Analisis dan Perancangan Software Pengukuran Matriks Skala dan Kompleksitas Kode Program

Analysis and Desaign of Scale Matrix Measurement Software and Program Code Comlexity

R Billiyan Mulkan Ghifari[1], Siti Fitri[2], Afif Ardhyandoko[3], Muhammad Ainul Yaqin*[4]
[1,2,3,4,5] Program Studi Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim, Malang
*Correspondence author email: *yaqinov@ti.uin-malang.ac.id*

Abstrak

Permintaan software berkualitas semakin tinggi disebabkan oleh banyaknya software yang mengalami bahkan masih banyak kesalahan yang harus diperbaiki dan dikembangkan. Pengukuran kompleksitas kode program diperlukan sebagai proses pendeteksian kesalahan/bug sedini mungkin pada software dan menjamin kualitas pada software. Dalam perancangan software untuk menghitung kode program memerlukan metode yang dapat menghitung kompleksitas kode program. Analisi ini menggunakan salah satu metode pengukuran kompleksitas kode program yaitu Cyclomatic Complexity. Metode Cyclomatic Complexity digunakan sebagai metode untuk mengukur dan mengontrol jumlah alur melalui program sehingga dapat menghitung kompleksitas kode program. Dalam pengukuran kompleksitas kode program dilakukan dengan cara melakukan proses parsing code salah satunya menggunakan library Java Parser dan AST (Abstract Syntax Tree) untuk bahasa pemograman Java. Parsing code file kode program dilakukan untuk mendapatkan predicate nodes, operand dan operator kemudian dihitung kompleksitasnya metode Cyclomatic Complexity. Dalam menghitung kompleksitas kode program dapat menggunakan metrik skala yang dapat diterapkan sesuai dengan bahasa pemograman.
***Katakunci**: Pengukuran Kompleksitas, Kualitas Software, Cyclomatic Complexity, Deteksi Cacat Software, Skala Matrix*

Abstrack

The demand for quality software is increasing due to the large number of software experiencing problems and there are still many errors that need to be corrected and developed. Measuring program code complexity is necessary as a process of detecting errors/bugs as early as possible in the software and ensuring the quality of the software. In designing software to calculate program code, you need a method that can calculate the complexity of the program code. This analysis uses one method of measuring program code complexity, namely Cyclomatic Complexity. The Cyclomatic Complexity method is used as a method to measure and control the number of flows through a program so that it can calculate the complexity of program code. Measuring the complexity of program code is done by carrying out the code parsing process, one of which is using the Java Parser library and AST (Abstract Syntax Tree) for the Java programming language. Parsing the program code file is carried out to obtain predicate nodes, operands and operators, then the complexity is calculated using the Cyclomatic Complexity method. In calculating the complexity of program code, you can use scale metrics that can be applied according to the programming language
***Keyword**: Complexity Measurement, Software Quality, Cyclomatic Complexity, Software Defect Detection, Matrix Scale*

1. PENDAHULUAN

Software memiliki peran penting dalam pengembangan teknologi, namun banyak terjadi kesalahan dan kegagalan pada *software* yang sedang dikembangkan. Sehingga menyebabkan banyaknya permintaan untuk *software* berkualitas. Salah satu pengukuran kualitas dari suatu

