Instrumentasi Kode Program Secara Otomatis untuk *Basis Path Testing*

Raden Asri Ramadhina Fitriani(G64154007)*, Irman Hermadi

Abstrak/*Abstract*

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan dan bertujuan untuk menemukan kesalahan pada perangkat lunak. *Basis Path testing* merupakan salah satu metode pengujian struktural yang menggunakan *source code* dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Idealnya, pengujian dilakukan untuk semua kemungkinan dari perangkat lunak. Kumar dan Mishra (2016) mengatakan bahwa pengujian perangkat lunak menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Sehingga mengotomasi bagian dari pengujian akan membuat proses ini menjadi lebih cepat dan mengurangi kerawanan akan kesalahan. Pada penelitian ini, akan dibangun sebuah sistem untuk membangkitkan kemungkinan jalur-jalur dari sebuah program yang dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur yang dilalui program ketika dijalankan dengan masukan data uji tertentu, maka sistem ini juga akan melakukan instrumentasi *source code* program secara otomatis. Program yang akan diuji dalam penelitian ini adalah program yang dibangun dengan menggunakan bahasa Matlab. Dalam pengembangannya, aplikasi ini akan dibangun dengan menggunakan bahasa pemrograman Java dan library *Graphviz* untuk memvisualisasikan *Control Flow Graph*.

Kata Kunci

Basis Path Testing; *Control Flow Graph*; Instrumentasi

***Alamat Email**: radenasrirf@gmail.com

PENDAHULUAN

Latar Belakang

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan. Proses ini bertujuan untuk menemukan kesalahan pada perangkat lunak. Saat pengujian, bisa saja tidak ditemukan kesalahan pada Hasil pengujian. Hal ini dapat terjadi karena perangkat lunak yang sudah berkualitas tinggi atau karena proses pengujiannya berkualitas rendah. (Myers *et al.* 2012)

Teknik pengujian secara umum dibagi menjadi 2 kategori diantaranya *black box testing* dan *white box testing*. *Black box testing* bertujuan untuk memeriksa fungsional dari perangkat lunak apakah output sudah sesuai dengan yang ditentukan. Sedangkan *White box testing* atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur logika suatu proses. *Basis Path testing* merupakan salah satu metode pengujian struktural yang menggunakan *source code* dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu 2015). Untuk melakukan monitoring jalur mana yang diambil oleh sebuah masukan pada saat eksekusi program, maka diperlukan penanda yang dapat memberikan informasi cabang mana yang dilalui. Proses menyisipkan tanda tersebut disebut instrumentasi. Biasanya tanda tersebut disisipkan tepat sebelum sebuah percabangan (Tikir dan Hollingsworth 2011).

Idealnya, pengujian dilakukan untuk semua kemungkinan dari perangkat lunak. Tetapi untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Kumar dan Mishra (2016) mengatakan bahwa pengujian perangkat lunak menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Jika proses pengujian perangkat lunak dapat dilakukan secara otomatis, maka hal ini dapat mengurangi biaya pengembangan secara signifikan.

Hermadi (2015) melakukan penelitian untuk membangkitkan data uji untuk *path testing* menggunakan algoritma genetika. Dalam penelitian tersebut, Hermadi membangkitkan *Control flow Graph* (CFG) dan instrumentasi masih secara manual sehingga membutuhkan banyak waktu dan rawan akan kesalahan ketika program sudah semakin besar. Sehingga mengotomasi hal tersebut dapat membuat *path testing* menjadi lebih cepat dan mengurangi kerawanan akan kesalahan.

Pada penelitian ini, akan dibangun sebuah perangkat lunak untuk membangkitkan kemungkinan jalur dari

sebuah program. Jalur-jalur ini dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur mana yang dilalui ketika diberikan masukan data uji, maka sistem ini juga akan melakukan penyisipan tag-tag sebagai instrumentasi ke dalam *source code* secara otomatis.

Perumusan Masalah

Berdasarkan latar belakang di atas dapat dirumuskan masalahnya adalah bagaimana membangun sebuah aplikasi untuk melakukan instrumentasi secara otomatis untuk pengujian jalur dan *re-engineering* perangkat lunak.

Tujuan

Penelitian ini bertujuan untuk membangun sebuah aplikasi yang dapat digunakan untuk membangkitkan CFG dan melakukan instrumentasi secara otomatis.

Ruang Lingkup

Bahasa pemrograman yang diakomodasi adalah Matlab dan model diagram yang dibangkitkan adalah CFG.

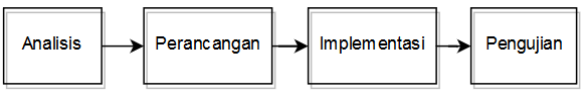
Manfaat

Hasil penelitian diharapkan dapat membantu pengembang dan penguji aplikasi untuk:

1. Menyisipkan tag-tag sebagai instrumentasi program ke dalam *source code* secara otomatis sehingga proses tersebut dapat dilakukan dengan lebih cepat.
2. Membangkitkan jalur-jalur dasar yang dapat digunakan sebagai dasar untuk pembangkitan data uji.
3. Membangkitkan diagram CFG yang dapat memudahkan pengembang dalam memahami struktur dan alur dari suatu program yang dapat dimanfaatkan ketika akan melakukan *re-engineering* perangkat lunak.

