Analisis dan Perancangan Software Pengukuran Metrik Skala dan Kompleksitas Diagram Class

Analysis and Design of Metric Measurement Software of Scale and Complexity of Class Diagram

Cika Nurqueen Paradis*[1], Muhamad Robert Yusuf[2], Muchamad Farhanudin[3] , Muhammad Ainul Yaqin[4]
[1,2,3] Program Studi Teknik Informatika, Fakultas Sains dan Teknologi, , Universitas Islam Negeri Maulana Malik Ibrahim Malang
*e-mail: *[1]185650039@student.uin-malang.ac.id, [2] 18650060@student.uin-malang.ac.id, [3]18650058@student.uin-malang.ac.id, [4]yaqinov@ti.uin-malang.ac.id*

Abstrak
Dalam pembuatan sebuah software haruslah dianalisa apakah software tersebut sudah kompleks dalam menyelesaikan masalah atau belum. Karena banyak software yang ada kurang bermanfaat untuk pengguna atau tidak bisa diimplementasikan dalam permasalahan yang dihadapi pengguna. Berdasarkan hal terebut maka penelitian ini akan mengimplementasikan matirk CWICC untuk menganalisa kompleksitas diagram class dalam menggambarkan alur program dan menghitug seberapa kompleks software yang dibuat dapat dimanfaatkan oleh user. Berdasarkan hasil pengujian diketahui bahwa dari 5 data uji dihasilkan akurasi sebesar 100% dengan rata-rata waktu yang diperlukan adalah 356 kali lebih cepat dibandingkan dengan perhitungan manual.
***Kata kunci**: Kompleksitas perangkat lunak, Software maintenance, Metrik Cognitive Weighted Inherited Class Complexity (CWICC)*

Abstrack
In making a software, it must be analyzed whether the software is complex in solving the problem or not. Because a lot of existing software is less useful for users or cannot be implemented in the problems faced by users. Based on this, this research will implement the CWICC matrix to analyze the complexity of the class diagram in describing the program flow and calculate how complex the software created can be utilized by the user. Based on the test results, it is known that from 5 test data an accuracy of 100% is produced with the average time required is 356 times faster than manual calculations.
***Keyword**: Software complexity, Software maintenance, Metrik Cognitive Weighted Inherited Class Complexity (CWICC)*

1. PENDAHULUAN

Kompleksitas perangkat lunak menunjukkan kesulitan dalam memahami, memelihara, memodifikasi dan menggunakan kembali perangkat lunak[1]. Apabila nilai kompleksitas tinggi maka akan kesulitan dalam memahami perangkat lunak pada tahap pemeliharaan maupun melakukan perubahan. Oleh karena itu kompleksitas perangkat lunak perlu dipantau untuk menetapkan kualitas dan kehandalan perangkat lunak tersebut menggunakan *software metric* [2]. Perangkat lunak yang dikembangkan dengan menerapkan pendekatan berorientasi objek akan berpusat pada objek dalam pemrogramannya dan mendukung penggunaan *inheritance*, *encapsulation*, *coupling*, dan sebagainya, yang dapat memudahkan *developer* dalam meningkatkan kualitas dan kehandalan perangkat lunak. Namun penggunaan *inheritance* dan *coupling* yang tinggi juga dapat mengakibatkan tingginya nilai kompleksitas[1].

Untuk meminimalisir *maintenance software* saat *software* sudah di publikasikan, maka perlulah analisa dalam berbagai aspek software tersebut sebelum software dipublikasikan

History of article:
Received: Juni, 2022 : Accepted: Juni, 2022

kepada user. Untuk memudahkan analisa software ini, program dalam software yang akan dipublish pertama digambarkan dengan diagram class, setelah digambar menggunakan diagram *class kompleksitas* dari *software* ini bisa diukur menggunakan *metric* yang sudah ditentukan, dan nantinya bisa didapatkan output berupa nilai kompleksitas dari software tersebut.

