Testing Conventional Applications

Testability

- Operability (Operabilitas) — beroperasi dengan bersih
- Observability (Observabilitas) — hasil dari setiap kasus uji mudah diamati
- Controllability (Dapat dikendalikan)— tingkat pengujian yang dapat diotomatisasi dan dioptimalkan
- Decomposability (Dekomposabilitas) — pengujian dapat ditargetkan
- Simplicity (Kesederhanaan) — mengurangi arsitektur dan logika yang kompleks untuk menyederhanakan tes
- Stability (Stabilitas) — beberapa perubahan diminta selama pengujian
- Understandability (Dapat dimengerti) — tentang desain

What is a “Good” Test?

- Tes yang baik memiliki probabilitas tinggi untuk menemukan kesalahan
- Tes yang baik tidak berlebihan.
- Tes yang baik harus “yang terbaik”
- Tes yang baik seharusnya tidak terlalu sederhana atau terlalu rumit

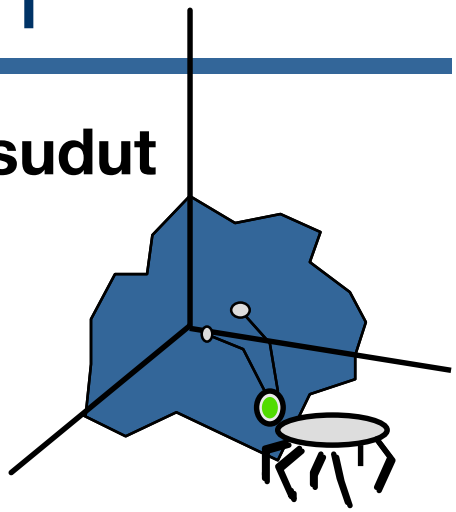
Internal and External Views

- Setiap produk rekayasa (dan sebagian besar hal lainnya) dapat diuji dengan salah satu dari dua cara:
  - Mengetahui fungsi yang ditentukan bahwa suatu produk telah dirancang untuk melakukan, pengujian dapat dilakukan yang menunjukkan masing-masing fungsi berfungsi penuh sementara pada saat yang sama mencari kesalahan di setiap fungsi;
  - Mengetahui cara kerja internal suatu produk, pengujian dapat dilakukan untuk memastikan bahwa "semua roda gigi," yaitu, operasi internal dilakukan sesuai dengan spesifikasi dan semua komponen internal telah dilaksanakan secara memadai.

Test Case Design

" Bug mengintai di sudut-sudut dan berkumpul di batas-batas ..."

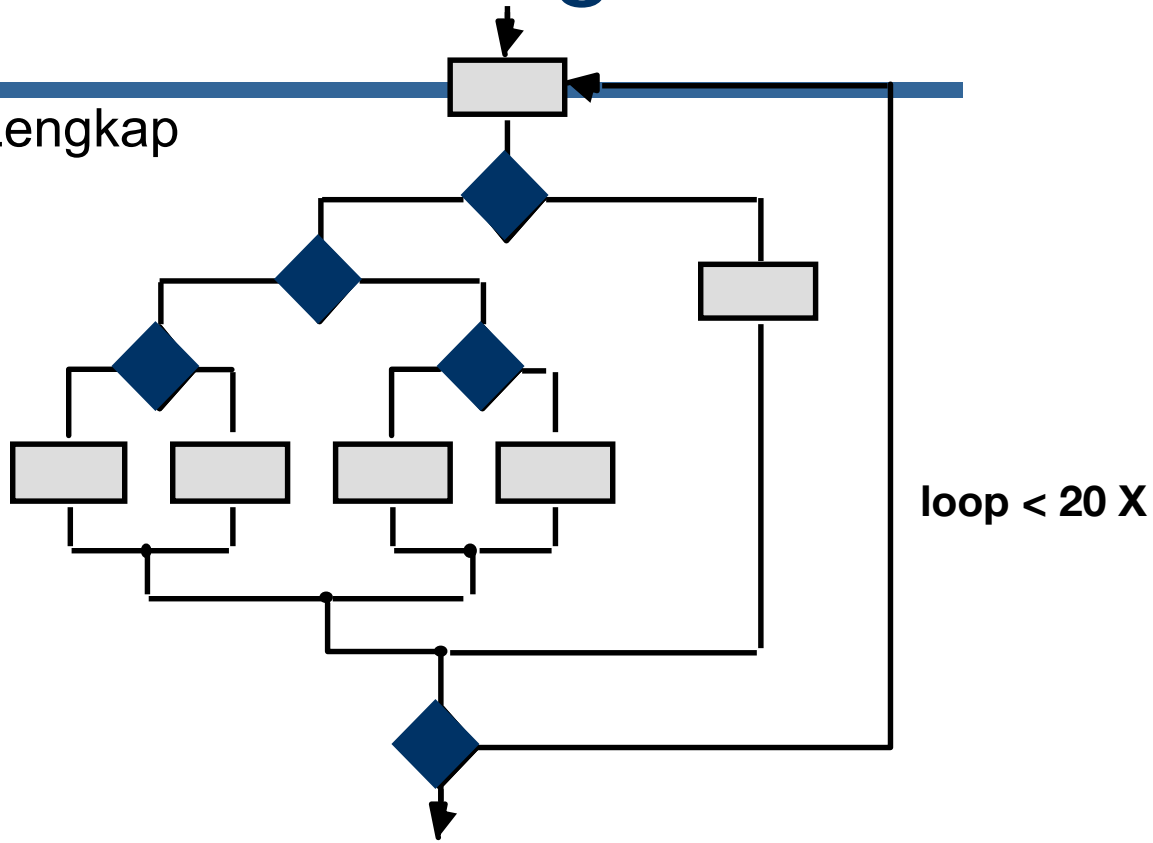
Boris Beizer



OBJECTIVE	**untuk mengungkap kesalahan**
CRITERIA	**Secara lengkap**
CONSTRAINT	**dengan usaha dan waktu minimum**

