REPOSITOR, Vol. 5, No. 1, Februari 2023, Pp. 565-574
ISSN : 2714-7975
E-ISSN : 2716-1382

Rancang Bangun Perangkat Lunak Perhitungan Metrik Cyclomatic Complexity Berdasarkan Control Flow Graph Berbasis Web

Fajarisma Asfiana Putri[*1], Gita Indah Marthasari[2], Ilyas Nuryasin[3]
[1,2,3]Universitas Muhammadiyah Malang
fajarismaasfianaputri@webmail.umm.ac.id[*1], gita@umm.ac.id[2], ilyas@umm.ac.id[3]

Abstrak

Pengujian perangkat lunak merupakan salah satu fase yang penting untuk dilakukan dalam membangun perangkat lunak. Salah satu jenis pengujian yang digunakan adalah whitebox testing dengan metode cyclomatic complexity (CC). Salah satu cara yang dapat digunakan untuk pengujian metode cyclomatic complexity adalah dengan menghitung banyaknya node dan edge dari control flow graph (CFG) yang merepresentasikan suatu kode program. Tujuan penelitian ini adalah membuat sebuah perangkat lunak perhitungan otomatis nilai CC dari CFG kode sumber perangkat lunak. Sistem yang dibuat ini akan mendeteksi serta menghitung banyaknya node dan edge dari sebuah gambar CFG yang kemudian diproses untuk mencari nilai cyclomatic complexity berdasarkan metrik CC. Sistem yang dibuat kemudian diuji menggunakan kode sumber website infotech.umm.ac.id. Hasil pengujian yang didapatkan dari implementasi sistem terhadap website infotech adalah nilai cyclomatic complexity sebesar 1,4881 dengan tingkat kompleksitas sistem rendah, nilai resiko rendah, dan bad fix probability sebesar 5%.

Kata Kunci: *Cyclomatic Complexity, CFG, Software Testing*

Abstract

Software testing is one of the important phases to do in software development. One kind of software testing is whitebox testing with the cyclomatic complexity (CC) method. One way that can be used to test the cyclomatic complexity method is to count the number of nodes and edge of a control flow graph (CFG) which represents a program code. To create ease and efficiency in calculation and testing, the author makes an automatic node and edge counting system from CFG. This system can detect and count the number of nodes and edges of a CFG image which it is processed to find the cyclomatic complexity value based on the CC metric. The case study used in the implementation of this system is the infotech.umm.ac.id website. The test results obtained from the implementation of the system on the infotech website are the cyclomatic complexity value of 1,4881 with levels system complexity is low, low value risk, and bad fix probability is 5%.

Keywords: *Cyclomatic Complexity, CFG, Software Testing*

1. Pendahuluan

Pengembangan *software* telah mengalami banyak kemajuan. Pada saat ini perangkat lunak yang berkualitas sudah banyak diciptakan. Sehingga permintaan dalam pembuatan perangkat lunak juga semakin meningkat. Namun terkadang masih ditemukannya perangkat lunak dengan kualitas yang kurang baik dikarenakan tidak melalui proses pengujian. Hasil yang ditemukan adalah terdapat banyak *bug* atau *error* pada perangkat lunak yang sedang dipakai. Selain itu, pada saat membuat atau pada tahap pengembangan sistem, pihak *programmer* juga memiliki peluang yang besar untuk melakukan kesalahan. Hal yang dapat dilakukan untuk menghindari adanya kesalahan berupa *bug* atau *error* adalah dengan dilakukannya pengujian perangkat lunak.

Pengujian perangkat lunak atau yang disebut *software testing* merupakan sebuah aktivitas yang digunakan untuk meningkatkan kualitas perangkat lunak [1]. Aktivitas pengujian ini salah satu fase penting dalam SDLC (*Software Development Life Cycle*) yang digunakan untuk menilai kualitas perangkat lunak [2]. Aktivitas pengujian perangkat lunak ini dilakukan untuk menghindari adanya *error* atau *bug* yang dapat menghambat dari kinerja sistem kedepannya. Manfaat dari

penerapan pengujian perangkat lunak ini adalah *stakeholder* dan *developer* dapat mengetahui secara tepat mengenai kualitas suatu produk yang sedang dikembangkan [3].

