# Metode *Refactoring* Untuk Meningkatkan Kualitas *Maintainability* Pada Sistem Informasi Pembayaran Dan Tunggakan SPP Sekolah

**Nita Siti Nuraliza[1], Puspita Nurul Sabrina[2], Herdi Ashaury [3]**
Fakultas Sains dan Informatika/Program Studi Informatika
Universitas Jenderal Achmad Yani
Cimahi, Indonesia
e-mail: [1]nitasnuraliza29@gmail.com, [2]puspita.sabrina@lecture.unjani.ac.id,
[3]herdi.ashaury@lecture.unjani.ac.id

Correspondence : *e-mail*: nitasnuraliza29@gmail.com

Diajukan: 20 Agustus 2024; Direvisi: 24 Agustus 2024; Diterima: 24 Agustus 2024

***Abstrak***

*Pengembangan perangkat lunak dan pemeliharaan rutin diperlukan untuk mencapai kualitas yang baik. Refactoring adalah salah satu dari banyak hal yang dapat dilakukan untuk menjamin kualitas. Pada hasil pengujian Sistem informasi pembayaran dan tunggakan SPP di SDIT Nuralima Karawang ini, sistem dibangun dengan menggunakan framework Laravel, serta mengirimkan informasi kepada pengguna melalui WhatsApp. Berdasarkan pengujian dengan PHPMetrics, ditemukan bahwa beberapa kelas dalam sistem ini menunjukkan nilai kompleksitas yang tinggi, di ambil dari nilai Weighted Method Class, Cyclomatic Complexity dan Max Cyclomatic Complexity. Masalah yang didapat dalam kategori smell code seperti Duplicate Code, Long Method, Magic Values, dan Deep Nesting, yang mempengaruhi maintainability sistem. Penelitian ini bertujuan untuk meningkatkan kualitas maintainability sistem informasi melalui penerapan teknik refactoring. Hasil penelitian menunjukkan bahwa teknik refactoring efektif dalam meningkatkan maintainability sistem dengan membuat kode lebih mudah dipahami, memudahkan pemeliharaan, serta meningkatkan kualitas sistem.*

***Kata kunci:*** *Refaktoring, Maintainability, Pemeliharaan Perangkat Lunak, Kualitas Perangkat Lunak, Kompleksitas Perangkat Lunak.*

***Abstract***

*Software development and routine maintenance are required to achieve good quality. Refactoring is one of the many things that can be done to ensure quality. In the testing results of the SPP payment and arrears information system at this school, the system was built using the Laravel framework and sent information to users via WhatsApp. Based on testing with PHPMetrics, it was found that several classes in this system showed high complexity values, taken from the Weighted Method Class, Cyclomatic Complexity, and Max Cyclomatic Complexity values. Problems found in the smell code category include Duplicate Code, Long Method, Magic Values, and Deep Nesting, which affect the system's maintainability. This research aims to improve the maintainability quality of the information system through the application of refactoring techniques. The results of the research show that refactoring techniques are effective in improving the system's maintainability by making the code easier to understand, facilitating maintenance, and enhancing the system's quality.*

***Keywords****: Refactoring, Maintainability, Software Maintainance, Software Quality, Software Complexity*.

## 1. Pendahuluan

Pengembangan perangkat lunak dan pemeliharaan rutin diperlukan untuk mencapai kualitas yang baik. *Refactoring* adalah salah satu dari banyak hal yang dapat dilakukan untuk menjamin kualitas. *Refactoring* dapat didefinisikan sebagai sebagai salah satu aktivitas pemeliharaan perangkat lunak untuk memperbaiki struktur internal kode, sekaligus mempertahankan perilaku eksternalnya[1]. Dalam 10 tahun terakhir, banyak penelitian menunjukkan bahwa *refactoring* mungkin mengurangi kompleksitas perangkat lunak, meningkatkan pemahaman pengembang serta meningkatkan waktu start-up dan efisiensi memori[2]. Oleh karena itu, para pengembang didorong untuk melakukan operasi *refactoring* secara teratur[1].

Sistem informasi pembayaran dan tunggakan SPP di SDIT Nuralima Karawang berbasis web ini digunakan oleh kepala sekolah dan staf tata usaha, dijalankan menggunakan local server dan whatsapp sebagai pengirim informasi kepada *user*, sistem ini digunakan untuk menyimpan data pembayaran SPP bulanan, tunggakan bagi yang belum melakukan pembayaran, pendapatan harian dari SPP bulanan dan pelaporan bulanan.