Exhaustive Testing

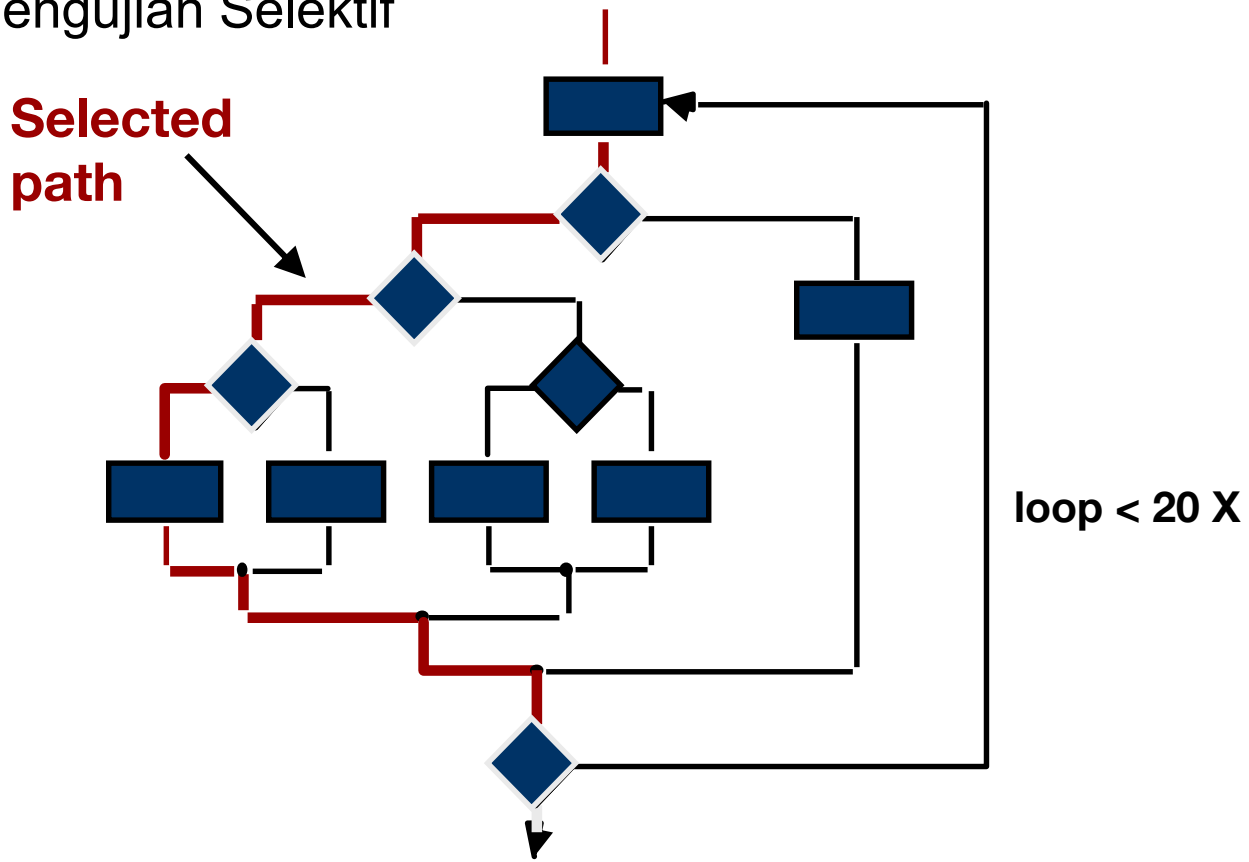
Pengujian Lengkap



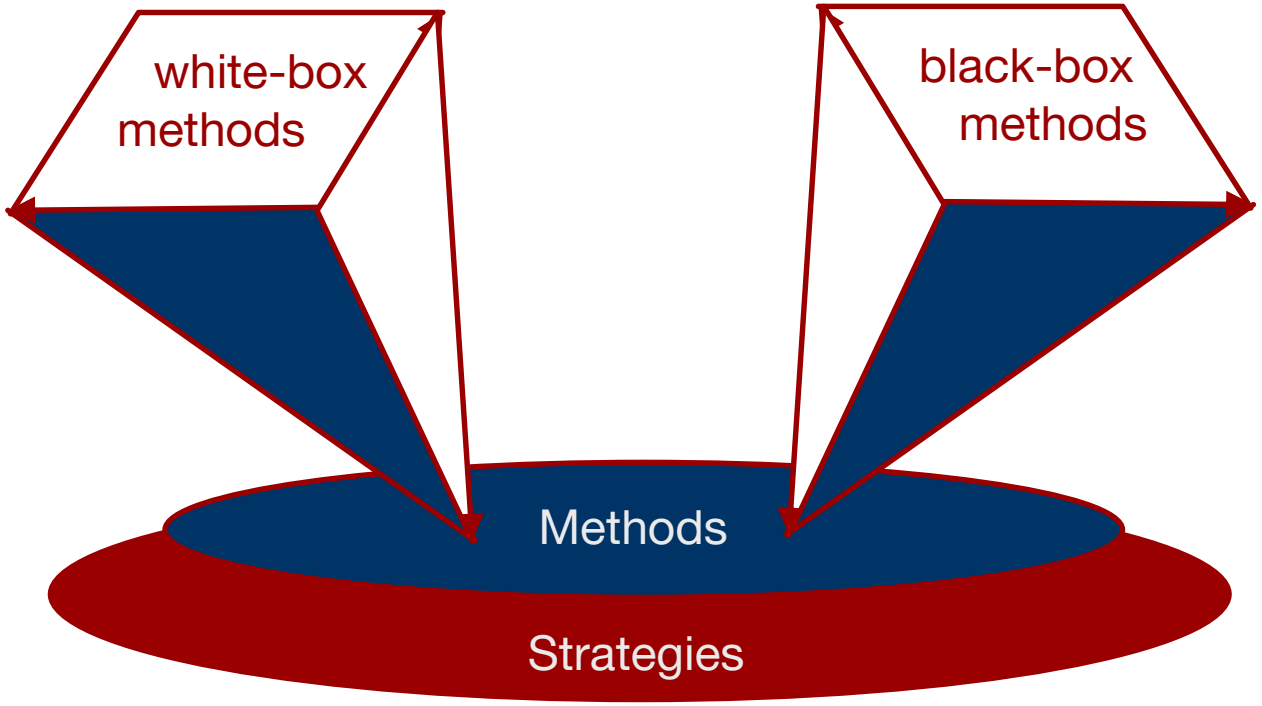
Ada 10^{14} jalur yang memungkinkan! Jika kita menjalankan satu tes per milidetik, ini akan memakan waktu 3.170 tahun untuk menguji program ini !!