Pada penelitian sebelumnya dilakukan pendeteksian garis dan lingkaran dengan menggunakan metode *Hough Trasnform*. *Hough trasnform* ini adalah dasar untuk mendeteksi garis lurus dari gambar berdasar hitam-putih. Penelitian ini menekankan pada dua poin penting yaitu *hough transform of line* dan *hough transform of circle*. Hasil penelitian ini adalah didapatkan hasil pendeteksian garis dan lingkaran secara akurat [4]. Pada penelitian sebelumnya, dilakukan pengujian untuk mengukur kualitas kegunaan pada *website*. Penelitian ini menghitung tingkat kemudahan dari *website* menggunakan *cyclomatic complexion*. Hasil yang didapat adalah tingkat kemudahan *website* dalam rentang angka 1-10 dari sepuluh *website e-commerce* yang diuji [5]. Penelitian selanjutnya yang digunakan penulis sebagai rujukan adalah penerapan *cycomatic complexity* test design pada sistem registrasi pasien IGD pada rumah sakit Wava Husada. Penelitian ini menghasilkan pengujian dari aplikasi tersebut dengan menerapkan *cyclomatic complexity*. Nilai hasil *cyclomatic complexity* menunjukkan hasil yang terukur dari proses penghitungan berdasarkan *flow control graph* [6].

Sebagai sistem dengan tingkat penggunaan tinggi oleh mahasiswa teknik informatika Universitas Muhammadiyah Malang, maka perlu dilakukan pengujian untuk menghitung tingkat kompleksitasnya yang dapat diukur dengan *cyclomatic*. Selain itu, belum ada penelitian yang menerapkan pengujian *whitebox* menggunakan inspeksi *cyclomatic* pada infotech.umm.ac.id ini. Pengujian *whitebox* pada infotech ini dilakukan untuk mendapatkan hasil pengukuran kuantitatif dari kompleksitas logika program. Hasil kuantitatif ini dapat digunakan untuk mengetahui kualitas perangkat lunak tertentu yang diwakili oleh *reusability*, *maintainability*, keamanan, dan prediksi kesalahan. Ini berfungsi sebagai panduan efisien untuk pengembangan kasus tes kedepannya [7].

Pengujian menggunakan inspeksi *cyclomatic* ini adalah pengujian dengan metrik terbaik jika dibandingkan dengan beberapa metrik kompleksitas yang lain, seperti LOC dan *Halstead's* [7]. Selain itu, penelitian yang lain juga menyebutkan, bahwa pengujian *cyclomatic* menghasilkan hasil pengujian secara kuantitatif yang mengukur jumlah jalur independen linear melalui kode sumber program [7].

Penggunaan *tools* dalam pengujian *whitebox* dinilai akan meningkatkan nilai efisiensi dan efektifitas pengujian. Hal ini dikarenakan pada pengembangan perangkat lunak akan memiliki banyak sumber kode program. Contohnya sistem Infotech yang berbasis *website*. Sistem ini memiliki banyak sumber kode program dengan ekstensi php dikarenakan menerapkan bahasa pemrogram php. adanya banyak sumber kode program maka akan memakan waktu yang lama dalam pengujian per *source code*. Sehingga muncullah ide untuk membuat *automated tools* untuk *whitebox testing* dengan menerapkan *cyclomatic complexcity* untuk menghitung nilai kompleksitas dari kode program. Sistem yang akan dibuat dapat menginputkan 3 file gambar *control flow graph* yang merepresentasikan 3 *source code*.php sebagai sample pengujian untuk membandingkan hasil prosentase antara pengujian pertama, kedua, dan seterusnya. Hasilnya adalah berupa CC *Metric* sebagai nilai kuantitatif kode program dan sistem yang diuji.

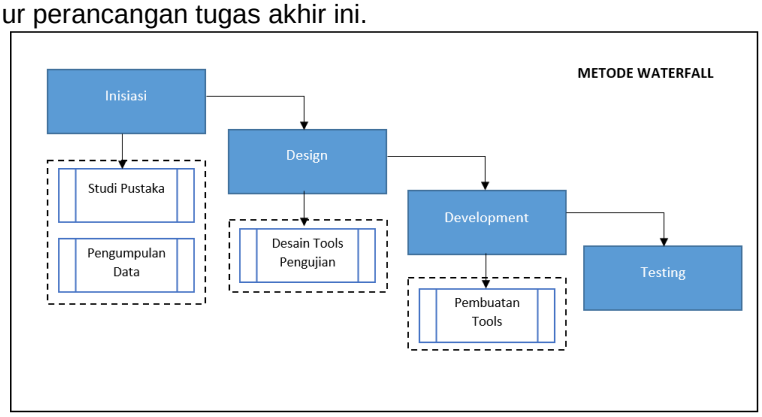
Penelitian ini bertujuan untuk membuat tools dalam membantu pengujian perangkat lunak dengan metode pengujian *whitebox* dengan menerapkan *cylomatic complexity*. Tools ini menerapkan *cyclomatic complexity* metode kedua, yaitu dengan Persamaan 1.