METODE PENELITIAN

Penelitian yang dilakukan terbagi menjadi beberapa tahapan proses. Gambar 1 menunjukan tahapan proses tersebut.



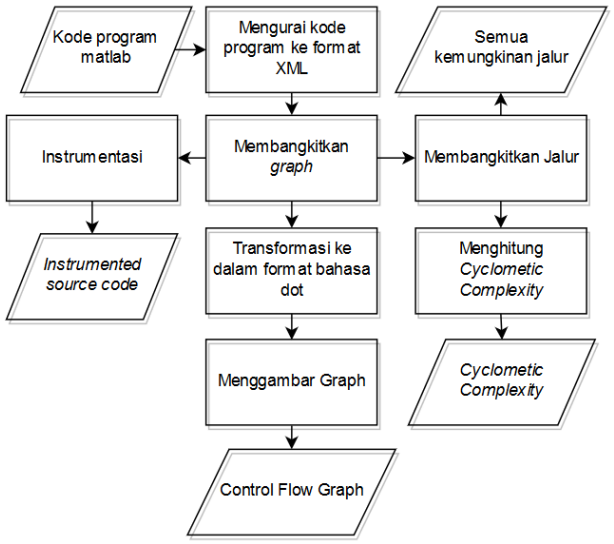
Gambar 1. Metode Penelitian

Analisis

Pada tahap ini dimulai dari membaca literatur terkait dan mendefinisikan kebutuhan dari aplikasi yang akan dibangun. Selain itu, pada tahapan ini juga dilakukan pengumpulan berapa contoh program yang akan digunakan dalam penelitian. Contoh program yang digunakan dalam penelitian ini didapatkan dari penelitian yang dilakukan oleh Hermadi (2015).

Perancangan

Pada tahap ini ditentukan bagaimana perangkat lunak akan dibangun. Ilustrasi arsitektur sistem dapat dilihat pada Gambar 2.



Gambar 2. Arsitektur Sistem

Kode Program

Kode program matlab akan dibaca sebagai inputan. Matlab memiliki empat struktur kontrol, yaitu IF-ELSE-END, SWITCH-CASE, FOR, dan WHILE. Kode program yang diinputkan harus sudah dipastikan dapat dijalankan jika di compile.

Mengurai Kode Progam ke Format XML

Penguraian kode program matlab dilakukan dengan menggunakan library MATLAB-PARSER. Ketika terdapat kesalahan pada kode program, library ini akan mengembalikan pesan error. Lalu kode program tersebut diurai menjadi file dengan format XML menggunakan library MATLAB-PARSER yang dibuat oleh Suffos (2015).

Extensible Markup Language (XML) adalah bahasa yang dapat mendeskripsikan sebuah dokumen. XML memiliki banyak bagian yang tidak memiliki struktur yang pasti. XML terdiri atas dua bagian utama, yaitu

elemen dan atribut. Elemen yang dapat disebut sebagai node merupakan bagian penting yang dapat menggambarkan struktur dari XML. Sedangkan atribut merupakan bagian yang dapat digunakan sebegai informasi tambahan dari setiap elemen (Hartwell 2017).

Membangkitkan *Graph*

Setiap dalam tag file XML tersebut akan ditelusuri satu persatu yang termasuk struktur kontrol di dalam bahasa matlab. Sehingga terbentuklah sebuah objek graph yang terdiri dari sekumpulan *node* dan *edge*.

Salah satu cara untuk membaca dan menulis dokumen XML pada framework .NET dan C# yaitu dengan menggunakan kelas XMLDocument yang terdapat dalam namespace System.XML. Setiap elemen XML yang merupakan struktur kontrol pada program akan menjadi nodes baru di dalam kelas graph. Setiap node berisi informasi nomor baris dan kolom yang akan digunakan untuk melakukan instrumentasi.

Membangkitkan Jalur

Basis path testing merupakan salah satu metode pengujian struktural yang menggunakan *source code* dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu 2015). Metode ini terbagi menjadi 4 tahapan, yaitu:

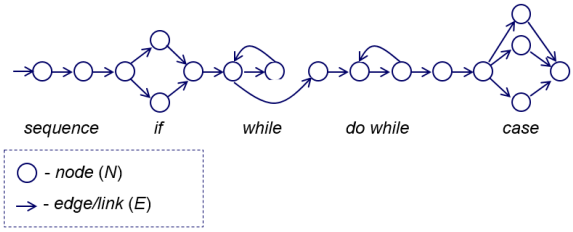
1. Menggambarkan jalur dalam bentuk *Control Flow Graph* (CFG)
2. Menghitung *cyclomatic complexity*
3. Memilih satu set jalur dasar
4. Membangkitkan data uji untuk setiap jalur dasar

Transformasi ke Dalam Format Bahasa Dot

Graph yang sudah terbentuk akan ditransformasikan ke dalam bentuk format bahasa permrograman dot. Bahasa dot adalah bahasa yang digunakan untuk mengambar graph berarah. Bahasa ini dapat mendeskripsikan 3 macam objek, yaitu graph, nodes, dan edges (Ganser et all 2015).