Selective Testing

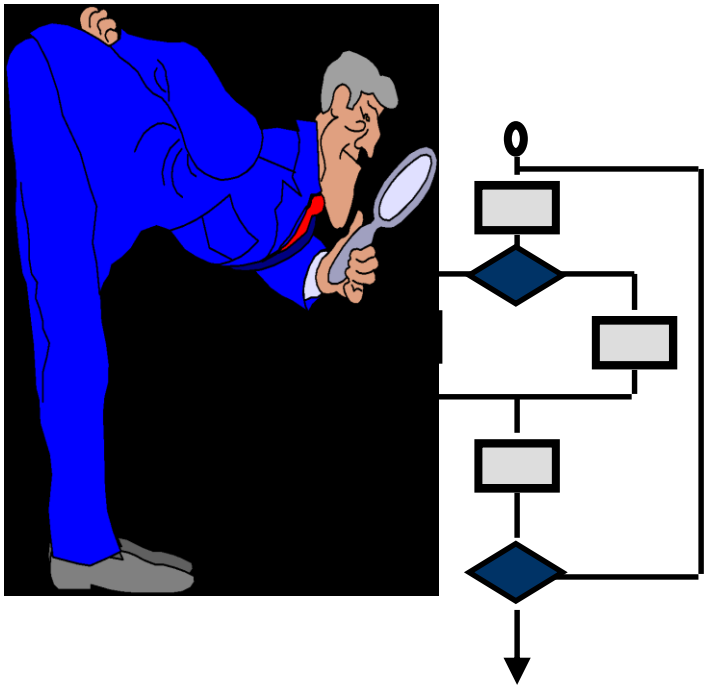
Pengujian Selektif



Software Testing



White-Box Testing



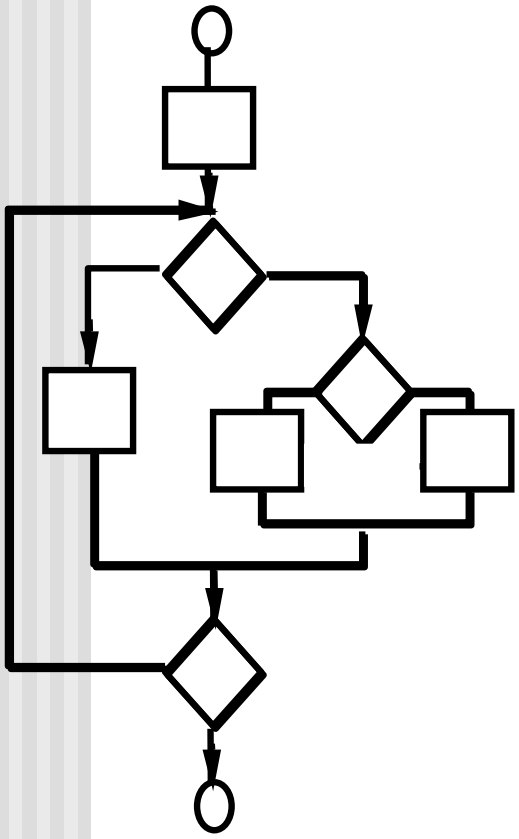
... tujuannya untuk memastikan bahwa semua pernyataan dan ketentuan telah dieksekusi setidaknya satu kali.....

Why Cover?

Mengapa menutupi?

- kesalahan logika dan asumsi yang salah berbanding terbalik dengan probabilitas eksekusi jalur
- kita sering percaya bahwa jalan (jalur) tidak mungkin dilaksanakan; pada kenyataannya, kenyataan seringkali berlawanan dengan intuisi
- kesalahan tipografi bersifat acak; kemungkinan jalur yang belum diuji akan mengandung beberapa

Basis Path Testing



Pertama, hitung kompleksitas cyclomatic:

sejumlah keputusan sederhana + 1

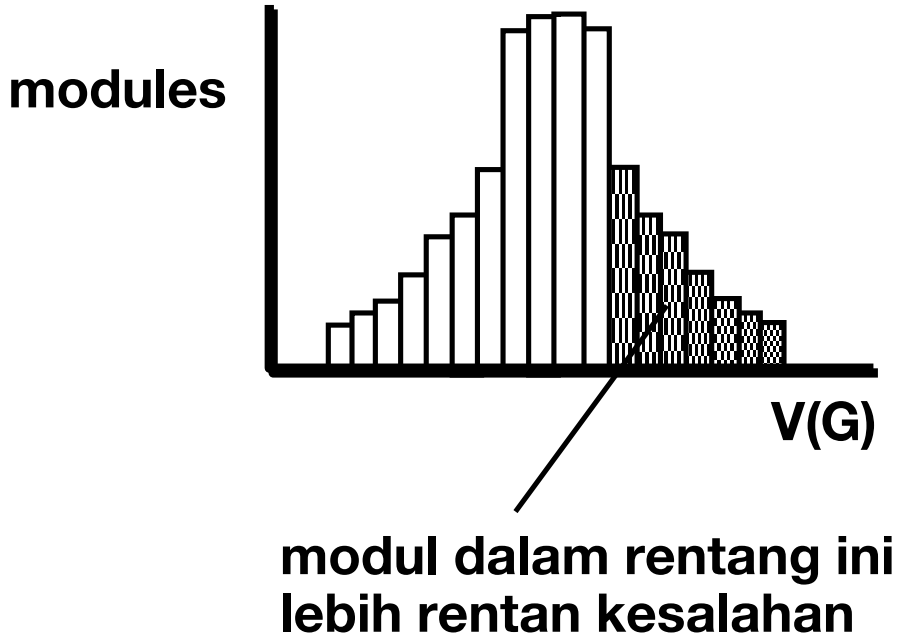
atau

jumlah area tertutup + 1

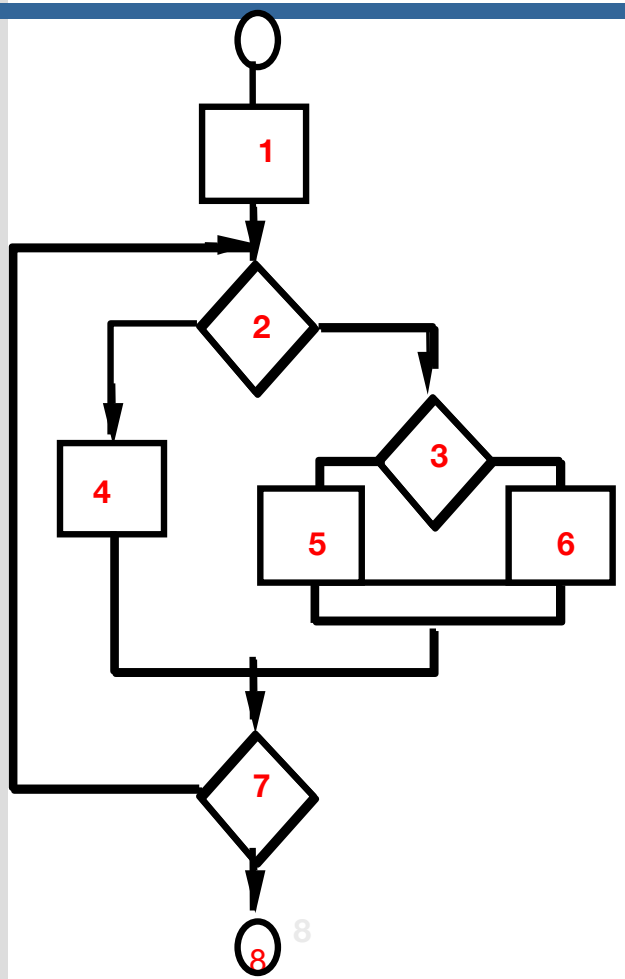
Pada kasus ini, V(G) = 4

Cyclomatic Complexity

Sejumlah studi industri telah menunjukkan bahwa semakin tinggi V (G), semakin tinggi probabilitas atau kesalahan.



Basis Path Testing



Berikutnya, didapatkan jalur independen:

Karena V (G) = 4, ada empat jalur

Path 1: 1,2,3,6,7,8

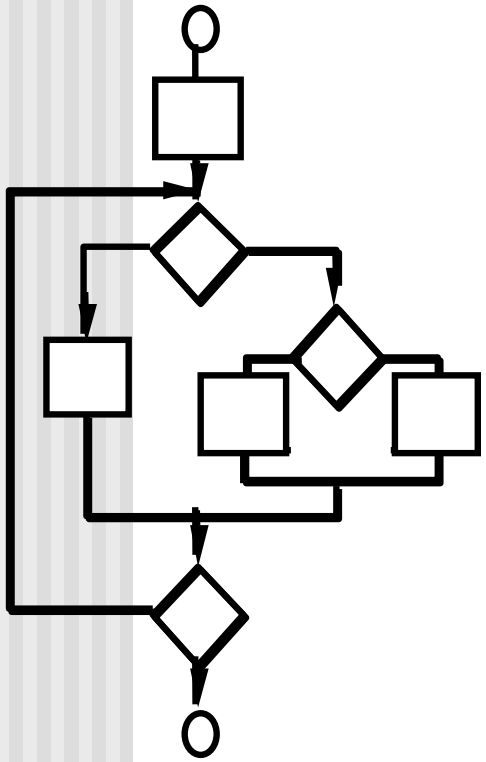
Path 2: 1,2,3,5,7,8

Path 3: 1,2,4,7,8

Path 4: 1,2,4,7,2,4,...7,8

Akhirnya, kami menurunkan kasus uji untuk menjalani jalur ini.

Basis Path Testing Notes



- Anda tidak memerlukan bagan alur, tetapi gambar akan membantu ketika Anda melacak jalur program
- hitung setiap tes logis sederhana, tes majemuk dihitung sebagai 2 atau lebih
- pengujian jalur dasar harus diterapkan pada modul kritis

Deriving Test Cases

Menurunkan Kasus Uji

- *Ringkasan:*
  - Dengan menggunakan desain atau kode sebagai fondasi, gambarkan grafik aliran yang sesuai.
  - Tentukan kompleksitas siklomatik dari grafik aliran yang dihasilkan.
  - Menentukan set dasar jalur yang bebas linear.
  - Mempersiapkan kasus uji yang akan memaksa eksekusi setiap jalur dalam set dasar.

Graph Matrices

- Matriks grafik adalah matriks kuadrat yang ukurannya (mis., Jumlah baris dan kolom) sama dengan jumlah node pada grafik aliran
- Setiap baris dan kolom berhubungan dengan node yang diidentifikasi, dan entri matriks berhubungan dengan koneksi (tepi) antara node.
- Dengan menambahkan bobot tautan ke setiap entri matriks, matriks grafik dapat menjadi alat yang ampuh untuk mengevaluasi struktur kontrol program selama pengujian