Class diagram adalah deskripsi yang paling penting dan paling banyak digunakan dari sebuah sistem berbasis objek[3]. Class diagram menunjukkan struktur statis dari class-class inti yang membangun sistem[4]. Class diagram menampilkan attribute dan method pada setiap class, selain itu class diagram juga menampilkan relation yang terdapat di antara setiap class[5].

Metrik skala kompleksitas class diagram dapat mendefinisikan tingkat kesulitan dalam memahami konsep inheritance pada kelas yang menyusunnya. Banyak peneliti yang mengusulkan berbagai metric baru dengan parameter yang berbeda dalam melakukan perhitungan nilai pada kompleksitas perangkat lunak. Salah satu dari metric tersebut adalah *Cognitive Weight Inherited Class Complexity* (CWICC) dimana matric ini digunakan untuk menghitung nilai kompleksitas diagram class yang berdasarkan bobot pemahaman pada attribute, method, dan inheritance pada suatu kelas. Sistem ini memiliki nilai akurasi 100% dan waktu untuk perhitungan kompleksitas lebih cepat daripada menggunakan perhitungan manual[6]. Selain itu kualitas dari suatu class diagram juga dapat digunakan melalui perhitungan nilai estimasi matriks *modifiability*[7]. Berdasarkan hal tersebut, maka penelitian ini bertujuan untuk menerapkan CWICC untuk menghitung kompleksitas suatu class diagram.

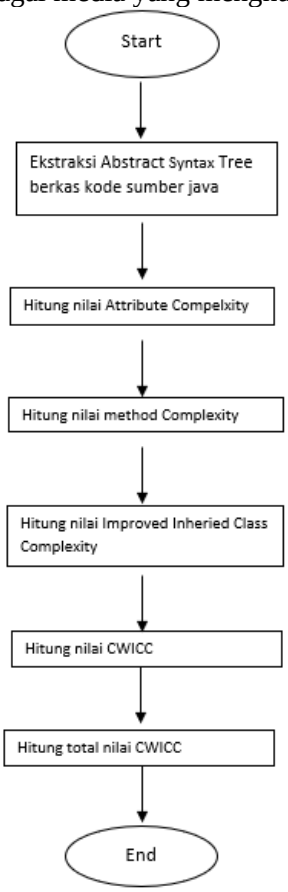
2. METODE PENELITIAN

Pada penelitian ini, peneliti akan melakukan perhitungan kompleksitas diagram class menggunakan metrik CWICC yang meliputi beberapa tahapan. Tahapan pertama dimulai dari melakukan ekstraksi *Abstract Syntax Tree* dari kode sumber yang akan diuji untuk menentukan *attribute, method, dan inheritance* yang menyusun kode sumber pada kelas tersebut dan kemudian menyimpan datanya. Setelah itu dilakukan perhitungan kompleksitas perangkat lunak menggunakan metrik CWICC, yaitu menghitung nilai *Attribute Complexity, Method Complexity, Improved Inherited Class Complexity*, dan *Cognitive Weighted Inherited Class Complexity* pada setiap kelas. Selanjutnya menghitung total nilai dari seluruh CWICC. Alur tahapan perhitungan kompleksitas dengan mentrik CWICC ditunjukkan pada Gambar 1.

1. Pada tahap awal peneliti akan membuat sebuat UML Diagram class dengan menggunakan tools tertentu seperti starUML ataupun bisa langsung didalam IDE yang telah di pasang sebuah pluggin yang memungkinkan untuk melakukan ekstraksi dari UML tersebut kedalam bahasa pemrograman yang dapat membaca UML Diagram Class tersebut. Contoh pluggin yang dimaksud seperti *easyUML* yang mana dapat ditemukan dalam IDE Netbeans yang berupa kode java.
2. Setelah itu menghitung nilai dari AC atau *atribut complexity* dari kode bahasa pemrograman tersebut yang telah di ekstraksi dengan berpusat pada jumlah dan bobot dari atribut serta tipe datanya
3. Selanjutnya menghitung MC atau *Method Complexity* yang menggunakan bobot kognitif dari BCs atau *Basic Control Structure* yang terdiri dari beberapa kategoriy
4. Berikutnya menghitung nilai dari IICC atau *Improved Inheried Class Complexity* yang merupakan bagian dari method dan atribut yang fungsinya telah digunakan kembali.
5. Setelah ketiga hal tersebut telah didapatkan, maka selanjutnya dapat menghitung nilai dari CWICC tersebut. Jika terdapat lebih dari satu system yang berhubungan maka dapat menghitung total dari kedua system tersebut menjadi nilai suatu kesatuan kompleksitas