Setelah dilakukan observasi terhadap sistem informasi pembayaran dan tunggakan SPP berbasis web di sekolah, didapatkan hasil bahwa sistem informasi ini menggunakan framework laravel, dilakukan juga pengujian menggunakan tools PHPMetrics untuk menghitung kualitas sistem, terdapat beberapa keluaran kualitas seperti *Weighted Method Class* (WMC), *Cyclomatric complexity* (CC), *Max Cyclomatric complexity*. *Weighted Method Class* (WMC) mengukur kompleksitas total dari semua metode dalam suatu kelas, sementara *Cyclomatic Complexity* (CC) mengukur kompleksitas sebuah program dengan fokus pada modularisasi[3]. Max *Cyclomatic Complexity* menunjukkan nilai CC tertinggi dari metode atau fungsi dalam kode.

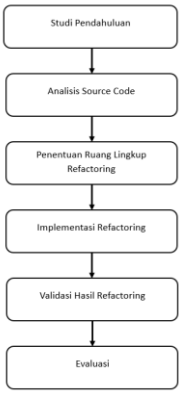
Setelah dilakukan pengujian menggunakan PHPMetrics, didapatkan class mana saja yang memiliki nilai kompleksitas yang besar, diantaranya *Controller* Admin.php, *Controller* Export.php, *Controller* Pembayaran.php, *Controller* Sekolah.php, *Controller* Siswa, *Controller* Tunggakan, dan *Controller User*. Ke tujuh class ini memiliki nilai kompleksitas yang tinggi jika dibandingkan dengan class yang lain.

Selanjutnya mencari *smell code* yang terindikasi di dalam class-class yang terindikasi memiliki kompleksitas yang tinggi, pengujian class untuk mencari *smell code* dilakukan secara mandiri. Setelah dilakukan pencarian *smell code*, maka didapatkan beberapa *smell code* yang terindikasi diantaranya, *Duplicate Code*, *Long Method*, *Magic Values*, Deep Nesting, dimana ini semua bisa mempengaruhi *maintainability*.

Adanya masalah tersebut, dapat disimpulkan bahwa perlu dilakukan *refactoring* terhadap sistem ini untuk meningkatkan kualitas dan efisiensinya. *Refactoring* dilakukan agar sistem lebih tertata rapi, mengurangi kompleksitas kode, dan mempermudah pemeliharaan di masa depan, tetapi tidak mengubah fungsi eksternal yang telah dibangun oleh *developer* dalam sistem informasi pembayaran dan tunggakan SPP di sekolah ini.

## 2. Metode Penelitian

Tahapan penelitian ini terdiri dari enam tahapan yaitu studi emphatize yang terdiri dari studi pendahuluan, analisis *source code*, penentuan ruang lingkup *refactoring*, implementasi *refactoring*, validasi hasil *refactoring* dan tahap terakhir adalah tahap evaluasi.




Gambar 1. Metode Penelitian

### 2.1. Studi Pendahuluan

Melakukan tinjauan awal untuk memahami konteks dan tujuan penelitian. Ini melibatkan pengumpulan informasi dasar tentang sistem yang akan diperiksa, termasuk fungsi dan penggunaan sistem tersebut. Tahapan selanjutnya akan mengkaji literatur terkait, mempelajari teknik *refactoring* yang relevan, dan menetapkan dasar teori yang akan mendukung penelitian. Tujuan dari studi pendahuluan adalah untuk membangun pemahaman yang kuat mengenai masalah yang ada dan bagaimana *refactoring* dapat diterapkan untuk memperbaikinya.

### 2.2. Analisis *Source Code*

Melakukan analisis mendalam terhadap kode sumber sistem yang ada. Ini mencakup identifikasi kode yang memiliki *smell code* seperti kode yang terlalu kompleks, duplikasi kode, metode yang terlalu panjang, terdapatnya string literal yang tidak jelas, atau cabang dari *if-else* yang terlalu panjang[4]-[7]. Dan juga akan menggunakan alat analisis seperti PHPMetrics untuk mengukur metrik kompleksitas seperti *Weighted Methods per Class* (WMC), *Cyclomatic Complexity* (CC) dan *Max Cyclometic Complexity*. Hasil dari analisis ini akan memberikan gambaran tentang area-area dalam kode yang memerlukan perbaikan[8].

