1. TEKNIK PENGUJIAN PERANGKAT LUNAK

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain dan pengkodean. Pengujian menyajikan anomali yang menarik bagi perekayasa perangkat lunak. Pada proses perangkat lunak, perekayasa pertama-tama berusaha membangun perangkat lunak dari konsep abstrak ke implementasi yang dapat dilihat, baru dilakukan pengujian. Perekayasa menciptakan sederetan *test case* yang dimaksudkan untuk "membongkar" perangkat lunak yang sudah dibangun. Pada dasarnya, pengujian merupakan satu langkah dalam proses rekayasa perangkat lunak yang dapat dianggap (paling tidak secara psikologis) sebagai hal yang destruktif daripada konstruktif.

1.1. Sasaran-sasaran pengujian

Beberapa sasaran pengujian diantaranya :

1. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan
2. Test case yang baik adalah test case yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya
3. Pengujian yang sukses adalah pengujian yang mengungkap semua kesalahan yang belum pernah ditemukan sebelumnya.

Sasaran tersebut mengimplikasikan adanya perubahan titik pandang yang dramatis. Sasaran itu berlawanan dengan pandangan yang biasanya dipegang yang menyatakan bahwa pengujian yang berhasil adalah pengujian yang tidak ada kesalahan yang ditemukan. Sasarannya adalah mendesain pengujian yang secara sistematis mengungkap kelas kesalahan yang berbeda dan melakkukannya dengan jumlah waktu dan usaha minimum.

Bila pengujian dilakukan secara sukses (sesuai dengan sasaran tersebut), maka akan ditemukan kesalahan di dalam perangkat lunak. Sebagai keuntungan sekunder, pengujian menunjukkan bahwa fungsi perangkat lunak bekerja sesuai dengan spesifikasi dan bahwa persyaratan kinerja telah dipenuhi. Sebagai tambahan, data yang dikumpulkan pada saat pengujian dilakukan memberikan indikasi yang baik mengenai reliabilitas perangkat lunak dan beberapa menunjukkan kualitas perangkat lunak secara keseluruhan.

1.2. Prinsip Pengujian

Prinsip Pengujian meliputi :

- *Semua pengujian harus dapat ditelusuri sampai ke persyaratan pelanggan*. Sebagaimana telah kita ketahui, sasaran pengujian perangkat lunak adalah untuk mengungkapkan kesalahan. Hal ini memenuhi kriteria bahwa cacat yang paling fatal (dari titik pandang pelanggan) adalah cacat yang menyebabkan program gagal memenuhi persyaratannya.
- *Pengujian harus direncanakan lama sebelum pengujian itu mulai*. Perencanaan pengjuian dapat dimulai segera setelah model pernyaratan dilengkapi. Definisi detail mengeani test case dapat dimulai segera setelah model desain diteguhkan. Dengan demikian, semua pengujian dapat direncanakan dan dirancang sebelum semua kode dibangkitkan.
- *Prinsip Pareto berlaku untuk pengujian perangkat lunak.* Secara singkat prinsip Pareto mengimplikasikan bahwa 80 persen dari semua kesalahan yang ditemukan selama pengujian sepertinya akan dapat ditelusuri sampai 20 persen dari semua modul program. Masalahnya, bagaimana mengisolasi modul yang dicurigai dan mengujinya dengan teliti.
- *Pengujian harus mulai "dari yang kecil" dan berkembang ke pengujian "yang besar".* Pengujian pertama yang direncanakan dan dieksekusi biasanya berfokus pada modul program individual. Selagi pengujian berlangsung maju, pengujian mengubah fokus dalam

usaha menemukan kesalahan pada cluster modul yang terintegrasi dan akhirnya pada sistem secara keseluruhan.

- *Pengujian yang mendalam tidak mungkin.* Jumlah jalur permutasi untuk program yang berukuran menengah-pun sangat besar. Karean itulah tidak mungkin untuk mengeksekusi setiap kombinasi jalus skema pengujian. Tetapi dimungkinkan untuk secara tepat mencakup logika program dan memastikan bahwa semua kondisi dalam desai prosedural telah diuji.
- *Untuk menjadi paling efektif, pengujian harus dilakukan oleh pihak ketiga yang independent.* Yang dimaksud dengan kata "yang paling efektif" adalah pengujian yang memiliki probabilitas tertinggi di dalam menemukan kesalahan (sasaran utama pengujian). Karena perekayasa perangkat lunak yang membuat sistem tersebut bukanlah orang yang paling tepat untuk melakukan semua pengujian bagi perangkat lunak.

1.3. Testabilitas

Testabilitas perangkat lunak adalah seberapa mudah sebuah program komputer dapat diuji. Karena pengujian sangat sulit, perlu diketahui apa yang dapat dilakukan untuk membuatnya menjadi mudah. Kadang-kadang pemrogram beresedia melakukan hal-hal yang akan membantu proses pengujian dan checklist mengenai masalah-masalah desai yang mudah, fitur dan lain sebagainya yang berguna dalam bernegosiasi dengan mereka.

Checklist berikut memberikan serangkaian karakteristik yang membawa kepada perangkat lunak yang dapat diuji :

- Operabilitas. "Semakin baik dia bekerja, semakin efisien dia dapat diuji"
  - Sistem memiliki beberapa bug (bug menambah analisis dan biaya pelaporan ke proses pengujian).
  - Tidak ada bug yang memblok eksekusi pengujian
  - Produk berkembang di dalam tahapan fungsional (memungkinkan pengemabngan dan pengujian secara simultan)
- Observabilitas. "Apa yang Anda lihat adalah apa yang Anda uji"
  - Output yang berbeda dikeluarkan oleh masing-masing input
  - Tahap dan variabel sistem dapat dilihat atau diantrikan selama eksekusi.
  - Sistem dan variabel yang lalu dapat dilihat atau diantrikan (misal : log transaksi)
  - Semua faktor yang mempengaruhi output dapat dilihat
  - Kesalahan itnernal dideteksi secara otomatis melalui mekanisme selftesting
  - Kesalahan internal dilaporkan secara otomatis
  - Kode sumber dapat diakses
- Kontrolabilitas. "Semakin baik kita dapat mengontrol perangkat luank, semakin banyak pengujian yang dapat diotomatisasi dan dioptimalkan"
  - Semua output yang mungkin dapat dimnculkan melalui beberapa kombinasi input
  - Semua kode dapat dieksekusi melalui berbagai kombinasi input
  - Keadaan dan varibale perangkat lunak dan perangkat keras dapat dikontrol secara langsung oleh perekayasa pengujian
  - Format input dan output konsistem dan terstruktur
  - Pengujian dapat dispesifikasi, dioptimasi dan direproduksi dengan baik
- Dekomposabilitas. "Dengan mengontrol ruang lingkup pengujian, kita dapat dengan lebih cepat mengisolasi masalah dan melakuakn pengujian kembali secara lebih halus"
  - Sistem perangkat luank dibangun dari modul-modul independen
  - Modul-modul perangkat lunak dapat diuji secara independen
- Kesederhanaan. "Semakin sedikit yang diuji, semakin cepat kita dapat mengujinya"
  - Kesederhanaan fungsional (seperti, kumpulan fitur adalah kebutuhan minimum untuk memenuhi persyaratan).

    - Kesederhanaan struktural (seperti, arsitektur dimodularisasi untuk membatasi penyebaran kesalahan)
    - Kesederhanaan kode (seperti, standar pengkodean diadopsi demi kemduahan inspeksi dan pemeliharaan).
- Stabilitas. “Semakin sedikti perubahan, semakin sedikit ganggunan dalam pengujian”
    - Pengujian ke perangkat lunak tidak sering
    - Perubahan ke perangkat lunak terkontrol
    - Perubahan ke perangkat lunak memvalidasi pengujian yang sudah ada
    - Kegagalan perangkat lunak dapat diperbaiki dengan baik
- Kemampuan untuk dapat dipahami. “Semakin banyak informasi yang kita miliki, semakin halus pengujian yang akan dilakukan”
    - Desain dipahami dengan baik
    - Ketergantungan di antara komponen internal, eksternal dan yang dipakai bersama, dipahami dengan baik.
    - Perubahan ke desai dikomunikasikan
    - Dokumentasi teknik dapat diakses dengan cepat
    - Dokumentasi teknis diorganisasikan dengan baik
    - Dokumentasi teknis spesifik dan detail
    - Dokumentasi teknis akurat

Berikut adalah atribut-atribut pengujian yang “baik” :

1. *Pengujian yang baik memiliki probabilitas yang tinggi untuk menemukan kesalahan.* Untuk mencapai hal ini, penguji harus memahami perangkat lunak dan berusaha mengembangkan gambaran mental mengenai bagaimana perangkat lunak dapat gagal. Idealnya kelas-kelas kegagalan itu diselidiki. Sebagai contoh, kelas kegagalan potensial pada GUI adalah kegagalan untuk mengenali posisi mouse yang sesuai. Serangkaian pengujian akan dirancang untuk menguji mouse untuk memperlihatkan kesalahan di dalam pengenalan posisi mouse.
2. *Pengujian yang baik tidak redudan.* Waktu pengujian dan sumber daya terbatas. Tidak ada manfaatnya melakukan pengujian dengan tujuan yang sama dengan pengujian lainnya. Setiap pengujian arus memiliki tujuan yang berbeda.
3. *Pengujian yang baik seharusnya “jenis terbaik”.* Dalam suatu kelompok pengujian yang memiliki tujuan yang serupa, batasan waktu dan sumber daya dapat menghalangi eksekusi hanya kelompok kecil dari pengujian tersebut. Pada kasus semacam ini maka pengujian yang memiliki kemungkinan paling besar untuk mengungkap seluruh kelas kesalahan yang tinggi harus digunakan.
4. *Pengujian yang baik tidak boleh terlalu sederhana atau terlalu kompleks.* Secara umum masing-masing test case harus dieksekusi secara terpisah.

1.4. Desain Test Case

Lebih dari dua dekade terakhir telah berkembang berbagai metode desai test case untuk perangkat lunak. Metode-metode tersebut memebrikan kepada pengembang sebuah pendekatan yang sistematik ke pengujian. Yang lebih penting, metode itu memberikan mekanisme yang dapat membantu memastikan kelengkapan pengujian dan memberikan kemungkinan tertinggi untuk mengungkap kesalahan pada perangkat lunak.

Semua produk yang direkayasa dapat diuji dengan satu atau dua cara : (1) dengan mengetahui fungsi yang ditentukan dimana produk dirancang untuk melakukannya, pengujian dapat dilakukan untuk memperlihatkan bahwa masing-masing fungsi beroperasi sepenuhnya, pada waktu yang sama mencari kesalahan pada setiap fungsi; (2) mengetahui kerja internal suatu produk, maka pengujian dapat dilakukan untuk memastikan bahwa “semua roda gigi berhubungan”, yaitu operasi

internal bekerja sesuai dnegan spesifikasi dan semua komponen ineternal telah diamati dengan baik. Pendekatan pengujian pertama disebut pengujian *black box* dan yang kedua disebut *white box.*

Jika perangkat lunak komputer dipertimbangkan, maka pengujian black box berkaitan dengan pengujian yang dilakukan pada interface perangkat lunak. Meskipun didesain untuk mengungkap kesalahan, pengujian black box digunakan untuk memperlihatkan bahwa fungsi-fungsi perangkat lunak adalah operasional, bahwa input diterima dengan baik dan output dihasilkan dengan tepat dan integrasi informasi eksternal (seperti file data) dipelihara. Pengujian balck box menguji beberapa aspek dasar suatu sistem dengan sedikit memperhatikan struktur logika internal perangkat lunak tersebut.