Sedangkan untuk perancangan sistemnya akan digunakan sebagai dasar melakukan tahap implementasi. Perancangan sistem dilakukan untuk mendapatkan *sequence* diagram dan *class* diagram. Perancangan komponen dilakukan untuk mendapatkan *pseudocode* berupa rancangan

algoritme method pada sistem. Perancangan antarmuka dilakukan untuk menghasilkan rancangan *layout* antarmuka sistem sebagai media yang menghubungkan pengguna dan sistem.



Gambar 1. Alur tahapan perhitungan

3. HASIL DAN PEMBAHASAN

Dalam analisis perhitungan metrik skala dan kompleksitas Digram Class, peneliti menggunakan metode Kalkulasi menggunakan metrik CWICC.

Hasil perancangan sistem

Pada Gambar 2 dijelaskan bahwa pengguna akan menginputkan berkasnya untuk dianalisa lalu program akan dihitung kompleksitasnya menggunakan metrik CWICC, selanjutnya pengguna akan mendapatkan output berupa Pengujian Akurasi Sistem.

1. Input Berkas

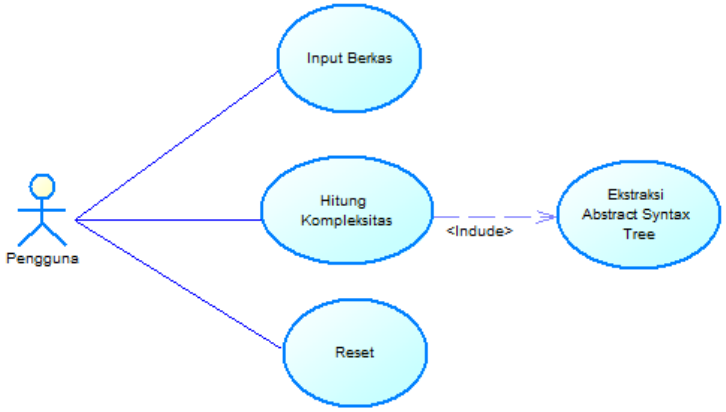
   Dalam tahap ini user menginputkan file berupa rancangan softwarenya dalam bentuk diagram class untuk dihitug berapa nilai kompleksitas dalam sftwarenya.

2. Hitung Kompleksitas

   Setelah file dimasukkan system akan menghitung jumlah class, attribute, dan method yang ada dalam software tersebut dengan meng ekstarak syntax yang ada dalam diagram class tersebut dan mulai menghitung menghitung dengan metric CWICC, setelah itu nilai kompleksitas akan keluar untuk dijadikan refrensi dalam evaluasi maintenance software.

3. Reset

   Jika output yang dihasilkan sudah keluar user bisa mereset ulang hasil analisa dan menginputkan data baru untuk dihitung nilai kompleksitasnya.



