Metode *Refactoring* Untuk Meningkatkan Kualitas *Maintainability* Pada Sistem Informasi Puskesmas

Ria Andriani[1], Puspita Nurul Sabrina[2], Herdi Ashaury[3]
[1,2,3]Fakultas Sains dan Informatika, Universitas Jenderal Achmad Yani, Cimahi, Indonesia
Email : riaandriani20@if.unjani.ac.id, puspita.sabrina@lecture.unjani.ac.id, herdi.ashaury@lecture.unjani.ac.id

Abstrak− Kemajuan teknologi informasi yang pesat telah mendorong pengembangan sistem informasi yang lebih kompleks, termasuk dalam sektor kesehatan. Sistem informasi puskesmas yang berfungsi untuk mengelola data pasien, informasi pegawai, dan layanan kesehatan memerlukan kualitas perangkat lunak yang tinggi agar mudah dipelihara dan dikembangkan. Salah satu tantangan dalam pengembangan sistem ini adalah menjaga *maintainability* kode, terutama dala menghadapi kompleksitas yang semakin meningkat dan munculnya *code smell.* Penelitian ini bertujuan untuk meningkatkan *maintainability* sistem informasi puskesmas melalui penerapan teknik *refactoring* serta mengevaluasi dampaknya terhadap metrik kualitas kode seperti kompleksitas siklomatik, ukuran kelas, dan penggunaan metode. Penelitian ini dilakukan melalui beberapa tahapan, termasuk identifikasi masalah, analisis kode sumber, penentuan ruang lingkup *refactoring*, pengukuran kualitas kode sebelum dan sesudah *refactoring*, implementasi *refactoring*, validasi hasil, serta evaluasi keseluruhan proses. Pengukuran kualitas kode menggunakan alat PHP Metrics untuk membandingkan data awal dengan hasil setelah *refactoring*. Hasil penelitian menunjukkan bahwa pengujian kualitas perangkat lunak memberikan peningkatan yang signifikan melalui perbaikan pada *Object Oriented Metrics.* Kompleksitas siklomatik rata-rata mengalami penurunan sebesar 41,25%, sedangkan Weighted Metods per Class (WMC) rata-rata berkurang sebesar 2,41%, dan efisiensi penggunaan metode meningkat. Analisis terhadap struktur kode mengidentifikasi kerentanan dan memberikan indikasi adanya *code smell* melalui pengujian menggunakan *Object Oriented Metrics.* Selanjutnya, dilakukan refactoring untuk memperbaiki struktur internal perangkat lunak dengan tujuan meningkatkan *maintainability* kode, mengurangi kompleksitas, meningkatkan keterbacaan, dan meminimalkan risiko kesalahan. Dampak dari refactoring ini secara langsung mendukung kelancaran operasional puskesmas dalam jangka panjang, memperkuat kehandalan sistem untuk mendukung kebutuhan pengelolaan data kesehatan yang lebih baik.

Kata Kunci: *Code smell*, Kompleksitas Kode, *Maintainability*, PHP Metrics, *Refactoring*, Sistem Informasi Puskesmas

Abstract− *The rapid advancement of information technology has encouraged the development of more complex information systems, including in the health sector. Health center information systems that function to manage patient data, employee information, and health services require high software quality to be easily maintained and developed. One of the challenges in developing this system is maintaining code maintainability, especially in the face of increasing complexity and the emergence of code smell. This research aims to improve the maintainability of the puskesmas information system through the application of refactoring techniques and evaluate its impact on code quality metrics such as cyclomatic complexity, class size, and method usage. This research was conducted through several stages, including problem identification, source code analysis, refactoring scope determination, code quality measurement before and after refactoring, refactoring implementation, result validation, and overall process evaluation. Code quality measurement uses the PHP Metrics tool to compare the initial data with the results after refactoring. The results show that software quality testing provides significant improvement through improvements in Object Oriented Metrics. The average cyclomatic complexity decreased by 41,25%, while the Weighted Methods per Class (WMC) decreased by 2,41% and method usage efficiency increased. Analysis of the code structure identified vulnerabilities and provided indications of code smell through testing using Object Oriented Metrics. Subsequently, refactoring was performed to improve the internal structure of the software with the aim of increasing code maintainability, reducing complexity, improving readability, and minimizing the risk of errors. The impact of this refactoring directly supports the smooth operation of the health center in the long run, strengthening the reliability of the system to support the need for better health data management.*

Keywords: *Code smell, Code Complexity, Maintainability, PHP Metrics, Refactoring, Health Center Information System*

1. PENDAHULUAN

Kemajuan teknologi perangkat lunak semakin merambah dan memakan biaya lebih besar karena tuntutan yang berubah dengan cepat dan penemuan teknologi baru. Selain itu, dalam proses pengembangan atau pemeliharaan, tidak cukup hanya menjamin keamanan perangkat lunak melainkan penting juga untuk menemukan metode yang lebih canggih dalam memastikan kualitas perangkat lunak tersebut. Perkembangan teknologi informasi terus bergerak menuju tingkat yang lebih tinggi dibandingkan dengan masa lalu yang tentunya masih memiliki banyak kelemahan dan kekurangan.

Teknologi merupakan alat untuk menyediakan kebutuhan manusia dalam menjalani hidup, dan pemanfaatan teknologi sangat bermanfaat bagi perkembangan manusia, sehingga menciptakan nilai-nilai baru didalam kehidupan bermasyarakat[1]. *Refactoring* dilakukan untuk pengembangan dan pemeliharaan perangkat lunak dan meningkatkan kualitas kode program yang sudah ada sebelumnya[2]. Aspek kualitas perangkat lunak

memiliki peran penting dalam proses pengembangan, evaluasi kualitas perangkat lunak tidak hanya memperhatikan hasil akhir produk tetapi juga fokus pada kualitas selama tahap pengembangan perangkat lunak[3]. Untuk menjamin kualitas perangkat lunak, penting untuk melakukan pengukuran yang terkait dengan berbagai aspek perangkat lunak tersebut. Sistem informasi puskesmas memiliki kemampuan untuk mengelola berbagai aspek penting, termasuk data pendaftaran pasien, informasi pegawai, pengelolaan pengaduan, dan penyediaan layanan kesehatan secara efisien. Salah satu faktor yang mempengaruhi *maintainability* adalah kualitas perangkat lunak yang digunakan. Namun, tekonologi perangkat lunak yang lebih tua seringkali memiliki struktur kode yang rumit dan tidak terorganisir, menyebabkan kesulitan dalam pemeliharaan dan pengembangan lebih lanjut. Dalam banyak kasus, sistem ini tidak dirancang dengan baik untuk beradaptasi dengan perubahan kebutuhan pengguna dan perkembangan teknologi. Ini mengakibatkan peningkatan biaya pemeliharaan dan risiko kesalahan yang lebih tinggi, serta mengurangi efisiensi dalam pengembangan.