Pengujian white box perangkat lunak didasarkan pada pengamatan yang teliti terhadap detail prosedural. Jalur-jalur logika yang melewati perangkat lunak diuji dengan memberikan test case yang menguji serangkaian kondisi dan atau loop tertentu. "Status program tersebut"dapat diuji pada pada berbgai titik untuk menentukan apakah status yang diharapkan atau dituntut sesuai dengan status aktual.

1.5. Pengujian White-Box

Pengujian white box, kadang-kadang disebut pengujian *glass box*, adalah metode desaintest case yang menggunakan struktur kontrol desain prosedural untuk memperoleh test case. Dengan menggunakan metode pengujian white box, perekayasa sistem dapat melakukan test case yang :

1. memberikan jaminan bahwa semua jalur independen pada suatu modal telah digunakan, paling tidak satu kali.
2. Menggunakan semua keputusan logis pada sisi *true* dan *false*
3. Mengeksekusi semua loop pada batasan mereka dan baas operasional mereka
4. Menggunakan struktur data internal untuk menjamin validitasnya

Pada titik ini, dapat diajukan pertanyaan yang beralasan, yaitu "Mengapa menghabiskan waktu dan energi untuk menguji logika jika kita dapat dengan lebih baik memperluas kerja yang dapat memastikan bahwa persyaratan program telah dipenuhi ?" Bila dinyatakan dengan cara lain, mengapa kita tidak menggunakan semua energi kita untuk melakukan pengujian black box ? Jawabannya ada pada sifat cacat perangkat lunak :

- *Kesalahan logis dan asumsi yang tidak benar berbanding terbalik dengan probabilitas jalur program yang akan dieksekusi.* Kesalahan cenderung muncul dalam kerja kita pada saat kita mendesain dan mengimplementasi fungsi, kondisi atau kontrol yang berada di luar mainstraim. Pemrosesan setiap hari cenderung dipahami dengan baik sementara pemrosesan "kasus khusus" cenderung berantakan
- *Kita sering percaya bahwa jalur logis mungkin tidak akan diseksekusi bila pada kenyataannya akan diseksekusi pada basis reguler*. Aliran logika dari suatu program kadang-kadang bersifat konterintuitif yang berarti asumsi kita yang tidak kita sasari mengenai aliran dan data kontrol dapat menyebabkan kita membuat kesalahan desai yang akan terungkap hanya setelah pengujian jalur mulai.
- *Kesalahan tipografis adalah random.* Bila sebuah program diterjemahkan ke dalam kode sumber bahasa pemrograman maka dimungkinkan akan terjadi banyak kesalahan pengetikan. Beberapa akan ditemukan dengan mekanisme pengecekan sintaks, tetapi yang lainnya akan tetap tidak terdeteksi sampai pengujian mulai.

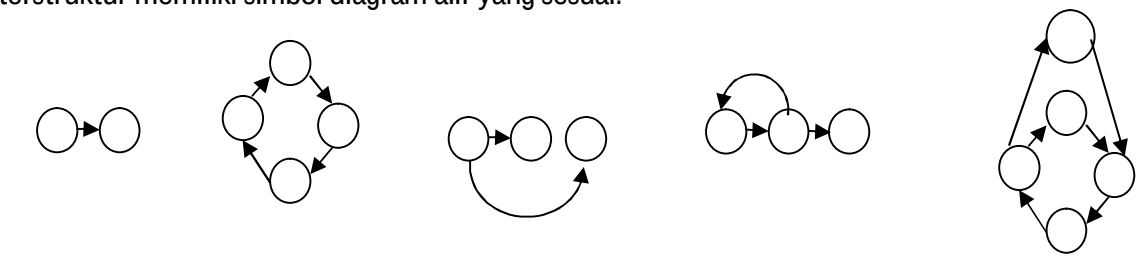
Masing-masing alasan tersebut memberikan suatu argurmen untuk melakukan pengujian white box. Pengujian black box, tidak peduli seberapa cermat dilakukan, dapat menangkap bentuk kesalahan tersebut.

1.6. Pengujian Basis-Path

Pengujian basis path adalah teknik pengujian white box yang diusulkan pertama kali oleh Tom McCabe. Metode basis ini memungkinkan desainer test case mengukur kompleksitas logis dari desai prosedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi. Test case yang dilakukan untuk menggunakan basis set tersebut dijamin menggunakan setiap statment di dalam program paling tidak sekali selama pengujian.

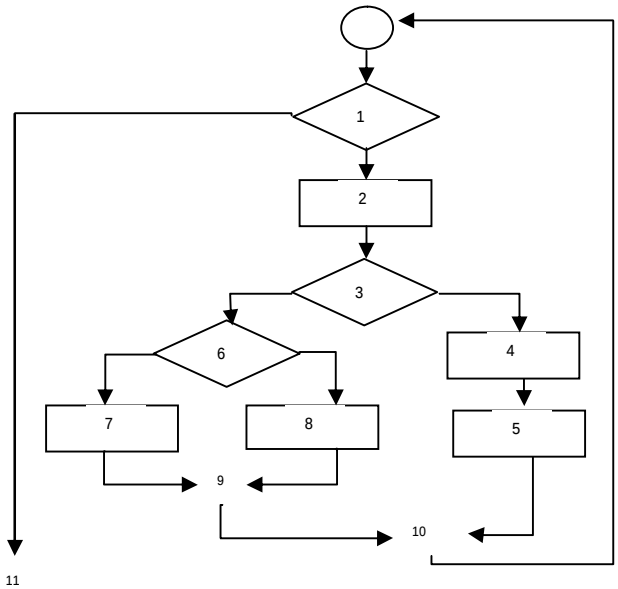
1.6.1. Notasi Diagram Alir

Sebelum metode babsis path diperkenalkan, terlebih dahulu akan dijelaskanmengenasi notasi sederhana dalam bentuk diagram alir (grafik alir). Diagral alir menggambarkan aliran kontrol logika yang menggunakan notasi seperti ditunjukkan pada gambar 1. Masing-masing gagasan terstruktur memiliki simbol diagram alir yang sesuai.

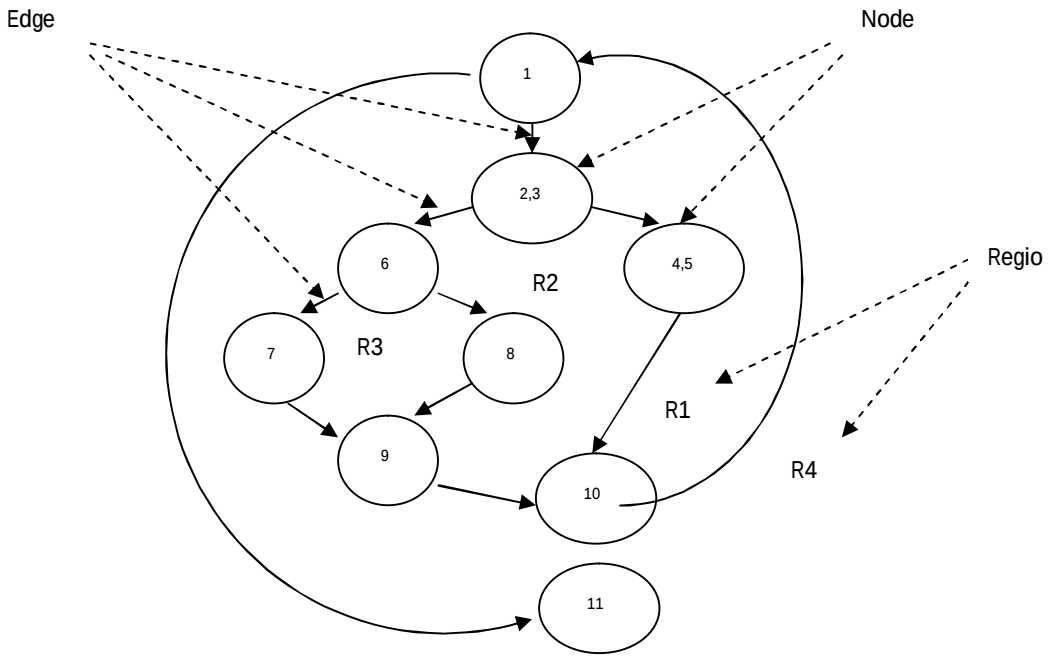


Gambar 1. Notasi diagram aliran

Gambar 2 merepresentasikan desain prsedural grafik alir. Paga gambar tersebut tampak struktrul kontrol progrm. Gambar 3 memetakan grafik alir tersebut ke dalam grafik alir yang sesuai. Pada gambar tersebut masing-masing lingkaran disebut simpulu grafik alir, merepresentasikan satu atau lebih statemen prosedural. Urutan kotak proses dan belahketupat keputusan dapat memetakan simpul tunggal. Anak panah pada grafik alir tersebut disebut edges atau links, merepresentasikan aliran kontrol dan analog dengan anak panah bagan alir. Edge harus berhenti pada suatu simpul, mesikipun nila simpul tersebut tidak merepresentasikan statemen prosedural (misal simpul untuk bangun if-then-else). Area yang dibatasi oleh edge dan simpul disebtu region. Pada saat menghitung region kita perlu memasukkan area di luar grafik dan menghitungnya sebgai sebuah region.

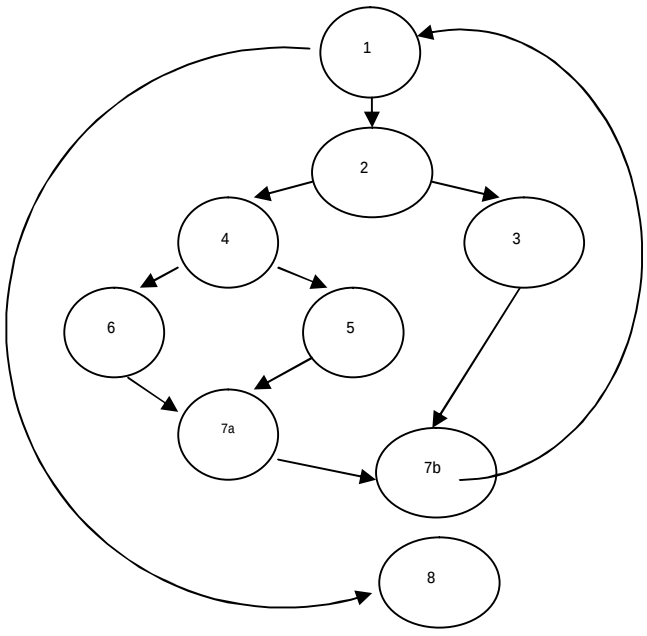


Gambar 2. Bagan Alir



Gambar 3. Grafik Alir

Setiap representasi desain prosedural dapat diterjemahkan ke dalam suatu grafik alir. Gambar 4 memperlihatkan segmen PDL dan grafik airnya yang sesuai. Tampak bahawa statemen PDL telah diberi nomor dan penomoran yang sesuai digunakan untuk grafik alir tersebut.

