White-Box Testing

SI-318 Testing dan Implementasi Sistem

Rosa Ariani Sukamto, ST

Pendahuluan

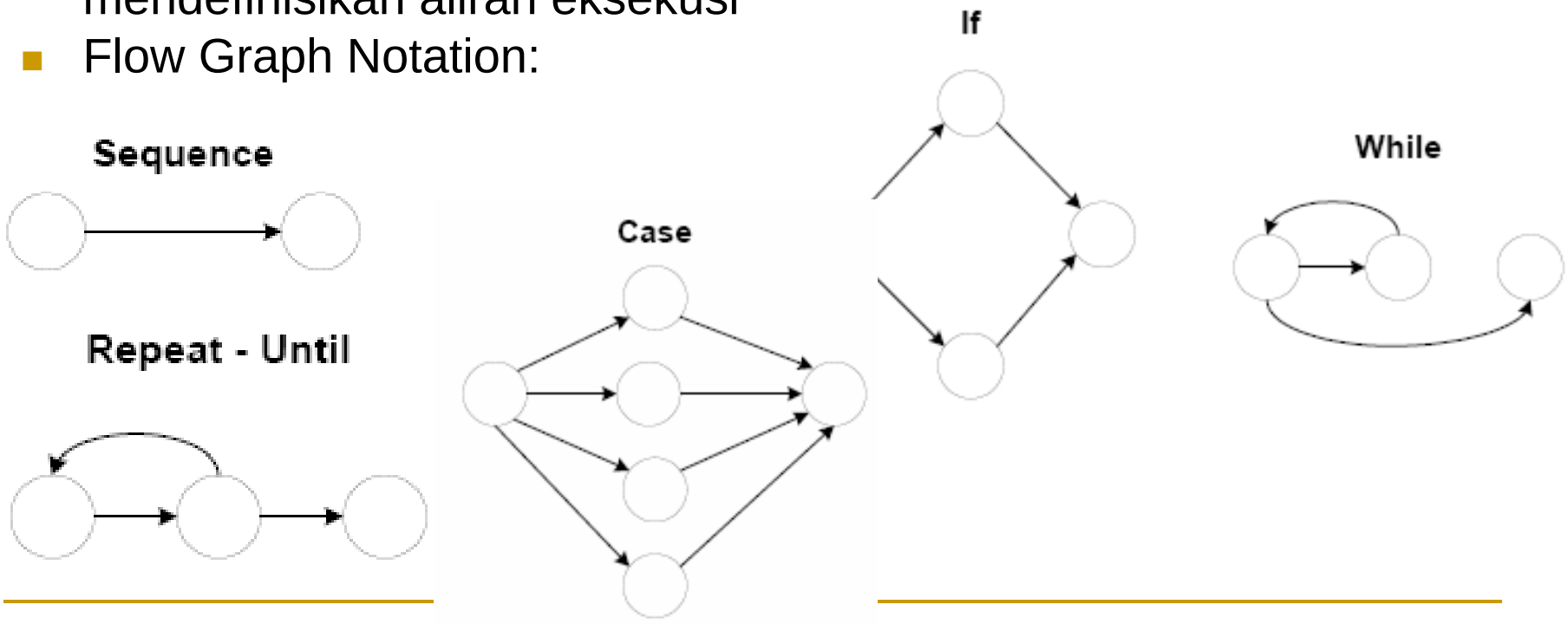
- White Box Testing atau Glass Box Testing
  - Metode mendesain kasus uji dengan menggunakan struktur kontrol dari desain prosedural (fungsi-fungsi) untuk membuat kasus uji
  - SW Engineer harus dapat membuat kasus uji yang menjamin semua aliran modul yang independen minimal diuji sekali
  - Menguji semua putusan lojik *(logical decisions)* pada ruang lingkup yang benarnya dan yang salah
  - Menguji semua perulangan *(looping)* dalam ruang lingkupnya dan ruang operasinya
  - Menguji struktur data internal untuk memastikan validitasnya

