Teknik Pengujian Software

Oleh :

Ir. I Gede Made Karma, MT

1

Pengujian Software

Pengujian adalah proses pelaksanaan program dengan penekanan khusus pada pencarian kesalahan sebelum diserahkan kepada pengguna akhir.

2

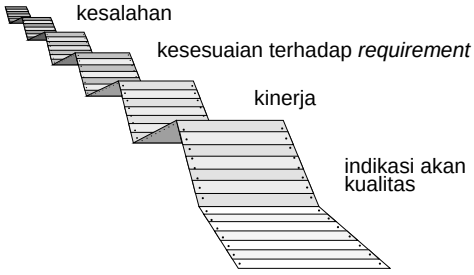
Tujuan Pengujian

1. Pengujian adalah sebuah proses pengeksekusian program khusus untuk pencarian kesalahan.
2. Kasus pengujian yang bagus adalah yang memiliki peluang tinggi untuk pencarian kesalahan yang belum ditemukan.
3. Pengujian yang berhasil adalah yang memecahkan kesalahan yang belum ditemukan.

Pandangan umum : pengujian yang berhasil adalah pengujian yang tidak menemukan kesalahan.

3

Apa yang Ditunjukan Pengujian



4

Prinsip Pengujian

- Semua pengujian harus dikaitkan dengan kebutuhan pelanggan.
- Pengujian harus direncanakan jauh hari sebelum pengujian dimulai.
- Prinsip Pareto diterapkan pada pengujian software.
- Pengujian harus dimulai dari yang kecil dan berkembang pada pengujian yang besar.
- Pengujian menyeluruh tidak dimungkinkan.
- Untuk efektifitas, pengujian sebaiknya dilakukan oleh pihak independen ketiga.

5

Testability : Kemudahan Diuji

- Operability—bekerja dengan baik, pengujian lebih efisien.
- Observability—apa yang dilihat itulah yang diuji.
- Controlability—menunjukkan tingkat pengujian yang dapat dilakukan secara otomatis dan optimal.
- Decomposability—pengujian dapat ditarget.
- Simplicity—kurangi kompleksitas arsitektur dan logika untuk menyederhanakan pengujian.
- Stability—selama pengujian diharapkan sedikit terjadi perubahan.
- Understandability—pemahaman rancangan akan mempermudah pengujian.

6

Atribut Pengujian Yang Baik

1. **Memiliki peluang tinggi untuk menemukan kesalahan.**
2. **Tidak berulang.**
3. **Menjadi *best of breed*.**
4. **Tidak terlalu sederhana atau tidak terlalu kompleks.**

7

Siapa Menguji Software?

developer

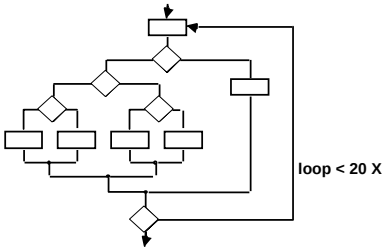
Memahami sistem tetapi akan menguji dengan baik dan dikendalikan oleh penyerahan

independent tester

Harus mempelajari sistem, tetapi akan berusaha untuk mengatasinya, dan dikendalikan oleh kualitas

8

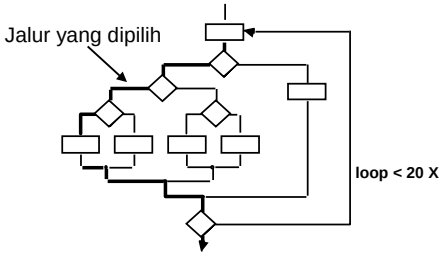
Pengujian Menyeluruh



Terdapat 10^{14} kemungkinan jalur! Apabila kita melakukan satu pengujian per milidetik, akan memerlukan 3,170 tahun untuk menguji program ini!!

9

Pengujian Selektif

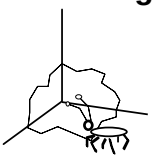


10

Perancangan Kasus Pengujian

"Bugs lurk in corners and congregate at boundaries ..."

