Analisis dan Sistem Informasi
Pengujian Perangkat Lunak

Teknik Informatika
UNIKOM

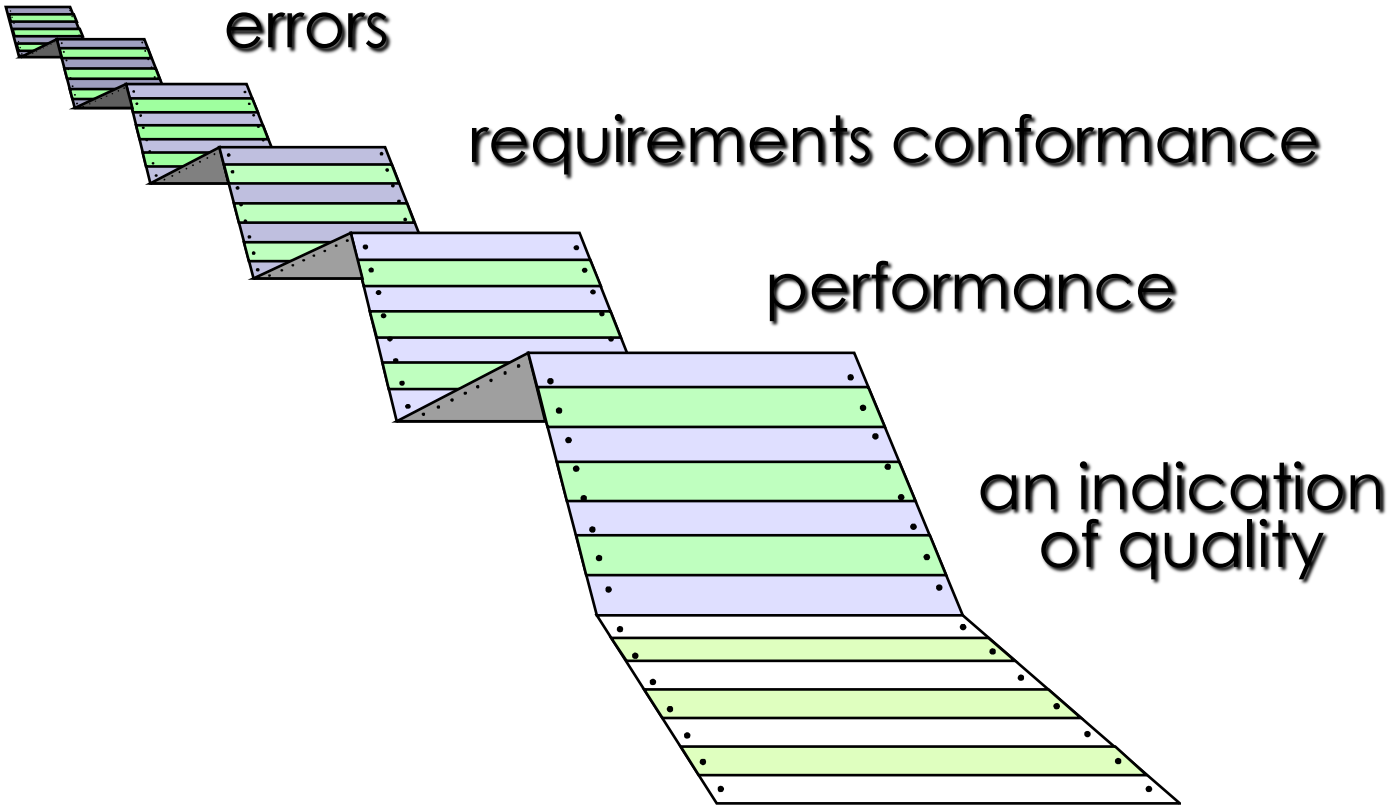
Pengujian Perangkat Lunak

1. Definisi pengujian perangkat lunak
2. Strategi pengujian perangkat lunak
3. Metode pengujian perangkat lunak
4. Pengujian white box
5. Teknik pengujian white box
6. Pengujian black box
7. Teknik pengujian black box

Definisi Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses menelusuri dan mempelajari sebuah program dalam rangka menemukan kesalahan pada perangkat lunak sebelum diserahkan kepada end user.

Apa yang dimunculkan pada pengujian?