Pendahuluan

- Mengapa tidak hanya menguji spesifikasi?
  - Kesalahan lojik (*logic errors*) dan asumsi yang salah berbanding terbalik dengan kemungkinan dieksekusinya sebuah alur program dengan baik
  - Kita sering berpikir bahwa sebuah aliran lojik *(logical path)* yang jarang dieksekusi tidak penting padahal sebenarnya proses itu sering dieksekusi
  - Penampilan letak kesalahan (*typographical errors*) biasanya dilakukan secara random

Basis Path/Control Structure Testing

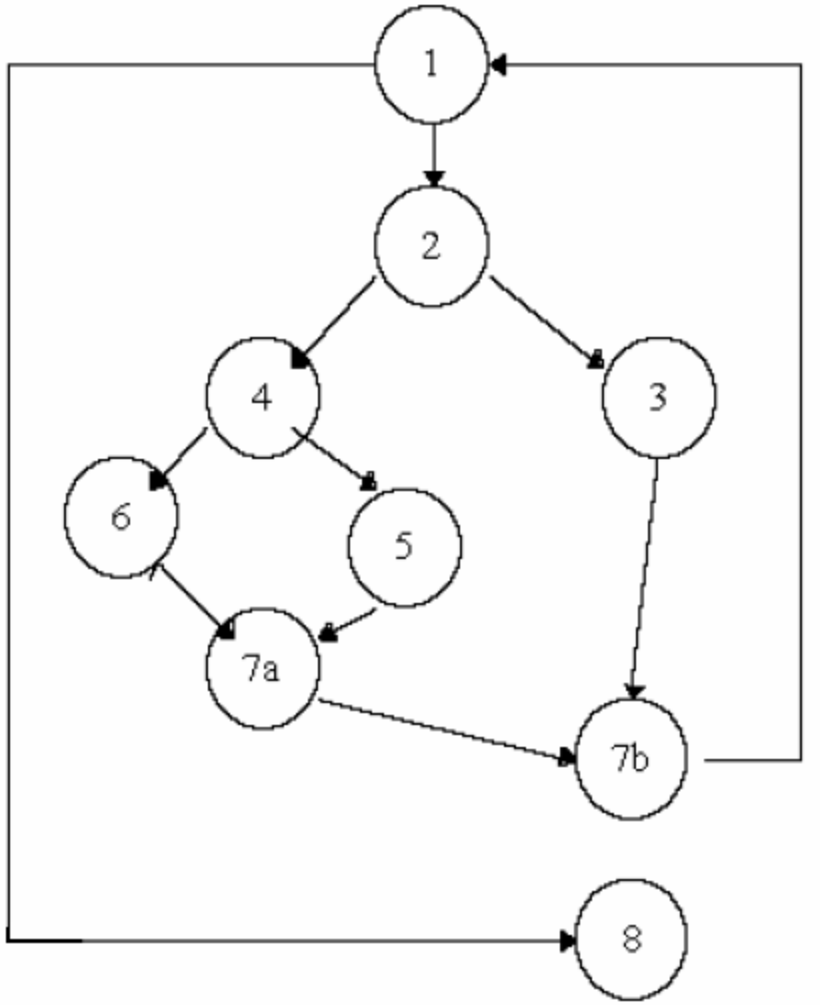
- Diperkenalkan oleh Tom McCabe
- Metode Basis Path mengijinkan pendesain kasus uji untuk membuat perkiraan lojik yang kompleks dari desain prosedural (fungsi-fungsi) dan menggunakan perkiraan ini untuk mendefinisikan aliran eksekusi
- Flow Graph Notation:



Graph Matrices

- Dapat secara otomatis menggenerasi *flow graph* dan mendeterminasi aliran dasar proses
- Dapat menggunakan *graph matrix* sebagai *tools*
- *Graph matrix*:
  - adalah bujursangkar dengan #sisi yang merepresentasikan #simpul
  - baris dan kolom merepresentasikan simpul
  - nilai 1 merepresentasikan adanya keterhubungan antar simpul

Graph Matrices



	1	2	3	4	5	6	7a	7b	8
1		1							1
2			1	1					
3								1	
4					1	1			
5							1		
6							1		
7a								1	
7b	1								
8									