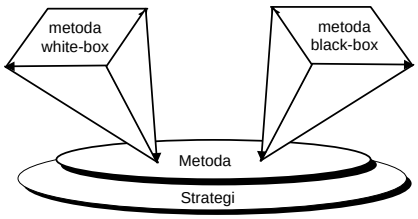
Boris Beizer



TUJUAN	**untuk mengatasi kesalahan**
KRITERIA	**secara lengkap**
BATASAN	**dengan usaha dan waktu minimum**

11

Pengujian Software



12

Black vs White Box Test

- Black-Box Test dipergunakan untuk mendemonstrasikan bahwa fungsi software beroperasi, input dengan baik diterima, output dihasilkan dengan benar, dan integritas informasi eksternal terjaga.
- White-Box Test merupakan pengujian detail prosedural. Jalur logika software diuji dengan kasus pengujian yang menguji sekumpulan kondisi dan/atau loop khusus.

13

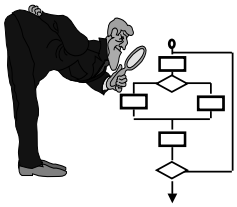
White-Box Testing (1)

Pengujian mempergunakan kasus pengujian yang :

1. Menjamin bahwa semua jalur independen dalam sebuah modul telah diuji minimal sekali.
2. Menguji semua kondisi logika pada sisi benar dan salah.
3. Mengeksekusi semua loop pada batas dan dalam batas operasional pengulangan.
4. Menguji struktur data internal untuk menjamin validitas.

14

White-Box Testing (2)



… sasarannya adalah untuk menjamin bahwa semua statement dan kondisi telah dieksekusi minimal sekali …

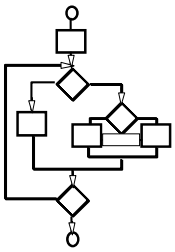
15

Mengapa Tertutup?

- Kesalahan logika dan asumsi yang keliru, secara proposional berlawanan dengan kemungkinan pengeksekusian jalur.
- Kita sering mempercayai bahwa jalur sepertinya tidak dieksekusi; kenyataanya, realitas sering bertentangan dengan intuisi.
- Typography kesalahan adalah acak; Sepertinya jalur yang tidak teruji akan berisi beberapa.

16

Pengujian Jalur Dasar (1)



Pertama, kita hitung kompleksitas *cyclomatic* (logikal program) :

jumlah pencabangan sederhana + 1

or

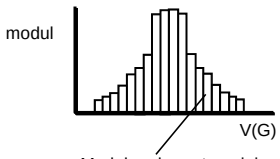
jumlah area tertutup + 1

Dalam kasus ini, V(G) = 4

17

Kompleksitas *Cyclomatic*

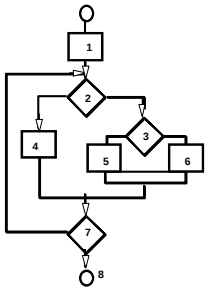
Sejumlah studi industri mengindikasikan bahwa semakin tinggi V(G), maka semakin tinggi kemungkinan atau kesalahan.



Modul pada rentang ini cenderung lebih banyak kesalahan

18

Pengujian Jalur Dasar (2)



Selanjutnya, kita tetapkan jalur independen :

Karena V(G) = 4,
berarti ada empat jalur

Jalur 1: 1,2,3,6,7,8
Jalur 2: 1,2,3,5,7,8
Jalur 3: 1,2,4,7,8
Jalur 4: 1,2,4,7,2,4,...7,8

Akhirnya, kita menetapkan *test cases* untuk menguji jalur-jalur ini.

19

Penetapan Kasus Pengujian

1. Pergunakan rancangan atau code sebagai dasar untuk menggambar *flow graph*.
2. Tentukan kompleksitas *cyclomatic* dari resultante *flow graph*.
3. Tentukan kumpulan jalur dasar independen secara linier.
4. Siapkan kasus pengujian yang akan memaksa pengeksekusian setiap kumpulan jalur dasar.