$$\mathrm{V(G)} = \mathrm{E} - \mathrm{N} + 2 \quad (1)$$

Rumus di atas dilakukan dengan menghitung hasil dari pengurangan dari *node* dan *edge* yang kemudian dijumlah dua. *Tools* ini merupakan sarana yang dapat membantu pengujian *whitebox*. Adanya *tools* ini diharapkan dapat meningkatkan efisiensi dan memudahkan dalam melakukan pengujian secara *whitebox*. Penerapan dari *tools* ini dilakukan pada sistem infotech Teknik Informatika Universitas Muhammadiyah Malang. Penerapan *tools* pada sistem Infotech adalah untuk mengetahui apakah *tools* yang dibuat telah sesuai dengan penerapan *whitebox cyclomatic complexity*. Selain itu, penerapan tools pada sistem infotech ini diharapkan dapat mengetahui tingkat kompleksitas dan kehandalan sistem sehingga dapat ditingkatkan kedepannya.

2. Metode Penelitian

Metode yang digunakan dalam penelitian ini merupakan dasar penyusunan rancangan penelitian dan merupakan penjabaran dari metode perancangan tools pada umumnya. Gambar 1 merupakan alur perancangan tugas akhir ini.



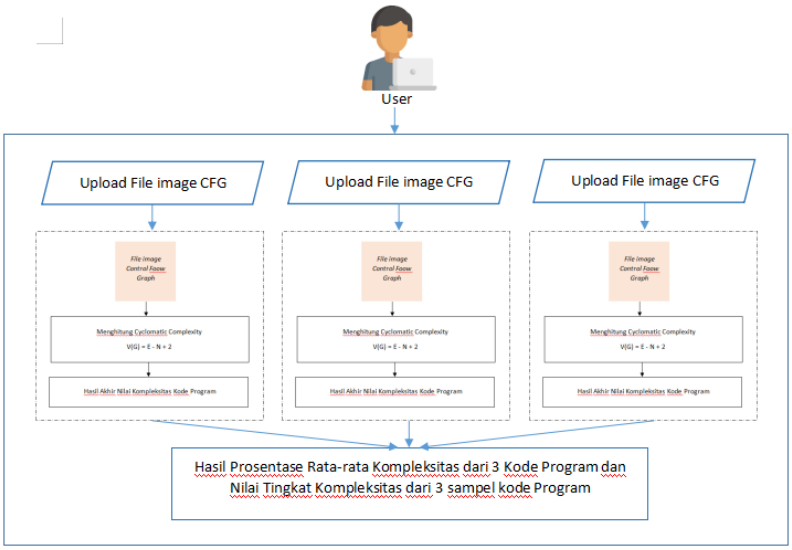
Gambar 1. Metode Pengerjaan Penelitian

2.1 Pengumpulan Data dan Analisis

Pengumpulan data ini dilakukan dengan melakukan diskusi dengan pihak *developer* dari sistem infotech Teknik Informatika Universitas Muhammadiyah Malang. Data yang dikumpulkan adalah berupa data mengenai kebutuhan *user* dari *tools* yang akan dikembangkan. Pengumpulan data dilakukan dengan proses perizinan kepada pihak laboratorium Universitas Muhammadiyah Malang. Hasil pengumpulan data yang akan diperoleh dari penelitian ini adalah dengan mengakses atau mendapatkan kode program (*source code*) sistem Infotech. Kode program ini akan diolah pada pengujian *whitebox* dengan *cyclomatic complexity* untuk memperoleh tingkat kompleksitas kode program yang merepresentasikan sistem tersebut. Selain itu, peneliti juga akan melakukan penggalian kebutuhan untuk memperoleh kebutuhan yang diinginkan oleh klien. Kebutuhan yang diperoleh dari klien ini digunakan sebagai dasar dalam memetakan kebutuhan user yang diselaraskan dengan fitur atau fungsi yang ada di sistem.

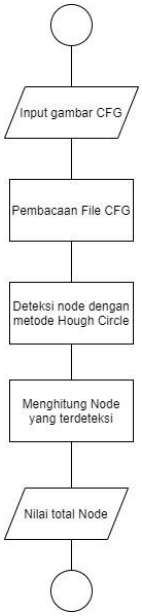
2.2 Desain

Tahap selanjutnya yaitu desain. Desain dilakukan sebelum proses *coding* dimulai. Ini bertujuan untuk memberikan gambaran lengkap tentang apa yang harus dikerjakan dan bagaimana tampilan dari sebuah sistem yang diinginkan. Sehingga membantu menspesifikan kebutuhan *hardware* dan sistem, juga mendefinisikan arsitektur sistem yang akan dibuat secara keseluruhan. Gambar 2 berikut adalah *design* dari *tools* pengujian yang akan dikembangkan.