History of article:
Received: Mei, 2024 : Accepted: Mei, 2024

software dapat digunakan dengan cara perhitungan kompleksitas program, efektifitas dari modularitas serta size dari program tersebut[1]. Sedangkan untuk mengetahui kualitas dan jaminan terhadap kehandalan software dapat digunakan 2 teknik pengembangan yaitu software testing serta software matriks[2]. Secara definisi software testing merupakan suatu teknik yang dapat digunakan untuk mengecek kesalahan setelah suatu perangkat lunak selesai dikembangkan, hal ini bertujuan untuk mencegah berbagai macam kesalahan / error yang timbul setelah perangkat lunak diimplementasikan sehingga kualitas dari software dapat lebih terjamin[3]. Sedangkan, software metrics merupakan suatu teknik yang digunakan untuk membantu peningkatan kuliatas suatu perangkat lunak melalui pemantauan kompleksitas perangkat lunak tersebut[4]. Namun kompleksitas perangkat lunak dapat dipandang dengan kemudahan penggunaan perangkat lunak untuk digunakan dan diverifikasi oleh user[5]. Metode yang paling sering digunakan dalam melakukan analisis kualitas perangkat lunak secara internal dapat dilakukan melalui analisis statis kode program[6]. Dimana analisis ini akan melakukan kajian terhadap perangkat lunak dan tidak dieksekusi[6]. Oleh karena itu analisis ini akan menekankan proses perancangan kompleksitas kode program guna menghasilkan perangkat lunak yang berkualitas. Kode program atau yang biasa disebut dengan *source code* merupakan kumpulan perintah yang dituliskan secara berurutan untuk menghasilkan output / informasi yang diperlukan oleh pengguna yang akan dieksekusi oleh perangkat lunak [7]. Selain itu juga, kode program juga digunakan sebagai informasi guna mengetahui dan memperbaiki bug yang terjadi selama proses eksekusi[8].

Kompleksitas kode program dapat diukur dengan beberapa metrik salah satunya yaitu *LOC* (*Line of Code*)[9], McCabe's *Cyclomatic Complexity* dan *Halstead's Volume*[10]. Dalam analisis ini, kompleksitas kode program akan dihitung menggunakan metode metrik yaitu McCabe's *Cyclomatic Complexity*. McCabe's *Cyclomatic Complexity* merupakan salah atribut yang biasa digunakan menghitung kompleksitas suatu kode program[11]. Dan cara ini juga dapat digunakan untuk mengontrol dan mengukur kompleksitas alur program[12]. Dalam menghitung kompleksitas kode program metode *Cyclomatic Complexity* dapat membaca kode program secara sekuensial, seleksi, maupun perulangan[13] sehingga dapat dipastikan bahwa seluruh kode program dapat terhitung sehingga dapat menyimpulkan bahwa program termasuk sederhana atau komplek, dibandingkan dengan metode lain yang menghitung secara fungsinya, seperti Metode *Halstead's Volume* yang menghitung kode program berdasarkan tingkat *volume* pada sebuah *software*[9], Metode *LOC* (*Line of Code*) yang menghitung kode program berdasarkan alur berjalannya program tanpa adanya perulangan dan sebagainya[4].

Untuk mendeteksi kesalahan pada kode program maka dapat digunakan kompleksitas kode program dimana pendekatan ini akan memberikan informasi defect yang terjadi[14]. Informasi terhadap pengukuran ini akan dapat mengetahui kesalahan ataupun kecacatan yang terjadi pada perangkat lunak serta mengetahui kesalahan dengan cepat sehingga menjamin kualitas perangkat lunak yang akan / telah digunakan. Ketika hal ini tidak dilakukan maka dapat memungkinkan menambah biaya perawatan perangkat lunak baik dalam jangka waktu pendek ataupun jangka panjang. Suatu perangkat lunak yang mengalami kesalahan baik dalam hal input ataupun output yang dihasilkan dapat berdampak pada penurunan kualitas perangkat lunak tersebut, hal ini tentu saja sangat dihindari oleh para developer ataupun tim pengembang perangkat lunak. Berdasarkan hal tersebut maka penelitian ini akan melakukan analisis terhadap suatu perangkat lunak guna menghitung kompleksitasnya khususnya pemrograman yang berbasis obyek oriented yaitu Java. Penelitian ini akan menggunakan Bahasa Pemrograman Java karena lebih fleksibel dan powerful serta paling sering digunakan oleh para developer dalam mengembangkan suatu pernagkat lunak[15]. Dalam analisis ini, akan mengambil bahasa *Java* sebagai objek studi kasusnya.

