

WHITE BOX TESTING DAN BLACK BOX TESTING

White Box



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%.

Pengujian dilakukan berdasarkan bagaimana suatu software menghasilkan output dari input. Pengujian ini dilakukan berdasarkan kode program.

Disebut juga struktural testing atau glass box testing

Teknik pengujian :

1. Menggambarkan kode program ke dalam graph yaitu node & edge.

Jika berhubungan bernilai 1, bila tidak bernilai nol.

Dalam pengujian ini akan diperoleh hasil :

- * Kemungkinan source code yang dieksekusi
- * Waktu yang dibutuhkan
- * Memori yang digunakan
- * Sumber daya yang digunakan

2. Basic path, yaitu pengukuran kompleksitas kode program dan pendefinisian alur yang akan dieksekusi.

Digambarkan sequence, if, atau while nya

Uji coba basis path adalah teknik uji coba white box yg diusulkan Tom McCabe. Metode ini memungkinkan perancang test case mendapatkan ukuran kekompleksan logical dari perancangan prosedural dan menggunakan ukuran ini sbg petunjuk untuk mendefinisikan basis set dari jalur pengerjaan. Test case yg didapat digunakan untuk mengerjakan basis set yg menjamin pengerjaan setiap perintah minimal satu kali selama uji coba.

3. Data flow testing, untuk mendeteksi penyalahgunaan data dalam sebuah program.

4. Cyclomatic Complexity

Cyclomatic Complexity merupakan suatu sistem pengukuran yang menyediakan ukuran kuantitatif dari kompleksitas logika suatu program. Pada Basis Path Testing, hasil dari cyclomatic complexity digunakan untuk menentukan banyaknya independent paths.

Independent path adalah sebuah kondisi pada program yang menghubungkan node awal dengan node akhir.

Terdapat 2 persamaan yang digunakan, yaitu:

$$V(G) = E - N + 2 \text{ atau } V(G) = P + 1$$

Keterangan:

$V(G)$ = cyclomatic complexity untuk flow graph G

E = Jumlah edge (panah)

N = Jumlah node (lingkaran)

P = Jumlah predicate node

- *Kelebihan White Box Testing*

Kesalahan logika. Digunakan pada sintaks 'if' dan pengulangan. Dimana White Box Testing akan mendeteksi kondisi-kondisi yang tidak sesuai dan mendeteksi kapan proses pengulangan akan berhenti.

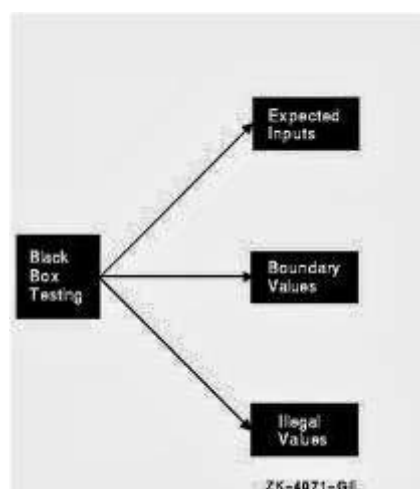
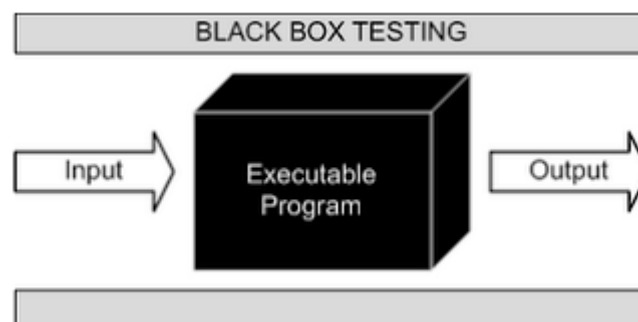
Ketidaksesuaian asumsi. Menampilkan asumsi yang tidak sesuai dengan kenyataan, untuk di analisa dan diperbaiki.

Kesalahan ketik. Mendeteksi bahasa pemrograman yang bersifat case sensitive.

- *Kelemahan White Box Testing*

- Untuk perangkat lunak yang tergolong besar, White Box Testing dianggap sebagai strategi yang tergolong boros, karena akan melibatkan sumber daya yang besar untuk melakukannya

Black Box



Pengertian Black Box Testing

Black box testing adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak. Jadi dianalogikan seperti kita melihat suatu kotak hitam, kita hanya bisa melihat penampilannya saja, tanpa tau ada apa dibalik bungkus hitamnya. Sama seperti pengujian black box, mengevaluasi hanya dari tampilan luarnya (interface nya), fungsionalitasnya. tanpa mengetahui apa sesungguhnya yang terjadi dalam proses detilnya (hanya mengetahui input dan output).

Black Box pengujian adalah metode pengujian perangkat lunak yang menguji fungsionalitas aplikasi yang bertentangan dengan struktur internal atau kerja (lihat pengujian white-box). Pengetahuan khusus dari kode aplikasi / struktur internal dan pengetahuan pemrograman pada umumnya tidak diperlukan. Uji kasus dibangun di sekitar spesifikasi dan persyaratan, yakni, aplikasi apa yang seharusnya dilakukan. Menggunakan deskripsi eksternal perangkat lunak, termasuk spesifikasi, persyaratan, dan desain untuk menurunkan uji kasus. Tes ini dapat menjadi fungsional atau non-fungsional, meskipun biasanya fungsional. Perancang uji memilih input yang valid dan tidak valid dan menentukan output yang benar. Tidak ada pengetahuan tentang struktur internal benda uji itu.

Metode uji dapat diterapkan pada semua tingkat pengujian perangkat lunak: unit, integrasi, fungsional, sistem dan penerimaan. Ini biasanya terdiri dari kebanyakan jika tidak semua pengujian pada tingkat yang lebih tinggi, tetapi juga bisa mendominasi unit testing juga.

Pengujian pada Black Box berusaha menemukan kesalahan seperti:

- Fungsi-fungsi yang tidak benar atau hilang
- Kesalahan interface
- Kesalahan dalam struktur data atau akses database eksternal
- Kesalahan kinerja
- Inisialisasi dan kesalahan terminasi

Teknik khas Black Box Testing desain meliputi:

1. DECISION TABLE

Decision Table adalah cara yang tepat belum kompak untuk model logika rumit, seperti diagram alir dan jika-then-else dan switch-laporan kasus, kondisi mengaitkan dengan tindakan untuk melakukan, tetapi dalam banyak kasus melakukannya dengan cara yang lebih elegan.

Pada tahun 1960-an dan 1970-an berbagai “Decision Table Based” bahasa seperti Filetab sangat populer untuk pemrograman bisnis.

2. ALL-PAIRS TESTING

All-pairs testing atau pairwise testing adalah metode pengujian perangkat lunak kombinatorial bahwa, untuk setiap pasangan parameter masukan ke sistem (biasanya, sebuah algoritma perangkat lunak), tes semua kombinasi yang mungkin diskrit parameter tersebut. Menggunakan vektor uji dipilih dengan cermat, hal ini dapat dilakukan jauh lebih cepat daripada pencarian lengkap semua kombinasi dari semua parameter, dengan “parallelizing” pengujian pasangan parameter. Jumlah tes biasanya $O(nm)$, dimana n dan m adalah jumlah kemungkinan untuk masing-masing dua parameter dengan pilihan yang paling.

