

Teknik Testing

Tujuan Testing

- ▶ Testing PL merupakan bagian terpenting dari *Software Quality Assurance (SQA)*.
- ▶ Yang harus diperhatikan adalah : testing tidak bisa menentukan apakah suatu PL bebas kesalahan, testing hanya menunjukkan bahwa ada kesalahan pada PL

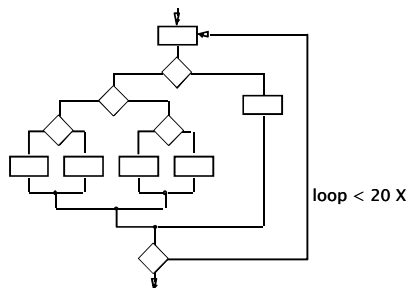
Prinsip Testing

- ▶ Testing harus memastikan semua *user requirement* terpenuhi
- ▶ Testing harus direncanakan sebelum coding dilakukan
- ▶ Memakai prinsip *Pareto* : 80% kesalahan yang ditemukan saat testing berasal dari 20% dari keseluruhan modul dalam PL. Sehingga modul yang 'dicurigai' berpotensi terhadap kesalahan harus diuji lebih teliti

Prinsip Testing (2)

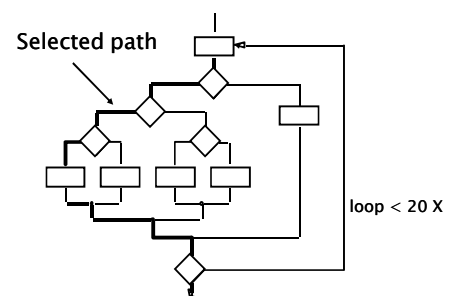
- ▶ Testing bergerak dari 'lingkup kecil' ke 'lingkup besar', fungsi setiap modul diuji terlebih dahulu, menuju integrasi antar modul
- ▶ Tidak mungkin menguji semua kombinasi '*path*' pada PL
- ▶ Testing paling efektif jika melibatkan pihak ketiga (selain user dan developer)

Exhaustive Testing



There are 10^4 possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

Selective Testing



Who Tests the Software?



developer

Understands the system
but, will test "gently"
and, is driven by "delivery"



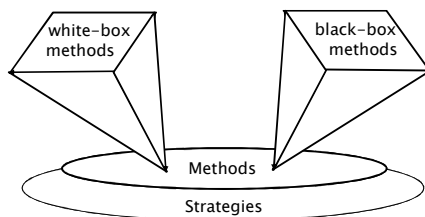
independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

PL yang 'TESTABILITY'

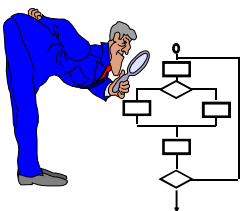
- ▶ Operability : fungsinya berjalan baik, pengujian lebih efisien
- ▶ Observability : "What you see is what you test"
- ▶ Controlability: bisa dilakukan otomatisasi dan optimasi testing
- ▶ Decomposability: untuk isolasi permasalahan
- ▶ Simplicity: lebih sederhana mempercepat testing
- ▶ Stability: sedikit perubahan, testing tanpa kekacauan
- ▶ Understandability

Software Testing



- ▶ White-box testing
 - Mengetahui proses apa yang terjadi di dalam PL
 - Penguji yang melakukan test melihat apakah inout diproses dengan benar sehingga menghasilkan output yang benar
- ▶ Black-box testing
 - Tidak peduli apa yang terjadi di dalam PL, yang penting adalah output
 - Yang dilihat adalah hasil akhir keluaran dan tampilannya saja

White-Box Testing (glass-box testing)



- ▶ Metode pengujian yang bekerja berdasar struktur kontrol pada perancangan prosedural
- ▶ Digunakan untuk pengujian unit-unit fungsional (modul)

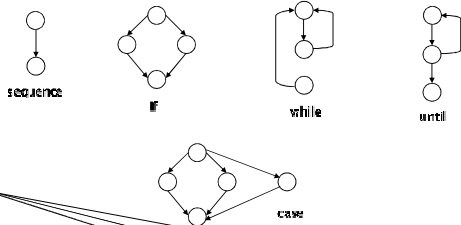
White-Box Testing (glass-box testing)

Pengujian yang diharapkan:

1. Menjamin bahwa semua *independence path* telah dilalui
2. Mencoba kemungkinan *true & false*
3. Mencoba *loop* pada batasan yang telah ditentukan
4. Menguji struktur data internal : data tetap benar selama eksekusi berlangsung

Basis Path Testing

- Pengujian berdasar graph alir dari program
- Pertama kali diperkenalkan oleh Tom Mc Cabe (1976) sebagai teknik untuk white box testing