Gambar 2. Diagram usecase

Perhitungan CWICC

Metrik CWICC pertama kali diusulkan untuk menghitung kompleksitas perangkat lunak yang disebabkan penggunaan *inheritance*[6]. Metrik CWIIC dapat dihitung dengan persamaan (1)

$$CWICC = \sum_{i=1}^{n} \mathrm{AC}\,i + \sum_{j=1}^{n} \mathrm{MC}\,j + \sum_{j=1}^{n} \mathrm{IICCk} \quad \ldots (1)$$

Keterangan :
AC = Kompleksitas Atribut (*Attribute Complexity*)
MC = Kompleksitas Sistem (Method Complexity)
IICC = Kompleksitas Kelas Turunan Lanjutan (*Improved Inherited Class Complexity*)

Sedangkan Untuk menghitung ***Attribute Complexity*** (AC) pada sebuah kelas dapat menggunakan persamaan (2) [8]

$AC = (PDT * Wb) + (DDT * Wd) + (UDDT * Wu)$ … (2)

Keterangan :

PDT (Total *Primary Data Type*) adalah tipe data mendasar dalam bahasa pemrograman seperti jika dalam bahasa C adalah integer(int), floating point(float), character(char) dan void.

Wb adalah Bobot Kognitif dari PDT.

DDT (Total *Derived Data Type*) adalah Tipe data yang ditentukan oleh pengguna itu sendiri. Seperti, mendefinisikan kelas dalam C ++ atau sebuah struktur. Dalam hal ini termasuk Arrays, Structures, Class, Union, Enumeration, Pointers dan sebagainya.

Wd adalah Bobot Kognitif dari DDT

UDDT (Jumlah *User Defined Data Type*) adalah tipe data yang berasal dari tipe data yang ada. UDDT dapat menggunakan DDT untuk memperluas tipe bawaan yang sudah tersedia dan membuat tipe data khusus sendiri. Ada 6 UDDT seperti Distinct type, Structured type, Reference type, Array type, Row type,dan Cursor type

Wu adalah Bobot Kognitif dari UDDT

Sedangkan Bobot dari faktor atributnya dapat ditunjukan seperti pada tabel 1

Tabel 1. Beban Faktor Attribute

Jenis Attribute	Beban
PDT	1
UDT	2
UDDT	3

Sedangkan untuk menghitung dari *Method Complexity* (MC) dapat dihitung dengan mengusulkan berat kognitif total komponen perangkat lunak, Wc, didefinisikan sebagai jumlah dari bobot kognitif dari blok linear q-nya yang terdiri dari BCS (*Basic Control Structure*) individu[9]. Karena setiap blok dapat terdiri dari m lapisan BCS bersarang, dan setiap lapisan n BCS linier, berat kognitif total, Wc, dapat dihitung dengan persamaan 3

$$MC = \sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{i=1}^{n} w_c(j,k,i)\right] \quad \dots (3)$$

Jika tidak ada BCS yang terpasang oada blok q, maka m=1 kemudian dapat disederhanakan menjadi persamaan 4

$$\mathrm{Wc} = \sum_{j=q}^{q}\sum_{i=1}^{m} Wc(j,i) \quad \dots (4)$$

Keterangan :
Wc= Bobot Kognitif Basic Control Structure

IICC adalah *Improved Inherited Class Complexity* yang mana perhitungannya dilakukan dengan menambahkan kompleksitas yang ada dari *Cognivite depth, method,* dan *Attribute* yang digunakan kembali[10]. Cara menghitung dari IICC sendiri dapat menggunakan persamaan 5.

$$ICC = CDC + \sum_{k=1}^{s}\mathrm{RMKk} + \sum_{l=1}^{t} RACl \quad \dots (5)$$

Keterangan :

CDC = Nilai Kompleksitas pada level Inheritance
RMK = Nilai Kompleksitas *Method* yang digunakan kembali
RAC = Nilai Kompleksitas *Attribute* yang digunakan kembali

Dimana CDC adalah *Cognitive depth Complexity* yang telah dihitung dengan menggunakan persamaan 6 [11]

$$CDC = DC * (CW_i)$$

… (6)

Keterangan :
DC = Level *Inheritance*
CW_1 = Bobot Kognitif Level *Inheritance*

Pengujian

Pengujian dilakukan dengan tujuan untuk menghitung nilai kompleksitas suatu program dengan metode CWICC dan untuk menemukan kesalahan pada perangkat lunak dan memperbaikinya. Selain itu juga untuk mengetahui apakah sistem dikembangkan berdasarkan kebutuhan yang telah didefinisikan. Pada penelitian ini, pengujian unit dan integrasi dilakukan dengan menggunakan metode *whitebox testing* yaitu dengan teknik *basis path testing*. Sedangkan metode *blackbox testing* digunakan pada pengujian validasi.