Alasan di balik semua-All-pairs testing ini: yang sederhana dalam sebuah program umumnya dipicu oleh parameter masukan tunggal. Kategori paling sederhana berikutnya bug terdiri dari mereka bergantung pada interaksi antara pasangan parameter, yang bisa ditangkap dengan menguji semua-pasangan. yang melibatkan interaksi antara tiga atau lebih parameter secara progresif kurang umum [2], sementara pada saat yang sama waktu semakin lebih

mahal untuk mencari oleh pengujian mendalam, yang sebagai batas pengujian lengkap semua input yang mungkin.

Banyak metode pengujian menganggap semua-pasang pengujian sistem atau subsistem sebagai kompromi biaya-manfaat yang wajar antara sering komputasi tidak layak tingkat tinggi metode pengujian kombinatorial, dan metode yang kurang lengkap yang gagal untuk menjalankan semua pasangan yang mungkin dari parameter. Karena tidak ada teknik pengujian dapat menemukan semua bug, semua-pasangan pengujian biasanya digunakan bersama dengan berbagai teknik jaminan mutu seperti unit testing, eksekusi simbolik, pengujian bulu halus, dan memeriksa kode.

3. STATE TRANSITION TABLE

Dalam teori automata dan logika sekuensial, state transition table adalah tabel yang menunjukkan apa yang negara (atau negara dalam kasus robot terbatas nondeterministic) suatu semiautomaton terbatas atau mesin finite state akan pindah ke, berdasarkan kondisi saat ini dan masukan lainnya. Sebuah tabel negara pada dasarnya adalah sebuah tabel kebenaran di mana beberapa input adalah kondisi saat ini, dan output termasuk negara berikutnya, bersama dengan keluaran lain.

state transition table adalah salah satu dari banyak cara untuk menentukan mesin negara, cara lain menjadi diagram negara, dan persamaan karakteristik.

4. EQUIVALENCE PARTITIONING

Equivalence partitioning adalah pengujian perangkat lunak teknik yang membagi data masukan dari unit perangkat lunak menjadi beberapa partisi data dari mana test case dapat diturunkan. Pada prinsipnya, uji kasus dirancang untuk menutupi setiap partisi minimal sekali. Teknik ini mencoba untuk mendefinisikan kasus uji yang mengungkap kelas kesalahan, sehingga mengurangi jumlah kasus uji yang harus dikembangkan.

Dalam kasus yang jarang Equivalence partitioning juga diterapkan pada output dari komponen perangkat lunak, biasanya itu diterapkan pada masukan dari komponen diuji. Partisi ekivalen biasanya berasal dari spesifikasi persyaratan untuk atribut masukan yang mempengaruhi pengolahan benda uji. Sebuah masukan telah rentang tertentu yang rentang sah dan lainnya yang tidak valid. Data yang tidak valid di sini tidak berarti bahwa data tidak benar, itu berarti bahwa data ini terletak diluar dari partisi tertentu. Hal ini mungkin lebih tepat dijelaskan oleh contoh fungsi yang mengambil sebuah parameter "bulan". Jangkauan bulan adalah 1 sampai 12, mewakili Januari-Desember. Jangkauan ini disebut partisi. Dalam contoh ini ada dua partisi lebih lanjut rentang tidak valid. Partisi pertama akan menjadi tidak valid ≤ 0 dan partisi tidak valid kedua akan menjadi > 13 .

5. BOUNDARY VALUES ANALYSIS

Boundary value analysis merupakan suatu teknik pengujian perangkat lunak di mana tes dirancang untuk mencakup perwakilan dari nilai-nilai batas. Nilai-nilai di tepi sebuah partisi kesetaraan atau sebesar nilai terkecil di kedua sisi tepi. Nilai dapat berupa rentang masukan atau keluaran dari komponen perangkat lunak. Karena batas-batas tersebut adalah lokasi umum untuk kesalahan yang mengakibatkan kesalahan perangkat lunak mereka sering dilakukan dalam kasus-kasus uji.

Dokumentasi komponen software, mencangkup pemeriksaan dokumen dari software itu sendiri, yaitu :

- * Flowchart yang dibuat
- * Deskripsi input yang digunakan
- * Deskripsi output yang digunakan
- * Deskripsi output yang dihasilkan
- * Kesesuaian penulisan (akurasi)
- * Kontrol/kendali terhadap sistem yang dibuat

Strategi Black Box System, meliputi :

- * **Batasan nilai untuk testing**, meliputi beberapa nilai, yaitu
 - Nilai minimum variabel input
 - Nilai di atas nilai minimum
 - Nilai normal
 - Nilai di bawah nilai maksimum
 - Nilai maksimum
- * **Equivalent Class Testing**, yaitu mengelompokkan input yang direpresentasikan sebagai hasil yang valid atau invalid. Contoh :
Rekrutasi pegawai berdasarkan pengalaman kerja :
<1 thn : diterima, part time
1-3 thn : diterima, sebagai tenaga kerja profesional
>4 thn : diterima, sebagai pegawai tetap

Kesalahan yang dapat terdeteksi melalui testing ini ialah :

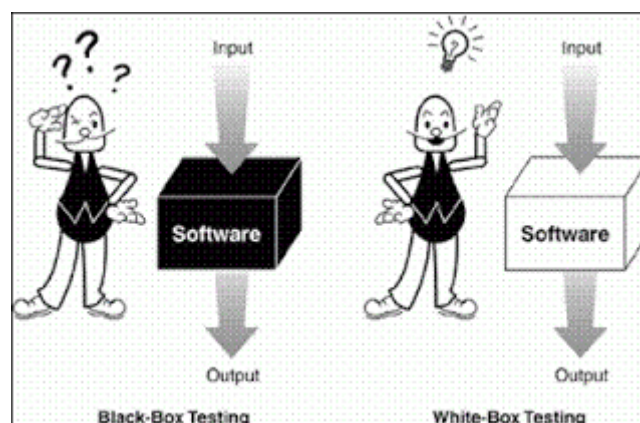
- * kebenaran dokumentasi
- * akses basis data
- * hasil akhir program

Kelebihan black box testing :

- * Spesifikasi program dapat ditentukan di awal
- * Dapat digunakan untuk menilai konsistensi program
- * Testing dilakukan berdasarkan spesifikasi
- * Tidak perlu melihat kode program secara detail

Kekurangan black box testing :

- * Bila spesifikasi program yang dibuat kurang jelas dan ringkas, maka akan sulit membuat dokumentasi setepat mungkin



Perbedaan White Box & Black Box

- *White box (Struktural)*
 - Dilakukan oleh penguji yang mengetahui tentang QA.

- Melakukan testing pada software/program aplikasi menyangkut security dan performance program tersebut (meliputi tes code, desain implementasi, security, data flow, software failure).
- Dilakukan seiring dengan tahapan pengembangan software atau pada tahap testing.
 - *Metode BlackBox (Fungsional)*
- Dilakukan oleh penguji Independent.
- Melakukan pengujian berdasarkan apa yang dilihat, hanya fokus terhadap fungsionalitas dan output. Pengujian lebih ditujukan pada desain software sesuai standar dan reaksi apabila terdapat celah-celah bug/vulnerabilitas pada program aplikasi tersebut setelah dilakukan white box testing.
- Dilakukan setelah white box testing.

Pengertian, Perbedaan White Box Dan Black Box Testing dan Contoh

- White Box

- *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%.

Pengujian white box:

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

Pelaksanaan pengujian white box:

- 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.
- Menjalani logical decision pada sisi dan false.
- Mengeksekusi pengulangan (looping) dalam batas-batas yang ditentukan.
- Menguji struktur data internal.

Berdasarkan konsep pengujian; White box (structural) testing / glass box testing : memeriksa kalkulasi internal path untuk mengidentifikasi kesalahan.