Memvisualisasi *Graph* dalam bentuk CFG

Control Flow Graph (CFG) adalah graph berarah yang merepresentasikan aliran dari sebuah program. Setiap CFG terdiri dari *nodes* dan *edges*. *Nodes* merepresentasikan *statement* atau *expressions*. Sedangkan *edges* merepresentasikan transfer kontrol antar *nodes* (Watson dan McCabe 1996). Notasi dari CFG dapat dilihat pada Gambar 3. Setelah file dengan format bahasa dot terbentuk, CFG akan divisualisasikan dengan menggunakan library Graphviz. Graphviz merupakan perangkat lunak open source untuk visualisasi grafik.



Gambar 3. Notasi Control Flow Graph (CFG)

Menghitung *Cyclometic Complexity*

Cyclomatic complexity merupakan suatu sistem pengukuran yang ditemukan oleh Watson dan McCabe untuk menentukan banyaknya *independent path* dan menunjukan tingkat kompleksitas dari suatu program. *Independent path* adalah jalur yang melintas dalam program yang sekurang-kurangnya terdapat kondisi baru. Perhitungan *Cyclomatic Complexity* dapat dilihat pada persamaan berikut:

$$V(G) = E - N + 2$$

Dimana, E menunjukkan jumlah *edges* dan N menunjukkan jumlah *nodes*.

Instrumentasi

Setelah jalur terbentuk, dilakukan juga proses instrumentasi. Instrumentasi merupakan sebuah proses menyisipkan sebuah penanda (tag) di awal atau di akhir setiap blok kode seperti awal setiap perintah, sebelum atau sesudah kondisi terpenuhi atau tidak. Dalam pengujian path testing, penanda ini dapat digunakan untuk memonitor jalur yang dilalui program ketika dijalankan dengan masukan data uji tertentu (Arkeman et al. 2014).

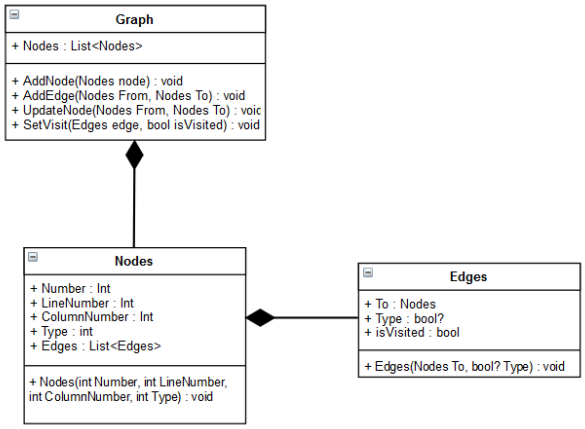
Instrumentasi dilakukan dengan cara menambahkan dulu variable keluaran bernama traversedPath. Variable in digunakan untuk menyimpan informasi node mana saja yang dilalui ketika diberikan inputan dengan nilai tertentu. Lalu setiap sebelum dan sesudah *node* percabangan, dilakukan penyisipan kode program berupa perintah untuk memasukkan nilai *node* yang dilalui. Sehingga ketika program tersebut dijalankan, akan menghasilkan keluaran tambahan bernama *traversedPath*.

Implementasi

Tahapan ini adalah melakukan implementasi dari tahap sebelumnya ke dalam bentuk aplikasi web. Aplikasi ini

akan dibangun dengan menggunakan bahasa pemrograman C dan menggunakan IDE Microsoft Visual Studio Ultimate 2013. Library yang dibutuhkan adalah library MATLAB-PARSER yang dibuat oleh Suffos (2015) untuk mengurai kode program ke dalam bentuk XML.

Setelah file dengan format bahasa dot terbentuk, CFG divisualisasikan dengan menggunakan library Graphviz.l Graphviz.Net adalah pembungkus C untuk generator grafik Graphviz yang dibuat oleh Dixon (2013). Keluaran yang dikembalikan ketika mengeksekusi Graphviz.Net berbentuk byte dalam array sehingga dapat diolah kembali sesuai dengan kebutuhan. Graphviz merupakan library yang dapat digunakan untuk menvisualisasi jalur ke dalam bentuk graph berarah (Gansner 2015).



Gambar 4. *Perancangan Class Diagram*

Testing

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Evaluasi dilakukan dengan membandingkan hasil yang dikeluarkan oleh sistem dengan pembangkitan secara manual dari segi waktu eksekusi.

HASIL DAN PEMBAHASAN

Analisis

Pada penelitian ini, akan dibangun sebuah perangkat lunak untuk membangkitkan kemungkinan jalur dari sebuah program. Jalur-jalur ini dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur mana yang dilalui ketika diberikan masukan data uji, maka sistem ini juga akan melakukan penyisipan tag-tag sebagai instrumentasi ke dalam kode program secara otomatis.

Sebelumnya sudah terdapat beberapa program yang dapat membangkitkan CFG seperti Eclipse Control Flow Graph Generator tetapi *library* tersebut hanya dapat digunakan di eclipse dan hanya membangkitkan CFG dari kode program java.

Data yang digunakan dalam penelitian ini didapatkan dari penelitian yang dilakukan oleh Hermadi (2015). Terdapat 15 contoh program yang akan digunakan pada penelitian ini dengan tingkat kompleksitas yang beragam. Contoh program yang akan digunakan dapat dilihat pada 1.

