



INDONESIA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER

Testing dan Implementasi Sistem

White Box Testing

Anggota Kelompok II :

Komang Dodik Gunawan 13101172

Daniel Eka Saputra 13101882

Teguh Wirawan 13101058

DW GD Surya Damanik 13101461

MD Adhi Parwata 12101242

Dosen: Agus Aan Jiwa Permana, S.Kom, M.Cs

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada **Tuhan Yang Maha Esa** atas selesainya makalah yang berjudul ***White Box Testing***. Makalah ini disusun sesederhana mungkin agar mudah dimengerti oleh pembaca. Atas dukungan yang diberikan dalam penyusunan makalah ini, maka penulis mengucapkan terimakasih kepada:

1. **Bapak Agus Aan Jiwa Permana, S.Kom, M.Cs**, selaku dosen Testing dan Implementasi Sistem, yang memberikan bimbingan, saran, masukan, dan ilmu yang beliau punya untuk menyelesaikan tugas makalah ini.
2. Kami juga berterimakasih kepada teman-teman yang sudah membantu memberikan ide yang sangat membantu pembuatan makalah ini.

Penulis menyadari bahwa makalah ini belumlah sempurna. Oleh karena itu, saran dan kritik yang membangun dari rekan-rekan sangat dibutuhkan untuk penyempurnaan makalah ini.

Denpasar, Mei 2016

Penulis

DAFTAR ISI

Judul	i
Kata Pengantar	ii
Daftar Isi	iii

BAB I PENDAHULUAN.....1

1.1 Latar Belakang	1
1.2 Ruang Lingkup.....	2
1.3 Tujuan dan Manfaat	2

BAB II PEMBAHASAN3

2.1 Pengertian <i>White Box Testing</i>	3
2.2 Tujuan <i>white box testing</i>	3
2.3 Pelaksanaan <i>white box testing</i>	3
2.4 Langkah-langkah <i>white box testing</i>	4
2.5 Kelebihan dan Kekurangan <i>White Box Testing</i>	4
2.6 Jenis–Jenis <i>White Box Testing</i>	5
2.6.1 Notasi Diagram Alir (<i>Path Graph Notation</i>)	5
2.6.2 Kompleksitas Siklomatis (<i>Cyclomatic Complexity</i>).....	7
2.7 Kesetaraan Partisi (EP)/Analisis Nilai Batas (BVA)	10
2.8 Contoh <i>White Box Testing Software</i>	11

BAB III ANALISIS DAN PERANCANGAN19

DAFTAR PUSTAKA20

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perangkat lunak diartikan sebagai kumpulan instruksi yang membentuk suatu program komputer yang apabila dijalankan akan memberikan hasil sesuai dengan target yang telah ditentukan. Kemajuan perkembangan perangkat lunak diberbagai bidang kehidupan manusia menyebabkan ketergantungan manusia terhadap perangkat lunak semakin besar. Kondisi tersebut meningkatkan arti penting keberadaan perangkat lunak yang berkualitas baik dan reliable.

Pengujian perangkat lunak secara otomatis dapat meningkatkan efisiensi proses pengujian untuk mengidentifikasi bagian dari perangkat lunak yang rawan mengalami kegagalan. Pengujian perangkat lunak secara otomatis bisa dilakukan dengan menggunakan berbagai metode pengujian perangkat lunak yang ada. Karakteristik ini memperluas area yang mampu diuji secara otomatis sehingga mampu mengurangi beban dari penguji perangkat lunak. Sistem penguji perangkat lunak otomatis harus mampu melakukan berbagai pengujian dalam skala besar dan mampu diulang berkali-kali untuk

memastikan kualitas perangkat lunak yang diuji. Dalam teori pengujian perangkat lunak terdapat metode yang bisa digunakan untuk melakukan pengujian, misalnya metode white-box testing.

Kondisi yang dipaparkan pada uraian diatas memunculkan kebutuhan akan adanya sistem pengujian perangkat lunak otomatis yang mampu melakukan berbagai pengujian pada perangkat lunak yang kompleks dan mampu diulang berkali-kali untuk memastikan kualitas perangkat lunak yang diuji dengan sumber daya yang sedikit.

1.2 Ruang Lingkup

Dalam pembuatan makalah materi yang dibahas adalah mengenai metode pengujian white-box testing.

1.3 Tujuan dan Manfaat

Adapun tujuan dan manfaat yang diharapkan adalah pengujian perangkat lunak otomatis dengan menggunakan pengujian basis patch untuk mendapatkan program yang benar secara 100%.