Langkah-langkah white box:

- Mendefinisikan semua alur logika
- Membangun kasus untuk digunakan dalam pengujian
- Melakukan pengujian.

- *Kelebihan White Box Testing*

- **Kesalahan logika.** Digunakan pada sintaks 'if' dan pengulangan. Dimana White Box Testing akan mendeteksi kondisi-kondisi yang tidak sesuai dan mendeteksi kapan

proses pengulangan akan berhenti.

- **Ketidaksesuaian asumsi.** Menampilkan asumsi yang tidak sesuai dengan kenyataan, untuk di analisa dan diperbaiki.

- **Kesalahan ketik.** Mendeteksi bahasa pemrograman yang bersifat case sensitive.

- *Kelemahan White Box Testing*

- Untuk perangkat lunak yang tergolong besar, White Box Testing dianggap sebagai strategi yang tergolong boros, karena akan melibatkan sumber daya yang besar untuk melakukannya.

Jenis white box;

<0>**Basis path**:: Metode identifikasi yang berdasarkan pada jalur,, struktur atau koneksi yang ada dari suatu sistem ini biasa disebut juga sebagai *branch testing*,, karena cabang-cabang dari kode atau fungsi logika diidentifikasi dan dites, atau disebut juga sebagai *control-flow testing*.

Ada 2 bentuk Basis path, yaitu:

- Zero Path: Jalur penghubung yang tidak penting atau jalur pintas yang ada pada suatu sistem.
- One Path: Jalur penghubung yang penting atau berupa proses pada suatu sistem.

<0>**Cyclomatic Complexity**

- Adalah pengukuran software yang memberikan pengukuran kuantitatif dari kompleksitas logika program.
- Pada konteks metode basis path testing , nilai yang dihitung bagi cyclomatic complexity menentukan jumlah jalur-jalur yang independen dalam kumpulan basis suatu program dan memberikan jumlah tes minimal yang harus dilakukan untuk memastikan bahwa semua pernyataan telah dieksekusi sekurangnya satu kali.
- Jalur independen adalah tiap jalur pada program yang memperlihatkan 1 kelompok baru dari pernyataan proses atau kondisi baru.

<0>**Graph Matrix**

- Adalah matrik berbentuk segi empat sama sisi, dimana jumlah baris dan kolom sama dengan jumlah node, dan identifikasi baris dan kolom sama dengan identifikasi node, serta isi data adalah keberadaan penghubung antar node (edges).
- Beberapa properti yang dapat ditambahkan sebagai pembobotan pada koneksi antar node di dalam graph matrix, sebagai berikut:
 1. Kemungkinan jalur (Edge) akan dilalui / dieksekusi.
 2. Waktu proses yang diharapkan pada jalur selama proses transfer dilakukan.
 3. Memori yang dibutuhkan selama proses transfer dilakukan pada jalur.
 4. Sumber daya (resources) yang dibutuhkan selama proses transfer dilakukan pada jalur.

Control Structur Testing, meliputi;

- Condition testing:: Suatu metode disain test case yang memeriksa kondisi logika yang terdapat pada modul program.
- Data flow testing:: Metode data flow testing memilih jalur program berdasarkan pada lokasi dari definisi dan penggunaan variabel-variabel pada program.
- Loop testing:: suatu teknik white box testing yang berfokus pada validitas konstruksi loop secara eksklusif. Ada 4 kelas dari loop, yaitu;
 - Simple Loops
 - Nested Loops
 - Concatenated Loops
 - Unstructured Loops

Contoh kasus : Imperial Taxi Services (ITS) :

$$V(G) = R = 6$$

$$V(G) = E - N + 2 = 21 - 17 + 2 = 6$$

$$V(G) = P + 1 = 5 + 1 = 6$$

$$\text{Rumus : } V(G) = R = E - N + 2 = P + 1$$

Keterangan :

$V(G)$ = cyclometric complexity graph

R = jumlah region dalam program flow graph

E = jumlah edge

N = jumlah node

P = jumlah decision (percabangan)

- Black Box

- *Pengertian Black Box Testing*

Black box testing adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak. Jadi dianalogikan seperti kita melihat suatu kotak hitam, kita hanya bisa melihat penampilan luarnya saja, tanpa tau ada apa dibalik bungkus hitam nya. Sama seperti pengujian black box, mengevaluasi hanya dari tampilan luarnya(interface nya) , fungsionalitasnya.tanpa mengetahui apa sesungguhnya yang terjadi dalam proses detilnya (hanya mengetahui input dan output).

- *Kelebihan Black Box*

- Dapat memilih subset test secara efektif dan efisien
- Dapat menemukan cacat
- Memaksimalkan testing investmen
- *Kelemahan Black Box*
 - Tester tidak pernah yakin apakah PL tersebut benar – benar lulus uji.

Contoh Black Box Testing dengan Equivalence Partitioning :

Pemeliharaan data untuk aplikasi bank yang sudah diotomatisasikan. Pemakai dapat memutar nomor telepon bank dengan menggunakan mikro komputer yang terhubung dengan password yang telah ditentukan dan diikuti dengan perintah-perintah. Data yang diterima adalah :

- Kode area : kosong atau 3 digit
- Prefix : 3 digit atau tidak diawali 0 atau 1
- Suffix : 4 digit
- Password : 6 digit alfanumerik
- Perintah : check, deposit, dll

Selanjutnya kondisi input digabungkan dengan masing-masing data elemen dapat ditentukan sebagai berikut:

- Kode area : kondisi input, Boolean –kode area mungkin ada atau tidak kondisi input, range –nilai ditentukan antara 200 dan 999
- Prefix : kondisi input range > 200 atau tidak diawali 0 atau 1
- Suffix : kondisi input nilai 4 digit
- Password : kondisi input boolean –password mungkin diperlukan atau tidak kondisi input nilai dengan 6 karakter string
- Perintah : kondisi input set berisi perintah-perintah yang telah didefinisikan

- Perbedaan White Box & Black Box

- *White box (Struktural)*
 - Dilakukan oleh penguji yang mengetahui tentang QA.
 - Melakukan testing pada software/program aplikasi menyangkut security dan performance program tersebut (meliputi tes code, desain implementasi, security, data flow, software failure).

- Dilakukan seiring dengan tahapan pengembangan software atau pada tahap testing.

- *Metode BlackBox (Fungsional)*

- Dilakukan oleh penguji Independent.
- Melakukan pengujian berdasarkan apa yang dilihat, hanya fokus terhadap fungsionalitas dan output. Pengujian lebih ditujukan pada desain software sesuai standar dan reaksi apabila terdapat celah-celah bug/vulnerabilitas pada program aplikasi tersebut setelah dilakukan white box testing.
- Dilakukan setelah white box testing.

WHITE BOX TESTING

- Pengertian White Box Testing

White Box Testing merupakan cara pengujian dengan melihat ke dalam modul untuk meneliti kode-kode program yang ada, dan menganalisis apakah ada kesalahan atau tidak. Jika ada modul yang menghasilkan output yang tidak sesuai dengan proses bisnis yang dilakukan, maka baris-baris program, variabel, dan parameter yang terlibat pada unit tersebut akan dicek satu persatu dan diperbaiki, kemudian di-compile ulang.

- Dengan menggunakan white box akan didapatkan kasus uji yang :

- Menguji semua keputusan logikal
- Menguji seluruh Loop yang sesuai dengan batasannya
- Menguji seluruh struktur data internal yang menjamin validitas

- Kelebihan White Box Testing

- Kesalahan Logika

Digunakan pada sintaks 'if' dan pengulangan. Dimana White Box Testing akan mendeteksi kondisi-kondisi yang tidak sesuai dan mendeteksi kapan proses pengulangan akan berhenti.