Tujuan Pengujian Perangkat Lunak (1)

1. Pengujian adalah proses menjalankan program dengan maksud untuk mencari kesalahan (*error*)
2. Kasus uji yang baik adalah kasus yang memiliki peluang untuk mendapatkan kesalahan yang belum ketahuan
3. Pengujian dikatakan berhasil bila dapat memunculkan kesalahan yang belum ketahuan

Tujuan Pengujian Perangkat Lunak (2)

4. Jadi, pengujian yang baik bukan untuk memastikan tidak ada kesalahan tetapi untuk mencari sebanyak mungkin kesalahan yang ada di program
5. Pengujian tidak dapat menunjukkan ke-tidak-hadir-an *defect*, pengujian hanya menunjukkan bahwa kesalahan perangkat lunak ***ada***.

Siapa yang melakukan pengujian?

Developer

Memahami sistem yang dibangunakan tetapi pengujiannya dilakukan dengan “HALUS” dan terkadang terburu-buru (karena batas waktu deliver)

Tester Independent

Butuh waktu untuk mempelajari sistem tetapi pengujiannya didasarkan pada kualitas dan tidak terburu-buru

Validation VS Verification

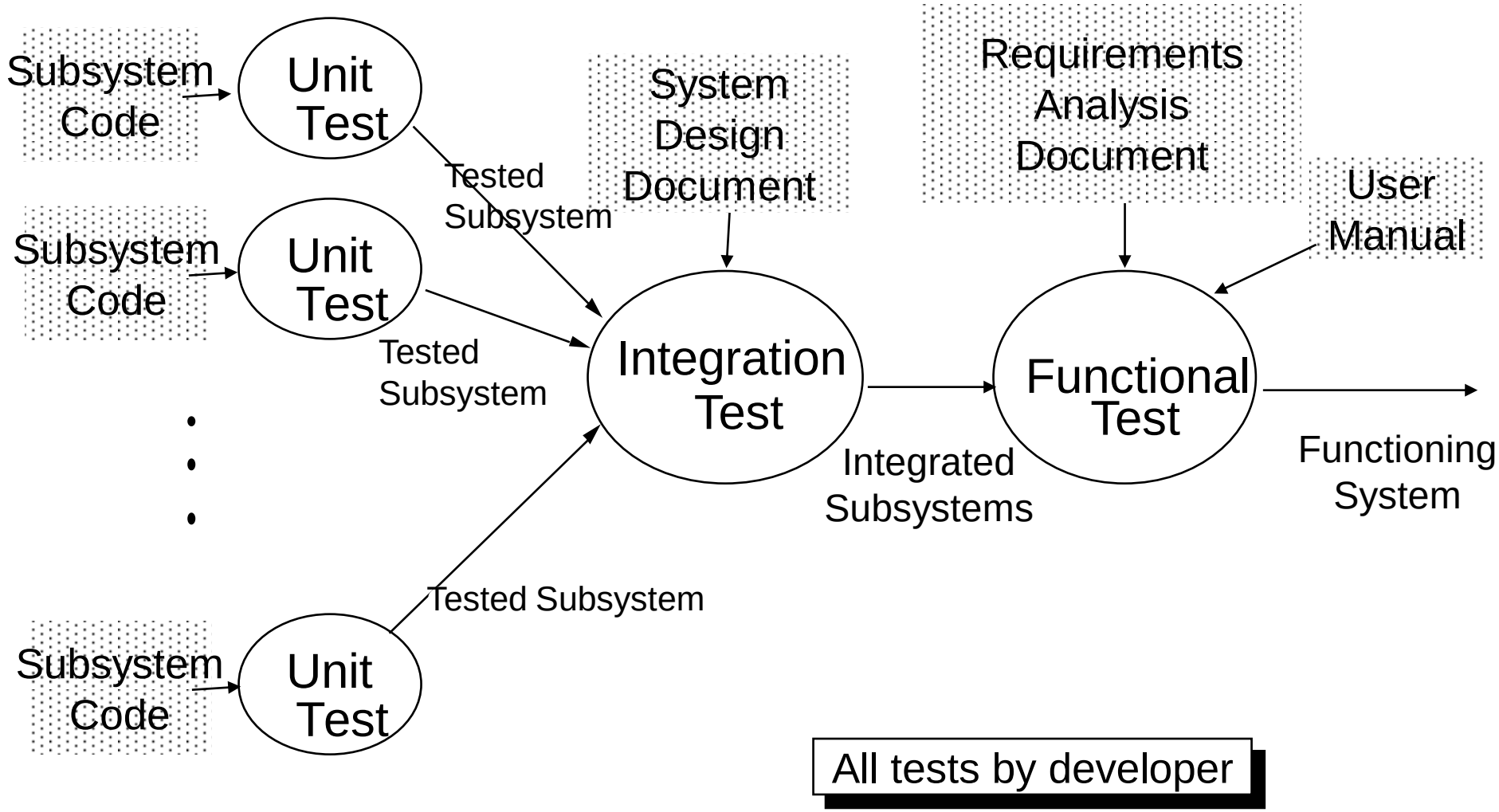
Verification – Apakah proses pembangunan produk dengan benar?

