PENDEKATAN *WHITE BOX TESTING* UNTUK MENENTUKAN KUALITAS PERANGKAT LUNAK DENGAN MENGGUNAKAN BAHASA PEMROGRAMAN C++

Sa'diyah Noor Novita Alfisahrin

Manajemen Informatika
Akademi Manajemen Informatika dan Komputer Bina Sarana Informatika Yogyakarta
Jl. Ringroad Barat, Ambarketawang, Gamping, Sleman, Yogyakarta
sa.snt@bsi.ac.id

Abstract

The development of software house have impact to the increment of software productivity circulating in the community, but the quality of software currently available is still questionable whether the software meet quality standards or not. White-box testing is a test that can be used to examine the internal structure of the program that access to the source code and can be done at any time in the software development cycle, so that the mistakes of the unwanted programs can be improved and can be a qualified software product. This research aimed to test the software using white box testing approach that can be used to find an unexpected error in the software are built. The research method in this research is the study of literature and experimental by testing the internal structure of the software that include control flow testing, branch testing, basis path testing, data flow testing, and loop testing. By doing white box testing, the improvement of the software can be done at any time and can produce qualified software.
Keywords: *software testing, white box testing, software quality.*

1. PENDAHULUAN

Perangkat lunak saat ini sudah banyak beredar di masyarakat baik yang berbayar maupun yang gratis. Meningkatnya penggunaan perangkat lunak sangat berdampak baik dalam mendukung aktifitas manusia, akan tetapi tidak semua perangkat lunak yang digunakan mempunyai kualitas yang baik sesuai dengan yang diharapkan, sehingga diperlukan jaminan bahwa perangkat lunak yang digunakan merupakan perangkat lunak yang berkualitas dan dapat beroperasi dengan baik.

Kualitas perangkat lunak yang beredar di masyarakat ini tergantung pada pembuat perangkat lunak itu sendiri. Ada perangkat lunak yang dibuat oleh pengembang yang sudah benar-benar mempunyai keahlian dan mempunyai tim pengembang sendiri, ada juga yang dibuat oleh programmer secara individu, bahkan ada juga yang dibuat hanya sebagai bahan uji coba untuk mendapatkan pengalaman dan tantangan sehingga kualitas perangkat lunak yang diciptakan berada dibawah standar kualitas perangkat lunak.

Pengujian perangkat lunak digunakan untuk menentukan kualitas perangkat lunak apakah sudah memenuhi persyaratan fungsional dan kinerja atau belum, untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil yang sebenarnya, menilai apakah tahap pengembangan perangkat lunak telah sesuai dengan metodologi yang digunakan, dan membuat dokumentasi hasil pengujian yang menginformasikan kesesuaian perangkat lunak yang diuji dengan spesifikasi yang ditentukan

White box testing merupakan salah satu teknik pengujian perangkat lunak yang paling umum digunakan, sangat efektif dalam memvalidasi desain, keputusan, asumsi dan menemukan kesalahan pemrograman dan kesalahan implementasi dalam perangkat lunak (Khan, 2011). *White box testing* membutuhkan akses ke kode sumber dan dapat dilakukan setiap saat dalam siklus pengembangan perangkat lunak. Pengujian *white box* dikenal juga dengan mana pengujian *clear box*, *glass box* ataua *open box* (Desikan & Ramesh, 2008). Berdasarkan latar belakang tersebut maka didapat rumusan masalah berupa banyaknya perangkat lunak dengan kualitas dibawah standar.

Penelitian ini hanya dibatasi pada pengujian struktur internal program dari perangkat lunak yang sedang dibangun yang meliputi *control flow testing, branch testing, basis path testing, data flow testing, loop testing*.