Dalam pengembangan sistem informasi berbasis web, salah satu hal yang perlu diperhatikan adalah kualitas kode. Kode yang berkualitas akan membuat sistem menjadi lebih mudah dibaca, dipahami, dan dirawat. Sebaliknya, kode yang berkualitas rendah akan membuat sistem menjadi lebih sulit dibaca, dipahami, dan dirawat. PHP Metrics adalah alat yang digunakan untuk mengukur berbagai aspek kualitas kode, seperti kompleksitas siklomatik, ukuran kelas, dan lain-lain. Dengan menggunakan PHP Metrics tidak mendeteksi *code smell* secara langsung, namun menyediakan sejumlah metrik untuk membantu mengidentifikasi masalah yang mungkin mengindikasikan *code smell*. PHP Metrics berfokus pada pengukuran kualitas kode berdasarkan kemampuan pemeliharaan, kompleksitas, dan lain-lain.

Refactoring adalah proses mengubah struktur internal perangkat lunak tanpa mempengaruhi fungsionalitas perangkat lunak. *Refactoring* adalah salah satu jenis modifikasi perangkat lunak yang bertujuan untuk meningkatkan kualitas perangkat lunak setelah dilakukan modifikasi, koreksi, penambahan, dan perubahan lainnya selama penggunaan[4]. *Code smell* menunjukkan adanya masalah pada kode yang perlu di*refactoring*. Hal ini merupakan teknik untuk memodifikasi kode sumber agar lebih mudah dibaca dan dipelihara dengan menghilangkan *code smell*. *Refactoring* digunakan untuk meningkatkan kualitas pperangkat lunak dengan mengurangi kompleksitas[5].

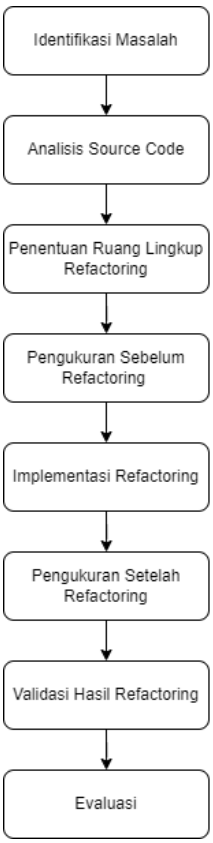
Salah satu dampak yang timbul akibat adanya *code smell* adalah menurunnya kemampuan untuk memelihara kode atau *maintainability*. Kode yang sulit dibaca, dipahami, dan diubah akan menyulitkan proses pemeliharaan dan pengembangan lanjutan. Rendahnya *maintainability* dapat menyebabkan peningkatan biaya pemeliharaan, risiko kesalahan yang lebih tinggi, dan penurunan efisiensi dalam pengembangan perangkat lunak. Oleh karena itu, mengidentifikasi dan mengatasi *code smell* adalah langkah penting untuk meningkatkan *maintainability* perangkat lunak.

Untuk meningkatkan *maintainability* sistem informasi puskesmas, penting untuk menggunakan berbagai alat yang dapat mengukur dan mengevaluasi berbagai aspek kinerja perangkat lunak termasuk PHP Metrics. Penelitian ini menekankan keunikan pada penerapan teknik *refactoring* pada sistem informasi puskesmas, yang merupakan area yang sangat penting dalam pelayanan publik, khususnya dibidang kesehatan. Mengingat peran puskesmas yang sangat penting sebagai fasilitas kesehatan tingkat pertama yang langsung melayani masyarakat, peningkatan kualitas *maintainability* sistem ini akan secara langsung mendukung kelancaran operasional puskesmas, mempercepat proses pelayanan kesehatan, dan memberikan manfaat yang lebih besar bagi petugas kesehatan serta masyarakat yang dilayani.

2. METODE PENELITIAN

2.1 Tahapan Penelitian

Berikut adalah tahapan penelitian yang dilakukan untuk meningkatkan kualitas *maintainability* dapat dilihat pada Gambar 1.



Gambar 1. Alur Penelitian

1. Identifikasi Masalah

Pada tahap ini, dilakukan identifikasi terhadap masalah yang ada dalam sistem informasi puskesmas, khususnya yang berkaitan dengan faktor pemeliharaan. Masalah yang umum mencakup kode yang sulit dipahami, kompleksitas yang tinggi, dan kesulitan dalam menambahkan fitur baru atau memperbaiki bug.

2. Analisis Source Code

Pada tahap ini, source code yang ada dianalisis untuk mengidentifikasi bagian-bagian yang membutuhkan perbaikan. Analisis ini dapat mencakup pemeriksaan manual kode, penggunaan alat bantu analisisi statis, dan melihat hasil pengujian sebelumnya.

3. Penentuan Ruang Lingkup *Refactoring*

Pada tahap ini menentukan bagian mana dari kode yang akan di *refactoring*. Ruang lingkup *refactoring* mencakup area spesifik yang membutuhkan perbaikan tanpa mengubah perilaku eksternal dari sistem.

4. Pengukuran Sebelum *Refactoring*

Pada tahap ini, pengukuran kualitas kode dilakukan sebelum *refactoring* untuk mendapatkan *baseline.* Alat seperti PHP Metrics digunakan untuk mengukur metrik-metrik seperti kompleksitas siklomatik, penggunaan metode, dan lain-lain. Data ini akan digunakan sebagai acuan untuk mengevaluasi hasil *refactoring* nanti.

5. Implementasi *Refactoring*

Pada tahap ini, *refactoring* dilakukan berdasarkan hasil analisis dan ruang lingkup yang telah ditentukan. *Refactoring* meliputi perubahan struktural pada kode tanpa mengubah fungsionalitas eksternal. Contohnya termasuk memecah metode yang besar, mengurangi duplikasi kode, dan memperbaiki arsitektur.

6. Pengukuran Setelah *Refactoring*

Setelah *refactoring* dilakukan, pengukuran kualitas kode dilakukan kembali untuk melihat perubahannya. Pengukuran ini menggunakan alat yang sala dan metrik yang sama dengan pengukuran sebelum *refactoring*. Hasil pengukuran ini akan dibandingkan dengan baseline untuk mengevaluasi efek *refactoring*.

7. Validasi Hasil *Refactoring*

Pada tahap ini validasi dilakukan untuk memastikan bahwa *refactoring* yang dilakukan tidak hanya meningkatkan metrik kualitas kode, tetapi juga tidak mengganggu fungsionalitas sistem. Validasi juga memastikan bahwa kode yang direfactor bekerja dengan baik dan tidak menimbulkan bug baru.

8. Evaluasi

Tahap terakhir adalah evaluasi keseluruhan proses *refactoring*. Evaluasi ini mencakup penilaian apakah tujuan dari *refactoring* telah tercapai, seperti peningkatan *maintainability*, penurunan kompleksitas, dan lain-lain. Selain itu, evaluasi juga melihat dampak keseluruhan dari *refactoring* terhadap pengembangan dan pemeliharaan sistem jangka panjang.