Apakah kode sudah dibuat sesuai dengan spesifikasinya?

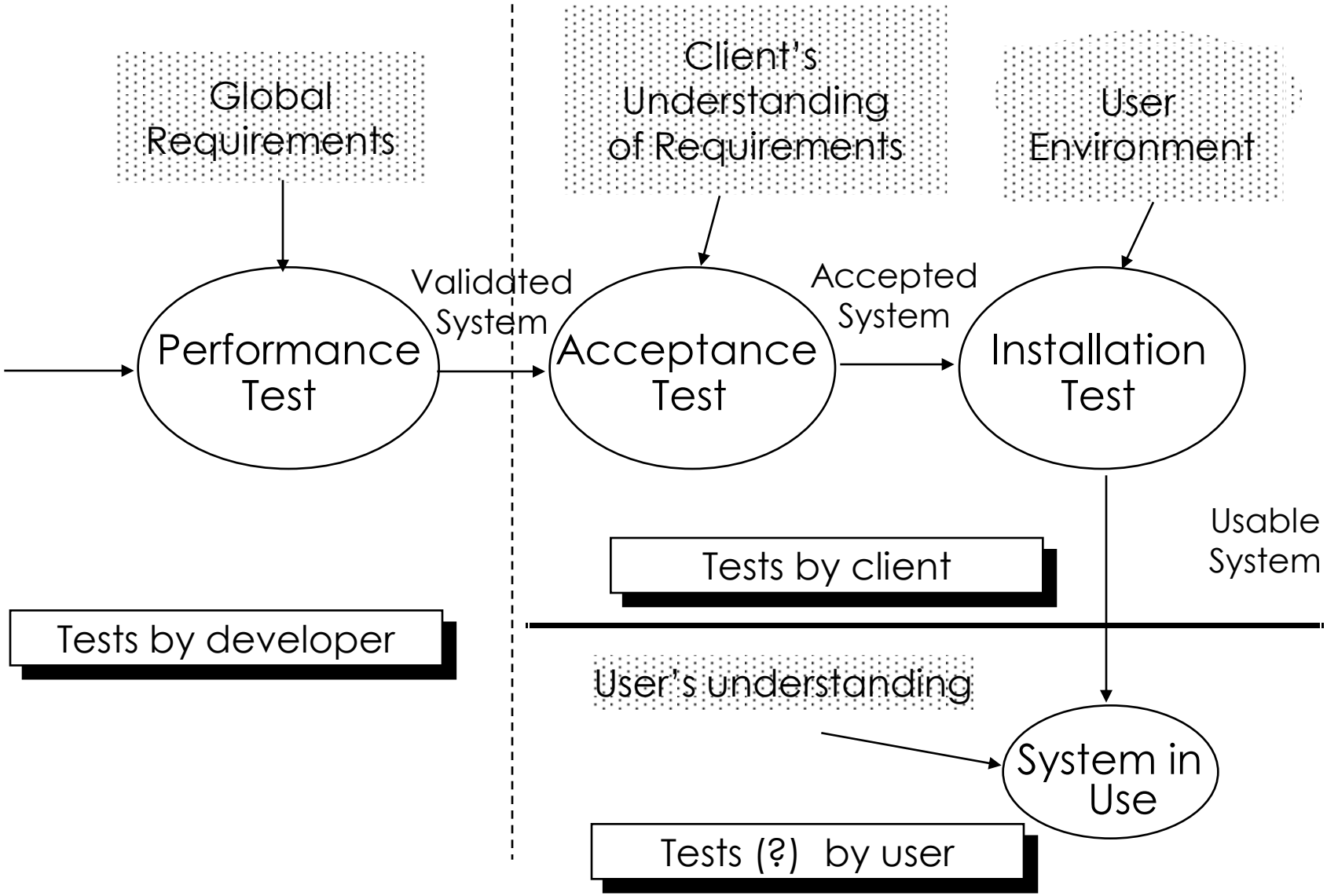
Validation – Apakah produk yang dibangun benar?