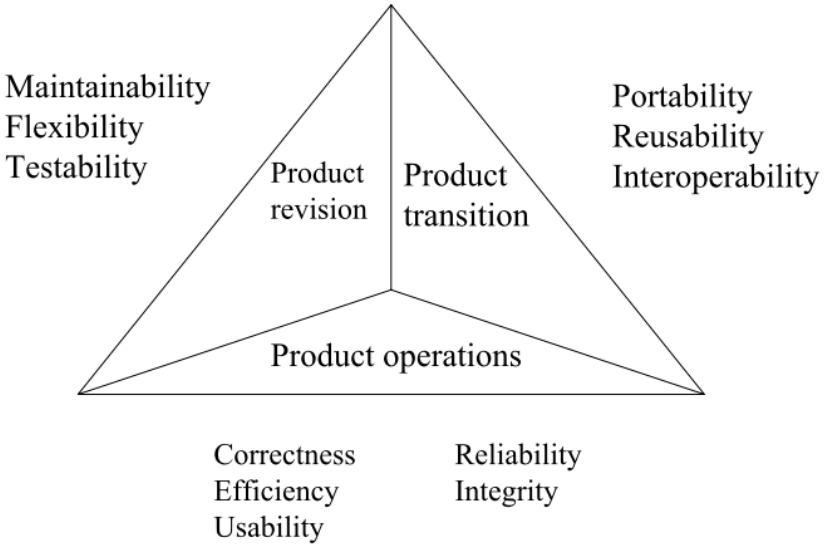
Tujuan dari penelitian ini adalah melakukan pengujian perangkat lunak dengan menggunakan pendekatan pengujian *white box* untuk menemukan kesalahan yang tidak diduga

yang mampu meningkatkan kualitas perangkat lunak.

2. TINJAUAN PUSTAKA

Kualitas perangkat lunak terbentuk dari sejumlah faktor. Menurut McCalls faktor-faktor yang dapat mempengaruhi kualitas software terbagi menjadi dua kategori, yaitu (pressman, 2010):

1. Faktor-faktor yang dapat diukur secara langsung.
2. Faktor-faktor yang dapat diukur secara tidak langsung.



Gambar 1. Faktor yang menentukan kualitas perangkat lunak (Pressman, 2010)

Menurut McCall terdapat tiga aspek penting dari suatu produk *software*, yaitu karakteristik operasional, kemampuan perubahan ketika *software* sudah berjalan, dan kemampuan beradaptasi terhadap lingkungan baru (Pressman, 2010).

Berdasarkan gambar diatas McCall menyediakan beberapa deskripsi yaitu:

1. *Correctness*: Tingkat pemenuhan program terhadap kebutuhan yang dispesifikasikan dan memenuhi tujuan/misi pengguna
2. *Realibility*: Tingkat kemampuan program yang diharapkan dapat menampilkan fungsi yang dimaksud dengan presisi yang ditetapkan
3. *Efficiency*: Jumlah sumber daya yang di proses dan kode yang diperlukan oleh program untuk melaksanakan fungsinya
4. *Integrity*: Tingkat kemampuan pengawasan akses terhadap data atau software oleh orang-orang tertentu
5. *Usability*: Usaha yang diperlukan untuk mempelajari, mengoperasikan, menyiapkan masukan dan mengartikan keluaran program.
6. *Maintainability*: Usaha yang diperlukan untuk menetapkan dan memperbaiki kesalahan dalam program.
7. *Flexibility*: Usaha yang diperlukan untuk memodifikasi program operasional
8. *Testability*: Usaha yang diperlukan untuk menguji program untuk memastikan bahwa program melaksanakan fungsi yang telah ditetapkan
9. *Portability*: Usaha yang diperlukan untuk memindahkan program dari *hardware*/lingkungan sistem *software* tertentu ke yang lainnya.
10. *Reusability*: Tingkat kemampuan program/bagian dari program yang dapat dipakai ulang dalam aplikasi lainnya, berkaitan dengan paket dan lingkup dari fungsi yang dilakukan oleh program.
11. *Interoperability*: Usaha yang diperlukan untuk menggabungkan sistem dengan sistem lainnya.

Menurut Myers (2004, hal. 12) pengujian program adalah sebuah proses destruktif yang mencoba menemukan kesalahan dalam sebuah program untuk meyakinkan bahwa program melakukan apa yang seharusnya dilakukan dan tidak melakukan apa yang seharusnya tidak dilakukan. Pengujian yang sukses adalah pengujian yang dapat menemukan kesalahan yang belum ditentukan, tujuan ini akan dicapai dengan baik melalui eksplorasi kesalahan yang terus menerus.