2.2 Model McCall

Model McCall merupakan salah satu model yang direkomendasikan untuk mengukur kualitas perangkat lunak. Berdasarkan prinsip McCall, pengukuran kualitas dilakukan dengan pendekatan hirarkis, dimana atribut tingkat atas (*high-level attribute*) disebut faktor, sementara atribut tingkat bawah (*low-level attribute*) disebut kriteria[6]. Faktor merupakan atribut kualitas produk dari sudut pandang pengguna, sedangkan kriteria adalah parameter kualitas produk yang diukur dari sudut pandang perangkat lunak itu sendiri.

Model faktor McCall mengelompokkan semua kebutuhan perangkat lunak ke dalam sebelas faktor kualitas[7]. Faktor-faktor tersebut meliputi *correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, dan interoperability.* Faktor-faktor ini dibagi menjadi tiga kelompok utama: *Product Operation* (*Correctness, Reliability, Usability, Integrity,* dan *Efficiency*), *Product Revision* (*Testability, Flexibility,* dan *Maintainability*) dan *Product Transition* (*Interoperability, Portability* dan *Reusability*)[8]. Dalam model ini, kriteria-kriteria tersebut dikelompokkan ke dalam faktor-faktor tertentu, dan untuk setiap faktor ditetapkan kriteria serta metrik yang sesuai.

2.2.1 Maintainability

Maintainability mengacu pada kemampuan untuk memperbaiki, memodifikasi, dan memelihara perangkat lunak dengan mudah dan efisien. Nilai *maintainability* suatu program dapat diukur dengan menggunakan metrik perangkat lunak untuk menilai sejauh mana perangkat lunak tersebut mudah dipelihara dan dimodifikasi[9]. Berdasarkan metode McCall, *maintainability* dibagi menjadi 5 sub attribute atau karakteristik. Karakteristik tersebut, yaitu *Conciseness, Self Descriptive, Modularity, Simplicity, dan Consistency*[10].

2.3 Object Oriented Metrics

Metrik pemrograman berorientasi objek adalah aspek penting yang harus diperhatikan. Metrik berfungsi sebagai seperangkat standar yang digunakan untuk mengukur efektivitas teknik analisis berorientasi objek dalam perancangan suatu sistem. Berbagai metrik telah diusulkan untuk sistem berorientasi objek. Metrik ini mengukur struktur prinsip yang jika tidak dirancang dengan baik dapat berdampak negatif pada kualitas desain dan kode. Metrik berorientasi objek yang ada umumnya diterapkan pada konsep kelas, kopling, dan pewarisan[11].

2.4 PHPMetrics

PhpMetrics adalah aplikasi perhitungan metrik dengan menggunakan bahasa pemrograman PHP. Aplikasi open source ini dikembangkan oleh Jean-François Lépine dan mencakup berbagai metrik untuk mengukur kompleksitas aplikasi, jumlah kode atau variabel tertentu, data terkait pemrograman berorientasi objek serta tingkat pemeliharaan[12].

2.4.1 *Weighted Method per Class* (WMC)

WMC adalah metrik yang berfokus pada kompleksitas method dalam suatu kelas. Kompleksitas ini dapat dihitung dengan rumus kompleksitas siklomatik. Semakin tinggi nilai WMC maka semakin rendah kualitas aplikasinya[13]. WMC menjumlahkan semua nilai kompleksitas ke semua method didalam sebuah class seperti pada rumus dibawah ini:

$$WMC = \sum_{i=1}^{n} ci$$

Dimana :
N : Jumlah method dalam sebuah class
Ci : Cyclomatic Complexity dari sebuah method

2.4.2 Siklomatik Kompleksitas

Siklomatik Kompleksitas atau *Cyclomatic Complexity* yang juga dikenal sebagai Conditional Complexity adalah alat ukur yang digunakan untuk menilai tingkat kompleksitas program dengan cara menghitung jumlah jalur independen yang dapat dilalui dalam kode sumber. Dikembangkan oleh Thomas J. McCabe, Sr. pada tahun 1976, siklomatik kompleksitas diterapkan dalam semua tahap siklus hidup perangkat lunak dimulai dari fase desain untuk memastikan perangkat lunak tetap dapat dipercaya, mudah diuji, dan terorganisir dengan baik[14].

Dalam menghitung Cyclomatic Complexity (CC) dalam method, pernyataan disimbolkan dengan node dan aliran program dilambangkan dengan edge. Nilai Cyclomatic Complexity (CC) dapat dilakukan perhitungan dengan menggunakan rumus[15]:

$$\mathbf{V(G) = E - N + 2}$$

V(G) : Cyclomatic Complexity

E : Total Jumlah Edge

N : Total Jumlah Node

Metrik ini dapat digunakan untuk menilai dan mengelompokkan tingkat kompleksitas dari kode program. Kode dengan siklomatik kompleksitas tinggi biasanya menunjukkan bahwa kode tersebut memiliki banyak jalur keputusan yang dapat mengindikasikan kebutuhan untuk refactoring agar kode lebih mudah dipahami dan dikelola.

2.5 Refactoring

Refactoring adalah proses mengubah struktur internal dari kode tanpa mengubah fungsionalitas eksternalnya[16]. Refactoring digunakan untuk melakukan perubahan yang membuat sistem lebih mudah untuk dipahami, dikembangkan, dan dipelihara[17]. Prinsip dasar refactoring adalah mengatur ulang kelas, variable, dan metode kedalam hierarki kelas untuk memfasilitasi adaptasi dan perluasan dimasa depan, menjadikan kode sumber lebih terstruktur, lebih mudah dibaca dan dipahami[18].

2.6 Extract Class

Extract class adalah teknik refactoring dalam pengembangan perangkat lunak yang digunakan untuk memindahkan sekelompok metode dari satu kelas ke kelas baru. Teknik ini diterapkan saat ada sekelompok metode yang berinteraksi erat dan membentuk fitur atau tanggung jawab yang terpisah dari kelas utama. Dengan memindahkan metode-metode ini ke kelas baru, kita dapat meningkatkan modularitas dan keterbacaan kode, serta mempermudah penggunaan ulang kode di tempat lain[19].

2.7 Extract Method

Extract method adalah teknik refactoring yang bertujuan untuk memindahkan sebagian kode dari satu metode ke metode yang berbeda. Dengan teknik ini, tanggung jawab yang beragam dapat dipisahkan menjadi metode-metode yang lebih kecil dan spesifik, sehingga meningkatkan pemahaman dan kemudahan pemeliharaan kode program[20].