Apakah spesifikasi sesuai dengan kebutuhan di awal?

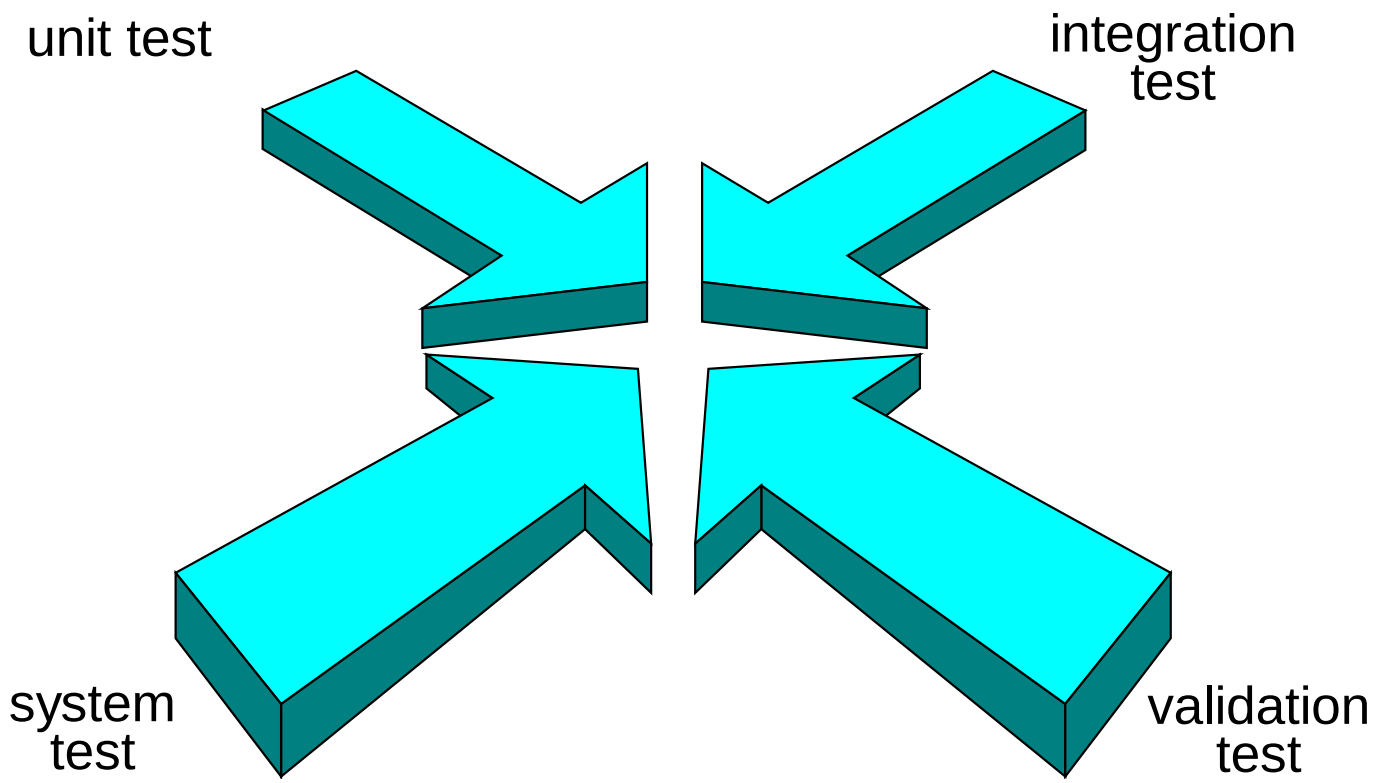
Aktivitas Pengujian (1)



Aktivitas Pengujian (2)



Strategi Pengujian (1)