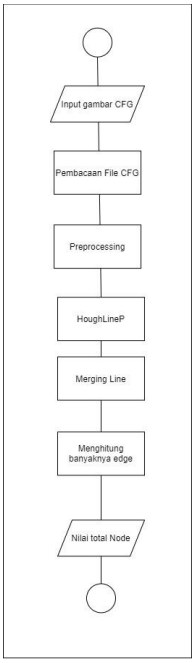
Gambar 2. Design Process dari Tools Pengujian

Selanjutnya adalah desain algoritma deteksi *circle* atau *node* dengan menggunakan *Hough Circle method*. Metode ini adalah metode sederhana yang digunakan untuk mendeteksi bentuk atau *shape* lingkaran. Gambar 3 berikut adalah *flowchart* sebagai penjelas alur deteksi *node* pada sistem ini yang mengimplementasikan *hough circle transform method* didalamnya.



Gambar 3. Flowchart deteksi node

Alur dalam pendeteksian node pada Gambar 4 ini adalah ketika gambar CFG telah selesai diinput, maka langkah selanjutnya adalah *pre-processing* yaitu pembacaan file CFG. Ketika pembacaan data berupa gambar CFG sudah selesai, maka akan dilanjutkan ke proses deteksi yang dilakukan oleh *method hough circle transform*. Metode ini mendeteksi *shape* atau bentuk lingkaran. *Node-node* yang terdeteksi ini kemudian dihitung dengan penggunaan fungsi *for* untuk *looping* yang kemudian akan mengembalikan nilai total keseluruhan *node* yang terhitung dalam satu gambar CFG.



Gambar 4. Flowchart deteksi edge

Setelah program berhasil menginput gambar *control flow graph*, maka selanjutnya dilakukan pembacaan *file image* tersebut. File gambar kemudian akan diteruskan ke tahap *preprocessing*. *Pre-processing* di sini mencakup merubah gambar menjadi *grayscale*, *treshold*, dan tahap awal implementasi *method HoughLine*. Konsep yang digunakan pada deteksi *edge* ini juga menerapkan *line merging dengan method HoughLine*. Implementasi dari *houghline* akan menghasilkan garis yang tidak beraturan. Kemudian konsep *line merging* akan diimplementasikan di tahap ini yaitu dari garis-garis tersebut dihitung jarak kedekatan, orientasi garis, dan perpotongan garis dan sudut garis. Garis dengan kedekatan tertentu dengan orientasi, dan sudut yang sama akan digabungkan menjadi sebuah garis lurus. Namun apabila ada perpotongan atau perbedaan sudut garis akan dianggap garis yang berbeda.

2.3 *Tools* Pengujian

Tools pengujian ini adalah berbasis website. Tools yang dibuat dapat mengambil 3 *source code* atau kode program dari suatu sistem. Kode program yang diambil adalah file dengan ekstensi.php. Masing-masing file.php ini akan diproses dengan melalui 3 proses, yaitu: (1) Mengkonversi Kode Program menjadi diagram alir atau *control flow graph*. Pengkonversian *source code* menjadi *control flow graph* adalah dengan merubah kode program menjadi *node* dan *edge*. Pengkonversian ini dilakukan secara manual oleh pihak penguji. *Node* dan *edge* ini kemudian diteruskan ke langkah berikutnya yaitu untuk menghitung nilai CC. (2) Menghitung *cyclomatic complexity* dengan menggunakan metode kedua seperti Persamaan 2 di atas. Nilai CC ini diperoleh dari hasil pengurangan dari banyaknya *node* dan *edge* yang kemudian ditambah dua. (3) Hasil akhir dari nilai kompleksitas kode program.

Setelah ketiga proses ini selesai, maka akan dilakukan pencarian nilai rata-rata prosentase dari ketiga program. Nilai prosentase ini kemudian didokumentasikan yang kemudian dilakukan lagi pengujian pada tiga sampel kode program lain. Nilai antara pengujian satu, dua, dan seterusnya dibandingkan untuk mengetahui nilai CC satu sistem secara penuh. Setelah nilai CC satu sistem diperoleh, kemudian akan dilakukan konversi dengan menggunakan CC *Metric* untuk mengetahui nilai parameter kompleksitas program tersebut.