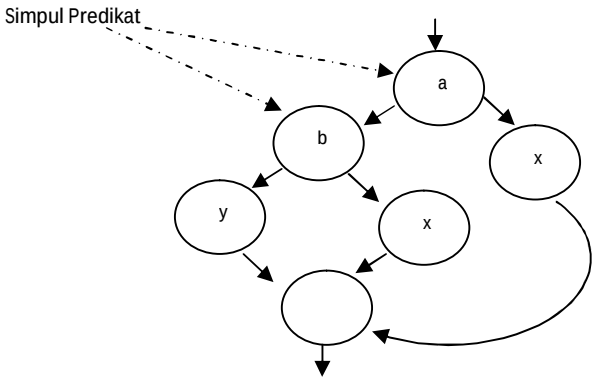


PDL
Procedure : sort
1 : do while records remain
Read record;
2: if record field 1 = 0
3 : then process record;
Store in buffer;
Increment counter;
4 : elseif record field 2 = 0
5 : then reset counter;
6: else process record;
Store in file;
7a: endif
Endif
7b: enddo
8 : end

Gambar 4. Menterjemahkan PDL ke grafik alir

Bila ada kondisi gabungan dalam desain prosedural, maka pembuatan grafik alir menjadi sangat rumit. Kondisi gabungan terjadi bila satu atau lebih operator Boolean (logika OR, AND, NAND, NOR) ada pada statemen kondisional. Gambar 5 menunjukkan sebauh saegmen PDL menterjemahkan ke dalam grafik alir. Tampak bahwa simpul (node) yang terpisah diciptakan untuk

masing-masing kondisi a dan b pada statemen IF a OR b. Masing-masing simpul yang berisi sebuah kondisi disebut simpul predikat dan ditandai dengan dua atau lebih edge yang berasal darinya.



.
.
IF a OR b
Then procedure x
Else procedure y
ENDIF

Gambar 5. Logika gabungan

1.6.2. Kompleksitas Siklomatis

Kompleksitas siklomatis adalah metriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila metriks ini digunakan dalam konteks metode pengujian basis path, maka nilai yang terhitung untuk kompleksitas siklomatis menentukan jumlah jalur independen dalam basis set suatu program an memberi batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua statemen telah dieksekusi sedikitnya satu kali.

Jalur independen adalah jalur yangmemlalui progarm yang mengintroduksi sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Bila dinyatakan dengan terminologi grafik alir, jalur independen harus bergerak sepanjang paling tidak satu edge yang tidak dilewatkan sebelum jalur tersebut ditentukan. Contoh, serangkaian jalur independen untuk grafik alir yang ditunjukkan pada gambar 3 adalah :

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

Tampak bahwa masing-masing jalur baru memeperkenalkan sebuah edge baru.

Jalur 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 tidak dianggap jalur independen karena merupakan gabungan dari jalur-jalur yang sudah ditentukan dan tidak melewati beberapa edge baru.

Jalur 1,2,3 dan 4 yang ditentukan di atas terdiri dari sebuah basis set untuk grafik alir pada gambar 3. Bila pengujian dapat dilakukan untuk memaksa adanya eksekusi dari jalur-jalur tersebut, maka setiap statemen pada program tersebut akan diseksekusi paling tidak satu kali dan setiap kondisi sudah akan dieksekusi pada sisi true dan fale-nya. Perlu dicatat bahwa basis set tidaklah unik. Pada dasarnya semua jumlah basis set yang berbeda dapat diperoleh untuk suatu desain prosedural yang diberikan.

Bagaimana kita tahu banyaknya jalur yang dicari ? Komputasi kompleksitas siklomatis memebrikan jawaban. Fondasi kompleksitas siklomatis adalah teori grafik dan memberi kita metriks perangkat lunak yang sangat berguna. Kompleksitas dihitunga dalam satu adari tiga cara berikut :

1. Jumlah region grafik alir sesuai dengan kompleksitas siklomatis

2. Kompleksitas siklomatis *V(G)* untuk grafik alir G ditentukan sebagai *V(G)=E-N+2* dimana E adalah jumlah edge grafik alir dan N adalah jumlah simpul grafik alir
3. Kompleksitas siklomatis *V(G)* untuk grafik alir G ditentukan sebagai *V(G)=P+1,* dimana P adalah jumlah simpul predikat yang diisikan dalam grafik alir G.

Pada gambar 3, kompleksitas siklomatis dapat dihitung dengan menggunakan masing-masing algoritma di atas :

1. Grafik alir mempunyai 4 region
2. *V(G)* = 11 edge – 9 simpul + 2 = 4
3. *V(G)* = 3 simpul yang diperkirakan + 1 = 4

Dengan demikian, kompleksitas siklomatis dari grafik alir pada gambar 3 adalah 4.

Yang lebih penting, nilai untuk *V(G)* memeberi kita batas atas untuk jumlah jalur independen yang membentuk basis set, dan implikasinya, batas atas jumlah pengujian yang harus didesain dan dieksekusi untuk menjamin semua statemen program.

1.6.3. Melakukan Test Case

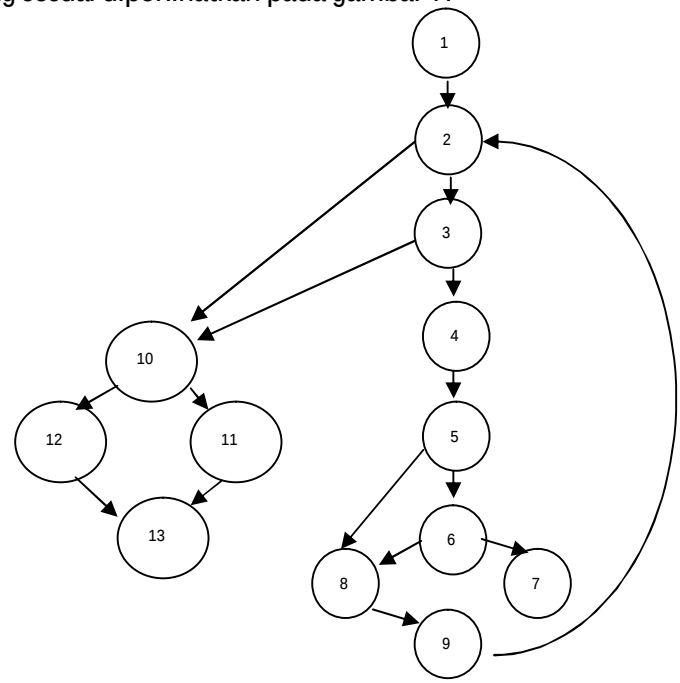
Metode pengujian basis path dapat diaplikasikan pada desain prosedural atau kode sumber. Pengujian basis path memiliki sederetan langkah. Average prosedur, yang digambarkan dalam PDL pada gambar 6 menggambarkan masing-masing langkah pada metode desain test case. Tampak bahwa average, meskipun merupakan suatu algoritma yang sederhana berisi kondisi gabungan dan loop.

```
PROCEDURE average
  *This procedure compute the average of 100 or fewer numbers that lie between bounding values; it also
   compute the sum and teh total number valid

INTERFACE RETURNS average, total.input, total valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:11] IS SCALAR ARRAY;
TYPE average, total.input, total valid;
     minimum, maximum, sum IS SCALAR
TYPE i IS INTEGER

i=1;                                                    1
total.input = total.valid = 0;                          2
sum = 0;
DO WHILE value[i] < > -999 and total.input < 100        3
   Increment total.imput by 1                           4
   IF value[i] >= minimum AND value[i] <= maximum       5, 6
      THEN increment total.valid by 1;                  7
           Sum = sum + value[i]
      ELSE skip
   ENDIF                                                8
   increment i by 1;                                    8
ENDDO                                                   9
   IF total.valid > 0                                   10
      THEN average = sum /total.valid;                  11
      ELSE average = -999;                              12
   ENDIF
END average                                             13
```

Gambar 6. PDL untuk desain test case dengan simpul-simpul yang diidentifikasikan

1. Dengan menggunakan desain atau kode sebagai dasar, gambarkan sebuah grafik alir yang sesuai. Grafik alir diciptakan dengan menggunakan simbol dan aturan konstruksi yang

disajikan pada subbab 1.6. PDL untuk average pada gambar 6 diciptakan dengan menomori statemen-statemen PDL yang akan dipetakan ke dalam simpul grafik alir yang sesuai. Grafik alir yang sesuai diperlihatkan pada gambar 7.



Gambar 7. Grafik aliran dari average prosedur

2. Tentukan kompleksitas siklomatis dari grafik alir resultan. Kompleksitas siklomatis , *V(G)* ditentukan dengan mengaplikasikan satu dari algoritma-algoritma yang telah digambarkan di atas. Perlu dicatat bahwa V9G) dapat ditentukan tantpa mengembangkan grafik alir dengan menghitung semua statemen kondisional dalam PDL (untuk average prosedur, kondisi gabungan menghitung 2) dan penambahan 1. Pada gambar 7 :
   V(G) = 6 region
   V(G) = 18 edge – 14 simpul + 2 = 6
   V(G) = 5 simpul prekdikat + 1 = 6
3. Tentukan sebuah basis set dari jalur independen secara linier. Harga V(G) memeberikan jumlah jalur independen secara linier melalui struktur kontrol program . Dalam kasus average prosedur, kita dapat menentukan enam jalur :
   Jalur 1 : 1-2-10-11-13
   Jalur 2 : 1-2-10-12-13
   Jalur 3 : 1-2-3-10-11-13
   Jalur 4 : 1-2-3-4-5-8-9-2- ....
   Jalur 5 : 1-2-3-4-5-6-8-9-2-....
   Jalur 6 : 1-2-3-4-5-6-7-8-9-2-.....
4. Siapkan test case yang akan memaksa adanya eksekusi setiap basis set. Data harus dipilih sehingga kondisi pada simpul-simpul predikat terpasang secara tepat pada saat masing-masing juluar diuji. Test case yang memenuhi basis set yang digambarkan di atas adalah :
   - Test case jalur 1 :
     Harga (k)= input valid, dimana k < i yang dteteapkan di bawah
     Harga (i) = -999 dimana $2 \leq 2\ i \leq 100$
     Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai k dan total yang tepat.
     Catatan : jalur 1 tidak dapat diuji sendirian karena harus diuji sebagai bagian dari pengujian jalur 4,5 dan 6

   - Test case jalur 2 :
     Harga (i) =-999
     Hasil yang diharapakan : rata-rata -999, total yang lain pada nilai awal

   - Test case jalur 3 :
     Usahakan untuk memproses 101 nilai atau lebih
     100 nilai pertama harus valid

Hasil yang diharapkan : sama seperti test case 1

- Test case jalur 4 :
  Nilai (i) = input valid dimana i < 100
  Nilai (k) < minimum, dimana k < i
  Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai-nilai n dan total yang tepat

- Test case jalur 5 :
  Nilai (i) = input valid dimana i< 100
  Nilai (k) > maksimum, dimana k ≤ i
  Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai-nilai n dan total yang tepat

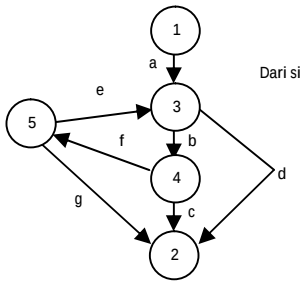
- Test case jalur 6 :
  Nilai (i) = input valid dimana i < 100
  Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai-nilai ni dan total yang tepat.