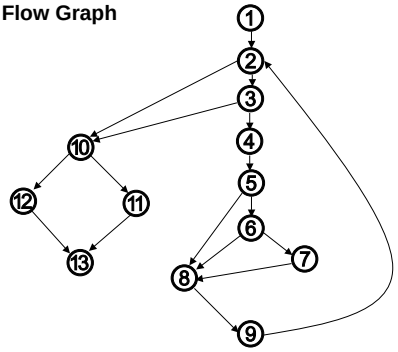
20

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

PDL untuk perancangan test case

21

1. Flow Graph



22

2. Menentukan kompleksitas cyclomatic :
   - ⇦ V(G) = 6 daerah
   - ⇦ V(G) = 17 pertemuan – 13 titik + 2 = 6
   - ⇦ V(G) = 5 predicate nodes + 1 = 6

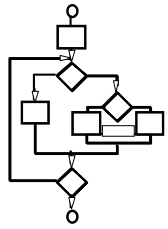
3. Menentukan kumpulan jalur dasar :
   - ⇦ Jalur 1 : 1-2-10-11-13
   - ⇦ Jalur 2 : 1-2-10-12-13
   - ⇦ Jalur 3 : 1-2-3-10-11-13
   - ⇦ Jalur 4 : 1-2-3-4-5-8-9-2- . . .
   - ⇦ Jalur 5 : 1-2-3-4-5-6-8-9-2- . . .
   - ⇦ Jalur 6 : 1-2-3-4-5-6-7-8-9-2- . . .

23

4. Menyiapkan test case :
   - Jalur 1
     value(k) = valid input, dimana k < i untuk $2 \leq i \leq 100$
     value(i) = -999 dimana $2 \leq i \leq 100$
     Tidak dapat diuji sendiri, tapi menjadi bagian jalur 4, 5 dan 6.
   - Jalur 2
     value(i) = -999, hasil yang diharapkan Average = -999.
   - Jalur 3
     usahakan memproses 101 atau lebih nilai. Ke-100 nilai pertama harusnya valid.
     Hasil yang diharapkan sama dengan kasus pengujian 1.
   - Jalur 4
     value(i) = valid input, dimana i < 100, value(k) < minimum dimana k < i
     Hasil : average benar berdasarkan nilai k dan total yang sesuai.
   - Jalur 5
     value(i) = valid input, dimana i < 100, value(k) < minimum dimana $k \leq i$
     Hasil : average benar berdasarkan nilai n dan total yang sesuai.
   - Jalur 6
     value(i) = valid input, dimana i < 100. Hasil : average benar berdasarkan nilai n dan total yang sesuai.

24

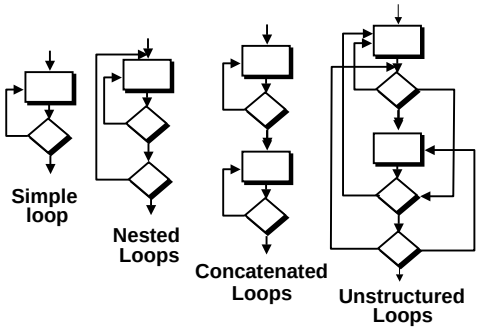
Catatan Untuk Pengujian Jalur Dasar



- **Kita tidak memerlukan *flow chart*, tetapi gambar akan membantu ketika menelusuri jalur program**
- **Hitung setiap *simple logical test*, pengujian gabungan dihitung 2 atau lebih**
- **Pengujian jalur dasar sebaiknya diterapkan pada modul kritis**

25

Loop Testing



26

Loop Testing: Simple Loops

Minimum kondisi — Simple Loops

1. **Lewati keseluruhan loop**
2. **Hanya sekali melalui loop**
3. **Dua kali melalui loop**
4. **m kali melalui loop, m < n**
5. **(n-1), n, dan (n+1) melalui loop**

dimana n adalah jumlah maksimum yang diijinkan melakukan loop

27

Loop Testing: Nested Loops

Nested Loops

Mulai pada loop yang paling dalam. Tetapkan semua loop luar pada nilai parameter pengulangan minimum.