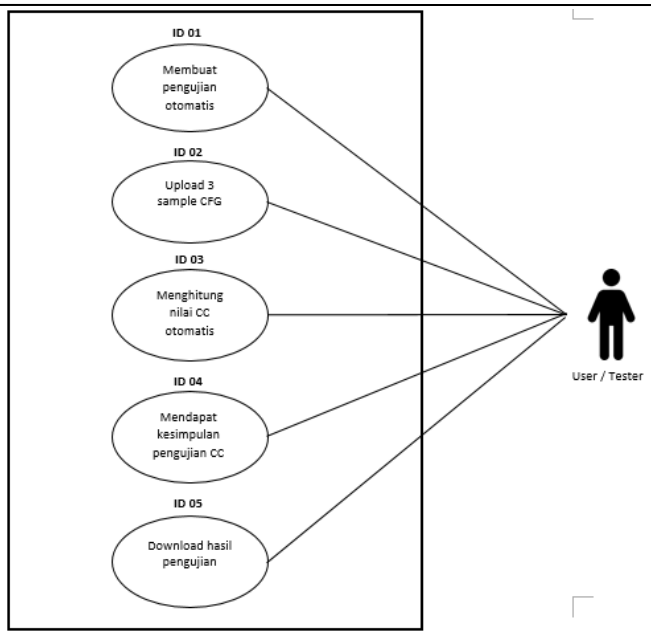
2.4 Pengujian

Pada tahap keempat ini akan dilakukan penggabungan modul-modul yang sudah dibuat sebelumnya. Setelah itu akan dilakukan pengujian yang bertujuan untuk mengetahui apakah *software* sudah sesuai desain yang diinginkan dan apakah masih ada kesalahan atau tidak. Pengujian ini akan dilakukan dengan metode *blackbox*. *Black Box Testing* atau yang sering dikenal dengan sebutan pengujian fungsional merupakan metode pengujian Perangkat Lunak yang digunakan untuk menguji perangkat lunak tanpa mengetahui struktur internal kode atau Program. Dalam pengujian ini, penguji menyadari apa yang harus dilakukan oleh program tetapi tidak memiliki pengetahuan tentang bagaimana melakukannya.

3. Hasil Penelitian dan Pembahasan

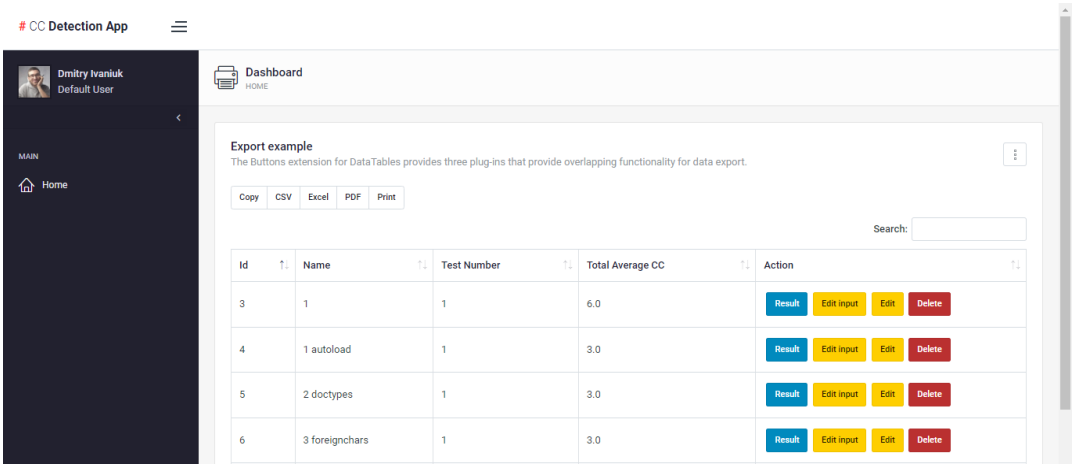
3.1 Hasil Analisis

Sistem infotech ini berbasis *website* dengan bahasa pemrograman.php pembangunnya. Sistem infotech ini menggunakan *framework Code Igniter* (CI) yang mana juga menerapkan konsep *model*, *view*, dan *controller* (MVC). Total file yang akan diimplementasikan dalam penelitian ini adalah sebanyak 84 file yaitu dari jumlah keseluruhan file *controller*. Penggunaan *file controller* pada penelitian ini adalah karena *file controller* berperan dalam bagian menerima perintah dan mengatur aplikasi untuk tugas dan tampilan yang sesuai, pengambilan data dari *model*, dan mengirimkan data ke *view*. Penggalian kebutuhan yang telah dilakukan oleh peneliti menghasilkan *use case* pada Gambar 5 berikut.