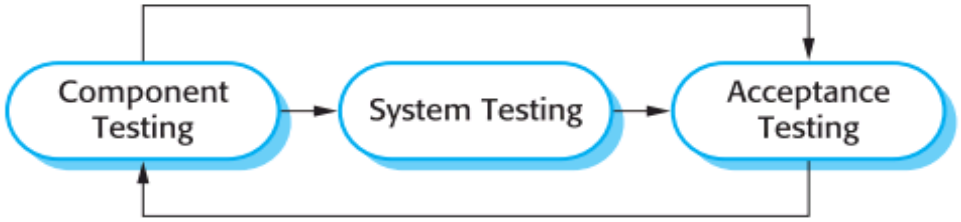
Pengujian perangkat lunak merupakan proses eksekusi program yang telah selesai dibuat

yang bertujuan untuk menemukan kesalahan yang tidak diduga, Proses pengujian memiliki dua tujuan yang berbeda yaitu (Sommerville, 2011):

1. Menunjukan kepada pengembang dan pengguna bahwa perangkat lunak telah memenuhi kebutuhan. Untuk perangkat lunak khusus, setidaknya harus ada satu pengujian untuk setiap persyaratan dalam dokumen persyaratan. Untuk produk perangkat lunak yang umum harus ada pengujian untuk semua fitur sistem.
2. Mengetahui situasi dimana *software* tidak berjalan dengan benar, tidak diinginkan, atau tidak sesuai dengan spesifikasinya seperti crash, interaksi yang tidak diinginkan dengan sistem yang lain, perhitungan yang salah, dan korupsi data.

Tujuan pengujian pertama mengarah pada pengujian validasi yang mengidentifikasi apakah sistem yang dibangun sudah bekerja dengan benar. Tujuan pengujian kedua merupakan pengujian verifikasi, pengujian ini digunakan untuk mengetahui apakah *software* yang dibangun sudah memenuhi kebutuhan pengguna, dimana uji kasus dirancang untuk menampilkan kerusakan atau cacat dari program.

Pengujian perangkat lunak dalam tahap pengembangan perangkat lunak telah menjadi bagian penting dari siklus pengembangan dan merupakan kunci dari metodologi agile. Kualitas kode dan perawatan perangkat lunak ditingkatkan dengan mengadopsi strategi pengujian terpadu yang menekankan pengujian unit program, Pengujian integrasi, pengujian sistem dan pengujian penerimaan pada keseluruhan proyek, selain itu pengujian unit keamanan merupakan investasi jangka panjang yang dapat meningkatkan kualitas perangkat lunak dan mengurangi kerentanan dan resiko yang terkait. (Vries, 2006).

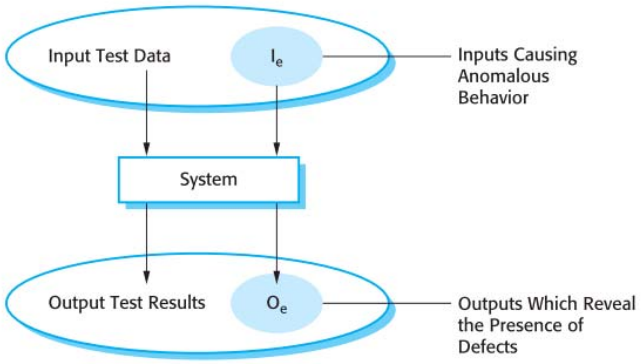


Gambar 2. Strategi pengujian (Sommerville, 2011)

Strategi pengujian

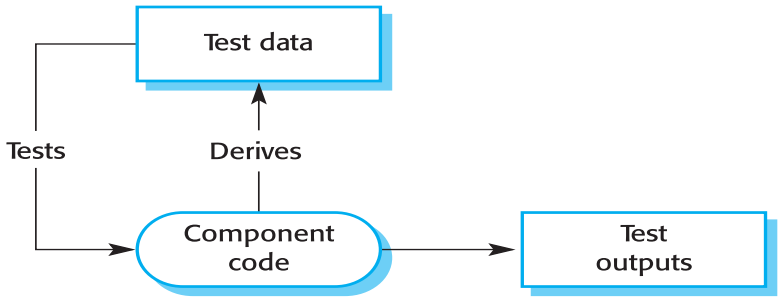
1. Pengujian Unit Program
   Pengujian difokuskan pada unit terkecil dari satu modul program, seperti metode, fungsi atau *class* yang dilaksanakan dengan menggunakan *driver* dan *stub*. *Driver* merupakan suatu program utama yang berfungsi mengirim atau menerima data kasus uji dan mencetak hasil dari modul yang diuji. *Stub* adalah modul yang menggantikan modul sub-ordinat dari modul yang diuji.
2. Pengujian Integrasi
   Pengujian integrasi merupakan pengujian terhadap unit-unit program yang saling berhubungan (terintegrasi) dengan fokus pada masalah *interfacing*
3. Pengujian Sistem
   Pengujian dilakukan terhadap integrasi sub sistem yaitu keterhubungan antar sub sistem. Pengujian ini meliputi *recovery testing, security testing dan stress testing*.
4. Pengujian Penerimaan
   Pengujian penerimaan merupakan pengujian terakhir sebelum sistem dipakai oleh *user*, melibatkan pengujian dengan data dari pengguna sistem. Pengujian ini dikenal sebagai *alpha test*, sedangkan untuk *software* komersial dimana pengujian dilakukan oleh potensial *user* dinamakan *beta test*.