BAB II

PEMBAHASAN

2.1 Pengertian *White Box Testing*

White box testing adalah pengujian yang didasarkan pada pengecekan terhadap detail perancangan, menggunakan struktur kontrol dari desain program secara procedural untuk membagi pengujian ke dalam beberapa kasus pengujian. Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

2.2 Tujuan *white box testing* :

1. Untuk mengetahui cara kerja suatu perangkat lunak secara internal.
2. Untuk menjamin operasi - operasi internal sesuai dengan spesifikasi yang telah ditetapkan dengan menggunakan struktur kendali dari prosedur yang dirancang.

2.3 Pelaksanaan *white box testing* :

1. Menjamin seluruh *independent path* dieksekusi paling sedikit satu kali. *Independent path* adalah

- jalur dalam program yang menunjukkan paling sedikit satu kumpulan proses ataupun kondisi baru.
2. Menjalani *logical decision* pada sisi dan *false*.
 3. Mengeksekusi pengulangan (*looping*) dalam batas - batas yang ditentukan.
 4. Menguji struktur data internal.

2.4 Langkah - langkah *white box testing* :

1. Mendefinisikan semua alur logika.
2. Membangun kasus untuk digunakan dalam pengujian.
3. Melakukan pengujian.

2.5 Kelebihan dan Kekurangan *White Box Testing*:

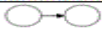

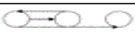

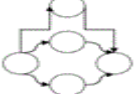
1. Mengetahui Kesalahan logika, dimana digunakan pada sintaks 'if' dan pengulangan. *White Box Testing* akan mendeteksi kondisi - kondisi yang tidak sesuai dan mendeteksi kapan proses pengulangan akan berhenti.
2. Ketidakesesuaian asumsi, yaitu menampilkan asumsi yang tidak sesuai dengan kenyataan, yang selanjutnya untuk di analisis dan diperbaiki.

3. Kesalahan ketik, berguna untuk mendeteksi bahasa pemrograman yang bersifat *case sensitive*.
4. Kekurangan dari pengujian ini adalah saat pengujian untuk perangkat lunak yang tergolong besar. *White Box Testing* dianggap sebagai strategi yang tergolong boros, karena akan melibatkan sumber daya yang besar untuk melakukannya.

2.6 Jenis – Jenis *White Box Testing*

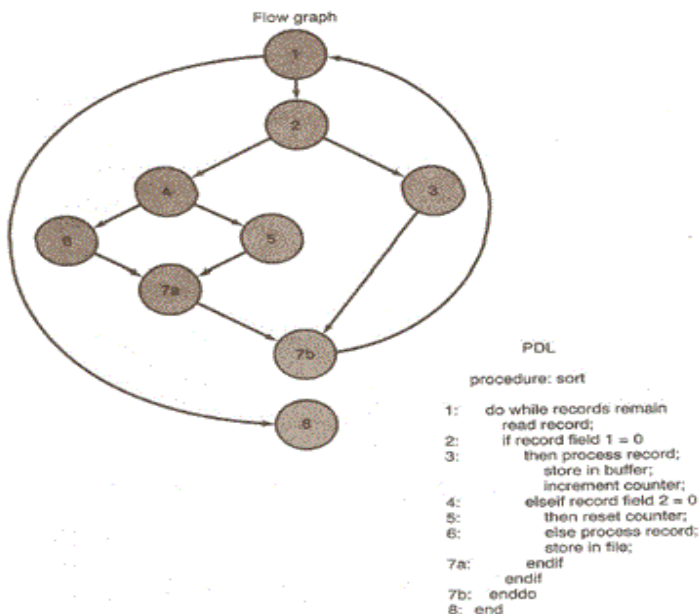
2.6.1 Notasi Diagram Alir (*Path Graph Notation*)

Notasi yang digunakan untuk menggambarkan jalur eksekusi adalah notasi diagram alir atau grafik program, yang menggunakan notasi lingkaran (simpul atau *node*) dan anak panah (*link* atau *edge*). Notasi ini menggambarkan aliran control logika yang digunakan dalam suatu bahasa pemrograman. Setiap representasi rancangan prosedural dapat diterjemahkan kedalam *flow graph*.

Notasi	Arti
	Skema Sequence
	Skema If
	Skema While (...) DO (...)
	Skema Repeat (...) Until (...)
	Skema Case (...) Of

Gambar Notasi Diagram Alir.

Gambar dibawah ini merupakan bagian dari PDL
(*Program Design Language*) dan *flow graph*-nya.