2. METODE PENELITIAN

Metode analisis ini dengan *waterfall model*. Adapun tahapan yang dilakukan adalah:

Studi Literatur
Studi literatur yang dilakukan dengan cara mencari dan mendapatkan teori-teori mengenai analisis terhadap tatacara dan tahapan perhitungan kompleksitas kode program serta perancangan kebutuhan pengembangan perangkat lunak yang akan dilakukan. Peneliti mendapatkan literatur ini melalui jurnal, e-book, buku dan lainnya. Secara spesifik penelitian ini merujuk pada jurnal dari Thomas J. McCabe, Sr. pada tahun 1976 yang berjudul *Transactions on Software Engineering. A Complexity Measure* sebab dalam jurnal tersebut merupakan ulasan mengenai pengembangan awal adanya metode *Cyclomatic Complexity* dan *e-book JavaParser: Visited Analyse, transformand generate your Java code base* yang dibuat oleh Nicholas Smith, Danny van Bruggen, and Federico Tomassetti sebab *e-book* tersebut sebagai dasar perancangan dalam menguraikan kode program *Java*.

Analisis Kebutuhan
Analisis kebutuhan akan dilakukan dengan mengamati sekaligus mempelajari segala macam kebutuhan pengguna perangkat lunak terhadap perangkat lunak yang digunakan. Hal ini bertujuan mempelajari berbagai macam kebutuhan dasar perangkat lunak yang dianalisis termasuk didalamnya beberapa input ataupun output dasar yang wajib dipenuhi baik sebelum ataupun sesudah perangkat lunak tersebut dianalisis. Hal ini juga harus mempertimbangkan berbagai macam kebutuhan antar pimpinan pengguna perangkat lunak.

Perancangan
Dalam perancangan perangkat lunak seorang developer harus dapat menghubungkan spesifikasi inti dari kebutuhan yang diperlukan dan implementai yang dilakukan sehingga suatu perangkat lunak dapat terbentuk sesuai keinginan. Tujuannya memberikan gambaran secara jelas terhadap pernagkat lunak yang dikembangkan serta dapat memenuhi kebutuhan input / output user. Dalam perancangan ini merupakan tahapan yang terpenting agar perangkat lunak sistem perhitungan kompleksitas kode program dapat dibuat sehingga dapat diimplementasikan.

Implementasi
Implementasi *software* dilakukan untuk memperoleh dan mengintegrasikan sumber daya fisik dan konseptual sehingga dapat menghasilkan suatu sistem yang bekerja. Tujuan dari tahap implementasi adalah mewujudkan permodelan dari hasil perancangan sistem sesuai dengan kebutuhan dan tahapan yang telah dirancang. Implementasi *software* disesuaikan dengan bahasa pemograman yang akan dihitung sehingga tujuan dari *software* sistem perhitungan kode program dapat terealisasikan.

Pemeliharaan
Pernagkat lunak yang telah diebrikan kepada pelanggan pasti akan mengalami perubahan. Perubahan tersebut biasanya terjadi karena penyesuaian perangkat lunak terhadap lingkungan atau yang biasa dikenal dengan peripheral sistem baru ataupun penyesuaian kebutuhan fungsional sistem pengguna terhadap perangkat lunak yang digunakan.

3. HASIL DAN PEMBAHASAN

Dalam menganalisis *software* perhitungan metrik skala dan kompleksitas kode program menggunakan metode *waterfall model* mendapatkan pembahasan berupa:

Studi Literatur
Beberapa referensi yang berupa jurnal dan buku yang terkait dalam analisis dan perancangan ini, diantaranya adalah:

ISSN: 2797-8443 (media online)