Perancangan

Perancangan *Class Diagram*

Class diagram dibangun untuk menggambarkan struktur sistem dari segi pendefinisian class-class. Perancangan Class diagram dapat dilihat pada Gambar 4. *Class graph* memiliki atribut berupa list dari class node. Dalam sebuah *class node* terdapat informasi nomor *node*, nomor baris dan kolom dari kode program, tipe dari perintah tersebut. Selain itu, terdapat list edge yang berisi *node* tujuan dan tipe dari *edge* yang digunakan jika terdapat percabangan apakah *true*, *false*, atau hanya garis penghubung biasa.

Perancangan Antarmuka

Perancangan antarmuka meliputi perancangan antarmuka form untuk pengguna memasukkan kode program yang akan di proses dan antarmuka hasil dari proses yang telah dilakukan. Perancangan antarmuka hasil dari proses yang telah dilakukan dapat dilihat pada Gambar 5.

Implementasi

Aplikasi dibangun dengan menggunakan bahasa pemrograman C# dan menggunakan IDE Microsoft Visual Studio Ultimate 2013.

Sebagai contoh, kode program yang digunakan adalah tA2008. Pada kode tA2008 terdapat perintah IF-THEN-ELSE bersarang sebanyak tiga tingkat.

Kode Program

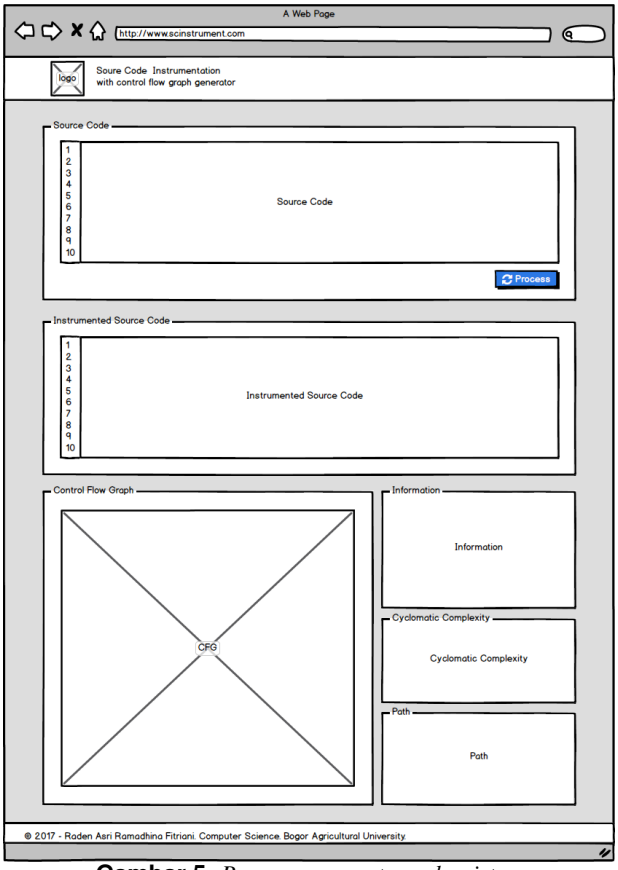
Gambar 6 merupakan kode program tA2008, yaitu fungsi untuk mencari jenis dari segitiga jika diketahui panjang dari setiap sisinya.

Mengurai Kode Progam ke Format XML

Penguraian kode program matlab dilakukan dengan menggunakan *library* MATLAB-PARSER. Kode program yang diinputkan harus sudah dipastikan dapat dijalankan jika di compile. Ketika terdapat kesalahan pada kode program, library ini akan mengembalikan pesan error. 7 menunjukkan potongan hasil penguraian kode program

Tabel 1. Contoh program uji

No	Program Uji	Nama	Deskripsi
1	triangleAhmed2008	tA2008	Menentukan tipe dari segitiga apakah termasuk equilateral, isosceles, scalene, atau not triangle
2	minimaxiAhmed2008	mmA2008	Menentukannilai minimal dan maksimal dari inputan berupa bilangan dalam array
3	insertionAhmed2008	iA2008	Mengurutkan bilangan dalam array menggunakan metode insertion sort
4	binnaryAhmed2008	binA2008	mencari indeks sebuah bilangan dalam array dengan mengembalikan indeks jika ditemukan dan tidak jika tidak ditemukan.
5	bubbleAhmed2008	bubA2008	Mengurutkan bilangan dalam array menggunakan metode bubble sort
6	gcdAhmed2008	gA2008	Menghitung GCD atau pembagi dua bilangan terbesar
7	expintBueno2002	eB2002	Fungsi exponensial yang dapat memproses bilangan integer dan float
8	quotientBueno2002	qB2002	Menghitung hasil bagi dan sisa hasil bagi dari dua buah bilangan bulat positif
9	findBueno2002	fB2002	Mengurutkan bilangan dalam array sebagian
10	fitnessMiniMaxiHermadi2014	fmH2014	Menghitung fungsi fitness dari fungsi minimaxiAhmed2008



