatkan kualitas serta efisiensi sistem.

c. Implementasi Refactoring

Pada tahap ini melibatkan penggunaan berbagai metode dan teknik refactoring yang sesuain dalam sistem Dinas Kesehatan Kota Cimahi. Focus utama adalah mengatasi kompleksitas kode dengan memperbaiki *code smells*. Prinsip *Continuous Integration/Continuous Deployment* (CI/CD) dapat diterapkan untuk memastikan perubahan kode dapat diintegrasikan dan diimplementasikan dengan lancar.

d. Pengujian

Setelah implementasi refactoring, dilakukan pengujian untuk mengukur kualitas dan kinerja perangkat lunak. Evaluasi mencakup nilai *bugs* dan *defects*, memberikan gambaran perbandingan dengan kondisi sebelumnya. Metrik tambahan seperti pengukuran keandalan dan skabilitas sistem juga dapat dimasukkan untuk mengevaluasi dampak dari refactoring. Juga melakukan pengujian fungsionalitas dan integrasi untuk memastikan bahwa proses refactoring tidak menyebabkan gangguan pada fungsi inti aplikasi.

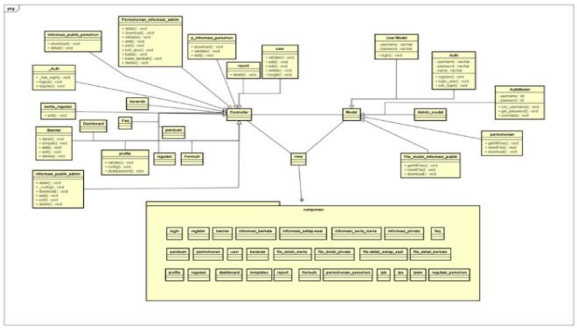
4. HASIL DAN PEMBAHASAN

Untuk melakukan refactoring, diperlukan analisis sistem dan analisis kode untuk menemukan *code smells*, sehingga mempermudah dalam proses refactoring.

4.1. Analisis Sistem

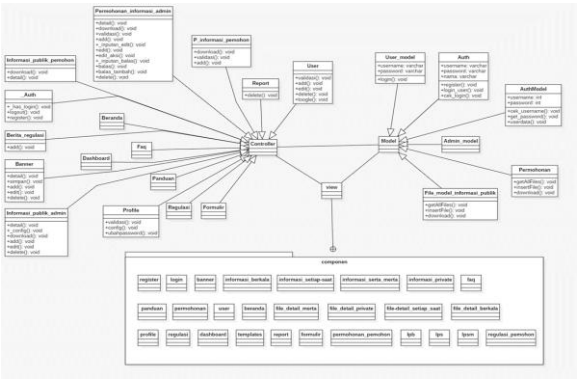
Pada tahap ini dilakukan analisis terhadap perangkat lunak dengan membuat class diagram sistem.

Class diagram merupakan penggambaran dari atribut dan *behaviour* yang disimpan kedalam entitas. Entitas-entitas ini dapat memiliki relasi dengan entitas lainnya agar dapat digunakan pada entitas yang berelasi. *Class diagram* ini terdiri dari dua komponen yaitu sebelum dan sesudah refactoring.



Gambar 4. *Class Diagram* Sebelum di Refactoring

Dan setelah di refactoring, maka gambar tesebut seperti gambar dibawah ini :



Gambar 5. *Class Diagram* setelah di Refactoring

Pada class diagram setelah di refactoring, hanya menambahkan function _inputan_edit dan _inputan-balas pada controller Permohonan_informasi_admin.

4.2. Code Smell

Setelah dilakukan analisis *code smell*, ditemukan beberapa jenis *code smell* pada kode sistem yaitu:

Tabel 1. Analisis Code Smells

No	Code Smells	Total
1	N1: *Choose Descriptive Names*	9
2	C5: *Commented-Out Code*	20
3	C3: *Redundant Comment*	11
4	G5: *Duplication*	2
5	G20: *Function Names Should say What They Do*	36
6	G12: *Clutter*	6
Total		84

Dari Tabel 1. dapat dilihat bahwa pada Sistem Dinas Kesehatan Kota Cimahi terdapat 84 code smell, yang berisikan beberapa code smells.