2.8 Code Smell

Code smell adalah istilah dalam rekayasa perangkat lunak yang merujuk pada indikasi potensial adanya masalah dalam desain atau implementasi kode. Pertama kali diperkenalkan oleh Kent Beck, *code smell* tidak langsung menunjukkan adanya bug atau kesalahan dalam kode, melainkan menjadi petunjuk bahwa struktur atau desain kode dapat ditingkatkan. Mendeteksi bau kode memungkinkan pengembang untuk secara proaktif meningkatkan kode dan mencegah terjadinya masalah yang lebih serius di masa mendatang[21]. Beberapa literatur mengidentifikasi berbagai jenis code smell yang meliputi:

1. Duplicate Code

   Duplicate code terjadi ketika satu bagian kode yang identik atau sangat mirip muncul diberbagai lokasi. Hal ini dapat menambah beban kerja saat melakukan perubahan, serta meningkatkan risiko bug jika satu salinan kode diperbaharui sementara yang lainnya tidak[22].

2. Long Method

   Long method adalah *code smell* yang terjadi ketika sebuah metode memiliki Panjang yang sangat besar. Metode ini cenderung kompleks dan sulit dipahami karena mengimplementasikan berbagai fungsionalitas dengan terlalu banyak baris kode[23].

3. Data Clumps

Data clumps atau gumpalan data merujuk pada sekelompok variabel yang sering muncul bersama di berbagai bagian dalam kode sumber, kehadiran data clumps menunjukkan potensi untuk menciptakan struktur data baru yang lebih terorganisir[24]. Karena data clumps tersebar di seluruh proyek perangkat lunak, mendeteksinya menjadi sulit bagi pengembang. Oleh karena itu, penting untuk mendeteksi dan merombak data clumps secara otomatis atau semi otomatis guna meningkatkan kualitas *maintainability* dari kode sumber.

4. Feature Envy

Feature envy terjadi ketika sebuah metode atau anggota dari suatu kelas lebih tertarik terhadap kelas lain daripada kelas tempat metode atau anggota tersebut didefinisikan[22]. Semakin tinggi penggabungan antar kelas, semakin tinggi jumlah kelas yang terpengaruh ketika perubahan dibuat pada sistem[25]. Perubahan kecil dalam sistem dapat menghasilkan serangkaian Panjang yang tidak diantisipasi perubahan dibanyak kelas. Oleh karena itu, saling ketergantungan kelas harus dijaga seminimal mungkin jika memungkinkan.

5. Conditional Complexity

Conditional complexity adalah salah satu jenis *code smell* yang terjadi ketika penggunaan struktur kondisional dalam kode menjadi terlalu rumit atau berlebihan. Hal ini terjadi ketika logika kondisional dalam sebuah fungsi atau blok kode menjadi sangat kompleks, sulit untuk dipahami, atau menyulitkan dalam pemeliharaan kode[17].

3. HASIL DAN PEMBAHASAN

3.1 Analisis Sistem Sebelum *Refactoring*

Analisis sistem sebelum *refactoring* terdiri dari tiga bagian yaitu pengumpulan data, analisis *code smell*, dan pengujian sistem informasi.

3.1.1 Pengumpulan Data

Pengumpulan data merupakan tahapan penting dalam penelitian ini yang bertujuan untuk memperoleh informasi yang akurat dan relevan mengenai *refactoring* kode. Metode pengumpulan data yang digunakan dalam penelitian ini meliputi pengujian sistem informasi dan analisis *code smell*. Pengujian sistem informasi dilakukan dengan menggunakan alat bantu, yaitu PHP Metrics untuk menguji berbagai metrik kualitas kode dalam hal struktur kode, kompleksitas, dan kualitas perangkat lunak secara keseluruhan. PHP Metrics membantu mengidentifikasi area kode yang memerlukan perbaikan untuk meningkatkan keterbacaan dan *maintainability*. Analisis *code smell* dilakukan dengan memeriksa langsung kode sumber dan mengidentifikasi *code smell* yang ada. *Code smell* merupakan indikasi masalah mendasar pada kode yang mungkin perlu dikerjakan ulang. Contohnya termasuk metode yang terlalu panjang, kode duplikat, dan ketergantungan yang berlebihan antar kelas atau modul.

3.1.2 Pengujian Sebelum *Refactoring*

Pengujian sistem informasi dilakukan menggunakan alat bantu yaitu PHP Metrics. Pengujian ini dilakukan untuk mengukur berbagai metrik kualitas kode seperti *Weighted Methos per Class (WMC), Class Cyclomatic*, dan *Max Method Cyclomatic.* Selain itu pengujian ini membantu mengidentifikasi area dalam kode yang memerlukan perbaikan guna meningaktkan keterbacaan, dan pemeliharaan.

3.1.3 Analisa *Code smell*

Code smell adalah tanda-tanda keputusan desain atau implementasi yang kurang optimal dalam kode sumber. Meskipun tidak selalu ada bug yang sebenarnya, *code smell* sering kali membuat kode lebih sulit dipelihara, lebih rentan kesalahan, dan lebih sulit diubah. Contoh *code smell* mencakup metode yang terlalu panjang, kompleksitas logika yang tinggi, penggunaan variabel global, dan lain-lain. Analisis *code smell* merupakan elemen penting dalam pengembangan sistem informasi. Mengidentifikasi dan memperbaiki kesalahan kode dapat membantu meningkatkan kualitas kode, sehingga lebih mudah untuk dipahami dan diuji. Hal ini membuat pemeliharaan sistem menjadi lebih efisien dan mengurangi risiko kesalahan saat melakukan perubahan. Oleh karena itu, memperhatikan *code smell* adalah bagian penting dari praktik terbaik dalam pengembangan perangkat lunak dan pemeliharaan sistem.

a. Duplicate Code

Duplicate code atau Kode duplikat terjadi ketika ada potongan kode yang hampir sama atau sama persis yang muncul di beberapa bagian perangkat lunak. Ini bisa disebabkan oleh beberapa faktor, seperti kurangnya pemahaman tentang cara yang efisien untuk menerapkan fungsi yang sama di berbagai bagian

program, kurangnya perencanaan yang baik, atau kurangnya pengetahuan tentang penerapan pola desain yang sesuai. Pada kelas 'admin' terindikasi adanya *code smell duplicate code* yaitu kedua metode 'TambahAdminPus' dan 'UpdateAdminPus' terdapat duplikasi kode. Pada kedua metode tersebut bisa disederhanakan dengan membuat metode baru yang menggabungkan logika berulang.

b. Long Method
Long method adalah kondisi di mana sebuah metode dalam program memiliki panjang yang berlebihan atau terlalu banyak variabel, parameter, dan kondisi yang dieksekusi dalam satu metode. Kondisi ini dapat menyebabkan program sulit dipahami, sulit dipelihara, dan sulit untuk dikembangkan atau diubah dikemudian hari. Oleh karena itu, deteksi long method sangat penting untuk meningkatkan kualitas perangkat lunak dan memudahkan pengembangan dan pemeliharaan program. Pada kelas 'Auth' terindikasi adanya *code smell* long method dimana metode metode 'proseslogin', 'reset password', dan 'update password' memiliki banyak logika yang bisa dipecah menjadi bagian-bagian yang lebih kecil untuk meningkatkan kejelasan dan pemeliharaan.