Connection Matrix

Condition Testing (1)

- Condition Testing bertujuan untuk mengeksekusi semua kondisi lojik dari sebuah modul program
- Dapat mendefinisikan
  - Relational Expression (E1 op E2) : dimana E1 dan E2 adalah *arithmetic expression*
  - Simple Condition: variabel boolean atau *relational expression*, mungkin diawali dengan operator NOT
  - Compound condition: terdiri dari dua atau lebih *simple conditions,* operator boolean, dan tanda kurung
  - Boolean Expression: kondisi tanpa *relational expression*
- Tipe kesalahan pada sebuah kondisi dapat mencakup:
  - boolean operator error (adanya incorrect/missing/extra boolean operator)
  - boolean variable error
  - boolean parenthesis (tanda kurung) error
  - relational operator error
  - arithmetic expression error

Condition Testing (2)

- Metode c*ondition testing* fokus pada pengujian setiap kondisi yang ada pada program
- Keuntungan strategi condition testing
  - Memperkirakan pengujian berdasarkan kondisi adalah hal yang simpel
  - Cakupan pengujian dapat mengarahkan pada penambahan kasus uji untuk sebuah program
- Beberapa strategi yang termasuk *condition testing*
  - Branch Testing
  - Domain Testing
  - Branch and Relational Operator Testing – menggunakan kondisi sebagai batasan
- Contoh: C1 = B1 & B2
  - dimana B1, B2 adalah *boolean conditions*
  - Batasan kondisi D1, D2 dimana D1 dan D2 dapat bernilai *true* (t) atau *false* (f)
  - Cabang dan operator relasi membutuhkan batasan-batasan { (t,t), (f,t), (t,f) } harus dipenuhi untuk eksekusi C1
- Cakupan batasan-batasan menjamin pendekteksian kesalahan pada operator relasional (*relational operator errors*)

Branch Testing

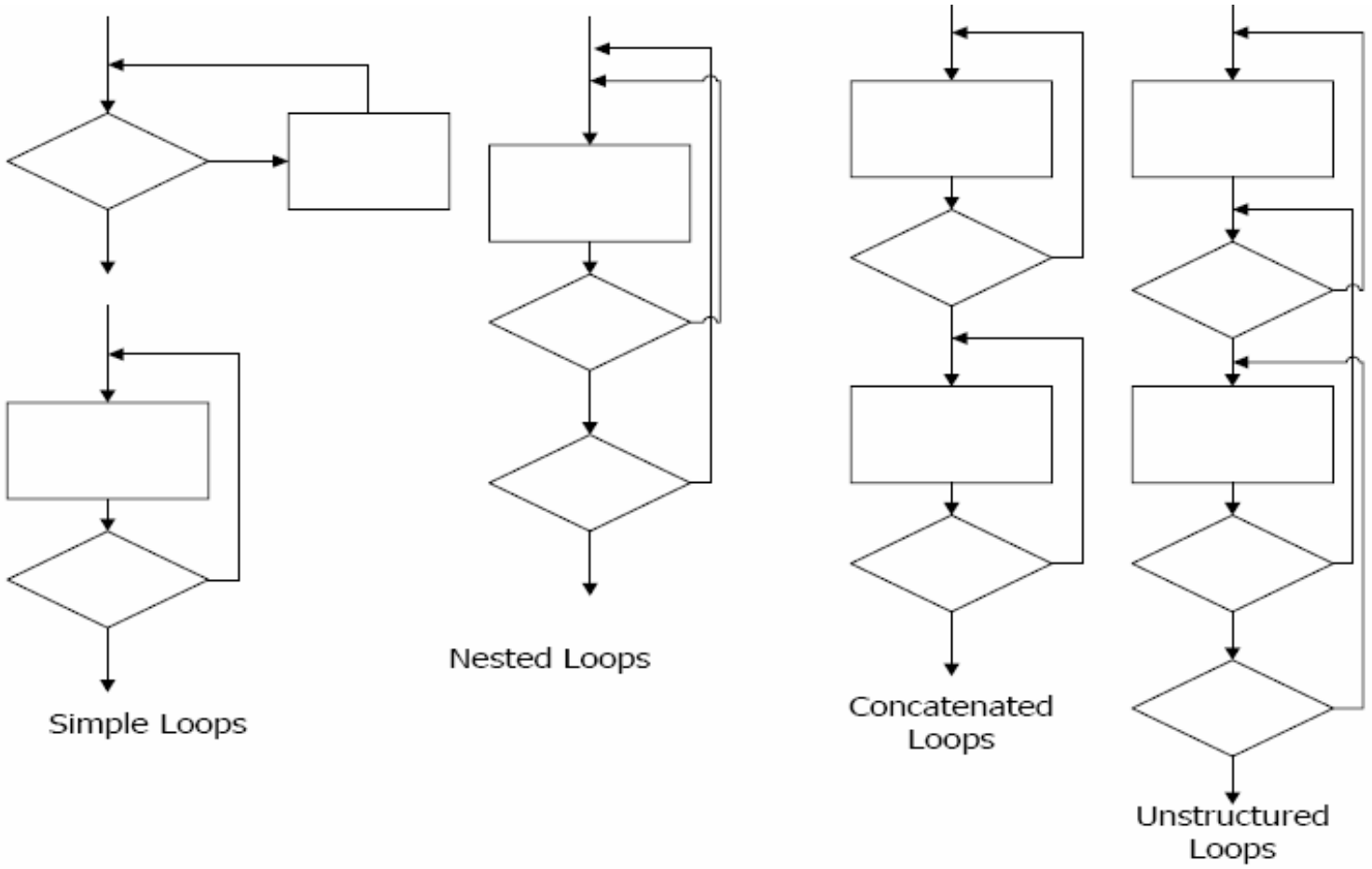
- Branch testing adalah strategi pengujian berbasis kondisi yang paling simpel
- Untuk setiap kondisi percabangan C, maka setiap cabangnya harus dieksekusi minimal sekali (yang bernilai *true* atau *false*)