- Ketidaksesuaian asumsi

Menampilkan asumsi yang tidak sesuai dengan kenyataan, untuk di analisa dan diperbaiki.

- Kesalahan ketik

Mendeteksi bahasa pemrograman yang bersifat case sensitive.

- Kelemahan White Box Testing

Untuk perangkat lunak yang tergolong besar, White Box Testing dianggap sebagai strategi yang tergolong boros, karena akan melibatkan sumber daya yang besar untuk melakukannya.

Pengujian White Box

Pengujian white box 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%.

Penggunaan metode pengujian white box dilakukan untuk :

- Memberikan jaminan bahwa semua jalur independen suatu modul digunakan minimal satu kali
- Menggunakan semua keputusan logis untuk semua kondisi true atau false

- Mengeksekusi semua perulangan pada batasan nilai dan operasional pada setiap kondisi.
- Menggunakan struktur data internal untuk menjamin validitas jalur keputusan.






Persyaratan dalam menjalankan strategi White Box Testing

- Mendefinisikan semua alur logika
- Membangun kasus untuk digunakan dalam pengujian
- Mengevaluasi semua hasil pengujian
- Melakukan pengujian secara menyeluruh

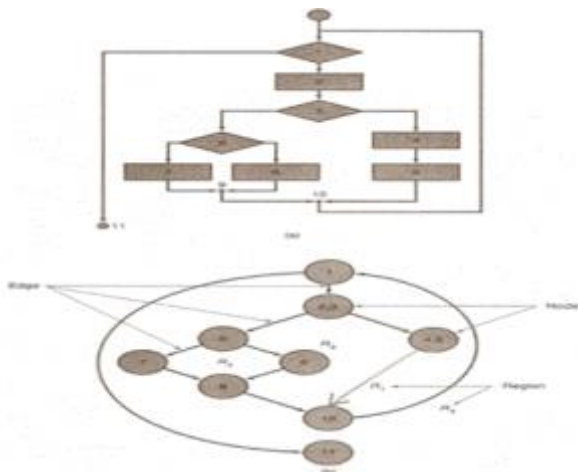
A) Notasi Diagram Alir (Path Graph Notation)

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.

Tabel 1. Notasi Diagram Alir

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

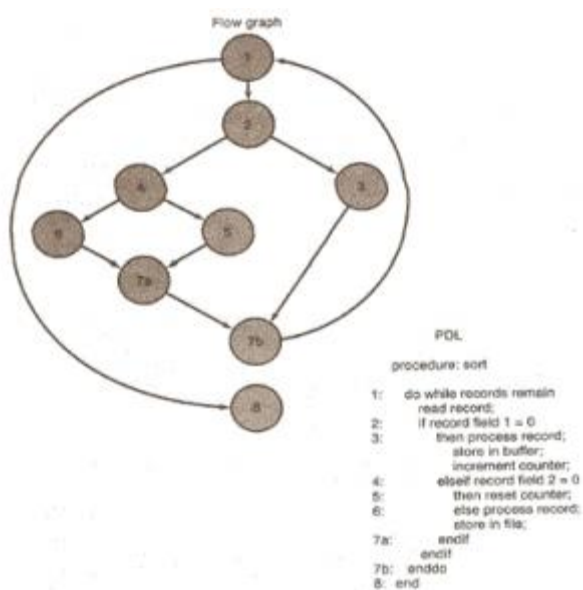
Untuk mengilustrasikan kegunaan dari diagram alir dapat dilihat pada gambar dibawah ini.



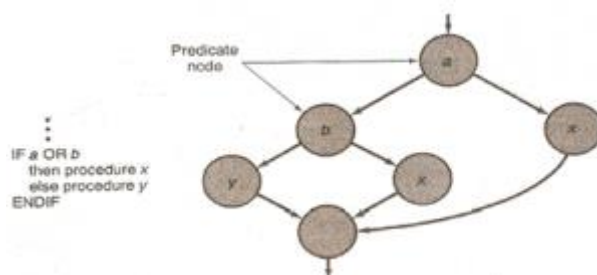
Gambar bagian (a) digunakan untuk menggambarkan struktur kontrol program, sedangkan gambar bagian (b) setiap lingkaran disebut dengan flow graph node, merepresentasikan satu atau lebih perintah prosedural. Urutan dari kotak simbol proses dan belah ketupat simbol keputusan dapat digambarkan menjadi sebuah node, sedangkan anak panah disebut edges, menggambarkan aliran dari kontrol sesuai dengan diagram alir.

Sebuah edge harus berakhir pada sebuah node walaupun tidak semua node merepresentasikan perintah prosedural. Area yang dibatasi oleh edge dan node disebut region, area diluar graph juga dihitung sebagai region.

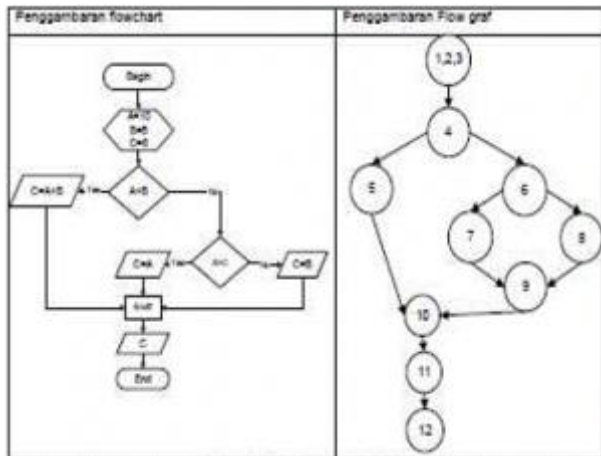
Setiap representasi rancangan prosedural dapat diterjemahkan kedalam flow graph. Gambar (a) dibawah ini merupakan bagian dari PDL (Program Design Language) dan flow graph-nya (perhatikan nomor untuk setiap perintahnya).



Ketika kondisi gabungan ditemukan, maka penggambaran flow graph akan menjadi lebih rumit. Kondisi gabungan biasanya muncul jika satu atau lebih operator Boolean (OR, AND, NAND, NOR) ditemukan dalam perintah, seperti terlihat pada gambar (b) dibawah ini :



Contoh yang kedua mengenai penggambaran flow chart dan flow graf:



Gambar 5. Contoh yang kedua metode basis path

B) Kompleksitas Siklomatis (Cyclomatic Complexity)

Kompleksitas Siklomatis adalah metrik 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 :

1. Grafik alir diatas mempunyai 4 region
2. $V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$
3. $V(G) = 3 \text{ predicates nodes} + 1 = 4$

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

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.

BLACK BOX TESTING

Black-box testing adalah metode pengujian perangkat lunak yang tes fungsionalitas dari aplikasi yang bertentangan dengan struktur internal atau kerja (lihat pengujian white-box). pengetahuan khusus dari kode aplikasi / struktur internal dan pengetahuan pemrograman pada umumnya tidak diperlukan. Uji kasus dibangun di sekitar spesifikasi dan persyaratan, yakni, aplikasi apa yang seharusnya dilakukan. Menggunakan deskripsi eksternal perangkat lunak, termasuk

spesifikasi, persyaratan, dan desain untuk menurunkan uji kasus. Tes ini dapat menjadi fungsional atau non-fungsional, meskipun biasanya fungsional. Perancang uji memilih input yang valid dan tidak valid dan menentukan output yang benar. Tidak ada pengetahuan tentang struktur internal benda uji itu.