2.6.2 Kompleksitas Siklomatis (*Cyclomatic Complexity*)

Kompleksitas Siklomatis adalah *metriks* perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Ketika digunakan dalam konteks metode ujicoba berbasis alur, nilai yang didapat akan menentukan jumlah jalur independen dalam himpunan *path*, serta akan memberi nilai batas atas bagi jumlah pengujian yang harus dilakukan, untuk memastikan bahwa semua pernyataan telah dieksekusi sedikitnya satu kali.

Jalur independent adalah jalur yang terdapat dalam program yang mengintroduksi sedikitnya satu rangkaian pernyataan proses atau kondisi baru.

Berdasarkan contoh PDL yang pertama, maka jalur independent yang didapat:

Jalur 1 : 1 – 11

Jalur 2 : 1 – 2 – 3 – 4 – 5 – 10 – 1 – 11

Jalur 3 : 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11

Jalur 4 : 1 – 2 – 3 – 6 – 7 – 9 – 10 – 1 – 11

Misalkan setiap path yang baru memunculkan edge yang baru, dengan path :

1 – 2 – 3 – 4 – 5 – 10 – 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1
– 11

Path diatas tidak dianggap sebagai independent path karena kombinasi path diatas telah didefinisikan sebelumnya Ketika ditetapkan dalam graf alur, maka independent path harus bergerak sedikitnya 1 edge yang belum pernah dilewati sebelumnya.

Kompleksitas cyclomatic dapat dicari dengan salah satu dari 3 cara berikut :

1. Jumlah region dari grafik alur mengacu kepada kompleksitas cyclomatic
2. Kompleksitas cyclomatic $V(G)$ untuk grafik alur G didefinisikan sebagai:

$V(G) = E - N + 2$, dimana E = jumlah edge, dan N = jumlah node

3. Kompleksitas cyclomatic $V(G)$ untuk grafik alur G didefinisikan sebagai:

$V(G) = P + 1$, dimana P = jumlah predicates nodes yang diisikan dalam grafik alor G

Simpul Predikat adalah penggambaran suatu node yang memiliki satu atau lebih inputan, dan lebih dari satu output.

Berdasarkan flow graph gambar (b) diatas, maka kompleksitas cyclomatic-nya dapat di hitung sebagai berikut :

Grafik alir diatas mempunyai 4 region

$$V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$$

$$V(G) = 3 \text{ predicates nodes} + 1 = 4$$

Hasil kompleksitas cyclomatic menggambarkan banyaknya path dan batas atas sejumlah ujicoba yang harus dirancang dan dieksekusi untuk seluruh perintah dalam program.

Berdasarkan contoh PDL yang kedua, maka jalur independent yang didapat :

Jalur 1 : 1,2,3 – 4 – 5 – 10 – 11 – 12

Jalur 2 : 1,2,3 – 4 – 6 – 7 – 9 – 10 – 11 – 12

Jalur 3 : 1,2,3 – 4 – 8 – 9 – 10 – 11 – 12

Contoh pengujian white-box

Menurut kebutuhan segitiga diberikan di bawah ini untuk menyelesaikan proses dan menyelesaikan tes:

1. Masukan kondisi:

1, kondisi 1: $a + b < c$

2, kondisi 2: $a + c < b$

3, kondisi 3: $b + c < a$

4, kondisi 04:00

5, kondisi 5-0

6, 7 kondisi 6-0, kondisi 7: $a == b$

8, kondisi 8: $a == c$

9, kondisi 9: $b == c$

10, kondisi 10: $a^2 + b^2 c^2 ==$

11, kondisi 11: $a^2 + b^2 c^2 ==$

12, kondisi 12: $c^2 + a^2 == b^2$

2. Output:

1, tidak dapat terbentuk segitiga

2, sebuah segitiga sama sisi

3, segitiga sama kaki

4, segi tiga siku-siku

5, segitiga umum

6, beberapa pihak tidak memenuhi pembatasan

2.7 Kesetaraan Partisi (EP) / Analisis Nilai Batas (BVA)

Partisi kesetaraan (EP) dan analisis nilai batas (BVA) memberikan strategi untuk menulis kasus pengujian white-box. Tidak diragukan lagi, setiap kali Anda menghadapi segala jenis nomor atau membatasi dalam persyaratan, Anda harus waspada untuk masalah EP / BVA.