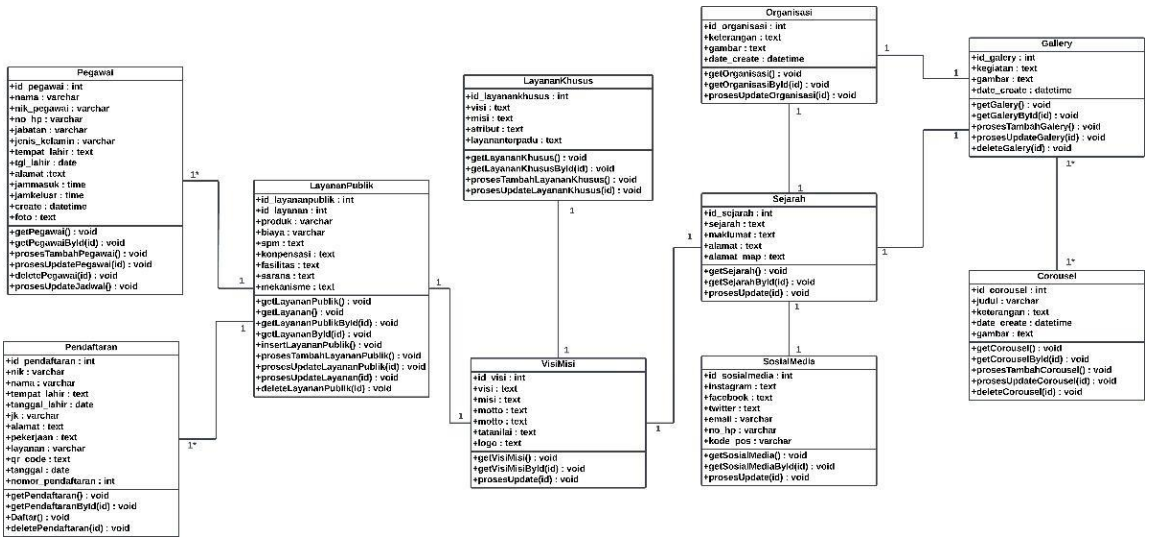
c. Data Clumps
Ketika dua atau lebih data atau primitive selalu dideklarasikan bersama dan saling bergantung. Ini menunjukkan adanya data clumps dalam source code. Data clumps biasanya terdiri dari pasangan nilai primitive yang sebenarnya bisa diubah strukturnya dengan menggabungkannya menjadi sebuah objek. Pada kelas 'PegawaiModel' data clumps dapat diidentifikasi dengan adanya sekelompok parameter yang sama digunakan berulang kali di berbagai metode. Contoh utama dapat dilihat pada metode prosesTambahPegawai dan prosesUpdatePegawai. Kedua metode ini menggunakan sejumlah besar parameter untuk menyimpan informasi pegawai.

d. Feature Envy
Pada kelas 'LoginModel' terindikasi adanya *code smell* feature envy. Feature envy terjadi ketika sebuah metode disatu kelas cenderung menggunakan data atau metode dari kelas lain secara berlebihan. Dalam kasus ini, beberapa metode di 'LoginModel' berhubungan dengan pengelolaan data pengguna (misalnya, mendapatkan data admin, mengupdate data, dan lain-lain) yang cenderung membuat kelas ini memiliki tanggung jawab yang terlalu banyak. Pada metode 'deleteAdmin' metode ini tidak hanya menghapus data dari basis data tetapi juga mengelola file sistem. Ini menujukkan ketergantungan berlebihan pada logika admin dan file sistem yang membuatnya sulit untuk dikelola.

e. Conditional Complexity
Conditional complexity mengacu pada kompleksitas yang muncul dari banyaknya percabangan (if-else) yang kompleks atau dalam lapisan yang dalam. Hal ini dapat membuat kode sulit untuk dipahami, diuji, dan dikelola. Di dalam kelas 'Content', terdapat beberapa metode seperti WebData(), admin(), DetailWeb(), TambahBerita() yang menggunakan struktur kondisional untuk memeriksa keadaan sesi (session). Pemakaian berulang-if ini bisa menandakan bahwa ada potensi untuk meningkatkan conditional complexity. Metode ini memeriksa apakah pengguna telah login sebagai super admin dengan memanggil `'$this->session->userdata('login_super')'`. Jika benar, maka akan memuat berbagai tampilan. Jika tidak, akan mengarahkan ke halaman login. Kondisi ini muncul hampir setiap metode dalam kelas ini yang menyebabkan duplikasi logika dan conditional complexity.

3.2 Class Diagram

Class Diagram adalah jenis diagram struktur di UML yang menggambarkan secara detail struktur class, termasuk atribut, metode, dan hubungan antar objek. Fungsi diagram kelas antara lain untuk menggambarkan secara jelas struktur sistem, meningkatkan pemahaman tentang gambaran umum atau skema dari sebuah program, dan menyajikan gambaran umum sistem atau perangkat lunak serta hubungan-hubungan didalam sistem. Penggunaan diagram kelas pada sistem informasi kesehatan untuk mengelola informasi pelayanan dan pendaftaran online pada suatu puskesmas ditunjukkan pada Gambar 2.



Gambar 2. *Class Diagram*

3.3 Pengukuran menggunakan OO Metrics

Pada tahapan ini peneliti melakukan pengukuran Sistem Informasi Puskesmas dengan menggunakan OO Metrics yaitu PHP Metrics, terdapat 3 kriteria yang akan diukur yaitu, *Weighted Method per-Class (WMC), Class Cycl,* dan *Max Method Cycl*.

Tabel 1. Pengukuran OO Metrics

Class	WMC	Class cycl	Max Method cycl
Auth	14	7	3
Admin	21	16	7
Content	26	12	2
SuperAdmin	26	21	8
AdminModel	7	3	3
BeritaModel	12	7	4
FeedModel	17	10	5
KepuasanModel	6	4	3
LoginModel	22	8	6
LayananKhususModel	4	1	1
LayananPublikModel	17	9	7
OrganisasiModel	6	4	4
PegawaiModel	16	9	4
PendaftaranModel	25	13	5
SejarahModel	6	4	4
SosialMediaModel	3	1	1
VisiMisiModel	6	4	4

Variabel yang ditandai dengan warna kuning menandakan bahwa perhatian utama dalam pengujian ini akan difokuskan pada kelas-kelas tersebut, yang dipilih berdasarkan nilai Cyclomatic Complexity (CC) yang melebihi 10. Nilai CC yang sama dengan atau lebih dari 10 menunjukkan potensi masalah. Kelas-kelas ini akan diuji secara

terpisah untuk menilai adanya indikasi *code smell* yang mungkin mempengaruhi *maintainability* sistem. Dengan cara ini, penguji dapat mengidentifikasi dan menangani potensi masalah yang dapat berdampak pada kualitas dan pemeliharaan kode secara keseluruhan.