Metode uji dapat diterapkan pada semua tingkat pengujian perangkat lunak: unit, integrasi, fungsional, sistem dan penerimaan. Ini biasanya terdiri dari kebanyakan jika tidak semua pengujian pada tingkat yang lebih tinggi, tetapi juga bisa mendominasi unit testing juga.

Metode ujicoba blackbox memfokuskan pada keperluan fungsional dari software. Karna itu ujicoba blackbox memungkinkan pengembang software untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba blackbox bukan merupakan alternatif dari ujicoba whitebox, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode whitebox.

Ujicoba blackbox berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya :

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan performa
5. kesalahan inisialisasi dan terminasi

SUMBER :

<http://universitaspendidikan.com/pengertian-white-box-dan-contoh-white-box-testing/>
<http://bangwildan.web.id/berita-176-white-box-testing--black-box-testing.html>

Pen jelaskan bagaimana jenis-jenis pengujian berikut ini dilakukan pada kasus anda:

1. Pengujian white box

2. Pengujian basis path

3. Pengujian struktur kontrol

/

4. Pengujian black box

jawab

1 .Pengujian White Box

Pengujian white box 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%.

Penggunaan metode pengujian white box dilakukan untuk :

Penggunaan metode pengujian white box dilakukan untuk :

- Memberikan jaminan bahwa semua jalur independen suatu modul digunakan minimal satu kali
- Menggunakan semua keputusan logis untuk semua kondisi true atau false
- Mengeksekusi semua perulangan pada batasan nilai dan operasional pada setiap kondisi.
- Menggunakan struktur data internal untuk menjamin validitas jalur keputusan.

Persyaratan dalam menjalankan strategi White Box Testing

- Mendefinisikan semua alur logika
- Membangun kasus untuk digunakan dalam pengujian
- Mengevaluasi semua hasil pengujian
- Melakukan pengujian secara menyeluruh

White Box Testing

Adalah meramalkan cara kerja perangkat lunak secara rinci, karenanya logikal path (jalur logika) perangkat lunak akan ditest dengan menyediakan test case yang akan mengerjakan kumpulan kondisi dan atau pengulangan secara spesifik. Secara sekilas dapat diambil kesimpulan white box testing merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

UJI COBA WHITE BOX

Uji coba white box adalah metode perancangan test case yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan test case. Dengan rnenggunakan metode white box, analis sistem akan dapat memperoleh test case yang:

menjamin seluruh independent path di dalam modul yang dikerjakan sekurang-kurangnya sekali

mengerjakan seluruh keputusan logikal

mengerjakan seluruh loop yang sesuai dengan batasannya

mengerjakan seluruh struktur data internal yang menjamin validitas

2. PENGUJIAN BASIS PATH

Uji coba basis path adalah teknik uji coba white box yg diusulkan Tom McCabe. Metode ini memungkinkan perancang test case mendapatkan ukuran kekompleksan logical dari perancangan prosedural dan menggunkan ukuran ini sbg petunjuk untuk mendefinisikan basis set dari jalur pengerjaan. Test case yg didapat digunakan untuk mengerjakan basis set yg menjamin pengerjaan setiap perintah minimal satu kali selama uji coba.

Notasi diagram alir

Sequence	if	while	until	case
-----------------	-----------	--------------	--------------	-------------

Gambar 9.1

Untuk menggambarkan pemakaian diagram alir diberikan contoh perancangan prosedural dalam bentuk flowchart

Gambar Diagram Alir

Selanjutnya diagram alir diatas dipetakan ke grafik alir

node

Gambar 9.3 Grafik Alir

Lingkaran/node :

menggambarkan satu/lebih perintah prosedural. Urutan proses dan keputusan dapat dipetakan dalam satu node.

Tanda panah/edge :

menggambarkan aliran kontrol. Setiap node harus mempunyai tujuan node

Region :

adalah daerah yg dibatasi oleh edge dan node. Termasuk daerah diluar grafik alir.

Contoh menterjemahkan pseudo code ke grafik alir

1: do while record masih ada

 baca record

2: if record ke 1 = 0

3: then proses record

 simpan di buffer

 naikan kounter

4: else if record ke 2 = 0

5 then reser kounter

6 proses record

 simpan pada file

7a: endif

endif

7b: enddo

8 : end

Gambar Menerjemahkan PDL ke grafik Alir

Nomor pd pseudo code berhubungan dengan nomor node. Apabila ditemukan kondisi majemuk (compound condition) pada pseudo code pembuatan grafik alir menjadi rumit. Kondisi majemuk mungkin terjadi pada operator Boolean (AND, OR, NAND, NOR) yg dipakai pada perintah if.

Contoh :

if A or B

then procedure x

else procedure y

endif

Gambar Logika Gabungan

Node dibuat terpisah untuk masing-masing kondisi A dan B dari pernyataan IF A OR B. Masing-masing node berisi kondisi yg disebut predicate node dan mempunyai karakteristik dua atau lebih edge darinya.

-

-

3. PENGUJIAN STRUKTUR KONTROL

PENGUJIAN KONDISI

Adalah sebuah metode disain test case yang menggunakan kondisi logis yang ada pada suatu program.

Contoh : Kondisi sederhana dari persamaan relasional

E_1 (Operator relasional) E_2

E_1 dan E_2 merupakan persamaan matematika

Operator Relasional adalah salah satu dari operator berikut ini :

$<, \leq, =, \neq, (-=), >, \geq$

Operator Boolean : OR ('|'), AND ('&'), NOT ('-')

Setiap bahasa pemrograman memiliki kemampuan untuk melakukan pengujian kondisi agar program dapat berjalan dinamis dan interaktif. Untuk menguji setiap kondisi, diperlukan pembandingan yang bisa sama dengan, lebih besar, lebih kecil, atau tidak sama dengan lainnya. Untuk mengujinya dibutuhkan operator yang dapat menyatakan kondisi tersebut, yaitu dengan operator :

<	Lebih kecil
>	Lebih besar
<=	Lebih kecil & sama dengan
>=	Lebih besar & sama dengan
=	Sama dengan
!=	Tidak sama dengan

1. 4. Pengujian Black Box

Black Box Testing dan Contoh Pengujian Black Box

Advertisement

Dalam testing dan implementasi sistem dikenal 2 metode pengujian yang populer, yakni pengujian black box dan pengujian white box, untuk

[maksud dan contoh pengujian white box](#) beberapa waktu lalu sudah pernah saya share, jadi untuk yang kali ini membahas [black box testing dan contoh dari pengujian black box](#) itu sendiri.

Black-Box Testing merupakan pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, tester dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program.

Ciri-Ciri Black Box Testing

- 1.Black box testing berfokus pada kebutuhan fungsional pada software, berdasarkan pada spesifikasi kebutuhan dari software.
- 2.Black box testing bukan teknik alternatif daripada white box testing. Lebih daripada itu, ia merupakan pendekatan pelengkap dalam mencakup error dengan kelas yang berbeda dari metode white box testing.