Control Structure Testing

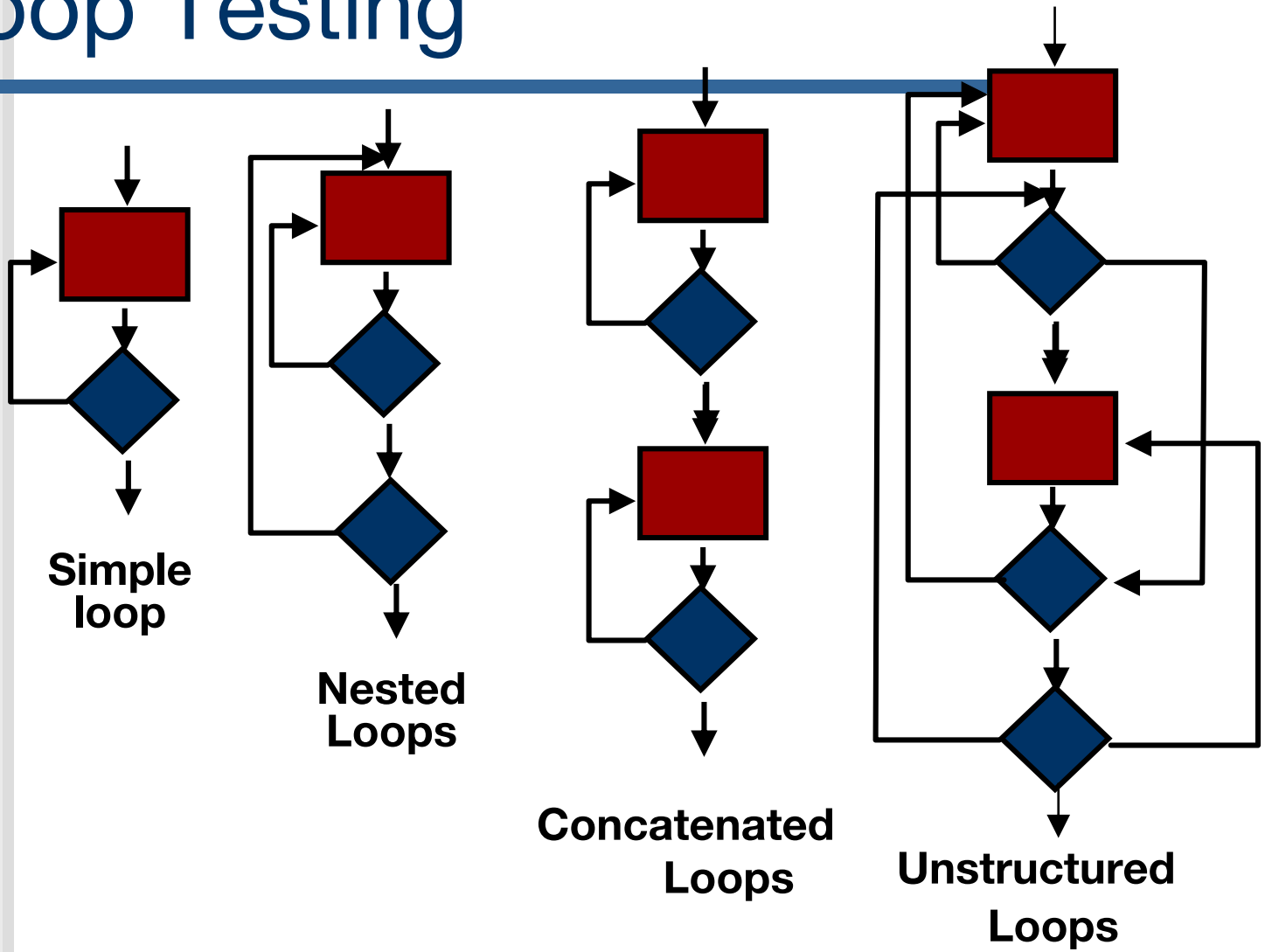
Pengujian Struktur Kontrol

- Condition testing - metode desain kasus uji yang melatih kondisi logis yang terkandung dalam modul program
- Data flow testing - memilih jalur pengujian suatu program sesuai dengan lokasi definisi dan penggunaan variabel dalam program

Data Flow Testing

- Metode pengujian aliran data memilih jalur uji suatu program sesuai dengan lokasi definisi dan penggunaan variabel dalam program.
  - Asumsikan bahwa setiap pernyataan dalam suatu program diberi nomor pernyataan unik dan bahwa setiap fungsi tidak mengubah parameter atau variabel globalnya. Untuk pernyataan dengan S sebagai nomor pernyataannya
    - DEF(S) = {X | statement S contains a definition of X}
    - USE(S) = {X | statement S contains a use of X}
  - A *definition-use (DU) chain* adalah dari bentuk [X, S, S'], di mana S dan S' adalah angka-angka pernyataan, X dalam DEF (S) dan USE (S'), dan definisi dari X dalam pernyataan S adalah langsung di pernyataan S'

Loop Testing



Loop Testing: Simple Loops

Minimum conditions—Simple Loops

1. lewati loop sepenuhnya
2. hanya satu melewati loop
3. dua melewati loop
4. m melewati loop m <n
5. (n-1), n, dan (n +1) melewati loop

di mana n adalah jumlah maksimum dari lintasan yang diizinkan

Loop Testing: Nested Loops

Nested Loops

Mulai dari loop paling dalam. Atur semua loop luar ke nilai parameter iterasi minimum mereka.

Tes min + 1, tipikal, max-1 dan max untuk loop paling dalam, sambil menahan loop luar pada nilai minimumnya.