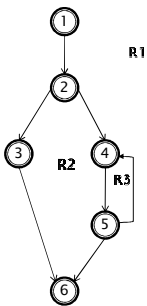
Contoh

```

Procedure Traversal (Q : Queue);
(*membaca setiap elemen queue untuk ditampilkan*)
var
  p: address;
begin
  ① p := Q.head
  ② if (p=Null) then
  ③   writeln ('Queue kosong');
  else
    repeat
      ④ writeln ('Elemen Queue: ', Q.TabQ[p].info);
      ⑤ p := Q.TabQ[p].Next;
    until (p=Null);
  ⑥ end

```

Graph alir untuk prosedur tsb:



Gambar (*)

Region (R) = 3
Busur = 7
Node = 6

Independent Path (IP)

- Yaitu: path yang mengandung paling sedikit sebuah *set of processing* yang **baru** atau kondisi yang **baru** (baru: belum pernah dilewati)
- Pada graph alir, IP merupakan jalur yang mengandung minimal sebuah busur yang belum pernah dilewati (oleh jalur sebelumnya)

Contoh:

IP dari gambar (*) adalah:

- path 1: 1-2-3-6 (semua busur belum pernah dilewati)
- path 2: 1-2-4-5-6 (busur 2-4-5-6 belum pernah dilewati)
- path 4: 1-2-(4-5)*-6 (busur 5-4 belum pernah dilewati)

Dalam testing PL, IP ini harus dilewati minimal sekali

Untuk mengetahui jumlah IP, maka digunakan bilangan kompleksitas siklomatik

Kompleksitas Siklomatik

- Bil. Siklomatik digunakan untuk mengukur kompleksitas PL
- Siklomatik bisa diperoleh dengan:
 1. Menghitung jumlah region pada graph alir
 2. Menghitung kompleksitas siklomatik $V(G)$ dari graph G:

$$V(G) = E - N + 2$$
 dengan E: jumlah busur, N: jumlah Node dari graph sebelumnya, $V(G) = 7 - 6 + 2 = 3$
 3. Menghitung $V(G)$ dengan $V(G) = P + 1$, dimana P adalah jumlah *predicate node* (simpul yang mempunyai busur keluar lebih dari satu) dalam graph G

$$V(G) = 2 + 1 = 3$$
, P = node 5 dan node 2

Testing Struktur Kontrol PL

- ▶ Selain *basis path testing*, testing terhadap struktur kontrol bisa dilakukan dengan:
 1. Testing kondisi
 - menguji kondisi logic dalam modul
 2. Testing aliran data
 - menyangkut lokasi definisi dan pemakaian data
 3. Testing loop
 - menguji kebenaran konstruksi pengulangan

Black-Box Testing

- ▶ Pengujian *black-box* ditujukan untuk melihat apakah PL berfungsi sesuai requirement .
- ▶ Kategori kesalahan yang mungkin ditemukan:
 1. Kesalahan fungsi/fungsi yang hilang
 2. Kesalahan antarmuka
 3. Kesalahan struktur/akses terhadap basis data eksternal
 4. Kesalahan performansi
 5. Kesalahan inisialisasi dan terminasi

Jika *white-box testing* dilakukan pada awal proses testing, maka *black-box testing* cenderung dilakukan pada akhir proses testing PL

Testing untuk keadaan dan aplikasi khusus

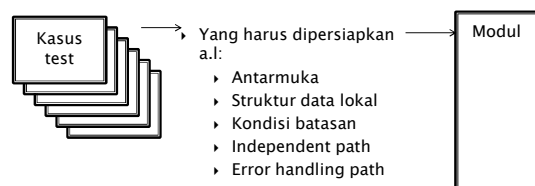
1. Testing GUI
2. Testing arsitektur *client-server*
3. Testing dokumentasi dan fasilitas *help*
4. Testing untuk sistem *real-time*

STRATEGI TESTING

- ▶ RPL dan strategi pengujian PL dapat digambarkan dalam bentuk spiral sbb:

Unit Testing

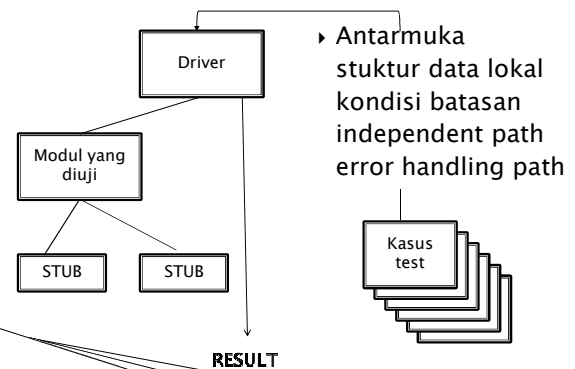
- ▶ Pengujian unit ditujukan menguji unit terkecil dalam PL, yaitu **modul**
- ▶ Orientasi pengujian adalah *white-box testing*:



Prosedur Pengujian Unit

- ▶ Modul bukan program yang '*standalone*' sehingga diperlukan *driver* & *stub* untuk menjalankan modul tersebut
- ▶ *Driver*: berfungsi sebagai '*main program*' yang menerima masukan data untuk pengujian modul
- ▶ *stub*: berfungsi sebagai submodul yang dipanggil oleh modul yang sedang diuji (stub merupakan dummy subprogram)
- ▶ Pengujian unit akan mudah jika modul tersebut mempunyai sifat kohesi tinggi
- ▶ Catatan: *driver* & *stub* bukan termasuk PL yang dipesan, sehingga menjadi overhead bagi developer