3. Black box testing melakukan pengujian tanpa pengetahuan detail struktur internal dari sistem atau komponen yang dites. juga disebut sebagai behavioral testing, specification-based testing, input/output testing atau functional testing

Pada [black box testing](#) terdapat jenis teknik disain tes yang dapat dipilih berdasarkan pada tipe testing yang akan digunakan, yang diantaranya :

1. Equivalence Class Partitioning
2. Boundary Value Analysis
3. State Transitions Testing
4. Cause-Effect Graphing

Kategori error yang akan diketahui melalui black box testing :

- Fungsi yang hilang atau tak benar
- Error dari antar-muka
- Error dari struktur data atau akses eksternal database
- Error dari kinerja atau tingkah laku
- Error dari inisialisasi dan terminasi

Equivalence Partitioning

Merupakan metode black box testing yang membagi domain masukan dari suatu program ke dalam kelas-kelas data, dimana test cases dapat diturunkan [BCS97a]. Equivalence partitioning berdasarkan pada premis masukan dan keluaran dari suatu komponen yang dipartisi ke dalam kelas-kelas, menurut spesifikasi dari komponen tersebut, yang akan diperlakukan sama (ekuivalen) oleh komponen tersebut. Dapat juga diasumsikan bahwa masukan yang sama akan menghasilkan respon yang sama pula. Nilai tunggal pada suatu partisi ekuivalensi diasumsikan sebagai representasi dari semua nilai dalam partisi.

Analisa partisi pada Equivalence Partitioning Black Box

1. Tester menyediakan suatu model komponen yang dites yang merupakan partisi dari nilai masukan dan keluaran komponen.
2. Masukan dan keluaran dibuat dari spesifikasi dari tingkah laku komponen.
3. Partisi adalah sekumpulan nilai, yang dipilih dengan suatu cara dimana semua nilai di dalam partisi, diharapkan untuk diperlakukan dengan cara yang sama oleh komponen (seperti mempunyai proses yang sama).

Partisi untuk nilai valid dan tidak valid harus ditentukan.

Contoh Black Box Testing dengan Equivalence Partitioning :

Pemeliharaan data untuk aplikasi bank yang sudah diotomatisasikan. Pemakai dapat memutar nomor telepon bank dengan menggunakan mikro komputer yang terhubung dengan password yang telah ditentukan dan diikuti dengan perintah-perintah. Data yang diterima adalah :

- Kode area : kosong atau 3 digit
- Prefix : 3 digit atau tidak diawali 0 atau 1
- Suffix : 4 digit
- Password : 6 digit alfanumerik
- Perintah : check, deposit, dll

Selanjutnya kondisi input digabungkan dengan masing-masing data elemen dapat ditentukan sebagai berikut:

Kode area : kondisi input, Boolean –kode area mungkin ada atau tidak kondisi input, range –nilai ditentukan antara 200 dan 999

Prefix : kondisi input range > 200 atau tidak diawali 0 atau 1

Suffix : kondisi input nilai 4 digit

Password : kondisi input boolean –password mungkin diperlukan atau tidak kondisi input nilai dengan 6 karakter string

Perintah : kondisi input set berisi perintah-perintah yang telah didefinisikan

- [Home](#)
- [Komputer](#)
- [Open Source](#)
- [Seo](#)
- [Kesehatan](#)
- [Akuntansi](#)

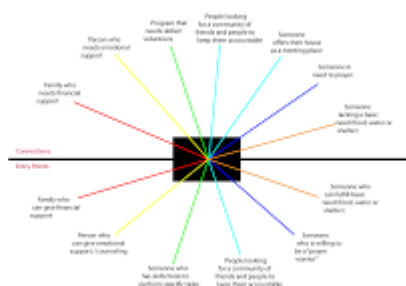
[Home](#) » [basis data](#) » [komputer](#) » Testing dan Implementasi "BLACKBOX TESTING"

Testing dan Implementasi "BLACKBOX TESTING"

Metode ujicoba blackbox memfokuskan pada keperluan fungsional dari software. Karna itu ujicoba blackbox memungkinkan pengembang software untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba blackbox bukan merupakan alternatif dari ujicoba whitebox, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode whitebox.

Ujicoba blackbox berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya :

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan performa
5. kesalahan inisialisasi dan terminasi



Tidak seperti metode whitebox yang dilaksanakan diawal proses, ujicoba blackbox diaplikasikan dibeberapa tahapan berikutnya. Karena ujicoba blackbox dengan sengaja mengabaikan struktur kontrol, sehingga perhatiannya difokuskan pada informasi domain. Ujicoba didesain untuk dapat menjawab pertanyaan-pertanyaan berikut :

1. Bagaimana validitas fungsionalnya diuji?
2. Jenis input seperti apa yang akan menghasilkan kasus uji yang baik ?
3. Apakah sistem secara khusus sensitif terhadap nilai input tertentu ?
4. Bagaimana batasan-batasan kelas data diisolasi?
5. Berapa rasio data dan jumlah data yang dapat ditoleransi oleh sistem?
6. Apa akibat yang akan timbul dari kombinasi spesifik data pada operasi sistem?

Dengan mengaplikasikan ujicoba blackbox, diharapkan dapat menghasilkan sekumpulan kasus uji yang memenuhi kriteria berikut :

- kasus uji yang berkurang, jika jumlahnya lebih dari 1, maka jumlah dari ujikasus tambahan harus didesain untuk mencapai ujicoba yang cukup beralasan

- Kasus uji yang memberitahukan sesuatu tentang keberadaan atau tidaknya suatu jenis kesalahan, daripada kesalahan yang terhubung hanya dengan suatu ujicoba yang spesifik
- Equivalence Partitioning Equivalence partitioning merupakan metode ujicoba blackbox yang membagi domain input dari program menjadi beberapa kelas data dari kasus ujicoba yang dihasilkan. Kasus uji penanganan single yang ideal menemukan sejumlah kesalahan (misalnya : kesalahan pemrosesan dari seluruh data karakter) yang merupakan syarat lain dari suatu kasus yang dieksekusi sebelum kesalahan umum diamati.

Equivalence partitioning berusaha untuk mendefinisikan kasus uji yang menemukan sejumlah jenis kesalahan, dan mengurangi jumlah kasus uji yang harus dibuat. Kasus uji yang didesain untuk Equivalence partitioning berdasarkan pada evaluasi dari ekuivalensi jenis/class untuk kondisi input. Class-class yang ekuivalen merepresentasikan sekumpulan keadaan valid dan invalid untuk kondisi input. Biasanya kondisi inputb dapat berupa spesifikasi nilai numerik, kisaran nilai, kumpulan nilai yang berhubungan atau kondisi boolean.

Ekuivalensi class dapat didefinisikan dengan panduan berikut :

1. Jika kondisi input menspesifikasikan kisaran/range, maka didefinisikan 1 yang valid dan 2 yang invalid untuk equivalence class
2. Jika kondisi input memerlukan nilai yang spesifik, maka didefinisikan 1 yang valid dan 2 yang invalid untuk equivalence class
3. Jika kondisi input menspesifikasikan anggota dari himpunan, maka didefinisikan 1 yang valid dan 1 yang invalid untuk equivalence class
4. Jika kondisi input adalah boolean, maka didefinisikan 1 yang valid dan 1 yang invalid untuk equivalence class

Misalkan, terdapat data terpelihara untuk sebuah aplikasi perbankan otomatis. User dapat mengaksesnya dari komputer pribadinya dengan menyediakan password 6 digit, dan mengikuti serangkaian perintah keyword yang mengakses berbagai fungsi perbankan. Software yang digunakan untuk aplikasi perbankan menerima data dalam bentuk :

- Area code – blank atau 3 digit nomor
- Prefix – 3 digit nomor yang tidak diawali oleh 0 atau 1
- Suffix – 4 digit nomor
- Password – 6 digit alphanumeric
- Commands – "check", "deposit", "bill pay", dsb

Kondisi input yang dihubungkan dengan setiap elemen data untuk aplikasi perbankan dapat dispesifikasikan sebagai :

1. Area code : kondisi input, Boolean – area code boleh ada maupun tidak Kondisi input, Range – nilai didefinisikan antara 200 dan 999, dengan beberapa pengecualian khusus (misal : tidak ada nilai > 905) dan syarat (misal : seluruh area code memiliki angka 0 atau 1 pada posisi digit ke-2)
2. Prefix : kondisi input, Range – nilai yang dispesifikasikan > 200
3. Suffix : kondisi input, Value – sepanjang 4 digit

4. Password : kondisi input, Boolean – Password boleh ada maupun tidak kondisi input, Value – 6 string karakter
5. Command : kondisi input, Set – mengandung perintah-perintah yang ada diatas

Aplikasikan panduan untuk derivasi dari class-class yang ekuivalen, kasus uji untuk setiap domain input data item dapat di bentuk dan dieksekusi. Kasus uji dipilih sehingga sejumlah atribut dari equivalence class dieksekusi sekali saja.