Lakukan pengujian ke min+1, tertentu, max-1 dan max untuk loop paling dalam, sambil menahan loop luar pada nilai minimum.

Pindah keluar satu loop dan lakukan seperti tahap 2, tahan semua loop lain pada nilai tertentu. Lanjutkan tahapan ini sampai loop terluar telah diuji.

28

Loop Testing: Nested Loops

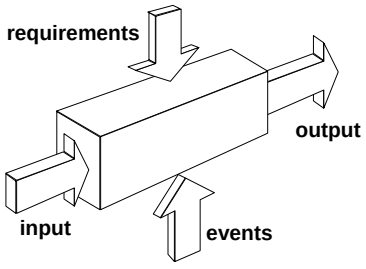
Concatenated Loops

If loop independen terhadap yang lain
then perlakukan masing-masing sebagai *simple loop*
else perlakukan sebagai *nested loop*
endif

Contoh, nilai akhir pencacah loop dari loop ke 1 dipergunakan untuk inisialisasi loop ke 2.

29

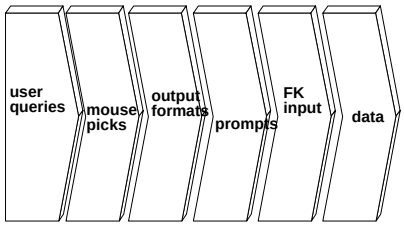
Black-Box Testing



30

Equivalensi Pemisahan

Metoda pengujian *black-box* yang membagi domain input program menjadi kelas data, yang dijadikan dasar penetapan kasus pengujian.



31

Equivalensi Kelas

Representasi dari sekumpulan valid dan invalid keadaan untuk kondisi input. Kondisi input dapat berupa nilai numerik, range nilai, sekumpulan nilai terkait atau boolean.

Petunjuk untuk menetapkan equivalensi kelas :

1. Bila kondisi input adalah range, tetapkan 1 valid dan 2 invalid equivalensi kelas.
2. Bila kondisi input memerlukan nilai tertentu, tetapkan 1 valid dan 2 invalid.
3. Bila kondisi input adalah anggota dari suatu kumpulan, tetapkan 1 valid dan 1 invalid.
4. Bila kondisi input adalah boolean, tetapkan 1 valid dan 1 invalid.

32

Sample Equivalence Classes

Valid data

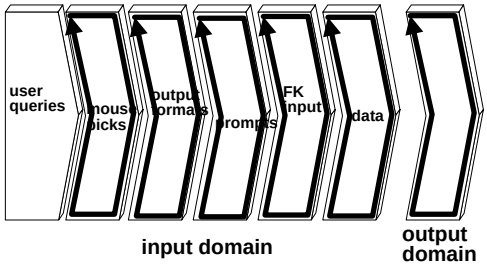
- **user supplied commands**
- **responses to system prompts**
- **file names**
- **computational data**
  - **physical parameters**
  - **bounding values**
  - **initiation values**
- **output data formatting**
- **responses to error messages**
- **graphical data (e.g., mouse picks)**

Invalid data

- **data outside bounds of the program**
- **physically impossible data**
- **proper value supplied in wrong place**

33

Boundary Value Analysis



34

Petunjuk BVA

1. Bila kondisi input menunjukkan batas range a dan b, kasus pengujian sebaiknya dirancang dengan nilai a dan b dan setingkat di atas dan di bawah a dan b.
2. Bila kondisi input menunjukkan sejumlah nilai, kasus pengujian dikembangkan dengan menguji jumlah minimum dan maksimum. Nilai di atas dan di bawah minimum dan maksimum juga diuji.
3. Terapkan petunjuk 1 dan 2 pada kondisi output.
4. Apabila struktur data internal program memiliki batas yang tetap (misal array 100 elemen), pastikan merancang kasus pengujian untuk mengujinya.

35