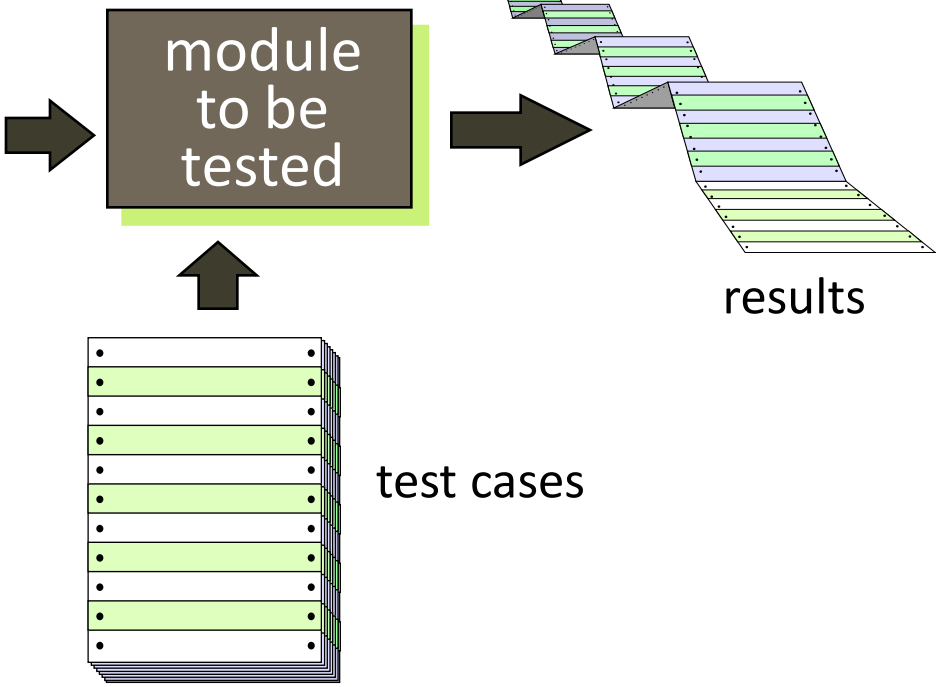


Strategi Pengujian (2)

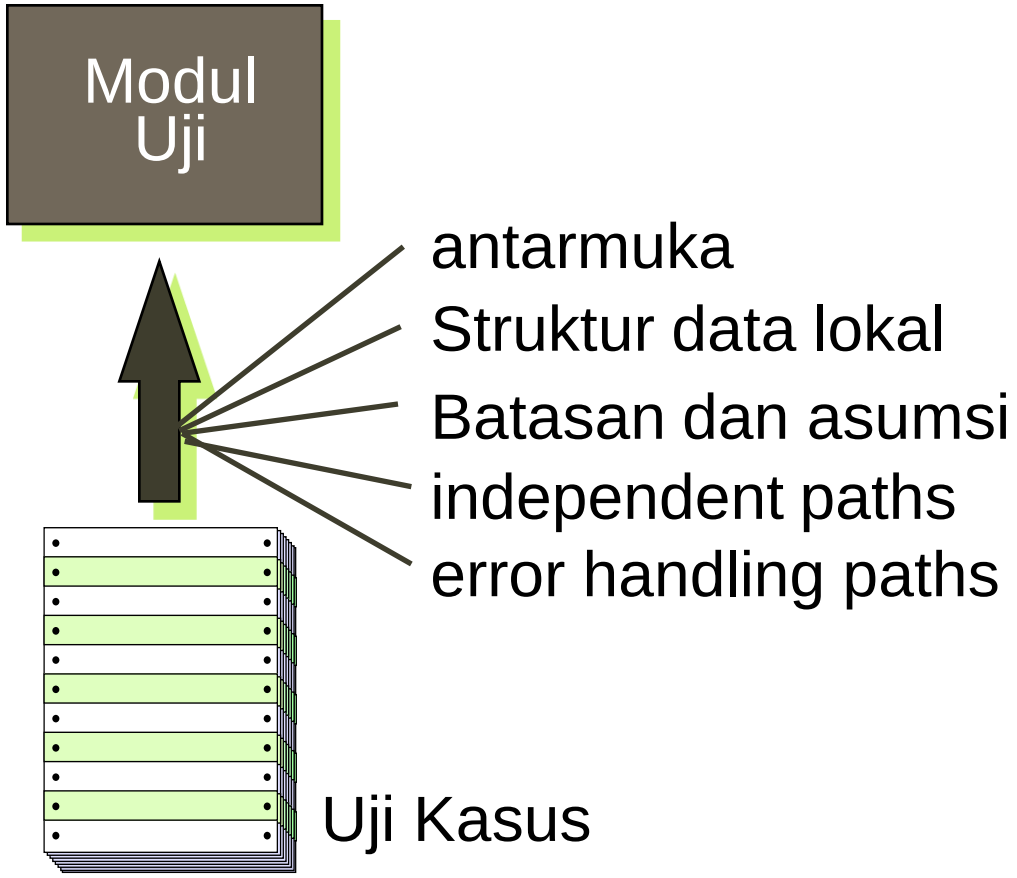
1. Dimulai dengan unit testing dan diakhiri dengan system testing.
2. **Unit testing:** pengujian komponen individual (modul di pemrograman prosedural atau class di OOP).
3. **Integration testing:** pengujian terhadap koleksi dari komponen-komponen yang bekerja bersamaan.
4. **Validation testing:** pengujian aplikasi terhadap kebutuhan pengguna.
5. **System testing:** pengujian aplikasi secara keseluruhan.