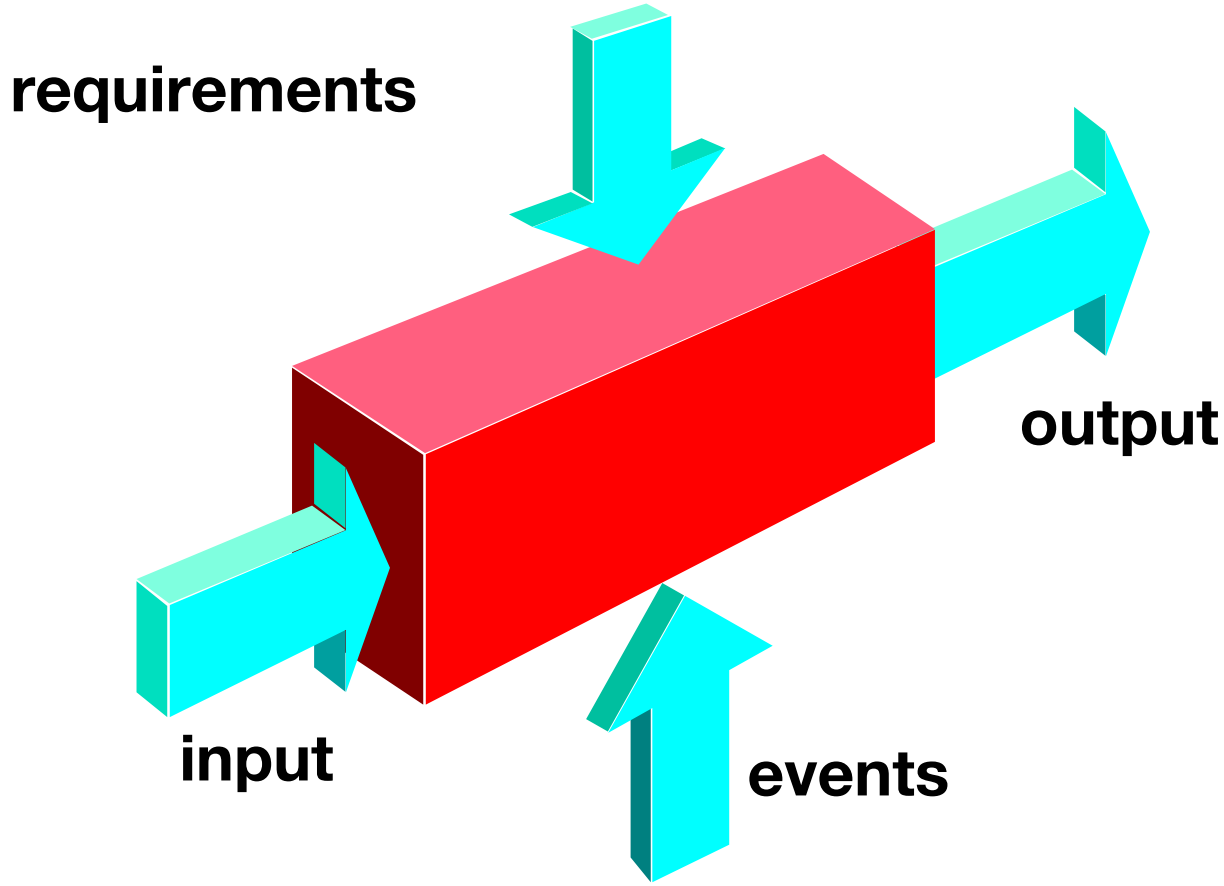
Pindahkan satu loop dan atur seperti pada langkah 2, tahan semua loop lainnya pada nilai tipikal. Lanjutkan langkah ini sampai loop terluar telah diuji.

Concatenated Loops

If loop independen satu sama lain
then perlakukan masing-masing sebagai loop sederhana
else* memperlakukan sebagai loop bersarang

misalnya, nilai penghitung putaran terakhir dari loop 1 digunakan untuk menginisialisasi loop 2.

Black-Box Testing



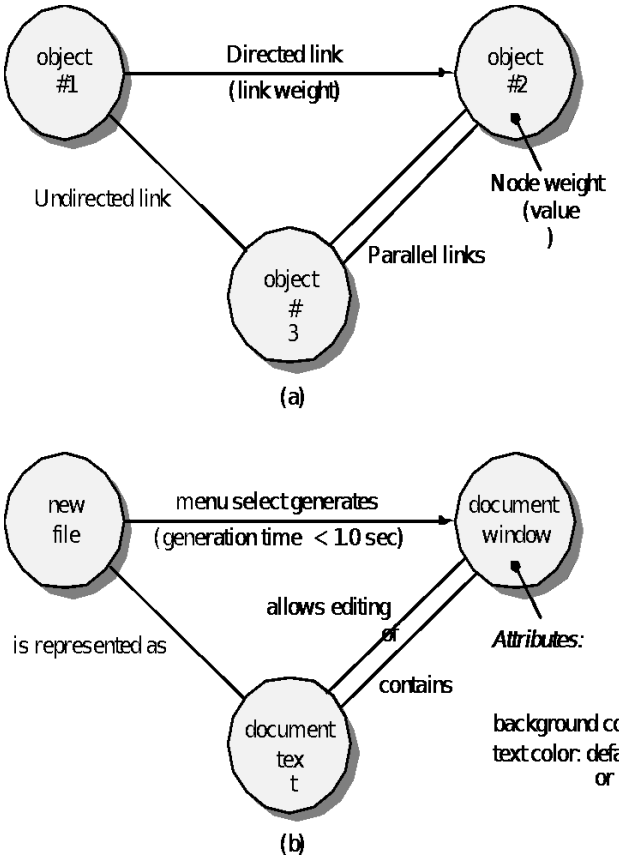
Black-Box Testing

- Bagaimana pengujian validitas fungsional?
- Bagaimana perilaku dan kinerja sistem diuji?
- Kelas input apa yang akan menghasilkan test case yang baik?
- Apakah sistem sangat sensitif terhadap nilai input tertentu?
- Bagaimana batas-batas kelas data diisolasi?
- Berapa kecepatan data dan volume data yang dapat ditoleransi sistem?
- Apa pengaruh kombinasi data tertentu terhadap operasi sistem?