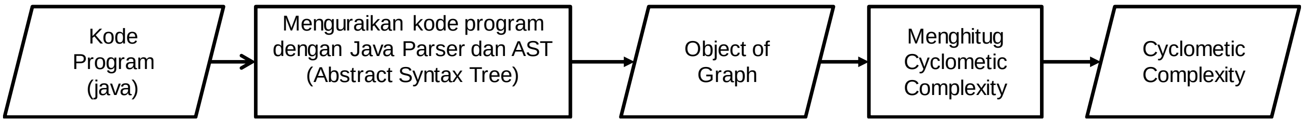
1. Pengukuran Kompleksitas Kode Program
   a. *Cyclomatic Complexity*
   b. *Java Parser* and *AST*
   c. Metrik Skala
2. *Software Process Model*
   a. Waterfall Model

Analisis Kebutuhan

Analisis kebutuhan pada *software* perhitungan metrik skala dan kompleksitas kode program dalam pembuatan membutuhkan library untuk mengurai file .Java dengan *Java Parser* and *AST*. *Cyclomatic Complexity* merupakan pendekatan ukuran kompleksitas yang dilakukan untuk mengukur dan mengontrol jumlah alur melalui program sehingga memerlukan skala metrik agar bisa lebih komplek dan teliti dalam menghitung kompleksitas kode program seperti metrik *Halstead, Path Testing*, dsb.

Perancangan

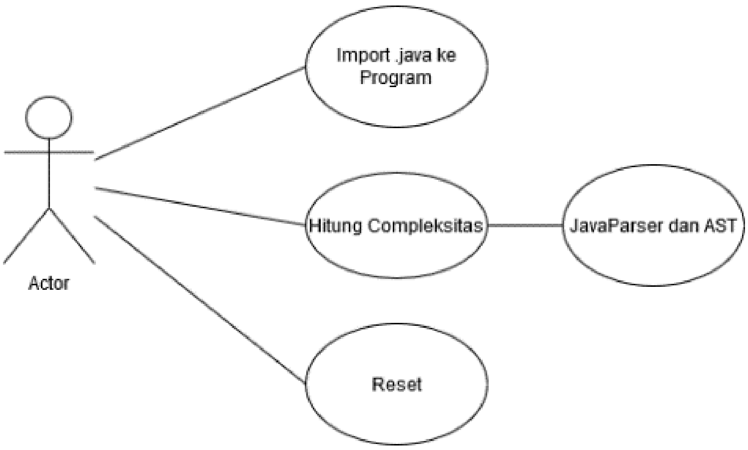
Gambar 1 adalah tahapan dari perancangan arsitektur sistem yang akan dilakukan. Mulai dari bagaimana perangkat lunak dibangun sampai dilakukan analisis.



Gambar 1. Perancangan arsitektur sistem

Usecase Diagram

Gambar 2 adalah perancangan diagram usecase yang dilakukan dimana diagram ini akan menghubungkan interaksi antar sistem dengan pengguna.