Myers mengemukakan ada dua teknik yang lazim digunakan dalam pengujian perangkat lunak, yaitu *black box testing* dan *white box testing*. *Black box testing* atau pengujian berbasis data dilakukan untuk antar muka perangkat lunak, pengujian ini dilakukan untuk memperlihatkan bahwa fungsi-fungsi perangkat lunak bekerja dengan baik, dalam arti bahwa semua masukan diterima dengan benar dan keluaran yang dihasilkan benar-benar tepat, pengintegrasian dari eksternal data berjalan dengan baik. Pengujian *black box* tidak menguji pada kode program tapi melihat perangkat lunak dari sisi *external* (Desikan & Ramesh, 2008).



Gambar 3. Black Box Testing (Sommerville, 2011)

Pengujian white box merupakan pengujian yang dilakukan lebih dekat lagi untuk menguji prosedur-prosedur yang ada, yang memungkinkan untuk memeriksa struktur internal program. Strategi ini berasal dari data uji hasil pemeriksaan logika program, pengujian *white box* dilakukan terlebih dahulu pada proses pengujian sedangkan pengujian *black box* dilakukan pada tahap akhir dari pengujian perangkat lunak. Keuntungan lain dari pengujian *white box* yaitu berlatar belakang matematika, kode sumber dari perangkat lunak dapat diubah menjadi grafik yang berarti bahwa semua teori grafik matematika dapat diterapkan dalam mengembangkan strategi pengujian *white box*.



Gambar 4. white box testing (Sommerville, 2011)

Mohd. Ehmer Khan berpendapat ada beberapa tipe penting dari pengujian *white box*, yaitu *control flow testing, branch testing, basis path testing, data flow testing, loop testing*.

1. *Loop Testing*

Loop testing merupakan teknik pengujian yang secara eksklusif berfokus pada validitas dari *loop*. Ada empat kelas dari *loop*, yaitu *simple loop, nested loop, concatenated loop*, dan *unstructured loop*.

Beberapa rangkaian *test* berikut dapat diterapkan pada *loop* sederhana, dimana n adalah jumlah maksimum yang diperbolehkan melewati *loop*

a. Lewati *loop* secara keseluruhan
b. Lewati *loop* hanya satu kali
c. Dua tahap melalui *loop*
d. m tahap melalui *loop* dimana m<n
e. n1, n, n+1 melewati *loop*

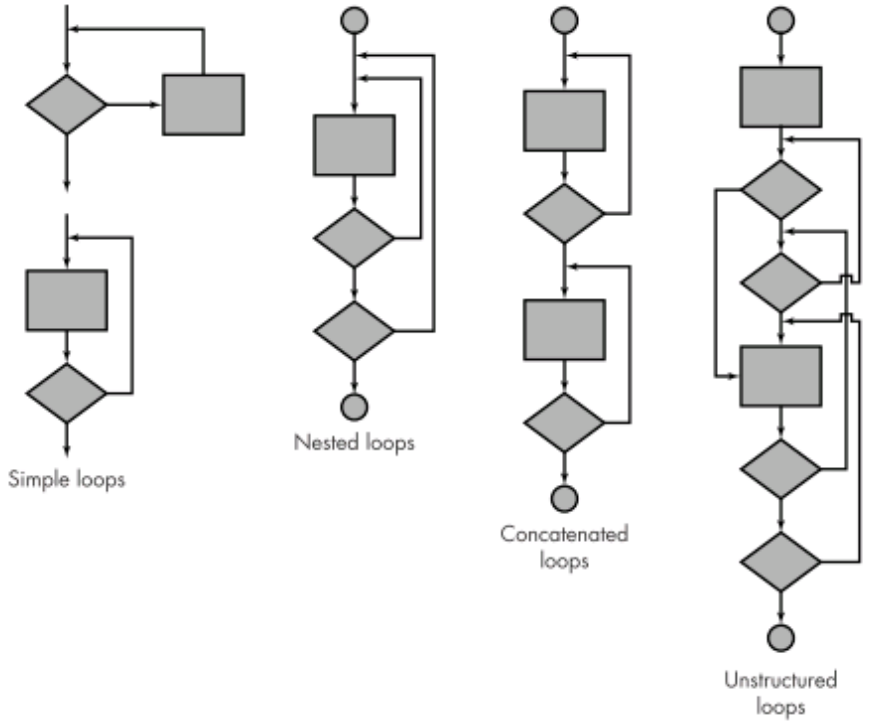
jika kita perluas pendekatan dari *simple loop* ke *nested loop*, jumlah pengujian mungkin akan meningkat secara geometris, pendekatan berikut akan membantu untuk mengurangi jumlah pengujian