Prosedur Pengujian Unit (2)



PERBAIKAN PL

- ▶ Perbaikan PL (*software maintenance*) merupakan proses perubahan PL setelah PL tersebut diterima dan dioperasikan oleh pemesan.
- ▶ Penyebabnya a.l:
 1. Perubahan terhadap *requirement* sebelumnya
 2. Perubahan *environment* (misal: PK)
 3. Error yang tidak ditemukan saat tahap pengujian, muncul setelah PL dioperasikan

Jenis Perbaikan PL, (Sommerville)

1. Corrective maintenance
2. Adaptive maintenance
3. Perfective maintenance

Corrective Maintenance

- ▶ Perbaikan karena adanya kesalahan dalam PL
- ▶ 3 jenis kesalahan yang mungkin terjadi:
 - Kesalahan coding: perbaikan paling sederhana
 - Kesalahan perancangan: harus menulis ulang beberapa bagian program
 - Kesalahan requirement: sistem / PL harus dirancang ulang

Adaptive Maintenance

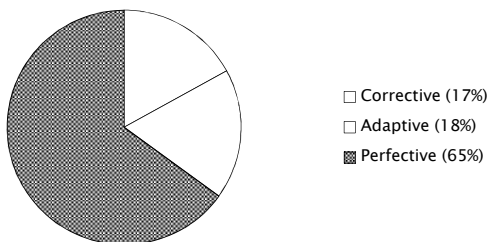
- ▶ Perbaiki PL karena adanya perubahan lingkungan PL
- ▶ Misal: HW atau sistem operasional
- ▶ Fungsionalitas PL tidak berubah

Perfective Maintenance

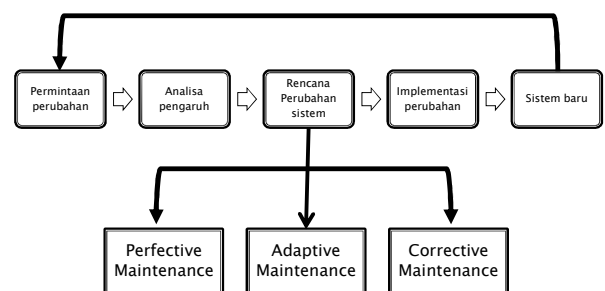
- ▶ Untuk menyempurnakan kemampuan PL
- ▶ Perubahan pada sistem organisasi dimana PL tersebut digunakan, menyebabkan perubahan terhadap fungsional maupun non fungsional PL sehingga PL memerlukan *requirement* yang baru

RPL Maintenance

Besarnya usaha yang harus dilakukan untuk tiap perbaikan



Proses Perbaikan PL

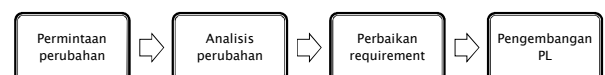


- ▶ "Perbaikan cepat" dilakukan karena adanya kesalahan dan keharusan segera memperbaiki PK agar tidak mengganggu aktifitas sistem
- ▶ Digambarkan dalam diagram sbb:



- ▶ Kelemahan perbaikan semacam itu adalah:
 - Perbaikan tidak didokumentasikan
 - Program secara bertahap 'keluar' dari spesifikasi
 - Lebih tertuju pada program yang 'asal jalan' dari pada program yang memenuhi solusi terbaik

- ▶ Oleh sebab itu akan lebih baik jika tahap perbaikan PL merupakan bagian dari proses rekayasa PL



- ▶ Kelemahannya: tidak sesuai untuk perbaikan yang sifatnya 'segera' (atau darurat karena sangat berpengaruh terhadap kelangsungan sistem). Karena proses ini memakan waktu yang lama.

Faktor Biaya Perbaikan PL

- Faktor teknis
- Faktor non-teknik

Faktor teknis

- Modul independen
- Bahasa pemrograman
- Gaya/cara pemrograman
- Pengujian (testing)
- Kualitas dokumentasi
- Pemakaian teknik manajemen konfigurasi (untuk konsistensi antara perbaikan sistem dengan dokumentasi sistem)

Faktor non-teknis

- Domain aplikasi
- Stabilitas team
- Usia program
- Dependensi program dengan lingkungannya
- Stabilitas HW

- Hal-hal yang bisa dijadikan dasar pengukuran *maintainability* suatu PL adalah:
 1. Jumlah permintaan untuk *corrective maintenance*
 2. waktu yang diperlukan untuk analisis dampak perubahan PL
 3. Waktu yang diperlukan untuk implementasi hasil perubahan PL
 4. Jumlah permintaan perubahan yang tidak terpenuhi.Untuk setiap parameter di atas, jika menunjukkan kenaikan berarti tingkat *maintainability* suatu PL turun