Data Flow Testing

- Metode Data Flow testing memilih aliran tes dari program berdasarkan lokasi pendefinisian dan penggunaan variabel dalam program
- dengan Data flow testing
  - Setiap statemen di dalam program diasumsikan diisi dengan angka yang unik dan diasumsikan tidak ada fungsi yang mengubahnya
- Defs(s) = { x | statemen S berisi pendefinisian X }
- Use(s) = { x | statemen S berisi penggunaan X }
- DU Chain (Definition - Use Chain) dari variabel X dengan bentuk {X, S, S'}, dimana S, S' adalah jumlah statemen, X di dalam Defs(S) dan Defs(S').
- Rangkaian yang mungkin lainnya:
  - DD (Definition-Definition Chain) – harus dihindari
  - UU (Use-Use Chain) - common chain
  - UD (Use-Definition Chain) - common chain
- Salah satu strategi yang simpel adalah Strategi DU Testing
  - Strategi ini menyatakan bahwa setiap DU chain harus dilalui minimal sekali

Loop Testing

- Loop adalah dasar dari banyak algoritma.
- Loop dapat didefinisikan menjadi berikut:



Loop Testing (2)

Untuk menguji:

- **Simple Loops** dengan n kali:
  - Abaikan perulangan
  - Masuki perulangan sekali
  - Masuki perulangan dua kali
  - Masuki perulangan m kali dimana m<n.
  - Masuki perulangan (n-1), n, dan (n+1) kali
- **Nested Loops**
  - Mulai dengan perulangan yang ada di dalam, set semua perulangan ke nilai minimum
  - Uji sebagai *simple loop* untuk perulangan yang ada di dalam
  - Pengujian dari dalam ke luar
  - Lanjutkan sampai semua loop diuji
- **Concatenated Loops**
  - Jika perulangan independen, gunakan pengujian *simple loop*
  - Jika perulangan tidak saling independen, gunakan nested loops.
- **Unstructured loops**
  - Jangan diuji – harus didesain ulang