Gambar 5. *Perancangan antarmuka sistem*

```
1  function type = triangle(sideLengths)
2  A = sideLengths(1); % First side
3  B = sideLengths(2); % Second side
4  C = sideLengths(3); % Third side
5  if ((A+B > C) && (B+C > A) && (C+A > B))
6        if ((A ~= B) && (B ~= C) && (C ~= A))
7              type = 'Scalene';
8        else
9              if (((A == B) && (B ~= C)) || ((B == C) && (C ~= A)) || ((C ==
                     A) && (A ~= B)))
10                   type = 'Isosceles';
11             else
12                   type = 'Equilateral';
13             end
14       end
15 else
16       type = 'Not a triangle';
17 end
```

Gambar 6. *Kode program tA2008*

ke dalam format XML. Potongan kode XML yang terlihat pada 7 menunjukkan hasil penguraian dari kode program pada baris ke 6 sampai baris ke 7.

Membangkitkan *Graph*

Salah satu cara untuk membaca dan menulis dokumen XML pada framework .NET dan C# yaitu dengan menggunakan kelas XMLDocument yang terdapat dalam *namespace System.XML*. Setiap elemen XML yang merupakan struktur kontrol pada program akan menjadi *node* baru di dalam kelas *graph*. Setiap *node* berisi informasi nomor baris dan kolom yang akan digunakan untuk melakukan instrumentasi. Setiap *node* juga dapat memiliki *edge* yang berisi informasi *node* tujuan dan tipe dari garis penghubung itu sendiri. Terdapat tiga macam tipe pada edge yaitu *null*, *true*, dan *false*. *True* dan *false* digunakan jika *node* asal merupakan percabangan.

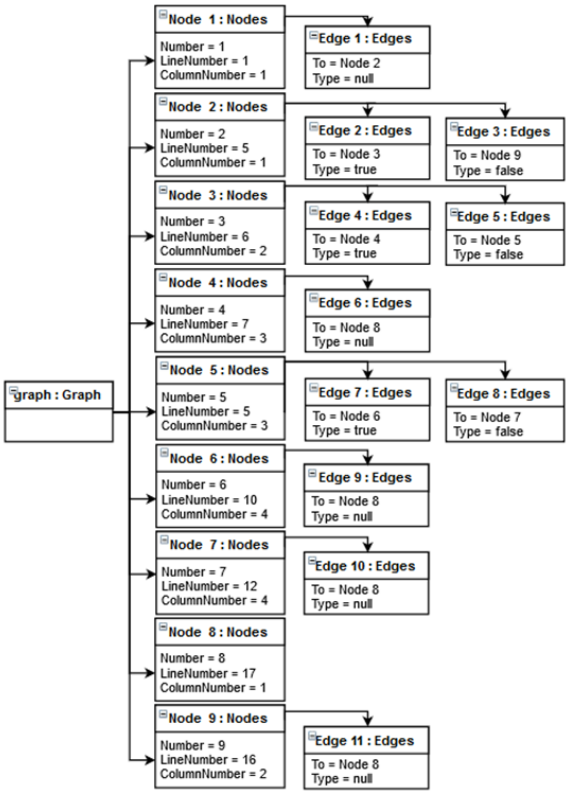
Hasil *node* yang dibentuk dari kode program tA2008 dapat dilihat pada 8. *Node* 1 dibentuk pada awal kode program sebagai inisialisasi. *Node* ditambahkan ketika bertemu dengan perintah yang termasuk ke dalam struktur kontrol seperti IF-ELSE-END, SWITCH-CASE, FOR, dan WHILE. Seperti dapat dilihat pada baris kode ke 5, terdapat perintah IF sehingga dibentuk node baru yaitu *node* 2. Representasi objek dari kelas *graph* yang terbentuk dari kode program tA2008 dapat dilihat pada 9. Terbentuk 9 buah *node* dan 11 buah *edge* yang menghubungkan antar *node* tersebut.

```
<IfPart.Statements>
  <If Line="6" Column="2" Text="if">
    <If.IfPart>
      <IfPart Line="6" Column="2" Text="if">
        <IfPart.Expression>
          <ShortAnd Line="6" Column="27" Text="&amp;&amp;">
        </IfPart.Expression>
        <IfPart.Statements>
          <Assignment Line="7" Column="8" Text="=">
            <Assignment.LValue>
              <Var Line="7" Column="3" Text="">
                <Var.Name>
                  <Name Line="7" Column="3" Text="">
                    <Name.Ids>
                      <Id Line="7" Column="3" Text="type" />
                    </Name.Ids>
                  </Name>
                </Var.Name>
              </Var>
            </Assignment.LValue>
            <Assignment.Value>
              <String Line="7" Column="10" Text="'Scalene'" />
            </Assignment.Value>
            <Assignment.Terminator>
              <NoPrint Line="7" Column="19" Text=";" />
            </Assignment.Terminator>
          </Assignment>
        </IfPart.Statements>
      </IfPart>
    </If.IfPart>
    <If.ElsePart>
      <ElsePart Line="8" Column="2" Text="else">
```