a. Mulai dari *loop* yang paling dalam, set semua *loop* lain dengan nilai minimum
b. Lakukan pengujian *loop* sederhana untuk *loop* pada bagian paling dalam, sambil mempertahankan *loop* bagian luar dengan nilai minimum iterasinya. Tambahkan pengujian lain supaya dapat keluar dari *loop*.
c. Kerjakan bagian luar, lakukan *test* untuk *loop* berikutnya, tetapi menjaga semua *loop* lain yang berada di luar berada pada nilai minimum.
d. Lanjutkan sampai semua *loop* teruji

Concatenated loop dapat diuji dengan menggunakan pendekatan yang didefinisikan untuk *loop* sederhana jika setiap *loop* tidak bergantung pada *loop* yang lainnya. Tetapi jika *loop* bergantung dengan *loop* yang lainnya

maka pendekatan *nested loop* sebaiknya digunakan.

Pada *unstructured loop*, jika memungkinkan *loop* ini harus didesain ulang untuk mencerminkan penggunaan konstruksi pemrograman terstruktur.



Gambar 5. kelas loop (Presman, 2010)

2. *Branch Testing*

Brancing testing atau pengujian percabangan merupakan pengujian yang bertujuan untuk menguji setiap pilihan (*true* atau *False*) pada setiap pernyataan. *Test case* didesain untuk menjalanan aliran kontrol percabangan atau *decision point* dalam setiap unit, semua cabang diuji minimal sekali.

3. *Data Flow Testing*

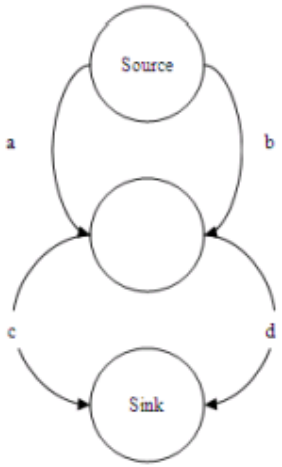
Data flow testing adalah pengujian yang melihat bagaimana data bergerak dalam sebuah program, *data flow testing* dapat juga didefinisikan sebagai teknik pengujian yang didasarkan pada pengamatan bahwa nilai-nilai yang terkait dengan variabel dapat mempengaruhi pelaksanaan program.

Pengujian aliran data merupakan teknik pengujian yang didasarkan pada pengamatan bahwa nilai-nilai yang terkait dapat mempengaruhi eksekusi program, untuk memastikan bahwa:

a. Setiap objek data telah diinisialisasi sebelum digunakan
b. Semua objek yang didefinisikan telah digunakan minimal sekali

Point penting dari pengujian data *flow* adalah:

a. Semua anomali dari data *flow* diselesaikan
b. Hindari masalah integrasi dengan melakukan semua operasi aliran data pada sebuah variabel dalam rutinitas yang sama
c. Bila memnungkinkan, gunakan deklarasi data eksplisit bukan implisit.



Gambar 6. control flow graph (Presman, 2010)

4. *Control Flow Testing*

Pengujian *control flow* lebih efektif untuk kode yang tidak terstruktur dari pada kode terstruktur, kebanyakan *bug* dapat menghasilkan *control flow* yang salah.

Beberapa batasan dalam pengujian *control flow* adalah:

a. Pengujian *control flow* tidak dapat menangkap seluruh kesalahan inisialisasi
b. Kesalahan spesifikasi tidak dapat ditangkap oleh pengujian *control flow*
c. Tidak mungkin menenukan path yang hilang dan fitur-fiturnya, jika model dan rogram yang dites dilakukan oleh orang yang sama.