Unit Testing (1)

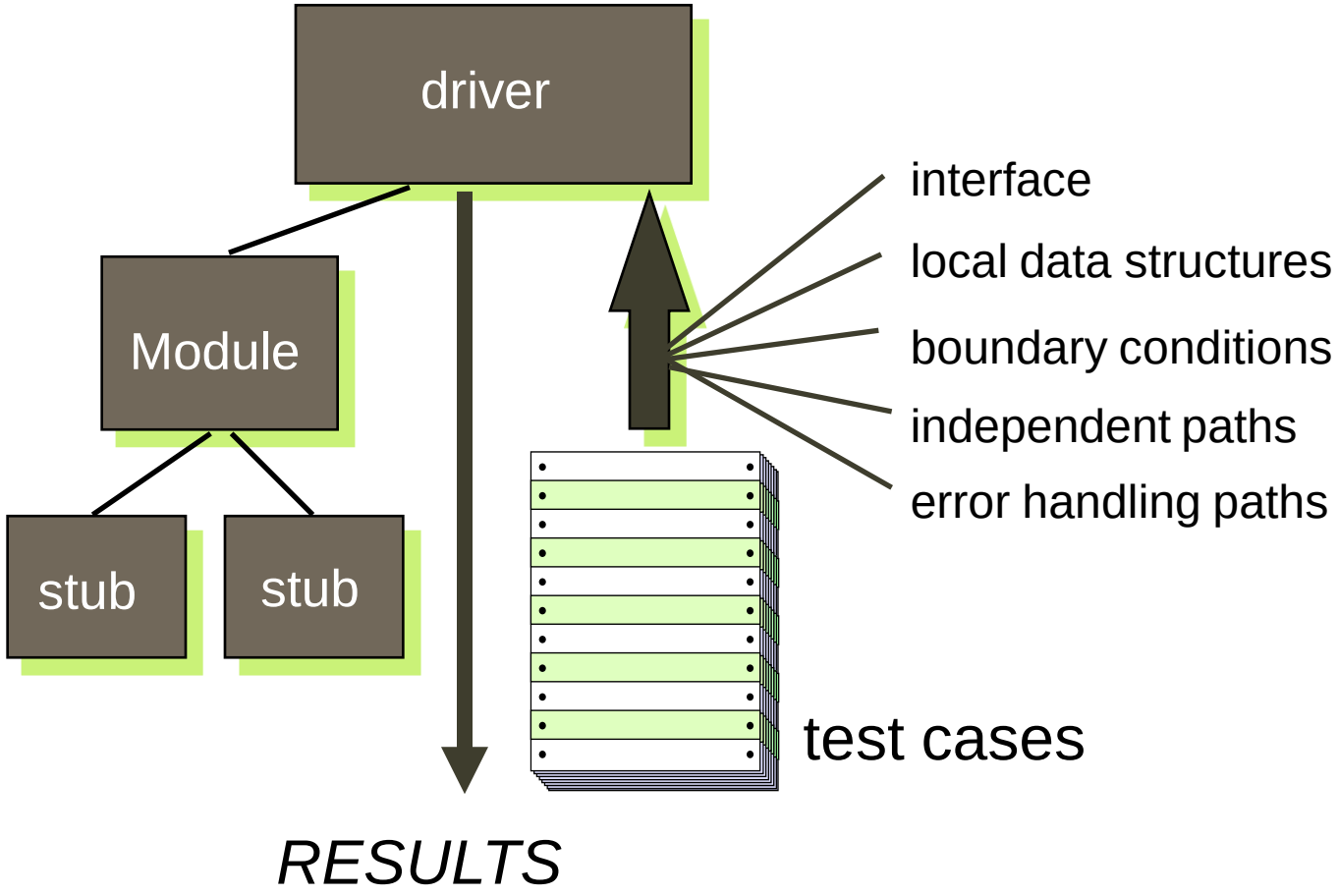
software engineer



Unit Testing (2)