Gambar 7. Potongan hasil penguraian kode program tA2008 ke dalam format XML

```
1   Node 1   function type = triangle(sideLengths)
2                A = sideLengths(1); % First side
3                B = sideLengths(2); % Second side
4                C = sideLengths(3); % Third side
5   Node 2       if ((A+B > C) && (B+C > A) && (C+A > B))
6   Node 3           if ((A ~= B) && (B ~= C) && (C ~= A))
7   Node 4               type = 'Scalene';
8                    else
9   Node 5               if (((A == B) && (B ~= C)) || ((B == C) && (C ~=
                         A)) || ((C == A) && (A ~= B)))
10  Node 6                   type = 'Isosceles';
11                       else
12  Node 7                   type = 'Equilateral';
13                       end
14                   end
15               else
16  Node 9           type = 'Not a triangle';
17               end
    Node 8   end
```

Gambar 8. Hasil nodes yang terbentuk dari tA2008



Gambar 9. Object diagram tA2008

```
digraph G {
graph [label="" nodesep=0.8]
1->2;
2->3 [ label="True"  fontsize=10 ];
3->4 [ label="True"  fontsize=10 ];
3->5 [ label="False"  fontsize=10 ];
5->6 [ label="True"  fontsize=10 ];
5->7 [ label="False"  fontsize=10 ];
7->8;
6->8;
4->8;
2->9 [ label="False"  fontsize=10 ];
9->8;
}
```

Gambar 10. Representasi tA2008 dalam bahasa dot

Membangkitkan Jalur

Jalur dibentuk dengan cara menelusuri objek graph yang sudah dibentuk sebelumnya. Jika *edge* memiliki tipe *true* atau *false*, maka jalur yang dibangkitkan akan ditambahkan informasi cabang yang dilalui. (T) ketika melalui edge yang memiliki tipe *true*, dan (F) ketika melalui *edge* yang memiliki tipe *false*. Setiap *edge* memiliki atribut *isVisited* yang digunakan untuk menandai apakah garis penghubung tersebut sudah dilalui atau belum. Jalur yang dibentuk ketika melalui perintah pengulangan seperti FOR dan WHILE akan dibatasi hanya satu kali pengulangan. Berikut merupakan semua kemungkinan jalur yang akan dilalui ketika diberikan suatu inputan yang dapat dijadikan sebagai dasar dalam pembangkitan data uji.

1. 1 2 (T) 3 (T) 4 8
2. 1 2 (T) 3 (F) 5 (T) 6 8
3. 1 2 (T) 3 (F) 5 (F) 7 8
4. 1 2 (F) 9 8

Transformasi ke Dalam Format Bahasa Dot

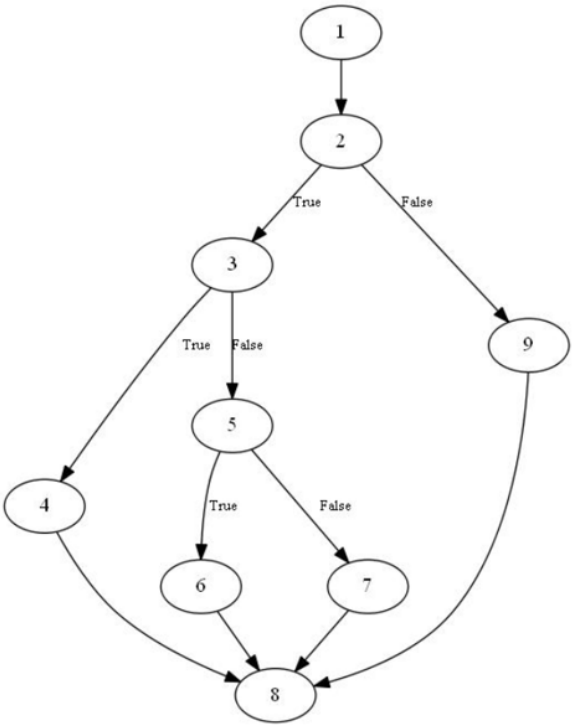
Transformasi ke dalam format bahasa dot dilakukan dengan cara menelusuri objek *graph* yang sudah dibangun sebelumnya. Yang didefinisikan dalam bahasa dot adalah *edge* yang terdapat pada *graph* yang dibangun. Seperti yang dapat dilihat pada 10, jumlah baris sebanyak jumlah *edge* pada objek graph yang telah didefinisikan sebelumnya.

Memvisualisasi *Graph* dalam bentuk CFG

Setelah file dengan format bahasa dot terbentuk, CFG divisualisasikan dengan menggunakan library Graphviz.Net. Hasil visualisasi bahasa dot kode program tA2008 ke dalam CFG dapat dilihat pada 11.

Menghitung *Cyclometic Complexity*

Cyclomatic complexity merupakan suatu sistem pengukuran yang menunjukkan banyaknya *independent path*. *Cyclomatic Complexity* dihitung dengan cara jumlah *edge* dikurangi dengan jumlah *node*, lalu ditambahkan dengan dua. Berdasarkan graph yang telah terbentuk dari kode program tA2008, dapat dilihat pada 9 bahwa jumlah



Gambar 11. CFG tA2008

node yang terbentuk adalah 9 dan jumlah *edge* yang terbentuk adalah 11. Sehingga hasil perhitungan cyclomatic complexity dapat dilihat pada persamaan dibawah ini.

$$Nodes(N) = 9$$

$$Edges(E) = 11$$

$$V(G) = E - N + 2$$

$$= 4$$

Instrumentasi

Instrumentasi dilakukan dengan cara menambahkan dulu variabel keluaran bernama traversedPath. Variabel in digunakan untuk menyimpan informasi node mana saja yang dilalui ketika diberikan inputan dengan nilai tertentu. Dan menyimpan informasi pilihan yang dilalui ketika ditemukan cabang yang terdapat pilihan *true* atau *false*.