4.3. Refactoring

Setelah ditemukan masalah pada kode, dilakukan refactoring pada kode untuk menghilangkan code smell guna membuat kode lebih mudah dipahami dan untuk pemeliharaan kode.

a. N1 : *Choose Descriptive Names*
   Untuk memenuhi aturan code smell N!: *Choose Descriptive Names* dengan memberikan nama yang lebih deskriptif sesuai dengan fungsi dari *function* tersebut.
b. C5: Commented-Out Code
   Untuk memenuhi aturan code smell C5: *Commented-Out Code* dengan cara melakukan penghapusan komentar yang tidak perlu.
c. C3: Redundant Comment
   Untuk memenuhi aturan *code smell* C3: *Redundant Comment* sama seperti pada aturan C5, yaitu dengan menghapus komentar yang tidak perlu pada source code.
d. G5: Duplication
   Untuk memenuhi aturan code smell G5: *Duplication*, dilakukan refactoring dengan membuat ekstrak komponen dan *helper* yang dipanggil berkali-kali agar mengurangi *duplicate code*.
e. G20: Function Names Should say What They Do
   Untuk memenuhi aturan code smell G20: *Function Names Should say What They Do*, dilakukan refactor pada nama function add menjadi function add_ban agar memenuhi aturan dalam menampilkan data tambah banner.
f. G12: *Clutter*
   Untuk memenuhi aturan code smell G12:Clutter, dengan cara dilakukan penghapusan komentar yang tidak perlu agar memenuhi aturan.

4.4. Pengujian

Pengujian ini dilakukan dengan menggunakan tools *PHP Metrics* dan *white box.*

a. Pengujian dengan PHP *Metrics*

Tabel 2. Pengujian sebelum refactoring

No	Class	WMC	Class cycl	Bugs	Defects
1	Banner	15	8	0.37	0.64
2	I_berkala	18	10	0.54	0.78
3	I_merta	18	10	**0.55**	0.78
4	I_private	18	10	**0.53**	0.78
5	I_saat	18	10	**0.55**	0.78
6	P_informasi	18	5	**0.50**	0.78
7	Permohonan informasi	18	8	**1.27**	0.64

Tabel 3. Pengujian sesudah refactoring

No	Class	WMC	Class cycl	Bugs	Defects
1	Banner	15	4	0.37	0.60
2	I_berkala	19	7	0.53	0.70
3	I_merta	19	7	0.43	0.70
4	I_private	19	7	0.33	0.70
5	I_saat	18	7	0.55	0.70
6	P_informasi	20	4	0.48	0.62
7	Permohonan informasi	17	8	0.6	0.55

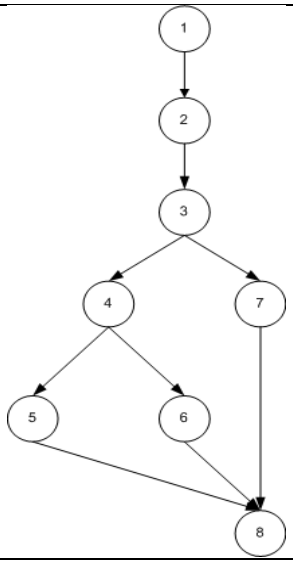
Dengan menggunakan *metode extract class* dan *extract* method, terlihat *class cycl* menurun, bug dalam sistem informasi juga dapat menurun pada beberapa kelas yang direfactoring. Dengan demikian, kode menjadi lebih modular dan mudah dipahami, sehingga memudahkan dalam mendeteksi dan memperbaiki bug. Pengurangan kompleksitas dan peningkatan keterbacaan kode mengurangi kemungkinan terjadinya kesalahan, yang pada akhirnya meningkatkan stabilitas dan kehandalan sistem informasi. Berikut merupakan data nilai bug yang didapatkan dengan menggunakan *tools PHPMetrics*.

b. Pengujian white box

Sistem diuji coba baik itu dari segi logika dan fungsi-fungsi agar layak untuk diimplementasikan. Adapun teknik pengujian sistem yang digunakan yaitu white box dengan menggunakan metode *Cyclomatic Complexity* (CC).