Masing-masing test case dieksekusi dan dibandingkan untuk mendapat hasil yang diharapkan. Sekali semua test telah dilengkapi maka penguji dapat yakin bahwa semua statemen pada program telah dieksekusi paling tidak satu kali. Penting untuk dicatat bahaw beberapa jalur independ (misal jalur 1 pada contoh di atas) tidak dapat diuji secara terpisah (sendiri) karena kombinasi data yang diperlukan untuk melintasi jalur tersebut tidak dapat dicapai di dalam aliran normal dari program. Dalam kasus semacam ini, jalur-jalur diuji sebgai bagian dari pengujian jalur yang lain.

1.6.4. Matriks Grafik

Prosedur untuk mendaptakan grafik alir dan menentukan serangkaian basis path , cocok dengan mekanisasi. Untuk mengembangkan peranti perangkat lunak yangmembantu pengujian basis patha, struktur data yang disebut matriks grafis dapat sangat berguna.

Matriks garfis adalah matriks bujur sangkar yang ukurannya sama dengan jumlah simpul pada grafik alir. Masing-masing baris dan kolom sesuai dengan simpul yang diidentifikasikan dan entri matriks sesuai dengan edge di atantara simpul. Contoh sederhana grafik air dan matriks grafisnya yang sesuai diperlihatkan pada gambar 8.



Grafik alir

Dari simpul \ ke simpul	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

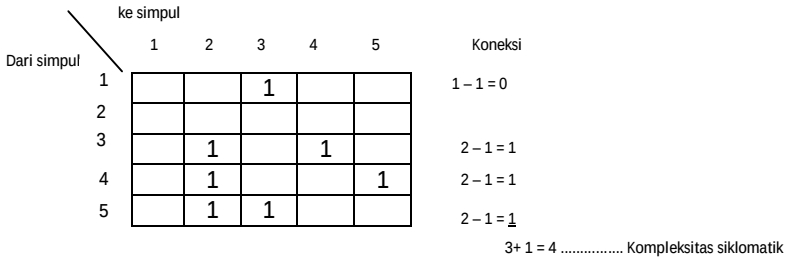
Matriks grafik

Gambar 8. Matriks koneksi

Pada gambar tersebut, masing-masing simpul pada grafik alir diidentifikasikan oleh bilangan, sementara masing-masing edge diidentifikas dengan huruf. Entri huruf dibuat di dalam atriks tersebut untuk pencocokan dengan koneksi di antara dua simpul. Sebgai contoh, simpul 3 disambungkan dengan simpul 4 oleh edge b. Untuk titik ini, matriks grafis tidak lebih dari sekedar representasi tabuler dari grafik alir. Tetapi denganmenambahkan sebuah link weighy pada masing-masing entri matriks, maka matriks grafis dapat menjadi alat yang sangat kuat untuk mengevaluasi struktur kontrol program selama pengujian. Link weight memberikan informasi tambahan mengenai aliran kontrol. Dalam bentuknya yang paling sederhana, link weight adalah 1 (ada hubungan) atau 0 (tidak ada hubungan). Tetapi link weight dapta ditetapkan lain, yaitu properti yang lebih menarik :

- Probabilitas di mana sebuah link (edge) akan dieksekusi
- Waktu perosesan yang digunakan slama pelewatan sebuah link
- Memori yang diperlukan selama peleawatan link
- Sumber daya yang diperlukan selama pelewatan link

Untuk menggambarkanya, kita gunakan pembebanan sederhana untuk menunjukkan hubungan (0 atau 1). Matriks grafis pada gambar 8 digambar lagi seperti ditunjukkan pada gambar 9. Masing-masng hruuf telah diganti dengan angka 1 yang menunjukkan bahwa ada hubungan (nol telah dihilangkan supaya jelas). Dengan bentuk seperti itu, matrriks grafis disebut matriks koneksi. Pada gambar 9 masing-masing baris dengan dua entri atau lebih merepresentasikan sebauah simpul predikat. Dengan demikian dengan mengejrjakan aritmatika yang diperlihatkan di sebelah kanan sambungan matriks akan memberi kita metode lain untuk menentukan kompleksitas siklomatis (subbab 1.6.2).



Gambar 9. Matriks Koneksi

Beizer memberikan perlakukan yang teliti dari algoritma matematika tambahan yang dapat diaplikasikan pada matriks-matriks grafis. Dengan menggunakan teknik tersebut, maka analisis yang dipergunakan untuk mendesain test case dapat diotomasi sebagian atau sepenuhnya.

1.7. Pengujian Struktur Kontrol

Teknik pengujian basis path yang digambarkan pada subbab di atas adalah salah satu dari sejumlah teknik untuk pengujian struktur kontrol. Meskipun pengujian basis path sederhana dan efektif, tetapi pengujian itu tidak memadai. Dalam bagian ini akan dibahas variasi lain pada pengujian struktur kontrol. Hal ini memperluas kupasan pengujian dan meningkatkan kualitas pengujian white-box.

1.7.1. Pengujian Kondisi

Pengujian kondisi adalah sebuah metode desain test case yang menggunakan kondisi logis yang ada pada suatu program. Kondisi sederhana adalah variabel Boolean atau suatu persamaan hubungan, dapat didahului dengan satu operator NOT (“¬”). Persamaan relasional mengambil bentuk :

$$E_1 \text{ (operator-relasional) } E_2$$

Dimana E_1 *dan* E_2 merupakan persamaan aritmatika dan (operator relasional) adalah salah satu dari operator berikut : “<”,”≤”,”=”,”≠”,””¬=”(pertidaksamaan),”>” atau “≥”. Kondisi gabungan terdiri dari dua atau lebih kondisi sederhana, oprator Boolean dan tanda kurung. Diasumsikan bahwa operator Boolean yang diijinkan dalam suatu kondisi gabungan meliputi OR (“|”), AND(“&”) dan NOT (“¬”). Kondisi tanpa persamaan realsional disebut persamaan Boolean.

Dengan demikian, tipe-tipe komponen yang mungkin di dalam suatu kondisi meliputi operator Boolean, sebuah variabel Boolean, sepasang tanda kurung Boolean (yang mengelilingi) suatu kondisi gabungan atau sederhana), sebuah operator relasional atau sebuah persamaan aritmatika. Bila suatu kondisi tidak benar, maka palling tidak satu komponen dari kondisi itu salah. Dengan demikian, tipe kesaahan pada suatu kondisi meliputi berikut ini :

- Kesalahan operator Boolean (adanya operator Boolenan yang salah/hilang/ekstra)
- Kesalahan variabel Boolean
- Kesalahan tanda kurung Boolean
- Kesalahan operator relasional
- Kesalahan persamaan aritmatika

Metode pengujian kondisi berfokus pada pengujian masing-masing kondisi di dalam program. Strategi pengujian kondisi biasanya memiliki dua keuntungan. Pertama, pengukuran kupasan pengujian dari suatu kondisi adalah sederhana. Kedua, cakupan pengujian terhadap kondisi di dalam suatu program memberikan pedoman untuk melakukan pengujian tambahan untuk program tersebut.

Tujuan pengujian kondisi adalah mendeteksi tidak hanya kesalahan di dalam kondisi program, tetapi juga kesalahan lain pada program. Jika pengujian ditentukan untuk program P efektif untuk mendeteksi kesalahan pada kondisi yang ada pada P, maka kemungkinan besar pengujian juga efektif untuk mendeteksi kesalahan lain pada P. Sebgai tambahan, jika strategi pengujian efektif untuk mendeteksi kesalahan-kesalahan pada suatu kondisi, maka kemungkinan bear strategi tersebut juga efektif untuk mendeteksi kesalahan pada suatu program.

Sejumlah strategi pengujian kondisi telah diusulkan. Pengujian cabang mungkin merupakan strategi pengujian kondisi yang paling sederhana. Untuk sudatu kondisi gabunga C, cabang-cabang true dan false dari C dan setiap kondisi sederhana pada C perlu dieksekusi paling tidak satu kali.

Pengujian domain membutuhkan tiga atau empat pengujian untuk dilaukan pada sebuah persamaan realsional. Karena perasamaan relasional mengambil bentuk :

$$E_1 \text{ (operator-relasional) } E_2$$

Maka diperlukan tiga pengujian untuk membuat nilai E_1 lebih tinggi, sama dengan atau kurang dari nilai E_2 secara berurutan. Bila operator relasional salah dan E_1 dan E_2 benar, maka ketiga pengujian itu menjamin pendeteksian kesalahan operator relasional. Untuk mendeteksi kesalahan pada E_1 dan E_2 maka tes yang membuat harga E_1 lebih besar atau kurang dari harga E_2 harus membuat perbedaan antara dua harga itu menjadi sekecil mungkin.

Untuk persamaan Booleand dengan n variabel, semua 2^n pengujian yang mungkin ($n > 0$) perlu dilakukan. Strategi itu dapat mendeteksi operator, variabel dan kesalahan tanda kurung Boolean, tatapi hanya praktis jika n kecil.

Pengujian error sensitive untuk persamaan Boolean dapat juga dilakukan. Untuk persamaan Boolean tunggal (persamaan Boolean dimana masing-masing variabel Boolean terjadi hanya sekali) dengan n variable Boolean ($n > 0$), kita dapat dengan mudah memunculkan serangkaian pengujian dengan kurang dari 2^n pengujian sehingga rangkaian pengujian itu menjamin pendeteksian kesalahan opertaor Boolean bertingkat serta efektif untuk mendeteksi kesalahan-kesalahan yang lain.

Tai mengusulkan strategi pengujiankondisi yang didasarkan atas teknik yang sudah diuraikan. Disebut pengujian BRO (branch and relational operator), tekni tersebut menjamin pendeteksian kesalahan cabgan dan operator relasional, dengan kondisi bahwa semua variabel Boolean dan opertaor relasional pada kondisi itu terjadi hanya sekali dan tidak memiliki variabel umum.

Strategi BRO menggunakan batasan kondisi bagi suatu kondisi C. Batasan kondisi untuk C dengan n kondisi sederhana ditentukan sebagai (D_1, D_2, ...D_n) dimana D_i ($0 < i \leq n$) merupakan simbol yangmenentukan batasan pada hasil akhir dari kondisi sederhana ke-i dalam kondisi C. Batasan kondisi D untuk kondisi C dikatakan dipenuhi oleh eksekusi dari C bila selama eksekusi dari C, hasil akhir dari masing-masing kondisi sederhana di dalam C memeneuhi batasan yang bersesuaian di dalam D.