Sebagai contoh, seseorang mungkin ingin membeli rumah, tetapi mungkin atau mungkin tidak memiliki cukup uang. Mengingat EP / BVA, saya ingin memastikan kasus uji kami meliputi:

1. Properti biaya \$ 100, telah memiliki \$ 200 (kelas kesetaraan “memiliki cukup uang”)
2. Properti biaya \$ 100, memiliki \$ 50 (kelas kesetaraan, “tidak punya cukup uang”)
3. Properti biaya \$ 100, \$ 100 maka (nilai batas)
4. Properti biaya \$ 100, memiliki \$ 99 (nilai batas)
5. Properti biaya \$ 100, memiliki \$ 101 (nilai batas)

Dengan loop pemrograman (seperti perulangan while), pertimbangkan EP dan melaksanakan loop di tengah operasional terikat mereka. Untuk BVA, Anda akan ingin memastikan bahwa Anda menjalankan loop tepat di bawah, sudah tepat, dan tepat di atas kondisi batas mereka.

2.8 Berikut Contoh White Box Testing Software

Contoh Testing White Box 1:

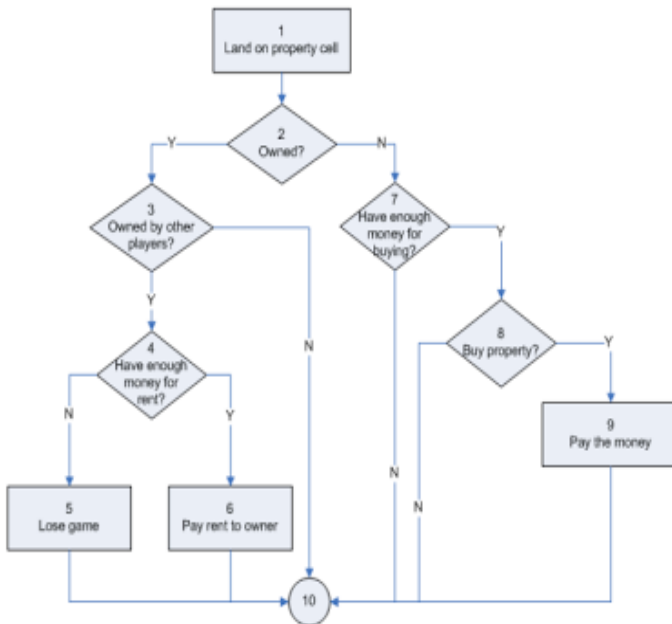


Figure 1: Flowgraph of purchasing property

Menggunakan grafik aliran ini, kita dapat menghitung jumlah jalur independen melalui kode. Kami melakukan ini dengan menggunakan metrik disebut nomor cyclomatic (McCabe, 1976), yang didasarkan pada teori grafik. Cara termudah untuk menghitung jumlah siklomatik adalah dengan menghitung jumlah *conditional* / predikat (diamond) dan tambahkan 1. Dalam contoh di atas, ada lima *conditional*. Oleh karena itu, jumlah *cyclomatic* kami adalah 6, dan kami memiliki enam jalur independen melalui kode. Jadi kita sekarang dapat menghitungnya:

1. 1-2-3-4-5-10 (properti yang dimiliki oleh orang lain, tidak mempunyai uang untuk sewa)
2. 1-2-3-4-6-10 (properti yang dimiliki oleh orang lain, membayar sewa)
3. 1-2-3-10 (properti yang dimiliki oleh pemain)
4. 1-2-7-10 (properti yang tersedia, tidak memiliki cukup uang)
5. 1-2-7-8-10 (properti yang tersedia, punya uang, tidak ingin membelinya)
6. 1-2-7-8-9-10 (properti yang tersedia, punya uang, dan membelinya)

Kami ingin menulis kasus pengujian untuk memastikan bahwa setiap jalur yang akan diuji setidaknya sekali. Seperti dikatakan di atas, jumlah siklomatik adalah batas bawah pada jumlah kasus uji yang akan kita tulis. Uji kasus yang ditentukan dengan cara ini adalah yang kami gunakan dalam pengujian basis patch.