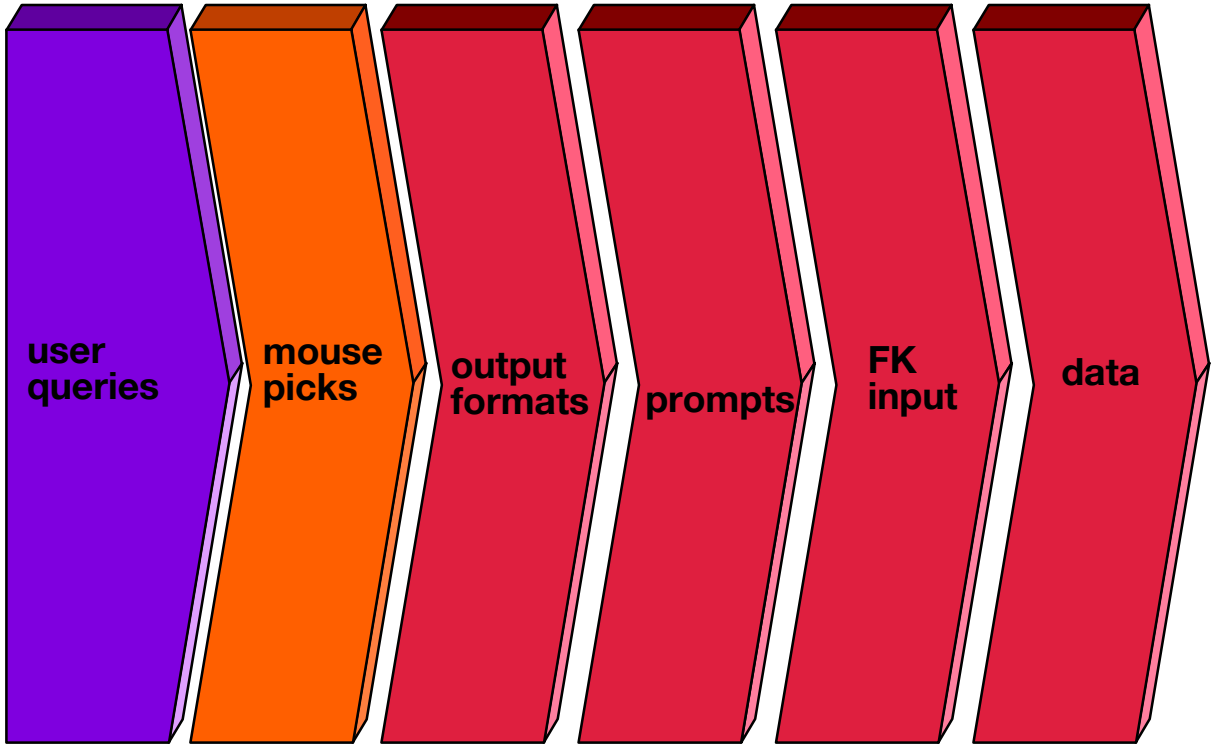
Graph-Based Methods

Untuk memahami objek yang dimodelkan dalam perangkat lunak dan hubungan yang menghubungkan objek-objek ini

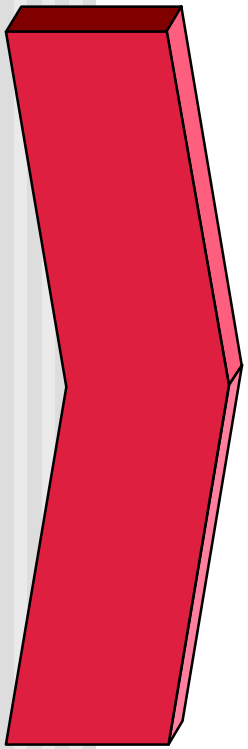
Dalam konteks ini, kita mempertimbangkan istilah "objek" dalam konteks seluas mungkin. Ini mencakup objek data, komponen tradisional (modul), dan elemen berorientasi objek dari perangkat lunak komputer.



Equivalence Partitioning



Sample Equivalence Classes



Valid data

perintah yang diberikan pengguna
tanggapan atas permintaan sistem
nama file
data komputasi

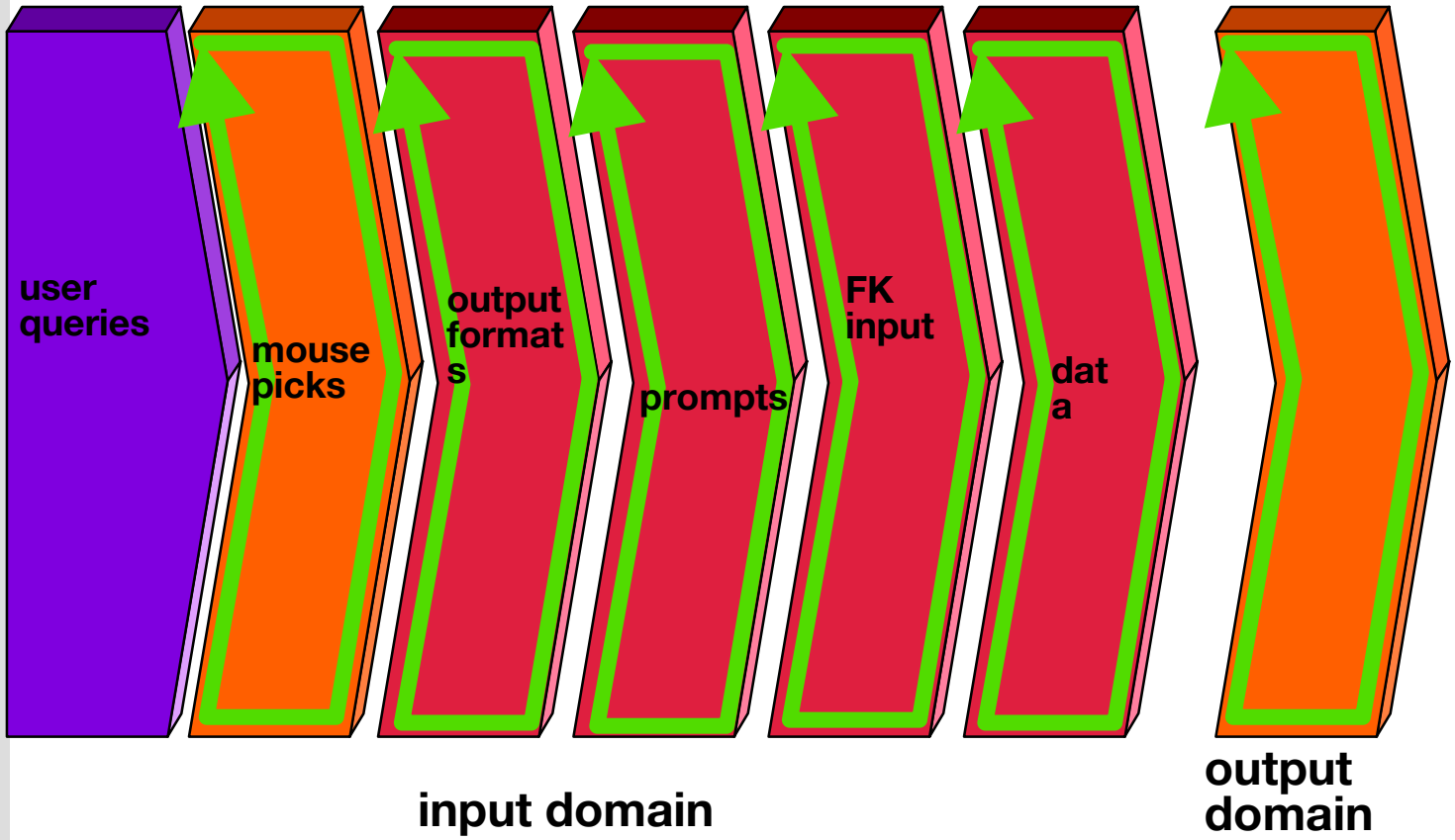
- **parameter fisik**
- **nilai pembatas**
- **nilai inisiasi**