Gambar 5. Skema Usecase

3.2 Hasil Desain *Tools* Pengujian

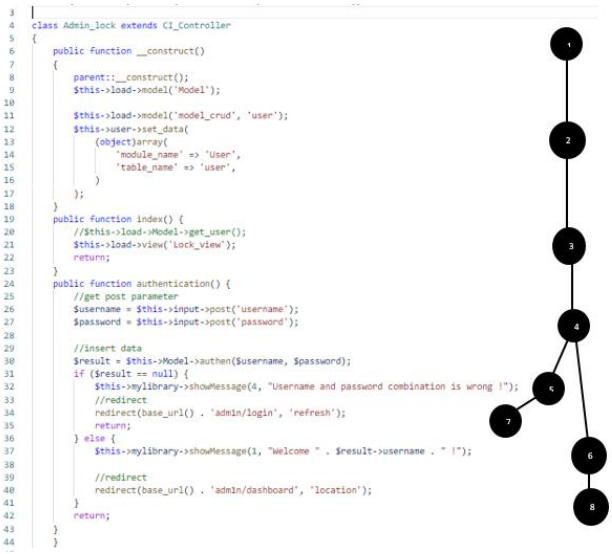


Gambar 6. Tampilan Antar Muka Sistem

Gambar 6 di atas menunjukkan tampilan antar muka sistem perhitungan nilai *Cyclomatic Complexity*. Tampilan ini akan menampilkan daftar nama pengujian yang disertai dengan nilai total rata-rata CC pada pengujian tersebut.

3.3 Hasil Konversi Kode Program *ke Control Flow Graph*

Peneliti menyiapkan beberapa sampel atau contoh hasil konversi kode program ke *control flow graph*. Konversi ini dilakukan secara manual dengan bantuan *tools online*. Berikut adalah beberapa sampel kode program beserta hasil konversi ke *control flow graph*.



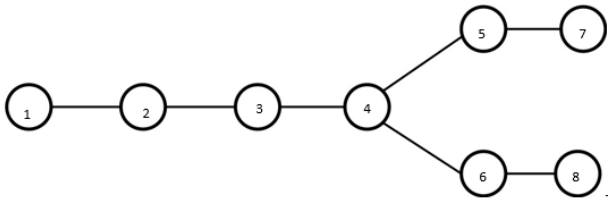
```
class Admin_lock extends CI_Controller
{
    public function __construct()
    {
        parent::__construct();
        $this->load->model('Model');

        $this->load->model('model_crud', 'user');
        $this->user->set_data(
            (object)array(
                'module_name' => 'User',
                'table_name' => 'user',
            )
        );
    }
    public function index() {
        //$this->load->Model->get_user();
        $this->load->view('Lock_view');
        return;
    }
    public function authentication() {
        //get post parameter
        $username = $this->input->post('username');
        $password = $this->input->post('password');

        //insert data
        $result = $this->Model->authen($username, $password);
        if ($result == null) {
            $this->mylibrary->showMessage(4, "Username and password combination is wrong !");
            //redirect
            redirect(base_url() . 'admin/login', 'refresh');
            return;
        } else {
            $this->mylibrary->showMessage(1, "Welcome " . $result->username . " !");

            //redirect
            redirect(base_url() . 'admin/dashboard', 'location');
        }
        return;
    }
    }
```

Gambar 7. Sample Konversi Kode program ke CFG



Gambar 8. Hasil Konversi admin_lock.php ke CFG

3.4 Hasil Implementasi Pembuatan *Tools*

Hasil implementasi pembuatan tools pengujian otomatis *cycomatic complexity* yang ditunjukkan oleh Gambar 9 di bawah ini merupakan hasil dari pengujian terhadap studi kasus yang diangkat dalam penelitian ini, yaitu sistem informasi infotech.umm.ac.id di Universitas Muhammadiyah Malang.

CC Calculation Result

edge 1	edge 2	edge 3	node 1	node 2	node 3	avg cc#
11.0	2.0	12.0	11.0	3.0	11.0	2.0
12.0	4.0	4.0	11.0	5.0	5.0	1.66667
12.0	4.0	3.0	11.0	5.0	4.0	1.66667
13.0	7.0	26.0	14.0	7.0	26.0	1.66667
13.0	3.0	3.0	16.0	4.0	4.0	0.333333
14.0	13.0	5.0	14.0	14.0	6.0	1.33333
17.0	5.0	10.0	16.0	6.0	10.0	2.0
19.0	7.0	10.0	19.0	8.0	8.0	2.33333
2.0	60.0	9.0	3.0	56.0	8.0	3.33333
3.0	11.0	4.0	4.0	10.0	7.0	1.0
3.0	49.0	6.0	4.0	50.0	7.0	1.0
31.0	12.0	12.0	26.0	13.0	11.0	3.66667
4.0	11.0	3.0	5.0	11.0	4.0	1.33333
40.0	6.0	51.0	47.0	6.0	52.0	1.66667
49.0	49.0	49.0	50.0	50.0	50.0	1.0
49.0	49.0	49.0	50.0	50.0	50.0	1.0
49.0	49.0	49.0	50.0	50.0	50.0	1.0
49.0	49.0	49.0	50.0	50.0	50.0	1.0
49.0	49.0	9.0	50.0	50.0	10.0	1.0
5.0	3.0	9.0	6.0	4.0	9.0	1.33333
6.0	6.0	21.0	7.0	7.0	19.0	2.0
6.0	15.0	21.0	7.0	16.0	20.0	1.66667
6.0	3.0	2.0	7.0	4.0	3.0	1.0
6.0	21.0	13.0	7.0	17.0	12.0	3.33333
7.0	4.0	11.0	8.0	5.0	13.0	0.666667
7.0	5.0	10.0	8.0	6.0	9.0	1.66667
7.0	6.0	5.0	8.0	7.0	6.0	1.0
9.0	9.0	4.0	9.0	10.0	5.0	1.33333
Summary:	Total avg CC	1.4881	Risk evaluation	Low risk testtable code	Bad fix probability	5%