Boundary	Value	Analysis
-----------------	--------------	-----------------

Sejumlah besar kesalahan cenderung terjadi dalam batasan domain input dari pada nilai tengah. Untuk alasan ini boundary value analysis (BVA) dibuat sebagai teknik ujicoba. BVA mengarahkan pada pemilihan kasus uji yang melatih nilai-nilai batas. BVA merupakan desain teknik kasus uji yang melengkapi equivalence partitioning. Dari pada memfokuskan hanya pada kondisi input, BVA juga menghasilkan kasus uji dari domain output.

Panduan untuk BVA hampir sama pada beberapa bagian seperti yang disediakan untuk equivalence partitioning :

1. Jika kondisi input menspesifikasikan kisaran yang dibatasi oleh nilai a dan b, kasus uji harus dibuat dengan nilai a dan b, sedikit diatas dan sedikit dibawah a dan b
2. Jika kondisi input menspesifikasikan sejumlah nilai, kasus uji harus dibuat dengan melatih nilai maksimum dan minimum, juga nilai-nilai sedikit diatas dan sedikit dibawah nilai maksimum dan minimum tersebut.
3. Aplikasikan panduan 1 dan 2 untuk kondisi output. Sebagai contoh, asumsikan tabel temperatur VS tabel tekanan sebagai output dari program analisis engineering. Kasus uji harus didesain untuk membuat laporan output yang menghasilkan nilai maksimum(dan minimum) yang mungkin untuk tabel masukan
4. Jika struktur data program internal telah mendeskripsikan batasan (misal : array ditetapkan maks. 100), maka desain kasus uji yang akan melatih struktur data pada batasan tersebut.

Kebanyakan pengembang software secara intuitif melakukan BVA pada beberapa tingkatan. Dengan mengaplikasikan panduan diatas, ujicoba batasan akan lebih lengkap, selain itu memiliki kemungkinan pendeteksian kesalahan yang lebih tinggi.

[lanjut part II](#)

sumber : denissopyan2004.blogspot.com

share this article to: [Facebook](#) [Twitter](#) [Google+](#) [Linkedin](#) [Technorati](#) [Digg](#)

[Posting Lebih Baru](#)

[Posting Lama](#)

Read more: <http://www.bukucatatn.web.id/2013/11/testing-dan-implementasi-blackbox.html#ixzz3pqCvVpqj>

Testing dan Implementasi "BLACKBOX TESTING" part II

Cause-Effect

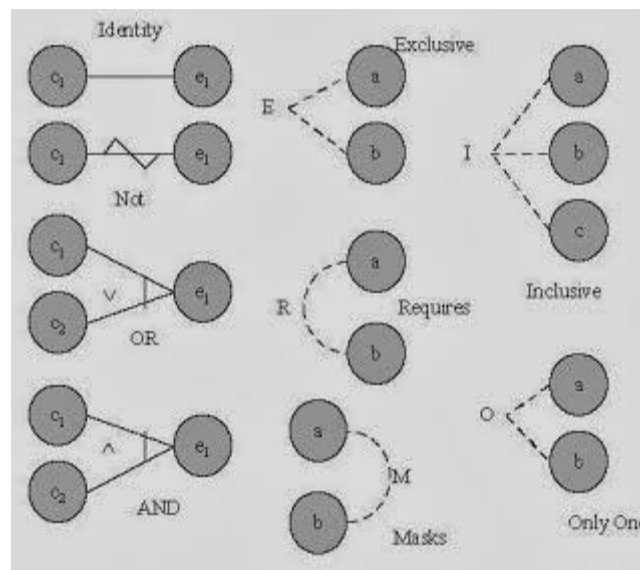
Graphing

Techniques

Cause-effect graphing merupakan desain teknik kasus ujicoba yang menyediakan representasi singkat mengenai kondisi logikal dan aksi yang berhubungan. Tekniknya mengikuti 4 tahapan berikut :

1. Causes (kondisi input), dan Effects (aksi) didaftarkan untuk modul dan identifier yang ditujukan untuk masing-masing
2. Cause-effect graph (seperti pada gambar dibawah) dibuat
3. Graph dikonversikan kedalam tabel keputusan
4. Aturan tabel keputusan dikonversikan kedalam kasus uji

Versi sederhana dari simbol graph cause-effect seperti dibawah ini. Terdapat hubungan causes c_i dengan effects e_i . Lainnya merupakan batasan relationship yang dapat diaplikasikan pada causes maupun effects.



gambar 1 : Cause-Effect Graphing

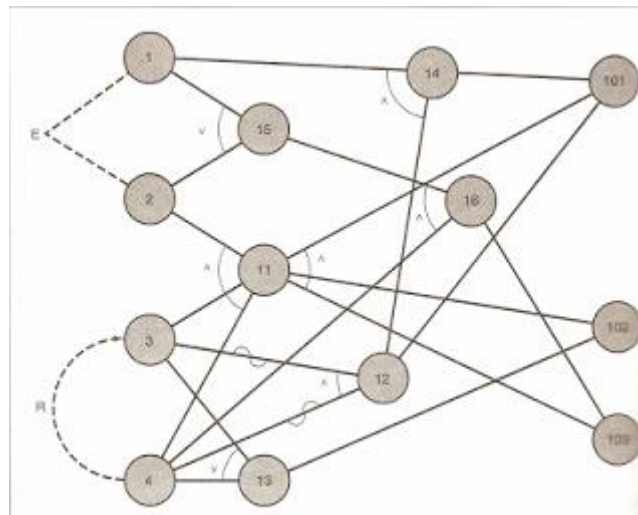
Untuk mengilustrasikan cause-effect graph perhatikan kasus tagihan berikut. Terdapat 4 kasus yang didefinisikan,

			yaitu		:
1	:		residential		indicator
2	:		commercial		indicator
3	:	peak	consumption	100	kWH
4	:	off-peak	consumption	100	kWH

Berdasarkan variasi kombinasi dari causes, maka effect berikut dapat terjadi :

101	:	Schedule	A	billing
102	:	Schedule	B	billing
103	:	Schedule	C	billing

Causes-effect graph untuk contoh diatas dapat dilihat pada gambar (2), causes 1, 2, 3, 4 direpresentasikan di sisi kiri graph dan effects 101, 102, 103 direpresentasikan disisi kanan. Secondary causes dibagian tengah (contoh : 11, 12, 13, 14, ...). Dari gambar (2) dapat dibentuk decision table seperti gambar (3).



gambar 2 : Cause-Effect Graph

		1	2	3	4	5	6	7	8	9	
Causes	1	1							0	1	
	2		1		1			1	1	0	
	3	0	1	0	1	0	1	1			
	4	0	1	0	1	1	0	1	1	1	
Intermediate causes	11		1		1			1			
	12	1		1							
	13					1	1				
	14	1									
	15								1	1	
	16								1	1	
Effects	101	1	1	1	0	0	0	0	0	0	
	102	0	0	0	1	1	1	0	0	0	
	103	0	0	0	0	0	0	1	1	1	