Untuk variable Boolean B, kita menentukan baasan pada hasil akhir B yang menyatakan bahwa B harus true (t) atau false (f). Dengan carayang sama, untuk persamaan realsional, simbol-simbol >, = dan < digunakan untuk menentukan batasan pada hasil akhir persamaan.

Contoh-1, diketahui kondisi berikut :

$$C_1 : B_1 \,\&\, B_2$$

Dimana B_1 dan B_2 adalah variabel Boolean. Batasan kondisi untuk C_i adalah bentuk (D_1,D_2) dimana masing-masing dari D_1 dan D_2 adalah “t” atau “f”. Nilai t, f adalah batasan kondisi untuk C1 dan dicakup oleh pengujian yang membuat harga B_1 menjadi true (t) dan harga B_2 menjadi false (f). Strategi pengujian BRO mengharuskan himpunan batasan [(t,t),(f,t),(t,f)] dicakup oleh eksekusi C_1. Bila C_1 tidak benar karena satu atau lebih kesalahan operator Boolean, maka sedikitinya satu anggota dari himpunan batasan akan memaksa C_1 untuk gagal.

Contoh-2 diketahui kondisi berikut :

$$C_2 : B_1 \& (E_3 = E_4)$$

Dimana B_1 adalah persamaan Boolean dan E_3 dan E_4 adalah persamaan aritmatika. Batasan kondisi untuk C_2 adalah bentuk (D_1,D_2) dimana D_1 adalah "t" atau "f" dan D_2 adalah >, = atau <. Karena C_2 sama seperti C_1 kecuali bahwa kondisi sederhana kedua di dalam C_2 adalah persamaan elasional, kita dapat membangun himpunan batasan untuk C_2 denganmemodifikasi himpunan pembatas [(t,t),(f,t),(t,f)] yang ditentukan untuk C_1. Perhatikan bahwa "t" untuk $(E_3 = E_4)$ mengimplikasikan "=" dan bahwa "f" untuk $(E_3=E_4)$ mengimplikasikan "<" atau ">". Dengan menggantikan (t,t) dengan (t,=) maka hasil himpunan batasan untuk C_2 adalah {t,=),(t,<), (t,>). Cakupan himpunan batasan tersebut akan menjamin pendeteksian Boolean dan kesalahan-kesalahan operator realsional pada C_2.

Contoh 3, diketahui kondisi sebagai berikut :

$$C_3 : (E_1 > E_2) \& (E_3 = E_4)$$

Dimana E_1,E_2, E_3 dan E_4 adalah persamaan aritmatika. Pembatas kondisi untuk C_3 adalah bentuk (D_1, D_2), dimana masing-masing dari D_1 dan D_2 adalah >, = atau <. Karena C_3 sama seperti C_2, kecuali bahwa kondisi sederhana yang pertama pada C_3 adalah persamaan relasional, maka kita dapat mmembangun himpunan batasan untuk C_3 denganmemodifikasi himpunan batasan untuk C_2 dan akan diperoleh :

$$\{(>,=), (=,=),(<,=),(>,>),(>,<)\}$$

Cakupan dari himpunan batasan tersebut akan menjamin pendeteksian kesalahan operator relasional pada C_3.

1.7.2. Pengujian Aliran Data

Metode pengujian aliran data memilih jalur pengujian dari suatu program sesuai dengan lokasi definisi da menggunakan variabel-variabel pada program. Sejumlah pengujian aliran data telah dipelajari dan dibandingkan.

Untuk menggambarkan pendekatan pengujian aliran data, diasumsikan bahwa masing-masaing statement pada suatu program diberi nomor statemen yang unik dan setiap fungsi tidak memodfikasi parameter atau variabel globalnya. Untuk statemen dengan S sebgai nomor statementnya :

DEF(S) ={X|statemen S berisi sebuah definisi dari X}

USE(S) = X|statemen S berisi suatu penggunaan dari X}

Bila statemen S adalah statemen if atau loop, maka himpunan DEF-nya kosong dan himpunan USE-nya didasarkan pada kondisi statemen S. Definisi variabel X pada statemen S dikatakan hidup pada staemen S' jika ada suatu jalur dari statemen S ke statemen S' yang tidak berisi defiisi yang lain dari X.

Rantai definition-use (atau rantai DU) dari variabel X berbentuk [X,S,S'] dimana S dan S' adalah nomor statemen X pada DEF(S') dan USE(S') dan definisi X pada statemen S hidup pada statemen S'.

Strategi pengujian aliran data sederhana mengharuskan setiap rantai DU dicakup paling tidak satu kali. Strategi ini disebut strategi pengujian DU. Pengujian DU tidak menjamin percakupan semua cabang sebuah program. Akan tetapi, satu cabang tidak dijamin dicakup oleh pengujian DU hanya pada situasi jarang seperti konstruksi if-then-else dimana bagian then tidak memilii definisi mengenai sembarang variabel dan bagian else tidak ada. Dalam situasi seperti ini, cabgan else dari statemen if tersebut tidak perlu dicakup oleh pengujian DU.

Strategi pengujian aliran data berguna untuk memilih jalur pengujian dari suatu program yang berisi statemen if dan loop yang tersarang. Untuk menggambarkannya, perhatikan aplikasi pengujian DU untuk memilih jalur pengujian pada PDL berikut :

```
Proc x
    B1;
```

```
Do while C1
If C2
   Then
         If C4
           Then B4;
            Else B5;
        Endif;
    Else
         If C3
              Then B2;
              Else B3;
        Endif;
   Endif
Enddo;
B6;
Endproc;
```

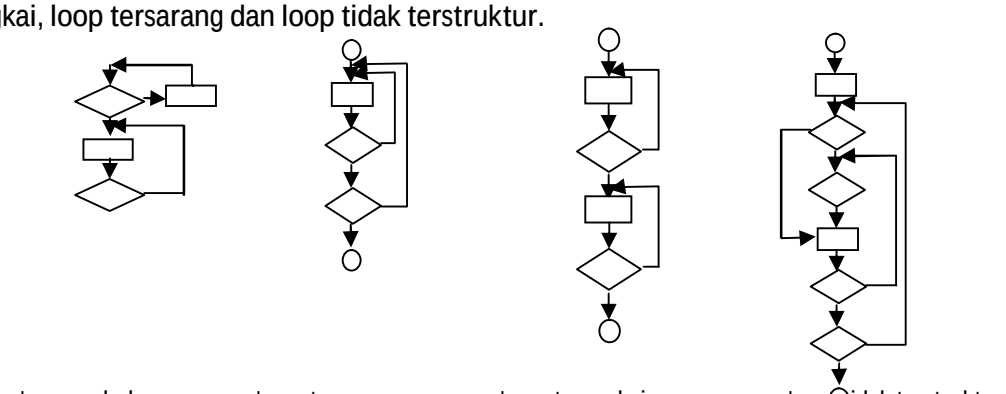
Untuk membuat strategi pengujian DU memilih jalur pengujian dari diagaram aliran kontrol, kita perlu tahu terdefinisi dan penggunaan masing-masing kondisi atau blok pada PDL. Anggap bahwa variabel X didefinisikan pada statemen terakhir dari blok B1, B2, B3, B4 dan B5 dan digunakan dalam statemen pertama dari blok B2, B3, B4, B5 dan B6. Startegi pengujian DU memerlukan ekseusi jalur terpendek masing-masing dari B_i, $0 < i \leq 5$ ke masing-masing dari B_j, $1 < j \leq 6$. Pengujian semacam itu mencakup banyak penggunaan variabel X dalam kondisi C1, C2, C3 dan C4). Meskipun ada 25 rantai DU dari variabel X, kita hanya membutuhkan lima jalur untuk mencakup rantai-ranatai DU tersebut. Alasannya adalah diperlukan lima jalur untuk mencakup rantai DU X dari B_i, $0 < i \leq 5$ ke B6 dan rantai Du yang lain dapat dicakup degan membuat lima jalur tersebut berisi iterasi dari loop.

Bila kita mengaplikasikan strategi pengujian cabang untuk memilih jalur pengujian dari PDL tersebut di atas, kita tidak memerlukan informasi tambahan. Untuk memilih jalur dari diagram pengujian BRO, kita perlu tahu struktur masing-masing kondisi atau blok (setelah pemilihan suatu jalur dari suatu program, kita harus menentukan apakah jalur tersebut mungkin dikerjakan dengan mudah untuk program itu, misal apakah ada paling tidak satu inut yang menggunakan jalur tersebut).

Karena statemen pada suatu program berhubungan satu dengan yang lainnya sesuai dengan definisi dan penggunaan variabel, maka pendekatan aliran data efektif untuk perlindungan dan kesalahan. Tetapi, masalah cakpan pengujian pengukuran dan pemilihan jalur pengujian untuk pengujian aliran data lebih sulit daripada masalah yang berhubugan dengan pengujian kondisi.

1.7.3. Pengujian Loop

Pengujian loop merupakan teknik pengujian white-box yang secara ekslusif berfokus pada validitas konstruksi loop. Empat kelas loop yang berbeda dapat didefinisikan : loop seerhana, loop terangkai, loop tersarang dan loop tidak terstruktur.



Gambar 10. Beragam jenis loop

- Loop sederhana. Himpunan berikut harus diaplikasikan pada loop sederhana, diman n adalah jumlah maksimum yang diijinkan melewati loop tersebut.
    1. Abaikan keselurhan loop
    2. Hanya satu yang melewati loop
    3. Dua yang melewati loop
    4. m melewati loop diman m < n
    5. n – 1, n, n+1 melewati loop
- Loop tersarang. Bila kita ingin memperluas pendekatan pengujian bagi loop sederhana ke loop tersarang, jumlah pengujian mungkin akan berkembang secara geometris sesuai tingkat pertambahan persarangan sehingga sejumlah pengujian menjadi tidak praktis. Beizer mengusulkan suatu pendekatan yang membantu mengurangi jumlah pengujian :
    1. Mulai pada loop yang paling dalam. Atur sema loop ke nilai minimum.
    2. Lakukan pengujian loop sederhana untuk loop yang paling dalam sementara menjaga loop yang paling luar pada nilai parameter iterasi minimumnya (misal pencacah loop). Tambahkan pengujian yang lain untuk nilai out of range aau nilaiyang tidak diperbolehkan
    3. Bekerja menuju ke luar, dengan melakukan pengujian untuk loop selanjutnya, teteapi menjaga semua loop bagian luar yang lain pada nilai minimumnya dan loop tersarang lainnya pada harga “tertentu”.
    4. Lanjutkan sampai semua loop telah tersarang.
- Loop terangkai. Loop terangkai dapat diuji denganmenggunakan pendekatan yang ditentukan untuk loop sederhana bila masing-masing dari loop itu independen terhadap yang lain. Tetapi bila dua loop dirangkai dan pencacah loop untuk loop 1 digunakan sebagai harga awal untuk loop 2, kemudian loop tersebut menjadi tidak independen, maka pendekatan diaplikasikan ke loop tersarang direkomendasi.
- Loop tidak terstruktur. Kapan saja memungkinkan kelas loop ini harus didesain lagi untuk mencerminkan penggunaan konsepsi pemrograman terstruktur.