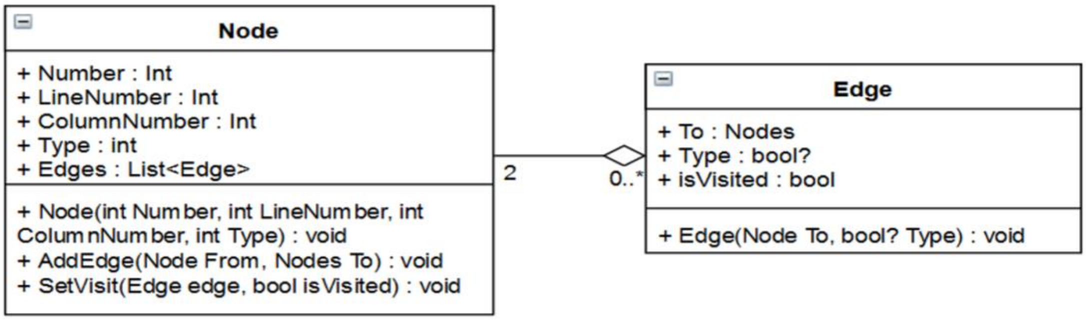


Gambar 2. ***Use Case Diagram***

Berdasarkan Gambar 2 diketahui bahwa aktor yang terlibat dari proses ini dapat melakukan import java kedalam program yang selanjutnya akan dilakukan perhitungan kompleksitas berdasarkan model *Cyclomatic Complexity*. Diawali dengan memilih dan membuka file kode kedalam sistem dengan ekstensi file java sebagai inputan datanya selanjutnya akan dihitung berdasarkan mode *Cyclomatic Complexity*. User juga dapat mengatur (mereset) file log yang berasal dari perhitungan kompleksitas kode sumber model *Cyclomatic Complexity*.

Class Diagram

Untuk mendapatkan hubungan antar clas yang dibangun maka diperlukan perancangan class diagram sebagai gambaran struktur perangkat lunaknya. Gambar 3 adalah perancangan class diagram yang digunakan.



Gambar 3. ***Class Diagram***

Dalam *class node* terdapat informasi mengenai nomor *node*, nomor baris dan nomor kolom dari kode program yang akan digunakan sebagi instrumentasi. Dalam *class node* juga, terdapat tipe dari perintah tersebut apakah termasuk percabangan, pengulangan, perintah biasa, atau akhir dari sebuah perintah. Selain itu, terdapat *list edge* yang berisi tipe dari *edge* yang digunakan jika terdapat percabangan *true*, *false*, atau hanya garis penghubung biasa dan *node* tujuan. Setiap *edge* memiliki atribut *is Visited* yang digunakan sebagai pertanda garis penghubung tersebut sudah dilalui

Implementasi

Implementasi perangkat lunak perhitungan metrik skala dan kompleksitas kode program menggunakan *software* Netbeans IDE 8.2. Ketika kode program yang akan dihitung kompleksitasnya maka terlebih dahulu diuraikan oleh *library* yang telah dimasukkan *software* Netbeans IDE 8.2, sehingga kode programnya dapat diuraikan. *Abstract Syntax Tree (AST)* terdiri dari beberapa node, setipa set berisi informasi yang dikenal sebagai structural properties[16]. Gambar 5 menunjukan *structural properties* dari *method declaration*.

```
public void start(BundleContext context)
throws Exception {
 super.start(context);
```

Gambar 5. ***Structural Properties Method Declaration***

ISSN: 2797-8443 (media online)

Dalam *Structural Properties Method Declaration* merupakan proses penguraian yang dilakukan oleh *Java Parser* dan *Abstract Syntax Tree (AST)* sehingga mendapatkan *node* yang akan dihitung oleh *Cyclomatic Complexity*, sehingga hasil *output* dari proses yang dilakukan *Java Parser* ditunjukkan tabel 2.

Tabel 1. Hasil Output Java Parser

Input	Output
PUBLIC VOID START(BUNDLECONTEXT CONTEXT) THROWS EXCEPTION { SUPER.START(CONTEXT); }	PublicAccessSpecifier, MethodDefinition, VoidDataType, FormalMethodParameter, ThrowsSpecification, ExceptionClass, SuperReference, SuperclassMethodCall, ExpressionStatement

Sumber [17]

Tabel 2 menunjukan konsep *output Java Parser* untuk fragmen kode pada *method declaration.* Setelah proses membangun pohon (*tree)* menggunakan *Eclipse AST API* selesai, *parser* melakukan analisis sematik menggunakan informasi dalam setiap *node*. Informasi tersebut digunakan untuk mengidentifikasi indeks detail dari *source code* program