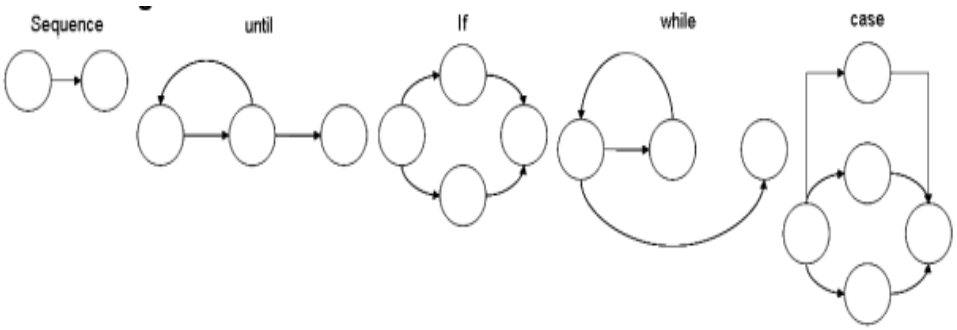
Kecukupan test case diukur dengan matrik yang dinamakan *coverage*. *Coverage* adalah ukuran kelengkapan dari himpunan uji kasus

5. *Basis Path Testing*

Basis path testing merupakan teknik pengujian yang memungkinkan perancang *test case* untuk mengukur kompleksitas logis dari desain prosedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi.

Konsep utama dari basis *path* yaitu tiap basis *path* harus diidentifikasikan, tidak boleh ada yang terabaikan setidaknya harus dites satu kali. Kombinasi dan permutasi dari suatu basis *path* tidak perlu dites.

Basis *path* menggunakan notasi aliran *graph* (node, link untuk merepresentasikan *sequence, if, while, untill,* dll).



Gambar 7. Notasi aliran graph (presman, 2010)

Lingkaran atau *node* menggambarkan satu atau lebih pernyataan prosedural, urutan proses dan keputusan dapat dipetakan dalam satu *node*. Tanda panah dalam aliran *graph* dinamakan *edge* atau *link* yang menggambarkan aliran kontrol, setiap *edge* harus terhubung dengan *node* meskipun *node* tidak mewakili pernyataan prosedural. Daerah yang dibatasi oleh *node* dan *edge* dinamakan *region*, termasuk daerah yang diluar grafik alir.

3. METODE PENELITIAN

Studi literatur dilakukan pada tahap awal dari penelitian ini. Studi literatur mengenai pengujian perangkat lunak bertujuan untuk mendapatkan informasi dasar tentang pengujian perangkat lunak melalui berbagai buku, jurnal maupun artikel yang berhubungan dengan masalah yang diteliti.

Metode penelitian eksperimen dapat diartikan sebagai metode penelitian yang digunakan untuk mencari pengaruh perlakuan tertentu terhadap yang lain dalam kondisi yang terkendalikan (Sugiyanto, 2008). Metode eksperimen dibagi menjadi dua, yaitu eksperimen absolut dan eksperimen komparatif. Eksperimen absolut mengarah pada dampak yang dihasilkan dari eksperimen, sedangkan eksperimen komparatif membandingkan dua objek yang berbeda (Kothari, 2004). Dalam penelitian ini digunakan metode eksperimen absolut.

4. HASIL DAN PEMBAHASAN

Pelaksanaan pengujian *white box* akan menghasilkan *test case* yang mengeksekusi pengulangan atau *looping* dalam batas-batas yang ditentukan, menjalani *logical decision* pada sisi *true* dan *false*, menguji struktur data internal dan menjamin seluruh *independen path* dieksekusi paling sedikit satu kali.

Test data adalah input yang direncanakan digunakan oleh sistem, sedangkan *test case* adalah input yang digunakan untuk menguji sistem dan memprediksi output dari input jika sistem beroperasi sesuai dengan spesifikasi. *Independen path* adalah jalur dalam program yang menunjukan paling sedikit satu kumpulan proses ataupun kondisi baru.

Flow graph atau grafik alir menggambarkan alur kontrol yang digunakan sebagai basis untuk menghitung kompleksitas siklomatik.

Kompleksitas siklomatik adalah metrik *software* yang menyajikan ukuran kuantitatif dari kompleksitas logika suatu program. Pada konteks metode *basis path testing*, nilai yang dihitung bagi kompleksitas siklomatik menentukan jumah jalur yang independen dalam kumpulan basis suatu program dan memberikan jumlah tes minimal yang harus dilakukan untuk memastikan bahwa semua pernyataan telah dieksekusi sekurang-kurangnya satu kali.