Gambar 9. Hasil Pengujian CC Sistem Infotech

Pengujian ini diberi nama “Pengujian Sistem Ilab” dengan banyak iterasi sebanyak 28 iterasi dengan tiap-tiap iterasi diwakili oleh 3 file *image control flow graph*. Setiap *file image* akan dideteksi nilai *node*, *edge*, dan nilai rata-rata *cyclomatic complexity*. Kemudian semua iterasi akan dihitung untuk mendapatkan nilai *cyclomatic complexity* satu sistem dan tingkat kompleksitas serta tingkat *bad fix probability*. Seperti yang ditunjukkan pada Gambar 9 menunjukkan hasil bahwa nilai CC sebesar 1,4881 dengan tingkat kompleksitas rendah, nilai evaluasi resiko yang rendah dan bad fix probability bernilai 5%. *File control flow graph* adalah hal yang menentukan dalam pengujian ini. Hal ini dikarenakan *image* CFG berperan sebagai *file* input yang akan diproses untuk dideteksi dan dihitung *node*, *edge*, serta nilai CC. Jadi, hasil akhir pengujian bergantung pada gambar CFG sebagai *file input*. *Error* yang muncul dalam proses deteksi dan mengakibatkan nilai CC tidak muncul, hal ini juga dikarenakan algoritma pendeteksian yang kurang benar sehingga perlu dilakukan peningkatan akurasi pendeteksian (Ye et al., 2016).

Pada tahap ini dilakukan pengujian fungsional menggunakan *blackbox testing*. Pengujian ini dilakukan untuk mengetahui apakah aplikasi ini sudah sesuai dengan kebutuhan user atau masih perlu ada perbaikan. Pengujian ini difokuskan pada fungsi dari setiap tombol yang ada pada aplikasi.

Tabel 1. Hasil Pengujian Black Box Sistem

Test Case	Hasil yang diharapkan	Hasil pengujian	*Status*
Fitur *input* nama pengujian	Dapat meng*input*kan nama pengujian serta banyak interasi pengujian	Dapat menginputkan nama pengujian	Sesuai
Fitur *input* gambar *control flow graph*	Dapat meng*input*kan 3 *file image* CFG untuk tiap iterasi	Dapat menginputkan 3 *file* tiap iterasi	Sesuai
Fitur *input* gambar *control fow graph* dengan *node* berbentuk selain lingkaran	Tidak dapat memproses CFG dengan *node* selain bentuk lingkaran dengan menampilkan *error*	*Error identified* (*attribute error*)	Sesuai
Fitur deteksi *node* dan *edge*	Dapat mendeteksi dan menampilkan nilai *node* dan *edge*	Dapat mendeteksi *node* dan *edge*	Sesuai
Fitur Nilai CC dan pengukuran dengan CC Metrik	Dapat menampilkan hasil nilai CC dan pengukuran sesuai CC Metrik	Dapat menampilkan nilai CC sesuai CC *metric*	Sesuai

Dari hasil pengujian menggunakan metode *black box* pada Tabel 1, maka hasilnya adalah keseluruhan fungsional dari aplikasi berfungsi dengan baik sesuai dengan apa yang dibutuhkan *user*.

4. Kesimpulan

4.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan dapat di ambil beberapa kesimpulan, yaitu:

a. *Tools* dapat berjalan sesuai dengan fungsinya.