Tabel 4. Pengujian White Box

1) Tambah Data Banner

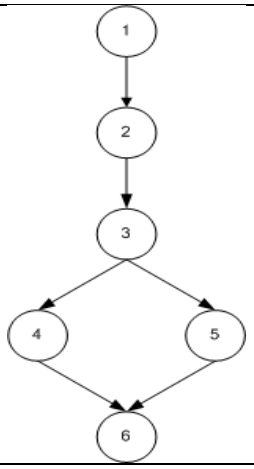


V(G) = E-N+2
V(G) = 9-8+2 -> Maka, memiliki 3 path
1-2-3-4-5-8
1-2-3-4-6-8
1-2-3-7-8

Ketiganya mempunyai kasus uji, seperti tabel dibawah ini:

Kasus Uji	*Expected Result*	Hasil Uji
1-2-3-4-5-8	*User* berhasil menyimpan data banner kedalam *database*	Sesuai
1-2-3-4-6-8	*User* gagal menyimpan data banner kedalam *database*	Sesuai
1-2-3-7-8	*User* gagal upload gambar banner	Sesuai

2) Hapus Data Banner



V(G) = E-N+2
V(G) = 6-6+2 -> Maka, memiliki 2 path
1-2-3-4-6
1-2-3-5-6

Keduanya mempunyai kasus uji, seperti tabel dibawah ini:

Kasus Uji	*Expected Result*	Hasil Uji
1-2-3-4-6	*User* berhasil menghapus data banner dari *database*	Sesuai
1-2-3-5-6	*User* gagal menghapus data banner dari *database*	Sesuai

Tabel 5. Kesimpulan pengujian White Box

Menu	Modul	Path
Menu Banner	Tambah Data Banner	Memiliki 3 Path
	Hapus Data Banner	Memiliki 2 Path
Menu I_berkala	Tambah Data I berkala	Memiliki 3 Path
	Edit Data I_berkala	Memiliki 6 Path
	Hapus Data I berkala	Memiliki 2 Path
	detail data I_berkala	Memiliki 1 Path
Menu I_merta	Tambah Data I merta	Memiliki 3 Path
	Edit Data I_merta	Memiliki 6 Path
	Hapus Data I_merta	Memiliki 2 Path
	detail data I_merta	Memiliki 1 Path
Menu I_private	Tambah Data I private	Memiliki 3 Path
	Edit Data I_private	Memiliki 6 Path
	Hapus Data I private	Memiliki 2 Path
	detail data I_private	Memiliki 1 Path
Menu I saat	Tambah Data I_saat	Memiliki 3 Path
	Edit Data I_saat	Memiliki 6 Path
	Hapus Data I_saat	Memiliki 2 Path
	detail data I_saat	Memiliki 1 Path
Menu P_informasi	Tambah Data P informasi	Memiliki 4 Path
Menu Permohonan _informasi	Tambah Data Permohonan_informasi	Memiliki 4 Path
	Hapus Data Permohonan_informasi	Memiliki 3 Path
	Balas Data Permohonan_informasi	Memiliki 2 Path

Tabel 6. Perbandingan antara cyclomatic complexity dan 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 mengganggu, prosedur tidak dapat diuji	Sangat Tinggi

Berdasarkan hasil pengujian yang sudah dilakukan diperoleh jumlah *region cyclomatic complexity*, dan *independent path* pada masing-masing menu sudah sesuai, sehingga dapat ditarik kesimpulan bahwa algoritma tersebut sudah benar. Dengan kata lain bahwa Sistem Dinas Kesehatan Kota Cimahi yang dibuat telah layak digunakan.

c. Pengujian *Integrity SQL Injection*

Pengujian integritas terhadap *SQL Injection* bertujuan untuk memastikan bahwa aplikasi tidak rentan terhadap serangan *SQL Injection*, yang dapat memungkinkan penyerang mengakses, memodifikasi, atau bahkan menghapus data dalam *database*.