Prinsip pengujian dengan menggunakan kompleksitas siklomatik adalah membatasi kompleksitas dari modul-modul internalnya kedalam bentuk yang dapat diuji dengan tepat.

Kompleksitas siklomatik memiliki landasan dalam teori *graph* dan memberikan metrik perangkat lunak yang sangat berguna. Kompleksitas siklomatik dapat dihitung dengan tiga cara:

1. Jumlah *region flow graph* sesuai dengan kompleksitas siklomatik
2. Kompleksitas siklomatik V(G) untuk *flow graph* dihitung dengan rumus:
   $V(G) = E - N + 2$
   Dimana E adalah jumlah *edge* pada *flow graph*, N adalah jumlah *node* dalam *flow graph*.
3. Kompleksitas siklomatik juga dapat diukur dengan rumus:
   $V(G) = P + 1$
   Dimana P adalah jumlah *predicate node* pada grafik alir

Metode pengujian basis *path* dapat diterapkan untuk desain prosedural atau ke kode sumber. Pada bagian ini penulis menyajikan pengujian basis *path* dan langkah-langkah yang dapat diterapkan untuk mendapatkan basis set dari kode program berikut adalah:

1. Menggunakan desain atau kode sebagai landasan, menggambarkan grafik alir yang sesuai

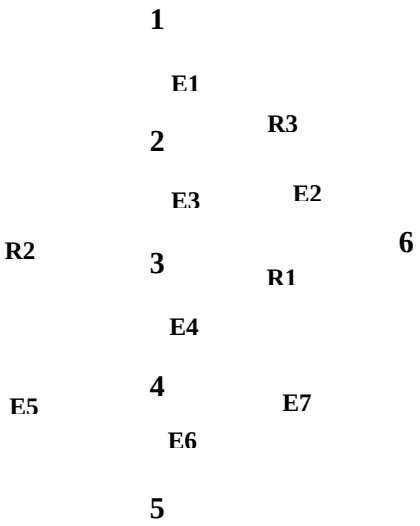
Misalkan akan diuji sebuah program yang dirancang untuk menampilkan nilai bilangan A, bilangan B dan bilangan C dengan iterasi dari satu hingga sepuluh, jika iterasi melebihi maka maka proses menampilkan nilai bilangan akan dihentikan. Jika nilai bilangan C sudah mencapai 13 maka proses menampilkan bilangan akan dihentikan meskipun nilai iterasi masih memenuhi.

Untuk memudahkan dalam pembentukan flow *graph*, maka kode program dibawah ini akan diberi nomor sesuai dengan urutan jalannya program.

```
/* ---------------------------- */
/* Perulangan FOR dengan break;  */
/* ---------------------------- */

#include <stdio.h>
#include <conio.h>

main()

{
  int a=3, b=2, c=1, bil;
  clrscr();                                          1
  printf("Bil-A   | Bil-B | Bil-C\n");
  printf("-------------------------");
  for(bil=1; bil<=10; ++bil)                         2
  {
   a+=b; b+=c; c+=2;                                 3
   printf("\n%d \t| %d \t| %d",a, b, c);
   if(c==13)                                         4
  break;                                        5
  }
  getche();                                          6
}
```

Listing program tersebut diberi nomor urut untuk memudahkan pembentukan *flow graph*, mulai dari inisialisasi nilai awal bilangan a,b, dan c, penentuan *node* statement perulangan, statement percabangan, hingga output yang dihasilkan. Baris kode program yang tidak menyatakan sebagai proses pengujian dapat dijadikan satu *node*.

Bentuk *flow graph* dari kode program diatas adalah sebagai berikut:



Gambar 8. Flow graph control

2. Menentukan kompleksitas siklomatik dari grafik alir yang dihasilkan

Bilangan siklomatik (V) ditentukan dengan cara:

a. Jumlah *region flow graph* sesuai dengan kompleksitas siklomatik
   V(G) = *Region*
   V(G) = 3
b. Kompleksitas siklomatik V(G) untuk *flow graph* dihitung dengan rumus:
   V(G) = E – N +2
   V(G) = 7 – 6 + 2
   V(G) = 3
c. Kompleksitas siklomatik juga dapat diukur dengan rumus:
   V(G) = P + 1
   V(G) = 2 + 1
   V(G) = 3

Sehingga didapat nilai kompleksitas siklomatik dari kode program diatas adalah 3.