3.4 Pengujian

Pada tahap pengujian ini, perangkat lunak dan kesesuaiannya dengan tujuan yang telah direncanakan diuji. Tujuan pengujian ini adalah untuk mengidentifikasi celah, kesalahan, kekurangan, atau cacat, serta memastikan bahwa semua komponen dari unit sistem hingga fungsi aplikasi berfungsi sesuai dengan fungsionalitas utama aplikasi.

3.4.1 Pengujian Kualitas Kode

Pengujian kualitas kode melibatkan evaluasi berbagai aspek yang mempengaruhi pemeliharaan kode. Proses ini mencakup pemeriksaan struktur kode, kepatuhan terhadap standar pengkodean, dan deteksi *code smell*. Alat seperti phpmetics sangat berguna dalam proses ini, karena menyediakan metrik dan visualisasi yang membantu mengidentifikasi kompleksitas, duplikasi, dan area kode yang memerlukan perbaikan. Phpmetrics dapat memberikan wawasan tentang kualitas kode dengan menghasilkan laporan yang mencakup metrik seperti *maintainability*, cyclomatic complexity, dan lain-lain. Dengan alat ini, pengembang dapat lebih mudah mengukur dan meningkatkan kualitas kode mereka secara efektif.

Tabel 2. Pengujian Kualitas Kode Sebelum *Refactoring*

No	Class	Sebelum *Refactoring*		
		WMC	**Class cycl**	**Max Method cycl**
1	Auth	14	7	3
2	Admin	21	16	7
3	Content	26	12	2
4	LoginModel	22	8	6
5	PegawaiModel	16	9	4

Tabel 3. Pengujian Kualitas Kode Sesudah *Refactoring*

No	Class	Sesudah *Refactoring*		
		WMC	**Class cycl**	**Max Method cycl**
1	Auth	25	7	2
2	Admin	16	13	7
3	Content	20	5	2
4	LoginModel	22	8	6
5	PegawaiModel	9	3	2

Pada Tabel 2 dan Tabel 3 diperlihatkan perbandingan kualitas kode sebelum dan sesudah refactoring untuk beberapa kelas terpenting dalam sistem. Tabel 2 menunjukkan data sebelum dilakukan *refactoring*, dimana kelas Admin dan PegawaiModel memiliki nilai tertinggi pada beberapa metrik seperti *Weighted Methods per Class* (WMC), *Class Cyclomatic Complexity* (Class cycl), dan *Max Method Cyclomatic Complexity* (Max Method cycl). Nilai yang tinggi pada metrik ini menunjukkan bahwa kelas tersebut memiliki tingkat kompleksitas yang tinggi, sehingga dapat menyulitkan proses pemeliharaan dan pengembangan lebih lanjut.

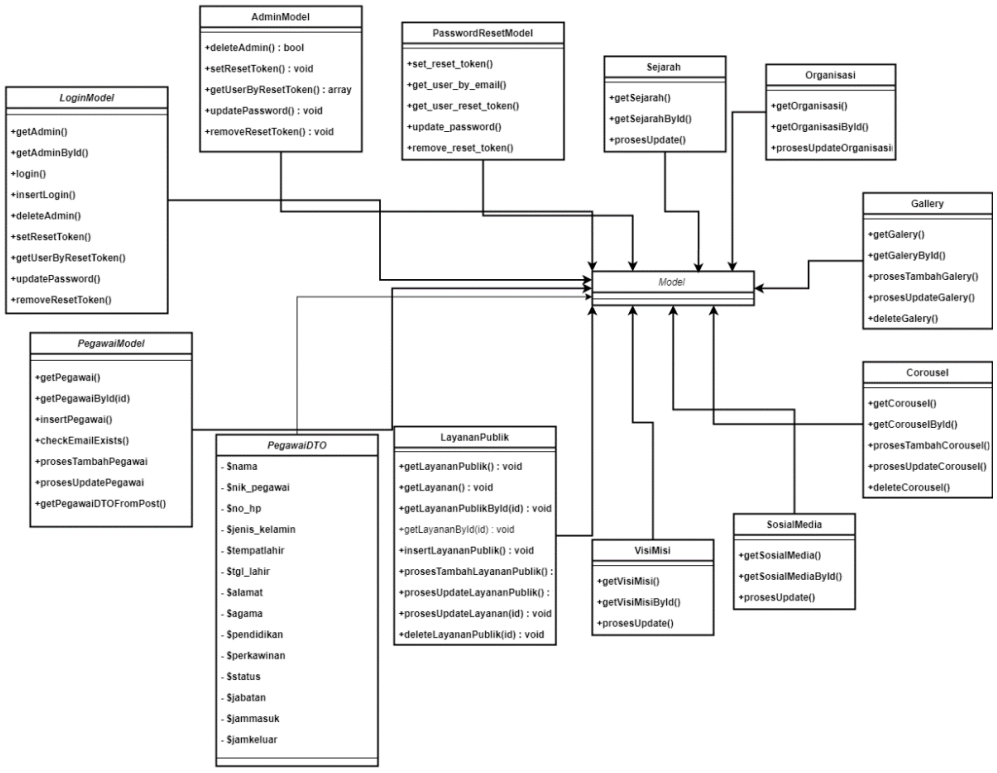
Setelah dilakukan *refactoring*, seperti yang ditunjukkan pada Tabel 3 terdapat peningkatan pada beberapa metrik. WMC mengalami peningkatan untuk beberapa kelas, seperti pada kelas Auth dan PegawaiModel. Peningkatan ini disebabkan oleh pemecahan fungsi menjadi metode yang lebih spesifik sehingga meningkatkan jumlah metode pada kelas tersebut. Namun, hal ini diimbangi dengan penurunan pada *Class Cyclomatic Complexity* dan *Max Method Cyclomatic Complexity*, yang menunjukkan bahwa kompleksitas ditingkat metode