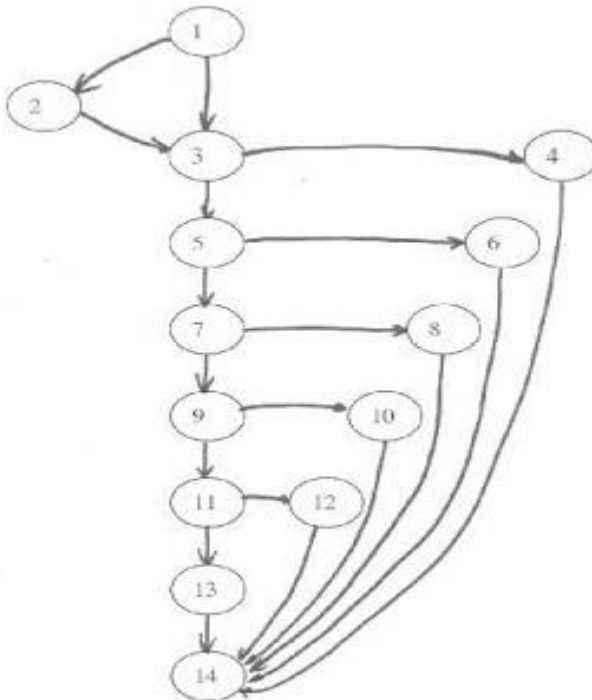
Contoh Pengujian White Box 2:

Step 1: Contoh prosedur di bawah ini menunjukkan bagaimana laporan algoritma dipetakan ke node grafik, nomor di sebelah kiri.

```
public double calculate(int amount)
{
-1- double rushCharge = 0;
-1- if (nextday.equals("yes") )
    {
-2-     rushCharge = 14.50;
    }
-3- double tax = amount * .0725;
-3- if (amount >= 1000)
    {
-4-     shipcharge = amount * .06 + rushCharge;
    }
-5- else if (amount >= 200)
    {
-6-     shipcharge = amount * .08 + rushCharge;
    }
-7- else if (amount >= 100)
    {
-8-     shipcharge = 13.25 + rushCharge;
```

```
    }  
-9- else if (amount >= 50)  
    {  
-10-     shipcharge = 9.95 + rushCharge;  
    }  
-11- else if (amount >= 25)  
    {  
-12-     shipcharge = 7.25 + rushCharge;  
    }  
    else  
    {  
-13-     shipcharge = 5.25 + rushCharge;  
    }  
-14- total = amount + tax + shipcharge;  
-14- return total;  
    } //end calculate
```

Dibawah ini adalah flowchart dari contoh program diatas:



Step 2: Menentukan kompleksitas *cyclomatic* dari grafik aliran.

$$\begin{aligned} V(G) &= E - N + 2 \\ &= 19 - 14 + 2 \\ &= 7 \end{aligned}$$

Keterangan:

E : Jumlah Busur atau Link

N : Jumlah Simpul

Ini menjelaskan bahwa batas atas pada ukuran basis set. Artinya, memberikan jumlah jalur independen yang perlu kita cari.

Step 3: Menentukan dasar jalur independen

Path 1: 1 - 2 - 3 - 5 - 7 - 9 - 11 - 13 - 14

Path 2: 1 - 3 - 4 - 14

Path 3: 1 - 3 - 5 - 6 - 14

Path 4: 1 - 3 - 5 - 7 - 8 - 14

Path 5: 1 - 3 - 5 - 7 - 9 - 10 - 14

Path 6: 1 - 3 - 5 - 7 - 9 - 11 - 12 - 14

Path 7: 1 - 3 - 5 - 7 - 9 - 11 - 13 - 14

Step 4: Menyiapkan test cases bahwa pelaksanaan kekuatan setiap jalur di set dasar.

<u>path</u>	<u>nextday</u>	<u>amount</u>	<u>expected result</u>
1	yes	10	30.48
2	no	1500	??????
3	no	300	345.75
4	no	150	174.125
5	no	75	90.3875
6	no	30	39.425
7	no	10	15.975

Penyataan pengulangan statement di tengah-tengah blok diperlukan meskipun ada gambaran sampai akhir, Jika itu adalah simbol terminal tambahan.

BAB III

KESIMPULAN

Dari hasil pembahasan tentang Makalah *White Box Testing*, maka diambil kesimpulan dengan adanya metode untuk pengujian *White Box Testing*, maka dapat dipastikan perangkat lunak harus diuji terlebih dahulu untuk menguji tingkat kesalahan sistem.

Sistem yang berjalan dengan optimal akan dapat membantu kinerja *user* tanpa adanya masalah yang dialami oleh sistem, oleh karena itu pengujian sistem sangat penting dilakukan untuk menguji kelayakan sistem yang akan di pakai oleh pengguna nantinya.

DAFTAR PUSTAKA

http://ayuliana_st.staff.gunadarma.ac.id/Downloads/files/25114/Pertemuan+04+-+%28Software+Testing+Techniques%29.pdf

<http://blog-arul.blogspot.com/2012/12/pengujian-whitebox-testing.html#ixzz2Nleivxjn>

Beizer, B. (1990). Software Testing Techniques. Boston, International Thompson Computer Press