Tiga *method* pada sistem ini yaitu method chooseFile, visit, dan calculateAC akan diuji menggunakan pengujian unit dan menghasilkan 19 kasus uji. Pengujian integrasi bertujuan untuk mengetahui apakah method calculateAC pada kelas ACController berintegrasi dengan method setListOfAC pada kelas ACData, yang menghasilkan 6 kasus uji. Sedangkan pada pengujian validasi untuk kebutuhan fungsional dihasilkan 8 kasus uji sedangkan untuk kebutuhan non-fungsional dihasilkan dua kasus uji yaitu *accuration* (akurasi) yang ditunjukkan pada Tabel 2 dan pengujian *performance* (performa) yang ditunjukkan pada Tabel 3. Pada pengujian yang selesai dilaksanakan diperoleh hasil berupa status valid pada seluruh kasus uji.

Tabel 2 . Pengujian Akurasi Sistem

No	Nama Data Uji	Perhitungan Manual				Perhitungan Program			
		AC	MC	IICC	CWICC	AC	MC	IICC	CWICC
1	JavaModul6	5	32	6	43	5	32	6	43
2	Football-Project	7	12	23	38	7	12	22	37
3	CubeProject	20	43	7	74	18	40	7	65
4	miniShop2	32	51	53	138	32	49	53	136
5	JavaShop1	73	162	63	298	73	161	60	294
Jumlah Kolom yang berbeda				10	Prosentase Perbedaan				25%
Jumlah Semua Kolom				40	Prosentase Kesesuaian				75%

Pengujian akurasi bertujuan untuk memperoleh perbandingan pada hasil perhitungan kompleksitas secara manual dengan hasil perhitungan menggunakan sistem. Pengujian ini menggunakan 5 data uji berupa proyek perangkat lunak yang didapatkan dari sisa project java penulis secara random. Hasil yang diperoleh dari pengujian akurasi, bahwa nilai akurasi pada sistem ini sebesar 100%.

Pengujian performa dilaksanakan dengan cara menganalisis perbandingan waktu yang diperlukan sistem untuk melakukan perhitungan kompleksitas dengan waktu yang diperlukan pada perhitungan manual. Pada pengujian ini menggunakan 5 data uji dari data uji pada pengujian akurasi. Pada pengujian performa diperoleh hasil bahwa sistem dapat melakukan perhitungan nilai kompleksitas perangkat lunak rata-rata 360 kali lebih cepat dibandingkan perhitungan secara manual. Hal ini membuktikan bahwa sistem memiliki keunggulan berupa tingkat akurasi yang cukup tinggi dan waktu perhitungan yang lebih cepat.

Tabel 3 . Pengujian performa sistem

No	Nama data Uji	Kecepatan Perhitungan (ms)	Kecepatan Perhitungan Sistem (ms)	Perbandingan
1	JavaModul6	122000	391	310:1
2	Football-Project	153000	465	329:1
3	CubeProject	213000	731	291:1

4	miniShop2	351000	850	413:1
5	JavaShop1	631000	1371	460:1
		Rata Rata		360:1

4. KESIMPULAN

Dari penelitian ini didapatkan kesimpulan bahwa, hasil pada tahap rekayasa kebutuhan, diperoleh 4 kebutuhan fungsional dan dua kebutuhan non-fungsional. Hasil pada tahap perancangan, diperoleh perancangan arsitektur, perancangan komponen, dan perancangan antarmuka. Sedangkan pada tahap implementasi diperoleh hasil berupa spesifikasi sistem, implementasi kode program dan implementasi antarmuka. Sedangkan pada keseluruhan pengujian yang dilakukan diperoleh status valid dari seluruh kasus uji. Pengujian akurasi dari 5 data uji dihasilkan nilai akurasi sebesar 100%. Sedangkan pada pengujian performa sistem diperoleh hasil bahwa waktu yang diperlukan sistem untuk melakukan perhitungan kompleksitas perangkat lunak rata-rata 356 kali lebih cepat dibandingkan perhitungan manual.