Hasil kode program yang telah diinstrumentasi dapat dilihat pada Gambar 13. Sebelumnya, kode program tA2008 hanya mengembalikan keluaran satu variabel bernama type yaitu menunjukan jenis dari segitiga ketika diberikan panjang dari ketiga sisi segitiga. Setelah dilakukan instrumentasi, kode program tA2008 akan mengembalikan keluaran dengan variabel tambahan bernama traversedPath. Sehingga ketika program

```
function [traversedPath,type] = triangle(sideLengths)
        traversedPath = [];
        traversedPath = [traversedPath '1 ' ];
        A = sideLengths(1); % First side
        B = sideLengths(2); % Second side
        C = sideLengths(3); % Third side
        % instrument Branch # 1
        traversedPath = [traversedPath '2 ' ];
        if ((A+B > C) && (B+C > A) && (C+A > B))
                traversedPath = [traversedPath '(T) ' ];
                % instrument Branch # 2
                traversedPath = [traversedPath '3 ' ];
                if ((A ~= B) && (B ~= C) && (C ~= A))
                        traversedPath = [traversedPath '(T) ' ];
                        traversedPath = [traversedPath '4 ' ];
                        type = 'Scalene';
                else
                        traversedPath = [traversedPath '(F) ' ];
                        % instrument Branch # 3
                        traversedPath = [traversedPath '5 ' ];
                        if (((A == B) && (B ~= C)) || ((B == C) && (C ~= A))
                        || ((C == A) && (A ~= B)))
                                traversedPath = [traversedPath '(T) ' ];
                                traversedPath = [traversedPath '6 ' ];
                                type = 'Isosceles';
                        else
                                traversedPath = [traversedPath '(F) ' ];
                                traversedPath = [traversedPath '7 ' ];
                                type = 'Equilateral';
                        end
                end
        else
                traversedPath = [traversedPath '(F) ' ];
                traversedPath = [traversedPath '9 ' ];
                type = 'Not a triangle';
        end
        traversedPath = [traversedPath '8 ' ];
end
```

Gambar 12. Hasil instrumentasi tA2008

```
>> [traversedPath, type] = triangle([3,4,4])

traversedPath =

    '1 2 (T) 3 (F) 5 (T) 6 8 '


type =

    'Isosceles'
```

Gambar 13. Hasil ekseskusi kode program tA2008 yang sudah diinstrumentasi

tersebut dijalankan dengan inputan tertentu akan menghasilkan keluaran nilai traversedPath dan type seperti yang dapat dilihat pada Gambar 12.

Sehingga ketika kode program hasil instrumentasi dijalankan dengan inputan tertentu akan menghasilkan keluaran variabel traversedPath dan type seperti yang dapat dilihat pada Gambar 13. Misalkan inputan adalah 3, 4, dan 4 akan menghasilkan isosceles dan dapat diketahui bagaimana cara menghasilkan keluaran tersebut dari traversedPath.

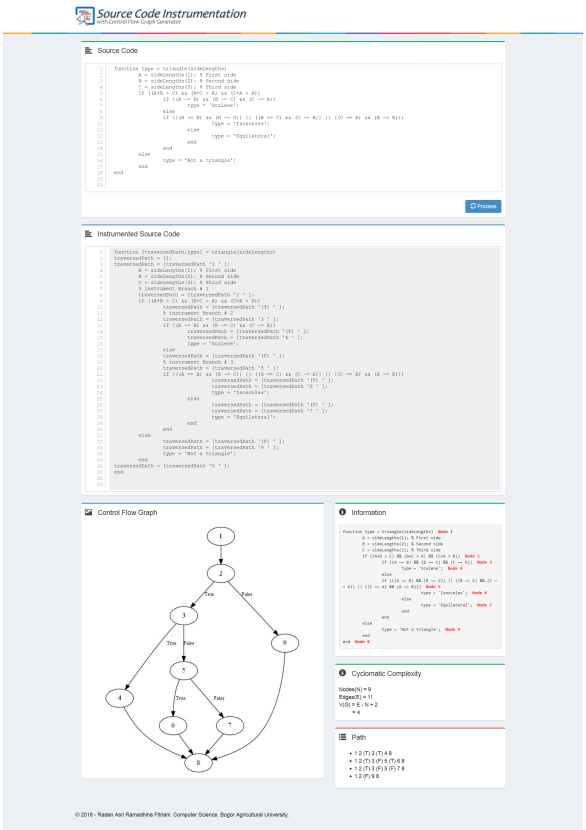
Implementasi Antarmuka

Tampilan hasil pembangkitan dapat dilihat pada Gambar 14.

Testing

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Evaluasi dilakukan dengan membandingkan hasil yang dikeluarkan oleh sistem dengan pembangkitan secara manual dari segi waktu eksekusi. Penguji terdiri dari dua orang yang berprofesi sebagai pengembang sistem.