Tools pendeteksian nilai *cyclomatic complexity* ini telah berjalan sesuai dengan fungsinya. Hal ini dibuktikan dengan pengujian black box terhadap sistem. Hasil pengujian *black box* telah dipaparkan pada Tabel 1.

b. *Tools* ini dapat mengetahui tingkat kompleksitas sistem infotech

Tools yang telah dibuat ini dapat mengetahui tingkat kompleksitas sistem infotech sebagai studi kasus yang diangkat dalam penelitian ini. Implementasi dari sistem pengujian otomatis ini adalah didapatkannya nilai CC sebesar 1,4881 dengan tingkat kompleksitas rendah, nilai evaluasi resiko yang rendah dan *bad fix probability* bernilai 5%.

c. Hasil pengujian yang didapatkan bergantung pada *image control flow graph* dan algoritma *Hough Line and Hough Circle Transform*

Tools yang telah dibuat ini dapat menerima dan memperoses image control flow graph (CFG). Oleh karena itu, hasil perhitungan *Cyclomatic Complexity* (CC) bergantung pada *image* CFG serta kinerja *algoritma hough line* dan *hough circle transfom*.

4.2 Saran

Untuk pengembangan sistem lebih lanjut lagi supaya mendapatkan hasil yang lebih baik, maka penulis menyarankan beberapa hal untuk dapat disempurnakan di penelitian yang lebih lanjut kedepannya :

a. Terdapat batasan ekstensi *image* yang dapat diinputkan ke dalam program. Program ini hanya menerima ekstensi file image berupa JPG atau JPEG saja. Maka dari itu diharapkan kedepannya sistem dapat lebih fleksibel memperoses *file* yang berisi control flow graph.
b. Sistem ini hanya dapat memproses gambar *control flow graph* dengan ukuran tertentu, yaitu gambar *node* dan *edge* harus jelas. Ukuran *border node* dan *edge* adalah minimal 6 ptx dan 4 ptx pada *border node* dan *edge*. *File image* dengan *control flow graph* yang kurang jelas atau ukuran *node* dan *edge* terlalu kecil akan mempengaruhi hasil pendeteksian sehingga menjadi tidak akurat. Maka dari itu, harapan kedepannya adalah meningkatkan tingkat pendeteksian yang dapat mendeteksi *node* dan *edge* dengan berbagai ukuran dan kualitas gambar.
c. Data yang harus disiapkan adalah gambar *control flow graph*. Namun pada penelitian ini proses pengubahan kode sumber program ke *control flow graph* masih dilakukan secara manual dengan bantuan *tools online*. Maka dari itu, diharapkan untuk penelitian selanjutnya adalah dapat dibuat konversi kode program menjadi *control flow graph* secara otomatis.

5. Daftar Notasi

di mana :
E = jumlah total edge pada flowgraph atau grafik alir
N = jumlah total node pada flowgraph

Referensi

[1] T. Jindal, “Importance of Testing in SDLC,” *Int. J. Eng. Appl. Comput. Sci.*, vol. 01, no. 02, pp. 54–56, 2016, doi: 10.24032/ijeacs/0102/05.
[2] M. S. A. Kesuma Jaya, P. Gumilang, T. Wati, Y. P. Andersen, and T. Desyani, “Pengujian Black Box pada Aplikasi Sistem Penunjang Keputusan Seleksi Calon Pegawai Negeri Sipil Menggunakan Teknik Equivalence Partitions,” *J. Inform. Univ. Pamulang*, vol. 4, no. 4, p. 131, 2019, doi: 10.32493/informatika.v4i4.3834.
[3] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, “Software testing techniques: A literature review,” *Proc. - 6th Int. Conf. Inf. Commun. Technol. Muslim World, ICT4M 2016*, pp. 177–182, 2017, doi: 10.1109/ICT4M.2016.40.
[4] H. Ye, G. Shang, L. Wang, and M. Zheng, “A new method based on hough transform for quick line and circle detection,” *Proc. - 2015 8th Int. Conf. Biomed. Eng. Informatics, BMEI 2015*, no. Bmei, pp. 52–56, 2016, doi: 10.1109/BMEI.2015.7401472.
[5] A. Saifudin, Y. Heryadi, and Lukas, “Ensemble Undersampling to Handle Unbalanced Class on Cross-Project Defect Prediction,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 662, no. 6, 2019, doi: 10.1088/1757-899X/662/6/062012.
[6] H. L. P. Y. T. A. Candra, “Cyclomatic Complexity Test Design Flowgraph Registration of Emergency Installation Patients in Wava Husada Hospital Using SEM,” *Int. J. Sci. Res.*, vol. 6, no. 8, pp. 1983–1987, 2017, doi: 10.21275/ART20176323.
[7] M. Najm and A. Jader, “Calculating Mccabe ’ S Cyclomatic Complexity Metric and Its Effect on,” *Int. J. Innov. Res. Creat. Technol.*, vol. 3, no. 5, pp. 10–22, 2018.