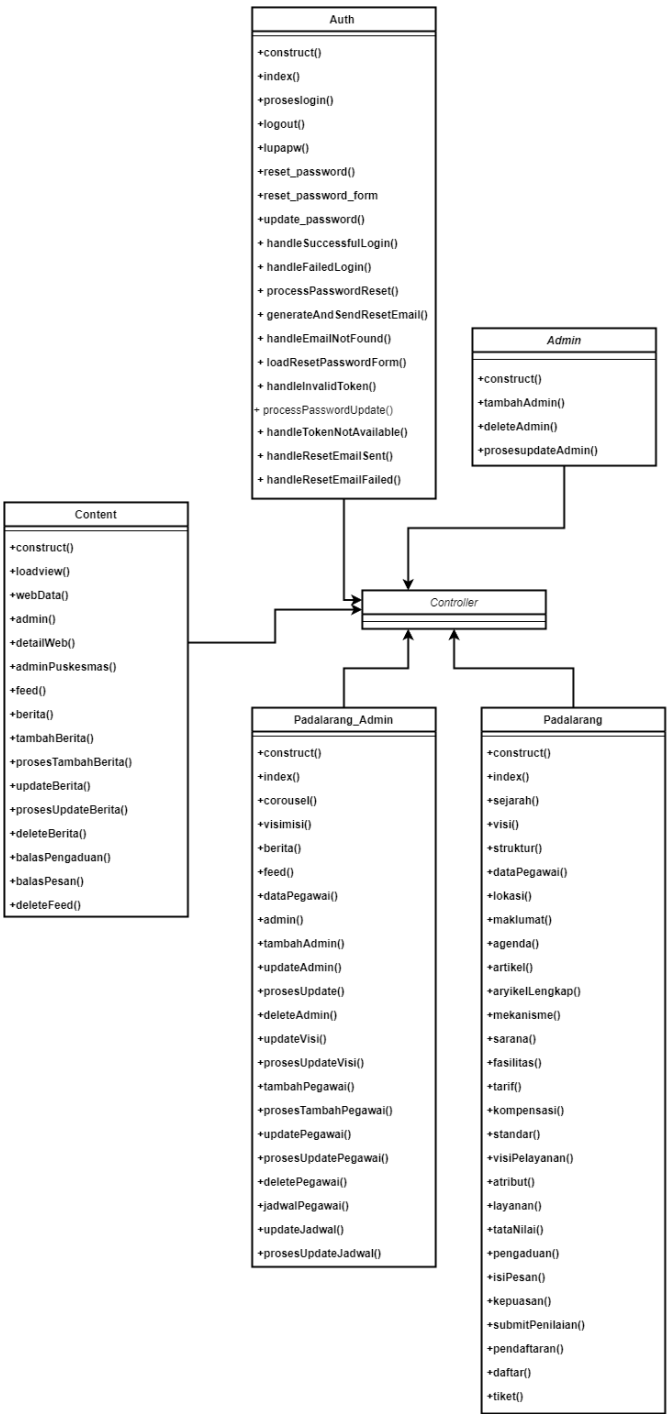
berkurang, membuat kode lebih mudah dipahami dan dipelihara. Dampak dari *refactoring* pada setiap metrik ini menunjukkan adanya perbaikan signifikan pada struktur internal perangkat lunak. Penurunana nilai *Class Cyclomatic Complexity* dan *Max Method Cyclomatic Complexity* mengindikasikan bahwa metode paling kompleks pada kelas-kelas tersebut menjadi lebih sederhana dan lebih mudah dikelola. Hal ini sejalan dengan tujuan *refactoring*, yaitu meningkatkan keterbacaan dan meminimalkan risiko kesalahan dalam pemeliharaan kode.

Keterkaitan antara metode penelitian dan hasil pembahasan sangat jelas, terutama dalam hal pengukuran kuantitatif sebelum dan sesudah *refactoring*. Setiap tahap metode peneltian, seperti identifikasi masalah, analisis kode, hingga penerapan *refactoring*, secara langsung mengarah pada hasil yang spesifik. Dengan menggunakan PHP Metrics sebagai alat pengukuran, dampak dari setiap perubahan dapat dinilai secara objektif. Penjelasan lebih rinci mengenai dampak setiap perbaikan, seperti penurunan kompleksitas ditingkat kelas dan metode, memberikan pemahaman yang lebih mendalam tentang bagaimana *refactoring* berkontirbusi pada peningkatan kualitas kode secara keseluruhan.

3.5 Class Diagram Baru

Class diagram saat ini pada Gambar 3 menunjukkan struktur dan hubungan antar kelas dalam sistem setelah *refactoring*. Diagram ini menunjukkan alokasi tanggung jawab kedalam kelas tertentu, seperti auth untuk autentikasi, admin untuk manajemen admin, dan konten untuk manajemen konten. Selain itu terdapat kelas-kelas yang menangani berbagai data terkait, seperti PegawaiModel, Organisasi, dan lainnya. Pendekatan ini meningkatkan modularitas, menyederhanakan pemeliharaan, dan memungkinkan pengembangan sistem yang lebih terstruktur.





Gambar 3. *Class Diagram Baru*

4. KESIMPULAN

Pada bagian akhir ini, peneliti akan menyampaikan kesimpulan berdasarkan hasil penelitian yang telah dilakukan. Data yang diperoleh melalui pengukuran dan pengolahan dengan PHP Metrics telah diklasifikasikan berdasarkan faktor *maintainability*. *Refactoring* yang dilakukan berhasil menunjukkan pola peningkatan kualitas *maintainability* yang konsisten. Penurunan kompleksitas, seperti *Weighted Methods per Class* (WMC) dan kompleksitas siklomatik pada tingkat kelas dan metode setelah refactoring, menunjukkan bahwa struktur dan fungsi kelas menjadi lebih teratur dan mudah dipahami. Ini mendukung hipotesis awal bahwa *refactoring* mampu mengurangi kompleksitas kode serta meningkatkan keterbacaan dan keteraturan struktur kode. Selain itu, berkurangnya jumlah *bug* dan *defect rate* mengindikasikan peningkatan keandalan dan kualitas kode secara keseluruhan yang mendukung tujuan penelitian untuk meningkatkan kualiatas perangkat lunak dalam hal

maintainability dan keandalan. Dengan menurunkan kompleksitas, *refactoring* juga meningkatkan kemudahan dalam pemehaman dan pemeliharaan (*understandability* dan *maintainability*) sistem secara keseluruhan.

Di samping itu, *refactoring* juga memberikan dampak positif terhadap reusabilitas (*reusability*) kode, di mana modul-modul yang telah disederhanakan dan dioptimalkan menjadi lebih mudah untuk digunakan kembali dalam pengembangan fitur baru atau proyek lainnya. Hasil analisis ini juga menunjukkan bahwa pengelolaan dependensi yang lebih baik serta pengorganisasian kode yang lebih modular membantu mengurangi risiko kerusakan saat melakukan perubahan atau pembaruan sistem. Secara keseluruhan, pendekatan refactoring yang sistematis terbukti efektif dalam meningkatkan *maintainability*, yang pada akhirnya dapat mengurangi biaya pemeliharaan jangka panjang dan memperpanjang umur sistem. Penenlitian ini menegaskan pentingnya penerapan *refactoring* dalam pengembangan perangkat lunak untuk menjaga kualitas dan keandalam kode seiring berkembangnya sistem.

Hasil analisis ini juga menunjukkan bahwa pengelolaan dependensi yang lebih baik serta pengorganisasian kode yang lebih modular membantu mengurangi risiko kerusakan saat melakukan perubahan atau pembaruan sistem. Secara keseluruhan, pendekatan *refactoring* yang sistematis terbukti efektif dalam meningkatkan *maintainability*, yang pada akhirnya dapat mengurangi biaya pemeliharaan jangka panjang dan memperpanjang umur sistem. Penelitian ini menegaskan pentingnya penerapan *refactoring* dalam pengembangan perangkat lunak untuk menjaga kualitas dan keandalan kode seiring berkembangnya sistem.