### 2.3. Penentuan Ruang Lingkup *Refactoring*

Proses ini melibatkan penilaian prioritas terhadap area yang paling rentan terhadap kesalahan atau memiliki tingkat kompleksitas yang tinggi, sehingga memberikan dampak terbesar jika dilakukan *refactoring*. Dengan memfokuskan upaya *refactoring* pada bagian-bagian ini, diharapkan perubahan yang dilakukan memberikan perbaikan yang berarti tanpa membuang waktu dan sumber daya pada bagian kode yang tidak terlalu bermasalah. Selain itu, menentukan ruang lingkup *refactoring* yang tepat juga membantu dalam perencanaan dan pelaksanaan proyek secara keseluruhan, memungkinkan pengembang untuk bekerja lebih efisien dan efektif dalam mencapai tujuan perbaikan kode.

### 2.4. Implementasi *Refactoring*

Menerapkan teknik *refactoring* yang telah dipilih untuk memperbaiki masalah yang diidentifikasi sebelumnya. Ini dapat melibatkan teknik seperti *Extract Method*, yang memindahkan potongan kode dari metode yang kompleks ke metode terpisah untuk meningkatkan keterbacaan dan *maintainability*[1]. Selama implementasi, penting untuk memastikan bahwa perubahan tidak mengubah perilaku eksternal sistem dan tetap mempertahankan fungsionalitas yang ada.

### 2.5. Validasi Hasil *Refactoring*

Setelah *refactoring* diterapkan, tahap berikutnya adalah memvalidasi hasilnya. Ini melibatkan pengujian sistem untuk memastikan bahwa perubahan yang dilakukan tidak menimbulkan bug atau masalah baru dan bahwa fungsionalitas yang ada tetap berfungsi seperti yang diharapkan. Validasi dapat dilakukan dengan menggunakan alat pengujian otomatis dan manual serta membandingkan metrik kualitas sebelum dan setelah *refactoring*.

## 3. Hasil dan Pembahasan

Pada hasil dan penelitian yang telah dilakukan, dilakukan pembahasan mengenai temuan-temuan didalam penelitian ini. Setiap hasil yang diperoleh akan dianalisis dan dikaitkan dengan tujuan penelitian untuk memberikan gambaran yang menyeluruh mengenai pengaruh *refactoring* terhadap *maintainability*. Pembahasan dalam bab ini bertujuan untuk menjelaskan sejauh mana *refactoring* mampu meningkatkan kualitas kode dan mempermudah pemeliharaan sistem secara keseluruhan.

### 3.1. Pengukuran Menggunakan OO *Metric*

Pada penelitian ini tahap pertama yaitu pengumpulan data dengan cara mengukur sistem informasi menggunakan tools OOMetrics yaitu PHPMetrics, terdapat beberapa kriteria yang akan diukur yaitu, *Weighted Method Class* (WMC), *Cyclomatic Complexity* (CC), dan *Max Cyclometic Complexity*. Pengujian ini membantu mengidentifikasi kemungkinan area dalam kode yang memerlukan perbaikan untuk meningkatkan *maintainability*. Hasil nilai WMC, CC dan Max CC yang di ambil dari tools PHPMetrics dapat dilihat pada tabel 1 dibawah ini :

Tabel 1. Pengujian awal *OOMetrics*

| No. | Class | WMC | CC | Max CC |
|---|---|---|---|---|
| 1 | *Controller* | 0 | 1 | 0 |
| 2 | ***Controller*Admin** | **25** | **20** | 6 |
| 3 | *Controller*Dashboard | 10 | 7 | 3 |
| 4 | ***ControllerExport*** | **53** | **48** | 17 |
| 5 | *Controller*ImportDataSiswa | 8 | 8 | 8 |
| 6 | *Controller*Login | 6 | 5 | 3 |
| 7 | *Controller*Logout | 2 | 2 | 2 |
| 8 | ***Controller*Pembayaran** | **124** | **103** | 14 |
| 9 | ***Controller*Sekolah** | **12** | **10** | 7 |
| 10 | ***Controller*Siswa** | **60** | **54** | 37 |

| | | | | |
|---|---|---|---|---|
| 11 | ***Controller*Tunggakan** | **67** | **59** | 15 |
| 12 | ***ControllerUser*** | **16** | **13** | 5 |