3. Menentukan basis set dari independen path linier

Nilai dari kompleksitas siklomatik menyajikan jumlah jalur independen linear melalui struktur kontrol program. Sehingga path yang didapat ada 3, yaitu :

Path 1: 1-2-6
Path 2: 1-2-3-4-2-6
Path 3: 1-2-3-4-5-6

Dari *path-path* tersebut dapat diketahui bahwa node 2 dan 4 adalah *predicate node*.

4. Menyiapkan *test case* yang akan memaksa eksekusi dari setiap path dalam basis path

Dalam menyiapkan *test case* data harus dipilih sehingga kondisi pada *predicate node* ditetapkan secara tepat pada tiap *path* yang diuji. Setiap *test case* dijalankan dan dibandingkan dengan hasil yang diharapkan, setelah semua *test case* telah selesai dijalankan penguji dapat memastikan bahwa semua *statement* dalam program telah dijalankan minimal sekali.

Berdasarkan jalur independen pada langkah ke tiga, maka dibentuk tiga *test case* yang akan digunakan dalam pengujian, yaitu:

a. *Test case path* 1
   Diberikan nilai bil > 10
   Hasil yang diharapkan adalah program tidak akan mencetak bilangan apapun
   Catatan: path 1 tidak dapat diuji sendiri secara langsung, tetapi harus diuji sebagai bagian dari *path* 2.
b. *Test case path* 2
   Diberikan nilai bilangan C != 13
   Hasil yang diharapkan adalah program akan melakukan *looping* dan mencetak nilai bilangan A, bilangan B dan Bilangan C
c. *Test case path* 3
   Diberikan nilai bilangan C == 13
   Hasil yang diharapkan adalah program akan berhenti melakukan *looping* meskipun nilai bil <= 13

5. PENUTUP

5.1. Kesimpulan

Pengujian *white box* sangat meningkatkan efektivitas tes secara keseluruhan, hal ini dapat meningkatkan produktivitas dalam mengungkapkan *bug* yang sulit ditemukan dengan pengujian *black box* atau metode pengujian lainnya. Pengujian *white box* seperti pada pembahasan diatas dapat dilakukan untuk mengeksekusi perulangan dan percabangan baik yang komplek maupun yang sederhana.

Banyaknya *path* dan *test case* ditentukan melalui perhitungan kompleksitas siklomatik suatu program, *test case* yang telah ditentukan kemudian dijalankan pada setiap *path* untuk membandingkan antara hasil yang diinginkan dengan keluaran dari program. Setelah mengetahui perbandingan yang dihasilkan, perbaikan dapat dilakukan supaya perangkat lunak yang dihasilkan menjadi lebih berkualitas.

5.2. Saran

Pengujian seharusnya tidak hanya dilakukan pada perangkat lunak yang berskala besar atau kompleks saja tetapi juga dilakukan pada semua produk perangkat lunak, hal ini dimaksudkan supaya perangkat lunak yang dihasilkan dapat menjadi produk yang berkualitas, handal dan pada akhirnya tidak mengalami kegagalan.

Pengujian dengan metode *white box* hanya dapat dilakukan untuk pengujian algoritma program, sehingga untuk meningkatkan kualitas perangkat lunak sebaiknya ditambahkan dengan pengujian yang lain misalnya dengan metode pengujian *black box*.

DAFTAR PUSTAKA

Desikan, Srinivasan & Ramesh, Gopalaswamy. 2008. *Software TestingPrinciples and Practices*. Pearson Education. India.

Khan, M. E. 2011. Different Approaches to White Box Testing Technique for Finding Errors. *International Journal of Software Engineering and Its Applications Vol.5 No. 3*.

Kothari, C. R. 2004. *Research Methology Methods and Techniques.* India: New Age International Limited.

Myers, G. J. 2004. *The Art of Software Testing, Second Edition.* John Wiley & Sons, Inc. Canada.

Pressman, R. S. 2010. *Software Engineering A Practitioner's Approach seventh edition.* McGraw-Hill. New York.

Sommerville, I. 2011. *Software Engineering Ninth Edition.* Pearson Education, Inc. United States of America.

Sugiyanto. 2008. *Metode Penelitian Kuantitatid Kualitatif dan R&D.* Bandung: Alfabeta.

Vries, S. d. 2006. Security Testing Web Aplication through Automated Software Test. *A Corsair White Paper*.