UCAPAN TERIMAKASIH

Pada kesempatan ini peneliti ingin mengucapkan terima kasih kepada seluruh pihak yang terlibat pada penelitian ini serta terimakasih kepada seluruh Dosen Informatika Universitas Jenderal Achmad Yani atas ilmu dan wawasan yang telah diberikan, yang sangat berperan penting dalam penyelesaian penelitian ini.

REFERENCES

[1] S. A. Putri and S. Syamsir, "Efektifitas Penyelenggaraan E-Puskesmas Di Puskesmas Lubuk Buaya Kota Padang ¬," *JISIP (Jurnal Ilmu Sos. dan Pendidikan)*, vol. 5, no. 2, 2021, doi: 10.58258/jisip.v5i2.2031.

[2] A. A. B. Baqais and M. Alshayeb, "Automatic software refactoring: a systematic literature review," *Softw. Qual. J.*, vol. 28, no. 2, pp. 459–502, 2020, doi: 10.1007/s11219-019-09477-y.

[3] R. K. Hapsari and M. J. Husen, "Estimasi Kualitas Perangkat Lunak Berdasarkan," *Semin. Nas. Sains dan Teknol. Terap. III*, pp. 425–434, 2015.

[4] R. Nindyasari, "METODE NON- HEURISTIC UNTUK DETEKSI REFACTORING NON-SOURCE CODE (SYSTEMATIC LITERATURE REVIEW)," vol. 6, no. 2, pp. 361–366, 2015.

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

[6] S. Hartini, "Metode Mc Call pada Pengujian Correctness dan Usability Sistem Informasi Pembelian Obat Klinik Graha Medika Bekasi," *Inf. Manag. Educ. Prof.*, vol. 1, no. 2, p. 234537, 2017.

[7] A. Hidayati, E. Oktariza, F. Rosmaningsih, and S. A. Lathifah, "Analisa Kualitas Perangkat Lunak Sistem Informasi Akademik Menggunakan McCall," *Multinetics*, vol. 3, no. 1, p. 48, 2017, doi: 10.32722/multinetics.vol3.no.1.2017.pp.48-53.

[8] A. Suhari Camara M, K. Aelani, and F. Dwi Juniar S, "Pengujian Kualitas Website menggunakan Metode McCall Software Quality," *J. Inf. Technol.*, vol. 3, no. 1, pp. 25–32, 2021, doi: 10.47292/joint.v3i1.43.

[9] S. Widodo, F. Pradana, and K. C. Brata, "Pembangunan Kakas Bantu Pengukuran Maintainability pada Tahap Perancangan Perangkat Lunak," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 3, no. 5, pp. 4787–4793, 2019, [Online]. Available: https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/5338

[10] J. Mona, "Pengujian Nonfungsional dengan Pendekatan McCall's Factor pada Perspektif Product Revision," *Lect. Notes Networks Syst.*, vol. 445, pp. 287–298, 2023, doi: 10.1007/978-981-19-1412-6_23.

[11] V. Lakshmi Narasimhan and B. Hendradjaya, "Detailed theoretical considerations for a suite of metrics for integration of software components," *Adv. Syst. Comput. Sci. Softw. Eng. - Proc. SCSS 2005*, pp. 257–264, 2006, doi: 10.1007/1-4020-5263-4-41.

[12] R. A. Wijaya and K. Karmilasari, "Pengukuran Kualitas Website Pengurus Cabang NU Depok Menggunakan

Software Metric," *J. Sisfokom (Sistem Inf. dan Komputer)*, vol. 10, no. 3, pp. 438–443, 2021, doi: 10.32736/sisfokom.v10i3.1267.

[13] B. B. Dr Parul Gandhi, "Metric Approach to Anticipate Reusability of Object-Oriented (O-O) Software Systems," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 6, pp. 2456–2467, 2021, doi: 10.17762/turcomat.v12i6.5690.

[14] C. Erlanson-Albertsson and A. Larsson, "Importance of the N-terminal sequence in porcine pancreatic colipase," *Biochim. Biophys. Acta (BBA)/Lipids Lipid Metab.*, vol. 665, no. 2, pp. 250–255, 1981, doi: 10.1016/0005-2760(81)90009-6.

[15] C. Vikasari, "Cyclomatic Complexity dan Graph Matrix dalam Pengujian Sistem Informasi Manajemen Rumah Sakit," *Infotekmesin*, vol. 14, no. 1, pp. 43–49, 2023, doi: 10.35970/infotekmesin.v14i1.1636.

[16] A. Almogahed *et al.*, "A refactoring categorization model for software quality improvement," *PLoS One*, vol. 18, no. 11 November, 2023, doi: 10.1371/journal.pone.0293742.

[17] M. Fowler, "Refactoring improving the design of existing code," *2017 IEEE 37th Cent. Am. Panama Conv. CONCAPAN 2017*, vol. 2018-Janua, pp. 1–3, 2017, doi: 10.1109/CONCAPAN.2017.8278488.

[18] M. Abebe and C. Yoo, "Trends , Opportunities and Challenges of Software Refactoring : A Systematic Literature Review," vol. 8, no. 6, pp. 299–318, 2014.

[19] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Identification and application of Extract Class refactorings in object-oriented systems," *J. Syst. Softw.*, vol. 85, no. 10, pp. 2241–2260, 2012, doi: 10.1016/j.jss.2012.04.013.

[20] N. Tsantalis and A. Chatzigeorgiou, "Identification of extract method refactoring opportunities for the decomposition of methods," *J. Syst. Softw.*, vol. 84, no. 10, pp. 1757–1782, 2011, doi: 10.1016/j.jss.2011.05.016.

[21] H. P. Putro and I. Liem, "Deteksi Code Smell pada Kode Program dalam Representasi AST dengan Pendekata By Rules," *Semin. Nas. Apl. dan Teknol. Inf.*, vol. 2010, no. Snati, p. 6, 2010.

[22] M. Fowler, "Refactoring: Improving the Design of Existing Programs.," p. 337, 1999.

[23] A. Alazba and H. Aljamaan, "Code smell detection using feature selection and stacking ensemble: An empirical investigation," *Inf. Softw. Technol.*, vol. 138, no. May, p. 106648, 2021, doi: 10.1016/j.infsof.2021.106648.

[24] N. Baumgartner, F. Adleh, and E. Pulvermüller, "Live Code Smell Detection of Data Clumps in an Integrated Development Environment," *Int. Conf. Eval. Nov. Approaches to Softw. Eng. ENASE - Proc.*, vol. 2023-April, no. Enase, pp. 64–76, 2023, doi: 10.5220/0011727500003464.

[25] K. Nongpong, "Feature envy factor: A metric for automatic feature envy detection," *Proc. 2015-7th Int. Conf. Knowl. Smart Technol. KST 2015*, pp. 7–12, 2015, doi: 10.1109/KST.2015.7051460.