Variabel yang ditulis tebal menunjukkan bahwa perhatian khusus dalam pengujian ini akan difokuskan pada kelas-kelas tersebut, yang dipilih berdasarkan nilai *Cyclomatic Complexity* (CC) yang lebih dari 10. Nilai CC yang lebih dari atau sama dengan 10 menunjukkan kemungkinan masalah, sesuai dengan kategori klasifikasinya[9]. Kelas-kelas yang di tebalkan akan diuji secara mandiri untuk mengevaluasi adanya indikasi *smell code* yang dapat mempengaruhi *maintainability* sistem. Dengan demikian, peneliti dapat mengidentifikasi dan menangani kemungkinan masalah yang dapat mempengaruhi kualitas dan pemeliharaan kode secara keseluruhan.

### 3.2. Analisis *Smell Code*

*Smell code* tidak serta merta menyebabkan kesalahan pada sistem namun dapat menyebabkan konsekuensi negatif lainnya, yang berdampak pada pemeliharaan dan peningkatan perangkat lunak[10]. Dalam tabel 2 hasil analisis ini berisi data class dan *method* yang telah dianalisis, yang termasuk kedalam *smell code*.

Tabel 2. Hasil Analisis

| No | Class | Jenis *smell code* | Keterangan |
|---|---|---|---|
| 1 | *Controller*Admin | *Duplicate Code* | *Method* loadDataSekolah, dataKelas memiliki pemanggilan model yang sama, ini dapat di gabungkan di satu *method* yang sama. |
| 2 | *ControllerExport* | *Duplicate Code* | Terdapat kesamaan code pada pada beberapa *method* dimana code itu memuat setingan tampilan pdf. Variabel itu dipisahkan menjadi beberapa *method* agar fokus ke satu tanggungjawab. |
| 3 | *Controller*Pembayaran | *Duplicate Code* | *Method* getSekolahData, dataKelas memiliki pemanggilan model yang sama, ini dapat di gabungkan di satu *method* yang sama. |
| 4 | *Controller*Sekolah | *Long Method* | Metode inputDataSekolah terlalu panjang dan mengandung beberapa logika yang berbeda, membuatnya sulit untuk dipahami dan dipelihara. |
| 5 | *Controller*Siswa | *Duplicate Code* | *Method* dataTunggakanBulanan memiliki logika if dan else if yang kodenya sama, ini dapat di gabungkan di satu *method* baru. Dan nanti dapat di panggil lagi jika diperlukan. |
| | | Deep Nesting | *Method* NaikKelasSiswa memiliki logika if, else dan foreach yang dalam, sehingga dapat di *refactoring* menggunakan *extract method*. |
| 6 | *Controller*Tunggakan | *Magic String* | String literal pada URL, message dan type yang memiliki nilai hard-coded diubah menjadi konstanta. |
| 7 | *ControllerUser* | *Duplicate Code* | *Method* loadSekolahData, dataKelas memiliki pemanggilan model yang sama, ini dapat di gabungkan di satu *method* yang sama. |

### 3.3. Implementasi *refactoring*

Implementasi *refactoring* merupakan langkah penting dalam penelitian ini untuk mengevaluasi efektivitas solusi yang telah dirancang. Pada tahap ini, proses *refactoring* diterapkan pada sistem informasi pembayaran dan tunggakan SPP, berdasarkan hasil perhitungan dan analisis *smell code* yang di dapat. Pada tahapan ini akan dijelaskan bagaimana *refactoring* dilakukan guna meningkatkan kualitas kode, seperti mengurangi kompleksitas, mengatasi code smells, dan memperbaiki struktur kode untuk mencapai tujuan yang lebih baik dalam hal *maintainability* sistem.