format data keluaran
tanggapan terhadap pesan kesalahan
data grafis (mis., pilihan mouse)

Invalid data

data di luar batas program
data yang secara fisik tidak mungkin
nilai yang tepat diberikan di tempat yang salah

Boundary Value Analysis

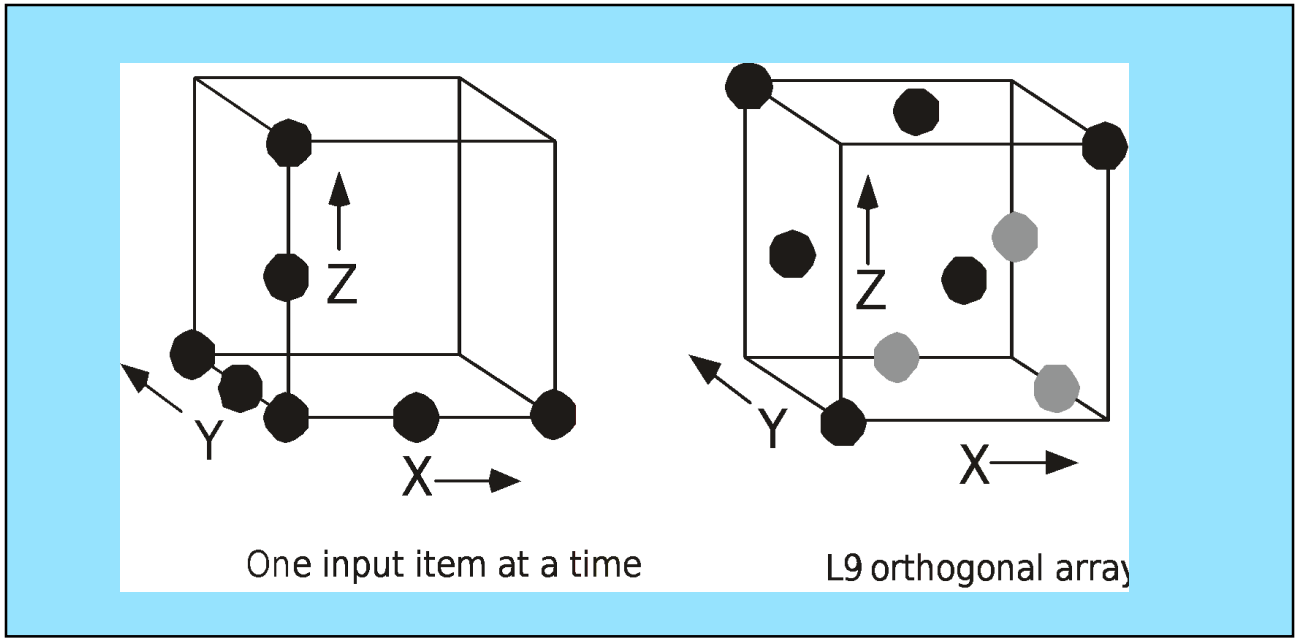


Comparison Testing

- Hanya digunakan dalam situasi di mana keandalan perangkat lunak sangat penting (mis., Sistem yang dinilai manusia)
  - Tim rekayasa perangkat lunak yang terpisah mengembangkan versi aplikasi yang independen menggunakan spesifikasi yang sama
  - Setiap versi dapat diuji dengan data uji yang sama untuk memastikan bahwa semua memberikan hasil yang identik
  - Kemudian semua versi dieksekusi secara paralel dengan perbandingan hasil waktu nyata untuk memastikan konsistensi

Orthogonal Array Testing

- Digunakan ketika jumlah parameter input kecil dan nilai yang diambil oleh masing-masing parameter dibatasi dengan jelas



Model-Based Testing

- Menganalisis model perilaku yang ada untuk perangkat lunak atau membuatnya.
  - Ingatlah bahwa model perilaku menunjukkan bagaimana perangkat lunak akan merespons peristiwa atau rangsangan eksternal.
- Lintasi model perilaku dan tentukan input yang akan memaksa perangkat lunak untuk melakukan transisi dari state ke state.
  - Input akan memicu peristiwa yang akan menyebabkan transisi terjadi.
- Tinjau model perilaku dan catat output yang diharapkan saat perangkat lunak melakukan transisi dari state ke state.
- Jalankan kasus uji.
- Bandingkan hasil aktual dan yang diharapkan dan lakukan tindakan korektif sesuai kebutuhan.

Software Testing Patterns

- Pola pengujian dijelaskan dengan cara yang hampir sama dengan pola desain
- *Contoh:*
  - *Pattern name:* **ScenarioTesting**
  - *Abstract: Setelah pengujian unit dan integrasi dilakukan, ada kebutuhan untuk menentukan apakah perangkat lunak akan tampil dengan cara yang memuaskan pengguna. Pola ScenarioTesting menjelaskan teknik untuk menjalankan perangkat lunak dari sudut pandang pengguna. Kegagalan pada level ini menunjukkan bahwa perangkat lunak telah gagal memenuhi persyaratan yang terlihat oleh pengguna.*