Tabel 5. Pengujian Integrity SQL *Injection*

Sintak yang belum diperbaiki
`<?php defined('BASEPATH') or exit('No direct script access allowed'); class Auth_model extends CI_Model { public function cek_username($username) { $data = "SELECT * FROM user WHERE username = '$username'"; return $this->db->query($data)->row_array(); } public function get_password($username) {`

```
        $data       =       $this->db-
>get_where('user',    ['username'    =>
$username])->row_array();
        return $data['password'];
          $data = "SELECT * FROM user
WHERE username = '$username'";
          return            $this->db-
>query($data['password'])-
>row_array();
    }
    public                        function
userdata($username)
    {
        $data = "SELECT * FROM user
WHERE username = '$username'";
          return            $this->db-
>query($data)->row_array();
    }
}
```

Dilakukan analisis dan perbaikan pada beberapa poin diantaranya:

a. SQL Injection di function cek `cek_username($username)` dan `userdata($username)`. Sintak sebelum diperbaiki:

```
$data = "SELECT * FROM user WHERE
username = '$username'";
return        $this->db->query($data)-
>row_array();
```

Akan menjadi masalah jika `$username` berisi `'OR '1'='1`, itu akan mengembalikan semua data pengguna. Oleh karena itu digunakan *Active Records* agar lebih aman dan bersih untuk menghindari SQL *Injection*. Maka dibuat sintak seperti ini:

```
Public                            function
cek_username($username)
{
    return                    $this->db-
>get_where('user',    ['username'    =>
$username])->row_array();
}
```

b. Kesalahan pada function `get_password($username)`. Terdapat dua return *statement*, sehingga kode dibawahnya tidak pernah dieksekusi yang menyebabkan tetap adanya SQL *Injection Risk*. Sintak sebelum diperbaiki:

```
$data = $this->db->get_where('user',
['username' => $username])-
>row_array();
return $data['password'];
$data = "SELECT * FROM user WHERE
username = '$username'";
return $this->db-
>query($data['password'])-
>row_array();
```

Sintak tersebut harus diubah agar tidak menjadi sasaran SQL *Injection*. Sintak yang diubah seperti ini:

```
public function
get_password($username)
{
    $data = $this->db-
>get_where('user', ['username' =>
$username])->row_array();
    return $data ? $data['password'] :
null;
}
```

Untuk lebih detailnya sintak yang sudah diperbaiki adalah sebagai berikut:

```
<?php
defined('BASEPATH') or exit('No direct
script access allowed');

class Auth_model extends CI_Model
{
    public                        function
cek_username($username)
    {
        return                $this->db-
>get_where('user',    ['username'    =>
$username])->row_array();
    }

    public                        function
get_password($username)
    {
        $data       =       $this->db-
>get_where('user',    ['username'    =>
$username])->row_array();
        return          $data          ?
$data['password'] : null;
    }

    Public                        function
userdata($username)
    {
        return                $this->db-
>get_where('user',    ['username'    =>
$username])->row_array();
    }
}
```

5. KESIMPULAN DAN SARAN

Hasil pengujian pada sistem informasi yang telah direfactoring menunjukkan peningkatan. Refactoring dengan metode extract class dan extract method berhasil membuat metode dalam kelas menjadi lebih spesifik dan saling terkait. Kode yang sebelumnya belum terintegrasi keamanannya, menjadi lebih aman. Struktur kode yang lebih baik ini juga mempermudah proses deteksi dan perbaikan bug, yang mengarah pada pengurangan jumlah bug dan peningkatan performa serta stabilitas sistem.