1.8. Pengujian Black-Box

Pengujian black-box berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengjian black-box memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua peryaratan fungsional untuk suatu program. Pengujian black-box bukan merupakan alternatif dari teknik white-box tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode white-box.

Pengujian black-box berusaha menemukan kesalahan dalam kategori sebagai berikut :

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalhan kinerja
5. Inisialisasi dan kesalahan terminasi

Tidak seperti pengujian white box yang dilakukan pada awal proses pengujian, pengujian black box cenderung diaplikasikan selama tahap akhir pengujian. Karena pengujianblack box memperhatikan struktur kontrol, maka perhatian berfokus pada domain informasi. Pengujian didesain untuk menjawab pertanyaan-pertanyaan berikut :

- Bagiamana vaiditas fungsional diuji ?

- Kelas input apa yang akan membuat test cse menjadi baik ?
- Apakah sistem sangat sensitif terhadap harga input tertentu ?
- Bagaimana batasan dari suatu data diisolasi ?
- Kecepatan data apa dan volume data apa yang dapat ditolerir oleh sistem ?
- Apa pengaruh kombinasi tertentu dari data terdahap operasi sistem ?

Dengan mengaplikasikan teknik black box, maka kita menarik serangkaian test case yang memenuhi kriteria berikut ini :

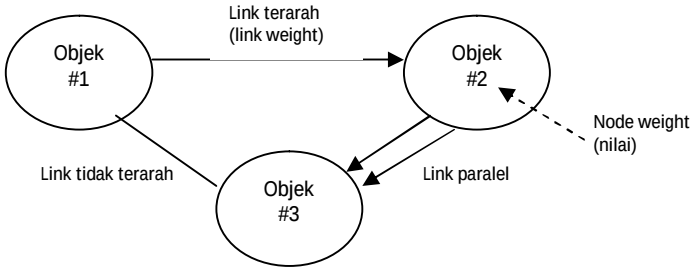
1. Test case yang mengurangi, dengan harga lebih dari saatu, jumlah test case tambahan yang harus didesain untuk mencapai pengujian yang dapat dipertanggungjawabkan.
2. Test case yang memberi tahu kita sesuatu mengenaikehadiran atau ketidakhadiran kelas kesalahan daripada memberi tahu kesalahan yang berhubugnan hanya dengan pengujian spesifik yang ada.

1.8.1. Metode Pengujian Graph-based

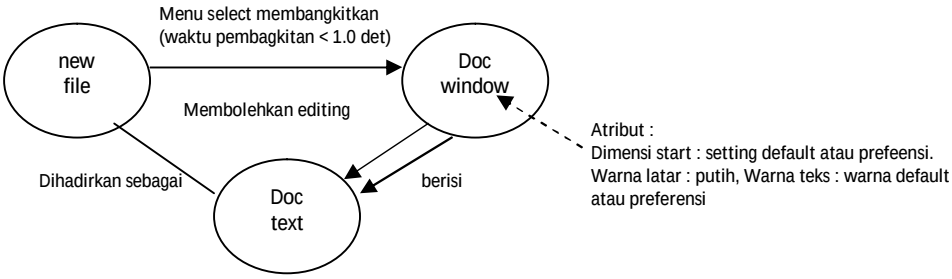
Langkah pertama pada pengujian black box adalah “memahami objek” yang dimodelkan di dalam perangkat lunak dan hubungan yang akan menghubungkan objek tersebut. Setelah hal itu dilakukan maka langkah selanjutnya adalah menentukan sederaetan pengujian yang membuktikan bahwa “semua objek memiliki hubungan yang diharapakan satu dengan yang lainnya”. Dengan kata lain, pengujian perangkat lunak dimulai dnegan membuat grafik dari objek-objek yang penting dan hubungan objek-objek serta kemudian memikirkan sedereetan pengujian yang akan mencakup grafik tersebut sehingga masing-masing objek dan hubungan digunakan dan kesalahan ditemukan. Untuk melakuka langkah-langkah tersebut, perekayasa perangkat lunak memulianya dengan membuat suatu grafik, sekkumpulan simpul yangmerepresentasikan objek; link yang merepresentasikan hubungan antar objek; node weight yang menggambarkan properti dari suatu simpul (misal : nilai data tertentu atau tingkah laku keadaan) dan links weight yang menggambarkan beberapa karakteristik suatu link.

Representasi simboki dari grafik diperihatkan pada gambar 11. Simpul-simpul direpresentasikan sebagai lingkaran yang dihubungkan oleh ling yang memakai sejumlah bentuk yang berbeda. Link terarah (direpresentasikan oleh sebuah anak panah) menunjukkan bahwa hubungan bergerak hanya dalam satu arah. Link dua arah yang disebut link simetris, mengimplikasikan bahwa hubungan tersebut berlaku dalam dua arah. Link paralel digunakan pada saat sejumlah hubungan yang berbeda dibangun di antara simpul-simpul grafik.

Sebagai contoh sederhana, perhatikan bagian dari suatu grafik untuk aplikasi pengolah kata gambar 12 :



Gambar 11. Notasi Grafik



Gambar 12. Contoh sederhana Notasi Grafik

Objek #1 = new file menu select
Objek #2 = document window
Objek #3 = document text

Seperti diperlihatkan pada gambar, pemilihan menu New File menghasilkan document window. Node weight dari document window memberikan daftar atribut window tersebut yang akan diharapakan pada saat window dimunculkan. Link weight menunjukkan bahwa window harus dimunculkan dalam kurang dari 1.0 detik. Link tak terarah membangun hubungan simetris di antara new file select menu dan text document dan lion parale menunjukkan hubungan antara document window dan document text. Kenyataanya grafik yang jauh lebih detail harus dimumculkan sebagai pendahuluan ke desain test case. Perekayasa perangkat lunak kemudian melakukan test case dengan melewatkan grafik tersebut dan mencakup masing-masing dari hubugan yang diperlihatkan. Test case itu didesain untuk menemukan kesalahan pada berbagai hubungan.

Beizer menggambarkan sejumlah metode pengujian behavioral yang dapat menggunakan grafik :

- Pemodelan aliran transaksi. Simpul-simpul merepresentasikan langkah-langkah pada beberapa transkasi (misal, langkah yang diperlukan untuk membuat reservasi pesawat dengan menggunakan layanan online) dan link merepresenasikan hubungan logis di antara langkah-langkah (misal flight.information.input diikuti oleh validation/availibility.processing). Diagram aliran data dapat digunakan untuk membantu menciptakan grafik tipe ini.
- Pemodelan keadaan terbatas. Simpul-simpul merepresentasikan keadaan perangkat lunak yang dapat diamati oleh pemakai yang berbeda (misal, masing-masing “layar muncul sebagai sebuah juru tulis entri order yang mengambil urutan telepon), dan link merepresentasikan transisi yang terjadi untuk bergerak dari satu keadaan ke keadaan lainnya (misal : order-information diperiksa selam inventory-availibiity-look-up diikuti dengan customer billing-information-input). Diagram transisi keadaan dapat digunakan untuk membantu membuat grafik tipe ini.
- Pemodelan aliran data. Simpul-simpul merupakan objek data, sementara link adalah transformasi yang terjadi untuk menterjemahkan satu objek data ke objek data yang lain. Contoh, simpul FICA.tax.withheld (FTW) dihitung dari Gross.wages (GW) dengan menggunakan hubungan FTW =0,062 x GW.
- Pengujian timing. Simpul-simpul merepresentasikan objek program dan link adalah hubungan sekuensial antara objek-objek tersebut. Link weight digunakan untuk menentukanwaktu eksekusi yang dibutuhkan pada saat program mengeksekusi.

Pengujian graph-base dimulai dengan definisi semua simpul dan node weight, dimana objek dan atribut diidentifikasi. Model data dapat digunakan sebagai titik awal, tetapi penting untuk dicatat bahwa banyak simpul dapat merupakan objek program (tidak secara eksplisit direpresentasikan dalam model data). Untuk memeberikan indikasi dari titik mulai dan berhenti untuk grafik tersebut, berguna sekali bila simpul masuk dan keluarnya ditentukan.

Sekali simpul diidentifikasi maka link dan link weight harus dibangun. Secara umum, link harus diberi nama, meskipun link yang merepresentasikan aliran kontrol di antara objek program tidak perlu diberi nama.

Dalam banyak kasus, model grafik dapat memiliki loop. Pengujian loop dapat diaplikasikan pada tingkat tingkah laku (black box). Grafik tersebut akan membantu mengindentifikasi loop-loop yang memerlukan pengujian tersebut.

Masing-masing hubungan dipelajari secara terpisah sehingga test case dapat dilakukan. Transitivitas hubungan sekuensial dipelajari untuk menentukan bagaimana pengaruh hubungan tersebut menyebar pada objek yang ditentukan pada suatu grafik. Transitivitas dapat digambarkan dengan memperhatikan tiga objek : X , Y dan Z. Perhatikan hubugnan berikut ini :

X diperlukan untuk menghitung Y
Y diperlukan untuk menghitug Z

Sehingga dibangun hubungan transitif antara X dan Z

X diperlukan untuk menghitung Z

Berdasarkan hubungan transitif ini, pengujian untuk menemukan kesalahan dala penghitungan Z harus memepertimbangkan berbagai harga untuk X maupun Y.

Simetri hubungan (lin grafik) juga merupakan panduan penting ke desain test case. Bila suatu link benar-benar dua arah (simetris), maka penting untuk menguji fitur tersebut. Fitur UNDO pada berbagai aplikasi personal komputer mengimplementasikan simetri yang terbatas, yaitu bahwa UNDO mengijinkan suatu aksi dihapuskan setelah aksi itu diselesaikan. Hal itu harus diuji secaramendalam dan semua perkecualian (yakni tempat dimana UNDO tidak dapat digunakan) harus dicatat. Akhirnya, setiap simpul pada grafik harus memiliki hubungan yang mengarah kepada dirinya sendiri : pada dasarnya dalah loop “no anction” atau”action null”. Hubungan refleksif itu juga harus diuji.

Pada saat disain test case dimulai, maka sasaran pertamanya adalah mencapai cakupan simpul (node coverage). Ini berarti bahwa pengujian harus didesain untuk memeprlihatkan bahwa tidak aa simpul yang dnegna sembrono diabaikan dan bahwa node weight (atribut objek) adalah benar.

Selanjutnya menenkankan cakupan link. Masing-masing hubungan diuji berdsarkan propertinya. Misal, hubungan simetri untuk memperlihatkan bahwa pada dasarnya hubungan itu memiliki dua arah. Hubungan transitif diuji untuk memeperlihatkan bahwa ada transitivitas. Hubunga relfreksif diuji untuk memastikan bahwa loop null ada. Bila link weight telah ditentukan, pengujian dilakukan utnuk menunjukkan bahwa weight telah ditentukan, pengujian dilakukan untuk menunjukkan bahwa weight itu valid. Akhirnya, pengujian loop dipanggil.