Tabel 3. Metode *Composing*

| No | Class | Jenis *smell code* | Metode *Composing* |
|---|---|---|---|
| 1 | *Controller*Admin | *Duplicate Code* | *Extract Method* |
| 2 | *ControllerExport* | *Duplicate Code* | *Extract Method* |
| 3 | *Controller*Pembayaran | *Duplicate Code* | *Extract Method* |
| 4 | *Controller*Sekolah | *Long Method* | *Extract Method* |
| 5 | *Controller*Siswa | *Duplicate Code* | *Extract Method* |
| | | Deep Nesting | Move State out of Composite, *Extract Method* |

| | | | |
|---|---|---|---|
| 6 | *Controller*Tunggakan | *Magic String* | Nilai hard-coded diubah menjadi konstanta, *Extract Method* |
| 7 | *ControllerUser* | *Duplicate Code* | *Extract Method* |

### 3.4. Pengujian akhir *OOMetrics*

Tahap pengujian akhir ini menunjukan perbedaan nilai WMC, CC dan Max CC sebelum dilakukannya *refactoring* dan sesudah *refactoring*. Berikut adalah tabel nilai dari pengujian *OOMetrics* menggunakan *PHPMetrics* :

Tabel 4. Pengujian *OOMetrics* sebelum *refactoring*

| **NO** | **Nama** | **Sebelum *Refactoring*** | | |
|---|---|---|---|---|
| | | **WMC** | **CC** | **Max CC** |
| **1** | *Controller*Admin | 25 | 20 | 6 |
| **2** | *ControllerExport* | 53 | 48 | 17 |
| **3** | *Controller*Pembayaran | 124 | 103 | 14 |
| **4** | *Controller*Sekolah | 12 | 10 | 7 |
| **5** | *Controller*Siswa | 60 | 54 | 37 |
| **6** | *Controller*Tunggakan | 67 | 59 | 15 |
| **7** | *ControllerUser* | 16 | 13 | 5 |
| Rata-rata Nilai | | 51 | 43,85 | 14,42 |

Tabel 5. Pengujian OOMetrics sesudah *refactoring*

| **NO** | **Class** | **Sesudah *Refactoring*** | | |
|---|---|---|---|---|
| | | **WMC** | **CC** | **Max CC** |
| **1** | *Controller*Admin | 25 | **18** | 6 |
| **2** | *ControllerExport* | 53 | **44** | 17 |
| **3** | *Controller*Pembayaran | **117** | **95** | **13** |
| **4** | *Controller*Sekolah | 12 | **8** | **5** |
| **5** | *Controller*Siswa | **53** | **40** | **9** |
| **6** | *Controller*Tunggakan | **63** | **53** | **11** |
| **7** | *ControllerUser* | **15** | **11** | **4** |
| Rata-rata Nilai | | **48,28** | **38,42** | **9,28** |

Terjadi perubahan nilai setelah melakukan *refactoring*, yang dimana pada tabel 5 memperlihatkan nilai sebelum di *refactoring* dan pada tabel 6 memperlihatkan nilai sesudah *refactoring*. Pada tabel 6 terdapat penanda nilai di tebalkan dan nilai di garis bawahi, untuk nilai yang di tebalkan berarti nilai sesudah di *refactoring* menurun dan untuk nilai yang di garis bawahi berarti nilai sesudah *refactoring* tetap.

Pada metrik perhitungan WMC di dapat 4 dari 7 kelas yang memiliki penurunan nilai dan 3 kelas sisanya memiliki nilai yang tetap, dengan nilai rata-rata yang awalnya 51 turun menjadi 48,28. Pada metrik perhitungan CC di dapat 7 dari 7 kelas yang nilainya turun, dengan nilai rata-rata yang awalnya 43,85 turun menjadi 38,42. Dan pada metrik perhitungan Max CC di dapat 5 dari 7 kelas yang memiliki penurunan nilai dan 2 kelas sisanya memiliki nilai yang tetap, dengan nilai rata-rata yang awalnya 14,42 turun menjadi 9,28.

Proses *refactoring* penting untuk menangani masalah yang muncul akibat nilai WMC dan CC yang tinggi. Semakin tinggi nilai WMC, semakin rumit kelas tersebut, dan ini bisa membuat maintainance jadi lebih sulit. Sementara itu, semakin tinggi nilai CC, semakin kompleks alur kontrol dalam metode, yang bisa menambah kemungkinan kesalahan dan membuat kode sulit dipahami serta diuji. Dan jika semakin tinggi nilai Max CC, menunjukkan bahwa metode memiliki banyak cabang logika, membuat kode menjadi kompleks dan sulit dipahami oleh pengembang.