Berdasarkan hasil penelitian dan pengujian yang telah dilakukan, disarankan untuk terus mengimplementasikan teknik refactoring secara

berkala guna menjaga kualitas kode yang optimal. Pengurangan jumlah bug yang tercatat juga memperlihatkan bahwa struktur kode yang lebih baik membantu dalam mendeteksi dan memperbaiki kesalahan lebih efisien. Untuk pengembangan ke depan, penting untuk terus memantau dan mengevaluasi kualitas kode menggunakan metrik yang sama, serta mempertimbangkan penggunaan alat bantu otomatis untuk mendeteksi potensi *code smell* lebih dini.

DAFTAR PUSTAKA

[1] F. N. Hasanah, *Buku Ajar Rekayasa Perangkat Lunak*. 2020. doi: 10.21070/2020/978-623-6833-89-6.

[2] M. B. I. T. Organizations, "An Approach to Software Maintenance : A Case Study," vol. 30, no. 5, pp. 603–630, 2020, doi: 10.1142/S0218194020500217.

[3] ZipDo, "Essential Test Automation Statistics In 2023." https://zipdo.co/statistics/test-automation/ (accessed Nov. 28, 2023).

[4] A. Mulyanto, "PENGUJIAN SISTEM INFORMASI AKADEMIK MENGGUNAKAN MCCALL ' S SOFTWARE QUALITY FRAMEWORK," vol. 1, no. 1, pp. 47–57, 2016.

[5] P. V. Nistala, K. V. Nori, S. Natarajan, N. R. Zope, and A. Kumar, *Quality management and Software Product Quality Engineering*. Elsevier Inc., 2016. doi: 10.1016/B978-0-12-802301-3.00006-5.

[6] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, "Code smells and refactoring: A tertiary systematic review of challenges and observations," *J. Syst. Softw.*, vol. 167, 2020, doi: 10.1016/j.jss.2020.110610.

[7] A. G. Approach, "Extract Class Refactoring Based on Cohesion and Coupling :," 2022.

[8] H. Fredianto, "Optimalisasi Perangkat Lunak Menggunakan Metode Refactoring," *J. Syntax Admiration*, vol. 2, no. 10, pp. 1885–1902, 2021, doi: 10.46799/jsa.v2i10.326.

[9] A. Kova *et al.*, "Automatic detection of Long Method and God Class code smells through neural source code embeddings," vol. 204, no. May, 2022, doi: 10.1016/j.eswa.2022.117607.

[10] U. Azadi, "Automation of duplicate code detection and refactoring Relatore : Prof . ssa Francesca Arcelli Fontana," no. April, 2020, doi: 10.13140/RG.2.2.22081.51043.

[11] S. Charalampidou, A. Ampatzoglou, and A. Chatzigeorgiou, "Identifying Extract Method Refactoring Opportunities based on Functional Relevance," vol. 5589, no. c, pp. 1–22, 2016, doi: 10.1109/TSE.2016.2645572.

[12] F. Tian, T. Wang, C. Wang, and M. A. Babar, "The impact of traceability on software maintenance and evolution : A mapping study," no. July, pp. 1–31, 2021, doi: 10.1002/smr.2374.

[13] A. Kaur and M. Kaur, "Analysis of Code Refactoring Impact on Software Quality," *MATEC Web Conf.*, vol. 57, 2016, doi: 10.1051/matecconf/20165702012.

[14] M. Mohan and D. Greer, "A survey of search-based refactoring for software maintenance," *J. Softw. Eng. Res. Dev.*, vol. 6, no. 1, 2018, doi: 10.1186/s40411-018-0046-4.

[15] "MetricForOOD_ChidamberKemerer94.pdf."

[16] PhpMetrics, "Metrics." https://phpmetrics.github.io/website/metrics/#cyclomatic-complexity-number-and-weighted-method-count (accessed Aug. 12, 2024).