Tabel 2. Hasil Percobaan Import Java

Percobaan pertama (Gagal)	Percobaan kedua (Berhasil)
File log cyclomatic complexity ditemukan. Prosedur	File log cyclomatic complexity ditemukan. Prosedur
Uji Set nama *file log Cyclomatic* sesuai dengan nama *file* di direktori penyimpanan *file*	Uji Set nama *file log Cyclomatic* yang berbeda dengan nama *file* di direktori penyimpanan *file*
Expected Result Mengembalikan nilai *true*	*Expected Result* Mengembalikan nilai *false*
Test Result Mengembalikan nilai *true*	*Test Result* Mengembalikan nilai *false*

Sumber [17]

Menghitung Kompleksitas Program

Setelah kode program diuraikan dan selanjutnya dihitung menggunakan *cyclomatic complexity* dengan kode program seperti tabel 2. Dalam penguraian kode program Java ini, *Java Parser* dan *Abstract Syntax Tree (AST)* menghasilkan *predicate nodes* maka rumus *Cyclomatic Complexity* yaitu:

$$\mathrm{V(G)} = P + 1 \quad \dots (1)$$

Dimana P = jumlah predicate nodes

Tabel 4. Hasil Perhitungan *Cyclomatic Complexity*

Percobaan	Data Kode Program	*Perdicates Node*	Hasil *Cyclomatic Complexity*
Percobaan Pertama	57	21	22
Percobaan Kedua	32	14	15
Percobaan Ketiga	70	28	29

Dalam table 4 dapat diketahui bahwa *Perdicates Node* yang telah dihasilkan *Java Parser* dan *Abstract Syntax Tree (AST)* akan dihitung oleh rumus *Cyclomatic Complexity* sehingga dapat disimpulkan bahwa sesuai daftar evaluasi resiko yang ditetapkan oleh *Software Engineering Institute* maka percobaan pertama dan ketiga memiliki kompleksitas yang tinggi dan percobaan kedua memiliki resiko kompleksitas yang sedang.

4. KESIMPULAN

Dalam analisis *software* perhitungan metrik skala dan kompleksitas kode program, dapat disimpulkan bahwa perhitungan kompleksitas kode program bisa membuat acuan terhadap pengembangan *software* terutama dalam mengetahui kesalahan/*bug* pada suatu program. Selain itu dalam menghitung komplesitas kode program tidak hanya menggunakan *Cyclomatic Complexity* bisa menggunakan skala metrik untuk menunjang pengukuran kompleksitas. Dalam perancangan software perhitungan metrik skala dan kompleksitas kode program, dapat menggunakan beberapa permodelan seperti *UML* (*Unified Modelling Language*) atau Model *Prototype*

DAFTAR PUSTAKA

[1] R. Parlika, R. R. Mahendra, M. R. A. R. Lutfi, R. K. Waritsin, and H. M. T. Ramadhan, “Pengukuran Kualitas Perangkat Lunak Website Pendataan Ekskul Siswa Menggunakan Function Point,” *J. Ilm. Inform.*, vol. 11, no. 01, pp. 1–14, 2023, doi: 10.33884/jif.v11i01.5578.

[2] M. A. Mude and M. A. Asis, “Evaluasi Kualitas Software Menggunakan Iso 1926-4,” *J. Inf. Syst. Manag.*, vol. 5, no. 1, pp. 58–63, 2023, doi: 10.24076/joism.2023v5i1.1165.

[3] F. Jihad, D. R. P. Lubis, and A. H. Lubis, “Perancangan Sistem Informasi Jadwal Kegiatan Pegawai Berbasis Web,” *Simtek J. Sist. Inf. dan Tek. Komput.*, vol. 8, no. 1, pp. 24–29, 2023, doi: 10.51876/simtek.v8i1.173.

[4] R. Parlika, D. C. M. Wijaya, H. Khariono, and R. A. Fernanda, “Studi literatur perbandingan antara metode LOC, COCOMO, FPA dalam ranah software metric,” *J. Pendidik. Inform. dan Sains*, vol. 9, no. 1, p. 66, 2020, doi: 10.31571/saintek.v9i1.1697.