1.8.2. Partisi Ekivalensi

Partisi ekivalensi adalah metode pengujian black-box yangmembagai domain input dari sautu program ke dalam kelas data dari mana test case dapat dialkukan. Test case yang ideal mengungkap kelas kesalahan (misal, pemrosesan yang tidak benar terhadap semua data karakter) yang akan memerlukan banyak kasus untuk dieksekusi sebelum kesalaha umum diamati. Partisi ekivalensi berusaha menentukan sebuah test case yang mengungkap kelas-kelas kesalahan, sehingga mengurangi jumlah total test case yang harus dikembangkan.

Desain test case untuk partisi ekivalensi didasarkan pada evaluasi terhdap kelas ekivalensi untuk suatu kondisi input. Dengan menggunakan konsep yan telah dijelaskan pada bagian sebelumnya, bila serangkaian objek dapat di-link oleh hubungan yan gsimetris, transitif dan refleksif,

maka ada kelas ekivalensi. Kelas ekivalensi merepresentasikan serangkaian keadaan valid atau invalid untuk kokdisi input. Secara khusus, suatu kondisi input dapat berupa harga numeris, suatu rentang harga atau serangkaian haraga terkait atau sebuah kondisi Boolean. Kelas ekivalensi dapat ditentukan sesuai pedoman berikut ini :

1. Bila kondisi input menentukan suatu range, maka satu kelas ekivalensi valid dan dua yang invalid ditentukan
2. Bila suatu kondisi input membutuhkan suatu harga khusus, maka satu kelas ekivalensi vaild dan uda yang invalid ditentukan
3. Bila suatu kondisi menentukan anggota suatu himpunan, maka satu kelas ekivalensi valid atau dua yang invalid ditentukan.
4. Bila suatu kondisi input adalah Boolean, maka satu kelas valid dan satu yang invalid ditentukan.

Sebagai contoh, perhatikan data yang dijaga sebagai bagian dari dari suatu aplikasi perbankan otomatis. Pemakai dapat "menghubungi" bank tersebut dengan menggunakan komputer personalnya, sebuah password enam digit dan diikuti dengn serangkaian perintah kata kunci yang memicu berbagai fungsi perbankan. Perangkat lunak yang dipasok untuk aplikasi perbankan menerima data dalam bentuk :

Kode area-kosong atau tiga nomor digit
Prefik – tiga nomor digit tidak mulai dengan 1 atau 0
Sufik - empat nomor digit
Password – enam nilai alfanumeris digit
Perintah "cek", "deposit","bayar pajak" dan sebagainya.

Kondisi input yang sesuai dengan masing-masing elemen data untuk aplikasi perbankan dapat ditentukan sebagai :

Kode area	:	Kondisi input, Boolean – kode area mungkin atau mungkin tidak ada kondisi. Input range – nilai yang ditentukan antara 200 dan 999 dengan perkecualian khusus.
Prrefiks	:	Kondisi input range – harga yang ditetapkan > 200 dengan tanpa digit 0
Sufiks	:	Kondisi input, harga – panjang empat digit
Password	:	Kondisi input, Boolean-password dapat ada atau tidak ada Kondisi input, harga – antrian enam karakter
Perintah	:	Kondisi input, himpunan – berisi perintah yang sudah ditulis di atas.

Dengan mengaplikasikan pedoman untuk derivasi kelas ekivalensi, test case untuk masing-masing item data domain input dapat dikemabngkan dan dieksekusi. Test case dipilih sehingga jumlah atribut yang terbesar dari suatu kelas ekivalensi digunakan saat itu juga.

1.8.3. Analisis Nilai Batas

Kareana alasan yang tidak jelas, jumlah kesalahan yang lebih besar terjadi pada batas domain input daripada di pusat. Karena itulah analisis nilai batas/boundary value analysis (BVA) telah dikembangkan sebgai teknik pengujian. Analisis nilai batas memunculkan pemilihan test case yang menggunakan nilai batas.

Analisis nilai batas adalah teknik desain proses yang melengkapi partisi ekivalensi. Daripada memilih sembarang elemen kelas ekivalensi, BVA lebih mengarah kepada pemilih test case pada "edge" dari kelas. Daripada hanya berfokus pada kondisi input, BVA melakukan test case dari domain output.

Dalam banyak hal, pedoman untuk BVA sama dengan yang diberikan untuk partisi ekivalensi:

1. Bila suatu kondisi input mengkhususkan suatu range dibatasi oleh nilai a dan b,maka test case harus didesain dengan nilai a dan b, persis di atas dan di bawah a dan b secara berkesesuaian.
2. Bila suatu kondisi input mengkhususkan sejumlah nilai, maka test case harus dikembangkan dengan menggunakan jumlah minimum dan maksimum. Nilai tepat di atas dan di bawah minimum dan maksimum juga diuji.
3. Pedoman 1 dan 2 diaplikasikan ke kondisi output. Contohnya anggap bahwa suhu vs.tabel tekanan diperlukan sebagai output dari program analisis rekayasa. Test case harus didesain untuk menciptakan laporan output yang menghasilkan jumlah minimum (dan maksimum) entri tabel yang diijinkan.
4. Bila struktur data program telah memesan batasan (misal, suatu array memiliki suatu batas yang ditentukan dari 100 entri), pastikan untuk mendesain test case yang menggunakan struktur data pada batasannya.

Sebagian besar perekayasa perangkat lunak secara intuitif melakukan BVA sampai beberapa tingaktaa. Dengan mengaplikasikan pedoman tersebut, pengujian batasan akan lebih lengkap, sehingga kemungkinan untuk berhasil mendeteksi kesalahan menjadi lebih besar.

1.8.4. Pengujian Perbandingan

Ada banyak situasi (seperti avionik pesawat udara) dimana reliabilitas perangkat lunaknya sangat kritis. Dalam aplikasi semacam itu, perangkat lunak dan perangkat keras redundan sering digunakan untuk meminimalkan kemungkinan kesalahan. Pada saat perangkat lunak redundan dikembangkan, tim rekayasa perangkat lunak yang terpisah mengembangkan versi-versi independen dari suatu aplikasi dengan menggunakan spesifikasi yang sama. Dalam situasi semacam itu, setiap versi dapat diuji dengan data uji yang sama untuk memastikan bahwa semua versi memberikan output yang identik. Kemudian semua versi dieksekusi secara paralel dnegan perbandingan real time hasil untuk memastikan konsistensi.

Dengan pelajaran mengenai sistem redundan, para peneliti mengusulkan agar versi perangkat lunak independen dikembangkanbagi aplikasi kritis, bahkan bila hanya sebuah versi tunggal saja yang akan digunakan pada sistem berbasis komputer yang disampaikan. Versi independen membentuk basis teknik pengujian black box yang disebut pengujian perbandingan atau pengujian back to back.

Bila implementasi bertingkat dari spesifikasi yang sama telah dihasilkan, maka test case yang didesain dengan menggunakan teknik black box yang lain (misal partisi ekivalensi) diberikan sebagai input untuk masing-masing versi perangkat lunak tersebut. Bila output dari masing-masing versi sama, maka diasumsikan bahwa implementasinya benar. Bila outputnya berbeda, maka masing-masing dari aplikasi itu diperiksa untuk menentukan cacat pada suatu versi atau agar perbedaan itudapat lebih jelas. Pada sebagian besar kasus, perbandingan output dapat dilakukan dengan suatu peranti yang diotomatisasi.

Pengujian perbandingan tidaklah mudah. Bila spesifikasi dari mana semua fungsi telah dikembangkan mengandung kesalahan, maka semua versi kemungkinan besar merefleksikan kesalahan. Lagi pula jika masing-masing versi independen identik tetapi tidak benar, maka pengujian kondisi akan gagal mendeteksi kesalahan.

1.9. Pengujian Untuk Aplikasi dan Lingkungan Khusus

Pada saat perangkat lunak komputer menjadi semakin kompleks,maka kebutuhan akan pendekatan pengujian yang khusus juga makin berkembanga. Metode pengujian black-box dan white box yang dibicarakan sebelumnya dapat diaplikasikan pada semua lingkungan, arsitektur dan aplikasi, tetapi kadang-kadang dalam pengujian diperlukan pedoman dan pendekatan yang unik.

Pada bagian ini akan dibahas pedoman pengujian bagi lingkungan arsitektur dan aplikasi khusus yang umumnya ditemui oleh para perekayasa perangkat lunak.

1.9.1. Pengujian GUI

Graphical User Interfaces (GUI) menyajikan tantangan yang menarik bagi para perekayasa. Karena komponen reusable berfungsi sebagai bagian dari lingkungan pengembangan GUI, pembuatan interface pemakai telah menjadi hemat waktu dan lebi teliti. Pada saat yang sama, kompleksitas GUI telah berkembanga, menimbulkan kesulitan yang lebih besar di dalam desain dan eksekusi test case.

Karena GUI modern memiliki bentuk dan cita rasa yang sama maka dapat dilakukan sederetan pengujian standar. Pertanyaan berikut dapat berfungsi sebagai panduan untuk serangkaian pengujian generik untuk GUI :

Untuk windows :

- Apakah window akan membuka secara tepat berdasarkan tipe yang sesuai atau perintah berbasis menu ?
- Dapatkah window di-resize, digerakkan atau digulung ?
- Apakah semua isi data yang diisikan pada window dapat dituju dengan tepat dengansebuah mouse, function keys, anak panah penunjuk dan keyboard ?
- Apakah window dengan cepat muncul kembali bila dia ditindih dan kemudian dipanggil lagi ?
- Apakah semua fungsi yang berhubungan dengan window dapat diperoleh bila diperlukan ?
- Apakah semua fungsi yang berhubungan dengan window operasional ?
- Apakah semua menu pull down, tool bar,scroll bar, kotak dialog, tombol ikon dan kontrol yang laina dapat diperoleh dan dengan tepat ditampilkan untu window tersebut ?
- Pada saat window bertingkat ditampilkan, apakah nama window tersebut direpresentasikan secara tepat ?
- Apakah window yang aktif disorot secara tepat ?
- Bila multitasking digunakan, apakah semua window diperbarui pada waktu yang sesuai ?
- Apakah pemilihan mouse bertingkat atau tidak benar di dalam window menyebabkan efek samping yang tidak diharapkan ?
- Apkaha audio prompt dan atau color prompt ada di dala window atau sebagai konsekuensi dari operasi windoe disaajikan menurut spesifikasi ?
- Apakah window akan menutup secara tepat ?