*Refactoring* membantu dengan memecah kelas besar menjadi kelas-kelas kecil yang memiliki tanggung jawab lebih spesifik. Ini menurunkan nilai WMC karena kode menjadi lebih terstruktur, dengan setiap kelas hanya fokus pada tugas-tugas tertentu yang membuatnya lebih mudah dikelola dan diubah jika

diperlukan. Selain itu, dengan teknik seperti extract method, *refactoring* tidak hanya mengurangi nilai CC dan Max CC, tetapi juga menyederhanakan alur kontrol dalam kode. Ini membuat logika kode lebih jelas, lebih mudah diikuti, dan meminimalkan potensi kesalahan, yang pada akhirnya meningkatkan maintainability dan kualitas keseluruhan dari sistem.

Secara keseluruhan, mengurangi CC, WMC, dan Max CC menjadikan perangkat lunak lebih mudah dipelihara, serta meningkatkan stabilitasnya. Kode yang lebih sederhana dan terstruktur mengurangi kemungkinan munculnya masalah di masa depan dan memudahkan pengembangan berkelanjutan. Dengan demikian, *refactoring* tidak hanya meningkatkan kualitas kode tetapi juga memungkinkan tim pengembang untuk lebih efisien dalam menambahkan fitur baru dan memperbaiki bug tanpa mengganggu fungsi yang sudah ada.

## 4. Kesimpulan

Hasil pengujian menggunakan PHPMatrics untuk sistem informasi pembayaran dan tunggakan SPP di SDIT menunjukkan bahwa sebelum di lakukannya *refactoring*, dari 7 class yang di uji nilai rata-rata WMC adalah 51, CC bernilai 43,85 dan Max CC bernilai 14,42. Setelah dilakukannya *refactoring*, nilai rata-rata dari tiga perhitungan itu menurun menjadi, WMC bernilai 48,28, CC bernilai 38,42 dan Max CC 9,28. Penurunan ini menunjukkan bahwa *refactoring* berhasil mengurangi kompleksitas kode, sehingga meningkatkan keterbacaan, mempermudah pengujian, dan memperbaiki pemeliharaan sistem. Peningkatan *maintainability* yang dicapai melalui *refactoring* membawa dampak positif yang signifikan pada kualitas sistem secara keseluruhan. Dengan berkurangnya nilai WMC, CC, dan Max CC, kode menjadi lebih sederhana dan terstruktur. Kode yang lebih mudah dipahami memungkinkan pengembang untuk lebih cepat mempelajari dan bekerja dengan kode tersebut, mengurangi waktu yang dibutuhkan untuk menemukan dan memperbaiki kesalahan.

## Daftar Pustaka

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

[2] M. A. Rodrigo, "Titre : Automated Improvement of Software Design by Search-Based Title : *Refactoring* Auteur : Rodrigo Morales Alvarado Author : Date : 2017 Document en libre accès dans PolyPublie AUTOMATED IMPROVEMENT OF SOFTWARE DESIGN BY SEARCH-BASED," 2017.

[3] J. Michura and M. A. M. Capretz, "Metrics suite for class complexity," *Int. Conf. Inf. Technol. Coding Comput. ITCC*, vol. 2, pp. 404–409, 2005, doi: 10.1109/itcc.2005.193.

[4] K. Hotta, Y. Sasaki, Y. Sano, Y. Higo, and S. Kusumoto, "An Empirical Study on the Impact of *Duplicate Code*," *Adv. Softw. Eng.*, vol. 2012, no. ii, pp. 1–22, 2012, doi: 10.1155/2012/938296.

[5] H. Zhang and T. Kishi, "*Long Method* Detection Using Graph Convolutional Networks," *J. Inf. Process.*, vol. 31, pp. 469–477, 2023, doi: 10.2197/ipsjjip.31.469.

[6] E. Soares *et al.*, "*Refactoring* Test Smells: A Perspective from Open-Source *Developer*s," *ACM Int. Conf. Proceeding Ser.*, pp. 50–59, 2020, doi: 10.1145/3425174.3425212.

[7] S. Engineering, *A concise introduction to software engineering*, vol. 46, no. 06. 2009.

[8] 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.

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

[10] 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, no. May, 2020, doi: 10.1016/j.jss.2020.110610.