Gambar 3 : Decision table

Comparison

Testing

Dianjurkan bahwa versi independen suatu software untuk aplikasi yang amat kritis harus dibuat, walaupun nantinya hanya satu versi saja yang akan digunakan dalam sistem. Versi independen ini merupakan basis dari teknik black box testing yang disebut comparison testing atau back-to-back testing. Ketika multiple implementasi dari spesifikasi yang sama telah diproduksi, kasus uji didesain dengan menggunakan teknik black box yang lain (misalkan equivalence partitioning) disediakan sebagai input untuk setiap versi dari software. Jika setiap outputnya sama, diasumsikan implementasinya benar, jika tidak, setiap versi di periksa untuk menentukan jika kerusakan terdapat pada satu atau lebih versi yang akan bertanggung jawab atas perbedaan tersebut. Jika setiap spesifikasi dari seluruh versi telah dibuat dalam kesalahan, maka seluruh versi akan merefleksikan kesalahan. Sebagai tambahan, jika setiap versi independent memberikan hasil identik, tetapi salah, ujicoba hasil dan kondisi akan gagal untuk mendeteksi kesalahan.

Karakteristik khusus untuk sistem realtime memberikan tantangan tersendiri ketika ujicoba dilaksanakan. Ketergantungannya dengan waktu, sifat alami dari beberapa aplikasi yang tidak sinkron, menambah kesulitan baru dan potensial sebagai elemen untuk ujicoba dengan waktu beragam. Tidak hanya ujicoba whitebox maupun blackbox, tetapi juga ketepatan waktu pengiriman data dan pemrosesan paralel. Contohnya software realtime yang mengontrol mesin fotocopy, yang dapat menerima interupsi dari operator (berupa penekanan tombol 'RESET' atau 'DARKEN') dengan tanpa kesalahan ketika mesin sedang berjalan ('COPYING' state). Operator yang sama akan menginterupsi, ketika mesin nberada dalam posisi 'jamned'. Sebagai tambahan, keterkaitan antara software real time dengan perangkat keras pendukungnya juga dapat menyebabkan masalah dalam ujicoba. Ujicoba software harus mempertimbangkan kesalahan perangkat keras yang disebabkan karena pemrosesan software. Belum ada uji kasus yang komprehensif yang dikembangkan untuk sistem realtime, tetapi 4 langkah strategi berikut dapat dilaksanakan :

1. Task Testing, yaitu dengan mengujicobakan setiap task secara independen. Dalam hal ini metode whitebox dan blackbox testing dapat digunakan untuk menemukan kesalahan logika dan kesalahan fungsional, tetapi untuk kesalahan ketepatan waktu dan perilaku software (timing or behavioral errors), tidak dapat terdeteksi.
2. Behavioral Testing, dengan menggunakan model sistem dengan CASE tool, memungkinkan untuk mensimulasikan perilaku sistem realtime dan menentukannya sebagai konsekuensi dari peristiwa eksternal. Aktivitas analisis ini dapat dilaksanakan sebagai dasar untuk desain kasus uji yang diadakan ketika sebuah sistem realtime berhasil dibuat. Dengan menggunakan teknik yang sesuai (seperti equivalence partitioning), event dikategorikan (misalnya : interrupts, control signals, data), misalkan event pada sebuah mesin fotokopi dapat berupa interupsi dari user ('reset counter'), interupsi mekanikal ('paper jammed'), interupsi sistem ('toner low'), dan kesalahan bentuk ('overheated'). Setiap

kesalahan yang terjadi diuji secara individual, dan perilaku executable sistem diperiksa. Perilaku software diperiksa untuk menentukan apakah terdeteksi kesalahan perilaku sistem

3. Intertask Testing, ketika sebuah kesalahan dari individual task berhasil diisolasi, ujicoba berlanjut kepada kesalahan pada waktu yang terkait. Task yang tidak sinkron yang berkomunikasi dengan task lainnya diuji dengan beberapa data dan pemrosesan untuk menentukan apakah kesalahan antar task akan terjadi. Sebagai tambahan task yang berkomunikasi via antrian pesan atau penyimpanan data, diujikan untuk menemukan kesalahan ukuran penyimpanan
4. System Testing, software dan hardware telah disatukan dan diujikan dalam uji sistem sebagai satu kesatuan. Uji ini dilakukan untuk menemukan kesalahan pada software/hardware interface.

AUTOMATED	TESTING	TOOLS
Dikarenakan ujicoba software menghabiskan sekitar 40% dari total usaha yang diperlukan untuk sebuah proyek pengembangan software, maka tools tools yang dapat mengurangi waktu uji sangat dibutuhkan. Dengan melihat manfaat potensialnya, maka dibentuklah generasi pertama dari automated test tools. Miller mendeskripsikannya menjadi beberapa kategori, diantaranya :		

1. Static Analyzer, program sistem analisis ini mendukung untuk pembuktian dari pernyataan tanpa bukti statis, perintah-perintah yang lemah dalam struktur program dan format.
2. Code Auditors, sebuah filter dengan kegunaan khusus yang digunakan untuk memeriksa kualitas software untuk memastikan bahwa software tersebut telah memenuhi standards pengkodean minimum
3. Assertion Processors, sistem preprocessor/postprocessor ini digunakan untuk memberitahukan programmer dengan menyediakan klaim, yang disebut assertion, yaitu mengenai suatu perilaku program yang benar-benar ditemukan saat pelaksanaan program riil
4. Test File Generators, merupakan pembangun processor, dan dipenuhi dengan definisi awal nilai, file masukan yang serupa untuk program yang sedang diujikan.
5. Test Data Generators. merupakan sistem analisis otomatis yang membantu pengguna aplikasi dalam memilih data uji yang menyebabkan program berperilaku khusus
6. Test Verifiers, tool ini mengukur cakupan uji internal, terkadang berhubungan dengan uji struktur kontrol dari objek uji, dan melaporkan cakupan nilai untuk para ahli jaminan kualitas
7. Test Harnesses. Tools ini mendukung pemrosesan uji coba dengan (1) meng-instal program kandidat dalam lingkungan uji, (2) berikan input data, dan (3) simulasikan dengan menggunakan stubs perilaku dari modul-modul subordinat Output Comparators, tool ini membuatnya sangat memungkinkan untuk membantingkan sekumpulan output dari suatu program dengan sekumpulan output dari proses sebelumnya(yang telah diarsipkan) untuk menentukan perbedaan diantara keduanya

Dunn menambahkan tool otomatis diatas, diantaranya :

1. 1. Symbolic Execution System, tool ini menampilkan ujicoba program dengan menggunakan input aljabar, dari pada nilai data numerik,. Software yang diuji akan menguji satu class data uji dari pada satu kasus uji spesifik. Output yang dihasilkan dalam bentuk aljabar dan dapat dibandingkan dengan output yang diharapkan yang telah ditulis dalam bentuk aljabar
2. Environment Simulator, merupakan tool untuk sistem berbasis komputer khusus, yang mampu untuk

3. menguji model lingkungan eksternal dari suatu software realtme, dan mensimulasikan kondisi operasi sesungguhnya secara dinamis.
4. Data Flow Analyzer, tool ini melacak aliran data yang melewati sistem, dan berusaha untuk menemukan data reference yang belum didefinisikan , peng-indeks-an yang salah, dan kesalahan data terkait lainnya. [kembali ke part I](#)

Read more: http://www.bukucatatatan.web.id/2013/11/testing-dan-implementasi-blackbox_3.html#ixzz3pqDIO28B