[5] Y. Lestari, A. Istiani, N. D. Farhanah, and M. A. Yaqin, “Survei Metrik Kompleksitas User Interface Menggunakan Sistematic Literature Review,” *Ilk. J. Comput. Sci. Appl. Informatics*, vol. 4, no. 2, pp. 146–161, 2022, doi: 10.28926/ilkomnika.v4i2.463.

[6] E. Suprapto, “User Acceptance Testing (UAT) Refreshment PBX Outlet Site BNI Kanwil Padang,” *J. Civronlit Unbari*, vol. 6, no. 2, p. 54, 2021, doi: 10.33087/civronlit.v6i2.85.

[7] D. F. Rizaldi, J. Abdillah, M. Naufal, M. A. Yaqin, and A. C. Fauzan, “Survei Pengukuran Fleksibilitas Software Menggunakan Metode Systematic Literature Review,” *Ilk. J. Comput. Sci. Appl. Informatics*, vol. 4, no. 1, pp. 53–66, 2022, doi: 10.28926/ilkomnika.v4i1.253.

[8] M. R. Aditya and C. Dewi, “Optimisasi Pengecekan Anomali pada Proses Job : Analisis Waktu dan Data untuk Identifikasi Anomali yang Efisien,” *J. Indones. Manaj. Inform. dan Komun.*, vol. 5, no. 2, pp. 1819–1832, 2024, doi: 10.35870/jimik.v5i2.737.

[9] R. A. Nugroho, F. Abadi, M. R. Faisal, R. Herteno, and R. Ramadhani, "Metrics Based Feature Selection for Software Defect Prediction," *J. Komputasi*, vol. 8, no. 2, 2020, doi: 10.23960/komputasi.v8i2.2670.

[10] C. N. Paradis, M. Robert Yusuf, M. Farhanudin, and M. Ainul Yaqin, "Analisis dan Perancangan Software Pengukuran Metrik Skala dan Kompleksitas Diagram Class," *J. Autom. Comput. Inf. Syst.*, vol. 2, no. 1, pp. 58–65, 2022, doi: 10.47134/jacis.v2i1.40.

[11] F. A. Putri, G. I. Marthasari, and I. Nuryasin, "Rancang Bangun Perangkat Lunak Perhitungan Metrik Cyclomatic Complexity Berdasarkan Control Flow Graph Berbasis Web," *J. Repos.*, vol. 5, no. 1, pp. 565–574, 2023, doi: 10.22219/repositor.v5i1.1469.

[12] M. G. AL Khamaeni, "Implementasi White Box Testing Berbasis Path Pada Aplikasi Berbasis Web," *J. Siliwangi*, vol. 9, no. 1, pp. 8–13, 2023, doi: 10.37058/jssainstek.v9i1.4109.

[13] R. Rosmiati and I. Reski, "Rancang Bangun Aplikasi Pembelajaran E-Learning Mata Kuliah Grafika Komputer," *BANDWIDTH J. Informatics Comput. Eng.*, vol. 01, no. 02, pp. 65–74, 2023, doi: 10.53769/bandwidth.v1i2.517.

[14] 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.

[15] S. R. Sistina and D. Safitri, "Sistem Pengendalian Internal Atas Biaya Operasional Pada BPJS Kesehatan KC Metro," *J. Ilmu Data*, vol. 1, no. 3, pp. 1–13, 2021.

[16] Arrijal Nagara Yanottama and Siti Rochimah, "Analisis Dampak Perubahan Artefak Kebutuhan Berdasarkan Kedekatan Semantik Pada Pengembangan XP," *J. RESTI (Rekayasa Sist. dan Teknol. Informasi)*, vol. 5, no. 4, pp. 721–728, 2021, doi: 10.29207/resti.v5i4.3281.

[17] F. N. Pranata, "Pengembangan Sistem Perhitungan Kompleksitas Kode Sumber Berdasarkan Metrik Halstead dan Cyclomatic Complexity," Universitas Brawijaya, 2016.