DAFTAR PUSTAKA

[1] W. Widyawati, B. Priyambadha, and F. Pradana, "Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik Cognitive Weighted Inherited Class Complexity," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 3, no. 5, pp. 5201–5206, 2019.

[2] D. W. Utomo, D. Kurniawan, and Y. P. Astuti, "Teknik Pengujian Perangkat Lunak Dalam Evaluasi Sistem Layanan Mandiri Pemantauan Haji Pada Kementerian Agama Provinsi Jawa Tengah," *Simetris J. Tek. Mesin, Elektro dan Ilmu Komput.*, vol. 9, no. 2, pp. 731–746, 2018, doi: 10.24176/simet.v9i2.2289.

[3] H. Harliana and W. Widayani, "Analisis Dempster Shafer Pada Sistem Pakar Pendeteksi ISPA," *FAHMA*, vol. 17, no. 2, pp. 60–69, 2019, [Online]. Available: https://stmikelrahma.e-journal.id/FAHMA/article/view/34/22.

[4] A. Riza, M. A. I. Anshori, F. Arrazy, and M. A. Yaqin, "Pengukuran Metrik Kompleksitas Web Service Sekolah," *Jurasik (Jurnal Ris. Sist. Inf. dan Tek. Inform.*, vol. 5, no. 1, p. 147, 2020, doi: 10.30645/jurasik.v5i1.179.

[5] R. Tarigan and D. Ardiansyah, "Perancangan Aplikasi Inventory Barang Pada Cv. Mr Lestari Berbasis Web," *J. Sist. Inf. dan Inform.*, vol. 3, no. 2, pp. 77–94, 2020, doi: 10.47080/simika.v3i2.985.

[6] K. Maheswaran and A. Aloysius, "Cognitive Weighted Inherited Class Complexity Metric," *Procedia Comput. Sci.*, vol. 125, pp. 297–304, 2018, doi: 10.1016/j.procs.2017.12.040.

[7] H. Apriadi, F. Amalia, and B. Priyambadha, "Pengembangan Aplikasi Kakas Bantu Untuk Menghitung Estimasi Nilai Modifiability Dari Class Diagram," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 3, no. 11, pp. 10605–10613, 2019.

[8] E. N. I. Sukarno, "Penerapan Material Requirement Planning (MRP) dalam Perencanaan Persediaan Bahan Baku Pembuatan Produk Pia Kawitan Menggunakan Metode Lot For Lot dan Part Period Balancing," *Pros. Manaj. Semin. Penelit. Sivitas Akad. Unisba*, vol. 6, no. 2, pp. 1011–1016, 2020, doi: 10.29313/.v6i2.24316.

[9] M. C. Saputra *et al.*, "Perbandingan Antara Metode Advance Use Case Point Dan Revised Use Case Point Untuk Evaluasi Biaya Pengembangan Sistem Informasi Reservasi Ruangan," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 7, no. 1, 2020, doi: 10.25126/jtiik.201961329.

[10] E. A. Alomar, M. W. Mkaouer, A. Ouni, and M. Kessentini, "On the Impact of

Refactoring on the Relationship between Quality Attributes and Design Metrics," *Int. Symp. Empir. Softw. Eng. Meas.*, vol. 2019-Septemer, pp. 1–11, 2019, doi: 10.1109/ESEM.2019.8870177.

[11] M. A. Yaqin, D. E. A. Pratama, M. Rofi'uruttab, and I. D. Cahya, "Pengukuran Metrik Kompleksitas Model Proses Bisnis Sekolah," *Jurasik (Jurnal Ris. Sist. Inf. dan Tek. Inform.*, vol. 5, no. 2, p. 217, 2020, doi: 10.30645/jurasik.v5i2.207.