Hasil jalur yang dibentuk dan nilai *cyclomatic com-*



Gambar 14. Tampilan hasil pembangkitan

plexity dari program uji dapat dilihat pada Tabel 2. Hasil perbandingan waktu yang dibutuhkan untuk melakukan pembangkitan secara manual dan oleh aplikasi dapat dilihat pada Tabel 3.

Jumlah jalur yang terbentuk bisa lebih besar dari cyclomatic complexity karena jumlah jalur yang tebentuk adalah dari semua kemungkinan. Sedangkan jumlah cyclomatic complexity menunjukkan independent path yaitu jalur yang setidaknya ada 1 jalur baru yang dilalui.

KESIMPULAN DAN SARAN

Tabel 2. Hasil jalur dan *cyclomatic complexity* dari program uji

No	Nama Program	Jumlah Node	Jumlah Edge	Cyclomatic Complexity	Jumlah jalur yang terbentuk
1	tA2008	9	11	4	4
2	mmA2008	8	10	4	5
3	iA2008	6	7	3	3
4	binA2008	12	15	5	10
5	bubA2008	8	10	4	4
6	gA2008	9	11	4	4
7	eB2002	15	20	7	21
8	qB2002	10	14	6	8
9	fB2002	22	29	9	12
10	fmH2014	10	13	5	9

Tabel 3. Perbandingan eksekusi secara menual dan oleh aplikasi

No	Nama Program	Waktu Eksekusi Oleh Aplikasi	Waktu Ekseskusi Manual			Kesesuaian Hasil	
			Penguji 1	Penguji 2	Rata-rata	Penguji 1	Penguji 2
1	tA2008	1.53 s	492 s				
2	mmA2008	1.71 s					
3	iA2008	1.56 s					
4	binA2008	1.45 s					
5	bubA2008	1.39 s					
6	gA2008	2.09 s					
7	eB2002	1.47 s					
8	qB2002	1.75 s					
9	fB2002	1.64 s					
10	fmH2014	1.81 s					

Kesimpulan

Penelitian ini berhasil membangun sebuah aplikasi yang dapat digunakan untuk melakukan instrumentasi secara otomatis, membangkitkan CFG, dan membangkitkan segala kemungkinan jalur yang dapat dilewati dari kode program matlab.

Hasil penelitian menunjukkan bahwa kecepatan eksekusi xxx% jauh lebih cepat dibandingkan dilakukan secara manual. Selain itu, dari 10 kode program yang digunakan sebagai program uji, terdapat 2 yang hasil jalur yang dilalui kurang dari seharusnya.

Saran

Penelitian selanjutnya diharapkan dapat mengakomodir bahasa lain selain bahasa matlab. Selain itu, instrumentasi juga dapat dilakukan dinamis sesuai dengan kondisi yang dibutuhkan. Seperti yang dilakukan oleh Hermadi, selain menyimpan informasi jalur mana yang dilewati, instrumentasi yang dilakukan pada penelitian tersebut juga menyisipkan nilai dari fungsi *fitness*.

DAFTAR PUSTAKA

Arkeman, Y, Herdiyeni, Y, Hermadi, I, dan Laxmi, G F. 2014. *Algoritma Genetika Tujuan Jamak (Multi-Objective Genetic Algorithm*. IPB Press.

Basu, A. 2015. *Software Quality Assurance, Testing and Metrics*. PHI Learning Privat Limited. [Internet]. [Diunduh tanggal 14/8/2017]. Dapat diunduh dari: `https://books.google.co.id/books`.

Hermadi, I. 2015. "Path Testing using Genetic Algorithm". Disertasi. University of New South Wales.

Khan, M E. 2011. "Different Approaches to White Box Testing Technique for Finding Errors" dalam: *International Journal of Software Engineering and Its Applications* 5, p. 3. [Internet]. [Diunduh tanggal 21/8/2017]. Dapat diunduh dari: `http://www.`

sersc.org/journals/IJSEIA/vol5_no3_2011/1.pdf.

Kumar, D dan Mishra, K K. 2016. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market" dalam: *Procedia Computer Science* 79, pp. 8–15. [Internet]. [Diunduh tanggal 20/8/2017]. Dapat diunduh dari: http://www.sciencedirect.com/science/article/pii/%5C%5CS1877050916001277.

Myers, G J, Sandler, C, dan Badgett, T. 2012. *The Art of Software Testing*. John Willey dan Sons, Inc, Hoboken, New York. [Internet]. [Diunduh tanggal 14/8/2017]. Dapat diunduh dari: https://books.google.co.id/books.

Tikir, M M dan Hollingsworth, J K. 2011. "Efficient Instrumentation for Code Coverage Testing" dalam: *International Journal of Software Engineering and Its Applications*. [Internet]. [Diunduh tanggal 21/8/2017]. Dapat diunduh dari: https://www.researchgate.net/publication/2835608_Efficient_Instrumentation_for_Code_Coverage_Testing.

Watson, A H dan McCabe, T J. 1996. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric)" dalam: *NIST Special Publication*. [Internet]. [Diunduh tanggal 14/8/2017]. Dapat diunduh dari: http://www.mccabe.com/pdf/mccabe-nist235r.pdf.