Untuk menu pull down dan operasi mouse :

- Apakah menu bar yang sesuai ditampilkan di dalam konteks yang sesuai ?
- Apakah menu bar aplikasi menampilkan fitur-fitur yang terkait dengan sistem (misal tampilan jam) ?
- Apakah operasi menu pull down bekerja secara tepat ?
- Apakah menu breakway, palette dan tool bar bekerja secara tepat ?
- Apakah semua fungsi menu dan subfungsi pulldown didaftar secara tepat ?
- Apakah semua fungsi menu dapat dituju secara tepat oleh mouse ?
- Apakah typface, ukuran dan format teks benar ?
- Mungkinkah memanggil masing-masing fungsi menu denganmenggunakan perintah berbasis teks alternatif ?
- Apakah fungsi menu disorot berdasarkan konteks operasi ang sedang berlangsung di dalam suatu window ?
- Apakah semua menu function bekerja seperti diiklankan ?
- Apakah nama-nama menu funvtion bersifat self explanatory ?

- Apakah help dapat diperoleh untuk masing-masing item menu, apakah dia sensitif terhadap konteks ?
- Apakah operasi mouse dikenali dnegna baik pada seluruh konteks interaktif ?
- Bila klik ganda diperlukan, apakah operasi dikenali di dalam konteks ?
- Jika mouse mempunyai tombol ganda, apakah tombol itu dikenali sesuai konteks ?
- Apakah kursor, indikator permosesan (seperti jam) dan pointer secara tepat berubah pada saat operasi yang berbeda dipanggil ?

Entri Data :

- Apakah entri data alfanumeris dipantulkan dan diinput ke sistem ?
- Apakah mode grafik dari entri (misal, slide bar) bekerja dengan baik ?
- Apakah data invalid dikenali dengan baik ?
- Apakah pesan input data sangat pintar ?

Sebagai tambahan untuk pedoman tersebut, grafik pemodelan keadaan yang terbatas dapat digunakan untuk melakukan sederetan pengujian yang menekankan objek program dan data spesifik yang relevan dengan GUI.

Karena sejumlah besar permutasi yang bekesesuaian dengan operasi GUI, maka pengujian harus didekati denga menggunakan peranti otomatis. Sudah banyak peranti pengujian GUI yang muncul dipasaran selama beberapa tahun terakhir.

1.9.2. Pengujian Arsitektur Client Server

Arsitektur client/server (C/S) menghadirkan tantangan yang berarti bagi para penguji perangkat lunakan. Sifat terdistribusi dari lingkungan client/server, masalah kinerja yang berhubungan dengan pemrosesan transaksi, kehadiran potensial dari sejumlah platform perangkat keras yang berbeda, kompleksitas komunikasi jaringan,kebutuahn aka layanan client multipel dari suatu database terpusat dan persyaratan koordinasi yang dibebabkan pada server, semua secara bersama-sama membuat pengujian terhadap arsitektur C/S dan perangkat lunak yang ada didalamnya menjadi jauh lebih sulit daripada pengujian aplikasi yang berdiri sendiri. Kenyataannya studi industri terakhir menunjukkan pertambahan berarti di dalam waktu pengujian dan biaya ketika lingkungan C/S dikembangkan.

1.9.3. Pengujian Dokumentasi dan Fasilitas Help

Istilah “pengujianperangkat lunak” memunculkan citra terhdapa sejumlah besar test case yang disiapkan untuk menggunakan program komputer dan data yang dimanipulasi oleh program. Dengan melihat kembali definisi perangkat lunak yang disajikan di awal, penting untuk dicatat bahwa pengujian harus berkembang ke elemen ketiga dari konfigurasi perangkat lunak, yaitu “dokumentasi”.

Kesalahan dala dokumentasi dapat menghancurkan penerimaan program seperti halnya kesalahan pada data atu kode sumber. Tidak ada yang lebih membuat frustasi dibanding mengikuti tuntuan pemakai secara tepat dan mendapatkan hasil atau tingkah laku yang tidak sesuai dengan yang diprediksi oleh dokumen. Karena itulah pengujian dokumentasi harus menjadi suatu bagian yang berarti dari setiap rencana pengujian perangkat lunak.

Pengujian dokumentasi dapat didekati dalam dua fase. Fase pertama, kajian teknis formal yang menguji kejelasan editorial dokumen. Fase kedua, live test, menggunakan dokumentasi dalam kaitannya dengan penggunaan program aktual.

Live test untuk dokumentasi dapat didekati dengan menggunaka teknik yang analog dnegan berbagai metode pengujian black box. Pengujian graph-based ddapat digunakan untuk menggambarkan penggunaan program tersebut; partisi ekivalensi dan analisis nilai batas dapat

digunakan untuk menentukan berbagai kelas input dan interaksi yang sesuai. Kegunaan program kemudian ditelusuri pada seluruh dokumen :

- Apakah dokumen tersebut secara akurat menggambarkan bagimana menyelesaikan masing-masing mode penggunaan ?
- Apakah deskripsi dari masing-masing urutan interaksi akurat ?
- Apakah contoh-contoh akurat ?
- Apakah terminologi, gambaran menu dan respon sistem konsisten dengan program aktual ?
- Apakah relatif mudah untuk menempatkan panduan di dalam dokumentasi ?
- Dapatkah trouble shooting dilakukan dnegan mudah dnegan dokumentasi ?
- Apakah tabel dokumen dari isi dan indeks akurat dan lengkap ?
- Apakah desain dokumen (layout, typeface, indetasi, grafik) kondusif untuk pemahaman dan asimilasi cepat terhadap informasi ?
- Apakah semua pesan kesalahan ditampilkan bagi pemakai dan digambarkan secara lebih detail di dalam dokumen ?
- Bila link hiperteks digunakan, apakah mereka akurat dan lengkap ?

Satu-satunya cara yang dapat berjalan untuk menjawab pertanyaan-pertanyaan tersebut adalah dengan menggunakan bagian ketiga yang independen (misal : pemakai yang diseleksi) yang menguji dokumentasi di dalam konteks kegunaan program. Semua diskrepansi dicatat dan area ambiguitas atau kelemahan dokumen ditentuka untuk penulisan ulang yang potensial.

1.9.4. Pengujian Sistem Real-Time

Sifat asinkron dan tergantung waktu yang ada pada banyak aplikasi real time menambahkan elemen baru yang sulit dan potensial untuk bauran pengujian-waktu. Tidak hanya desainer teset case yang harus memeprtimbangkan test case balck box dan white box tetapi juga penanganan kejadian (yaitu pemrosesan interupsi), timing data dan paralelisme tugas-tugas (proses) yang menangani data. Pada banyak situasi, data pengujian yang diberikan pada saat sebuah sistem real time ada dalam satu keadaan akan menghasilkan pemrosesan yang baik, sementara data yang sama yang diberikan pada saat sistem berada dalam keadaan yang berbeda dapat menyebabkan kesalahan.

Contohnya, perangkat lunak real time yang mengontrol alat foto kopi yang baru menerima interupsi operator (yakni operator mesin menekan kunci kontrol seperti reset) dengan tanpa kesalahan pada saat mesin sedang membuan kopian. Interupsi operator yang sama ini bila diinputkan pada saat mesin ada dalam keadaan “jammed” akan menyebabkan sebuah kode diagnostik yang menunjukkan lokasi jam yang akan hilang (kesalahan).

Hubungan erat perangkat lunak real time dan lingkungan perangkat kerasnya dapat juga menyebabkan pengaruh kegagalan perangkat keras pada pemrosesasn perangkat lunak. Kesalahan semacam itu dapat sangat sulit untuk bersimulasi secara realistis.

Metode desain test case yang komprehensif untuks istem real time harus berkembang. Tetapi strategi emapat langkah berikut dapat diusulkan :

- Pengujian tugas. Langkah pertama dalam pengujian perangkat lunak real time adalah menguji masing-masing tugas secara independen, yaitu pengujian white box dan black box yang didesain dan dieksekusi secara independen bagi masing-masing tugas. Masing-masing tugas dieksekusi secara independen selama pengujian ni. Pengujian tugas mengungkap kesalahan di dalam logika dan fungsi, tetapi tidak akan mengungkap timing atau kesalahan tingkah laku.
- Pengujian tingkah laku. Dengan menggunakan model yang idciptakan dengan peranti CASE, dimungkinkan untuk mensimulasi tingkah laku sistem real time dan menguji tingkah lakunya sebagai konsekuensi dari event eksternal. Aktivitas analisis ini dapat berfungsi sebgai dasar bagi desain test case yang dilakukan pada saat perangkat lunak real time dibangun. Dengan

menggunak teknik yang sama dengan partisi ekuivalensi, event-event (misal interupsi, signal, kontrol, data) dikategorikan untuk pengujian. Sebagai contoh, event untuk mesin fotokopi dapat merupakan inerupsi pemakai (misal, pencacah reset), interupsi mekanis (misal, paper jammed), interupsi sistem (misal tone rendah) dan mode kegagalan (misal roller yang terlalu panas). Masing-masing event tersebut diuji secara individual dan tingkah laku sistem yang dapat dieksekusi diperiksa untuk mendeteksi kesalahan yang terjadi sebagai akibat pemrosesasn yang terkait denga event tersebut. Perilaku model sisetm (dikembangkan selama aktivias analisis) dan perangkat lunaka yang dapat dieksekusi dapat dibandingkan untuk penyesuaian. Sekali masing-masing kelas event telah diuji,maka event-event disajikan pada sistem dalam urutan acak dan dengan frekuensi acak. Perilaku sistem diuji untuk mendeteksi kesalahan perilaku.

- Pengujian antar tugas. Setelah kesalahan-keaslahan pada tugas individual dan pada perilaku sistem diisolasi, maka pengujian beralih kepada kesalahan yang berkaitan dengan waktu. Tugas-tugas asinkronous yang dikenali untuk saling berkomunikasi diuji dnegna tingkat data yang berbeda dan pemrosesan dipanggil untuk menentukan apakah kesalahan sinkronisasi antar tugas akan terjadi. Sebagai tambahan, tugas-tugas yang berkomuniaksi melalui antrian pesan atau penyimpanan data, diuji untuk menemukan kesalahan dalam ukuran area penyimpanan data tersebut.
- Pengujian sistem. Perangkat lunak dan perangkat keras diintegrasi dan suatu rentang penuh dari pengujian sistem dilakukan di dalam usahan mengungkap keslahan pada interface perangkat lunak/perangkat keras.

Sebagian besar sitem real time memproses interupsi, karena itu pengujian penanganan terhadap kejadian-kejadian. Boolean merupakan hal yang penting. Dengan menggunakan diagram keadaan transisi dan spesifikasi kontrol, penguji mengembangkan daftar semua interupsi yang mungkin dan pemroesan yang terjadi sebagai konsekuensi dari interupsi. Kemudain pengujian didesain untuk menilai karakteristik sistem berikut ini :

- Apakah priortias interupsi ditetapkan dan ditangani secara tepat ?
- Apakah pemrosesan untuk masing-masing interupsi ditangani dengan tepat ?
- Apakah kinerja (misal waktu pemrosesasn) dari masing-masing prosedur penangan interupsi sesuai dengan persyaratan) ?
- Apakah volume interupsi yang tinggi yang terjadi pada waktu kritis menimbulkan masalah di dalam fungsi atau kinerja ?

Sebagai tambahan, area data global juga digunakan untuk mentransfe informasi sebagai bagian dari pemrosesan interupsi yang harus diuji untuk menilai potensi munculnya efek samping.