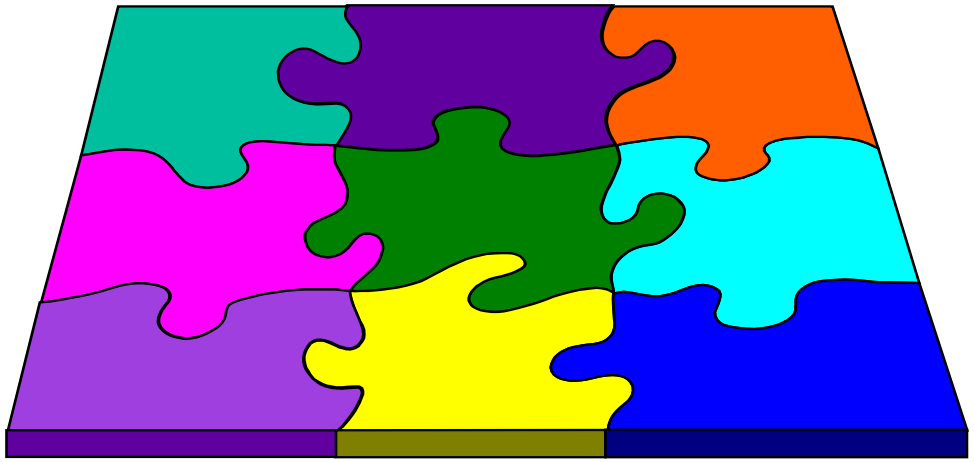
Unit Testing Environment



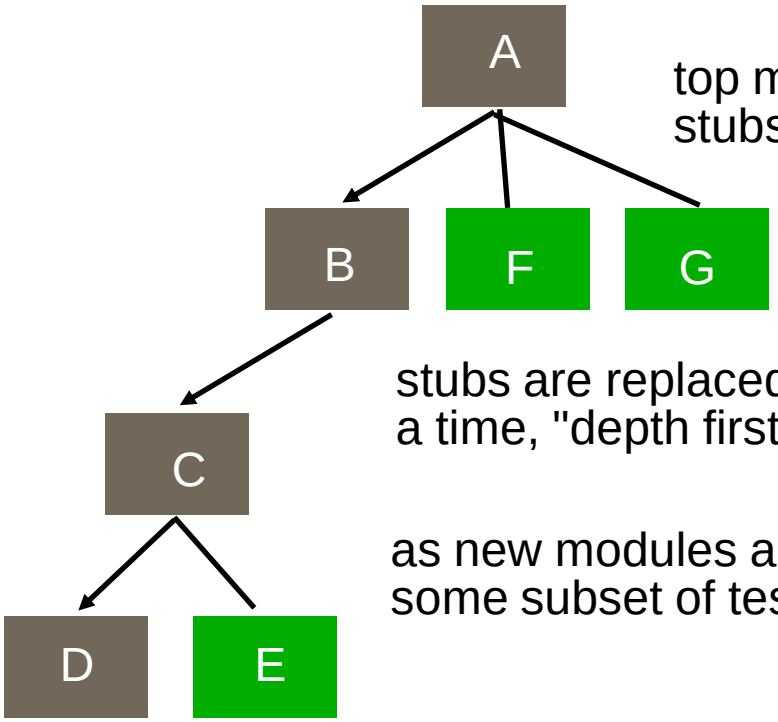
Integration Testing

Pilihan:

- Pendekatan “big bang”
- Strategi pembangunan incremental



Top Down Integration

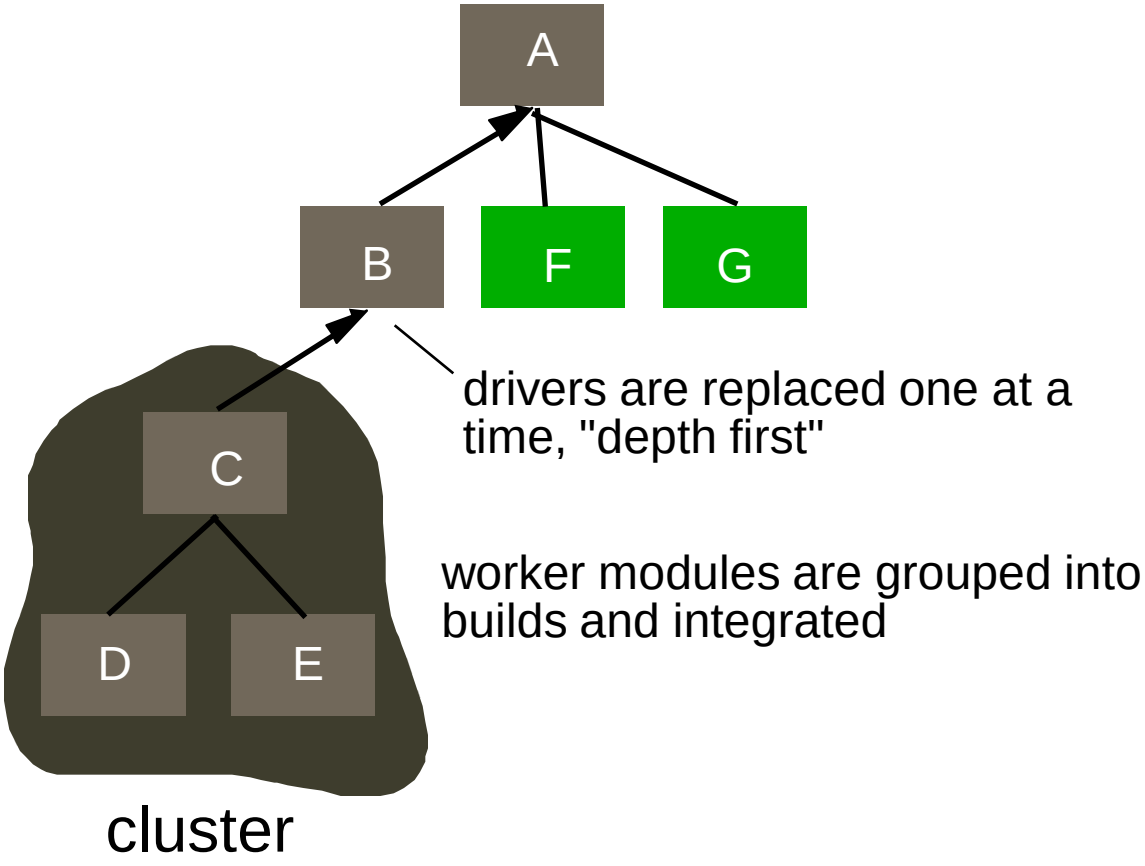


top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

Bottom Up Integration



Pengujian Regresi

1. Pengujian ulang yang dilakukan secara selektif dari sistem yang sudah dimodifikasi untuk meyakinkan tidak ada lagi bug yang timbul pada tahap modifikasi.
2. Alasan: memperbaiki satu bagian kode akan mempengaruhi yang lain.

High Order Testing (1)

- **Validation testing**

  Fokus terhadap kebutuhan perangkat lunak

- **System testing**

  Fokus terhadap integrasi sistem.

- **Alpha/Beta testing**

  Fokus terhadap keberdayagunaan.

- **Recovery testing**

  Membuat perangkat lunaknya menjadi gagal dalam menjalankan fungsinya dan perangkat lunak dapat menginformasikan hal-hal yang menyebabkan kegagalan tersebut.

High Order Testing (2)

- **Security testing**

  Memverifikasi mekanisme proteksi ke dalam sistem dalam rangka melindungi sistem dari penetrasi pihak luar

- **Stress testing**

  Mengeksekusi sistem dalam sebuah uji yang melebihi kemampuan sistem dalam rangka menguji kehandalan sistem.

- **Performance Testing**

  Menguji perfomansi run-time dari perangkat lunak sebagai sistem yang terintegrasi.

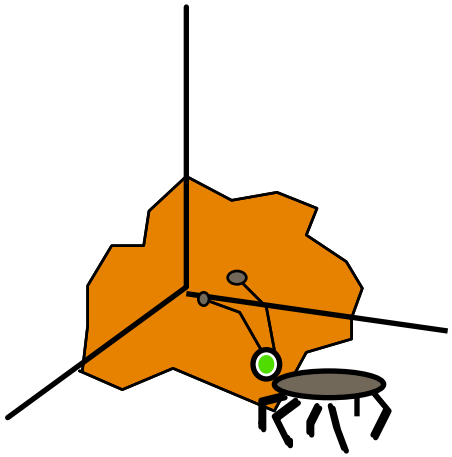
‘GOOD’ Test

Pengujian dikatakan sebagai good test jika mempunyai probabilitas yang tinggi dalam menemukan kesalahan dalam suatu perangkat lunak.

Perancangan Uji Kasus

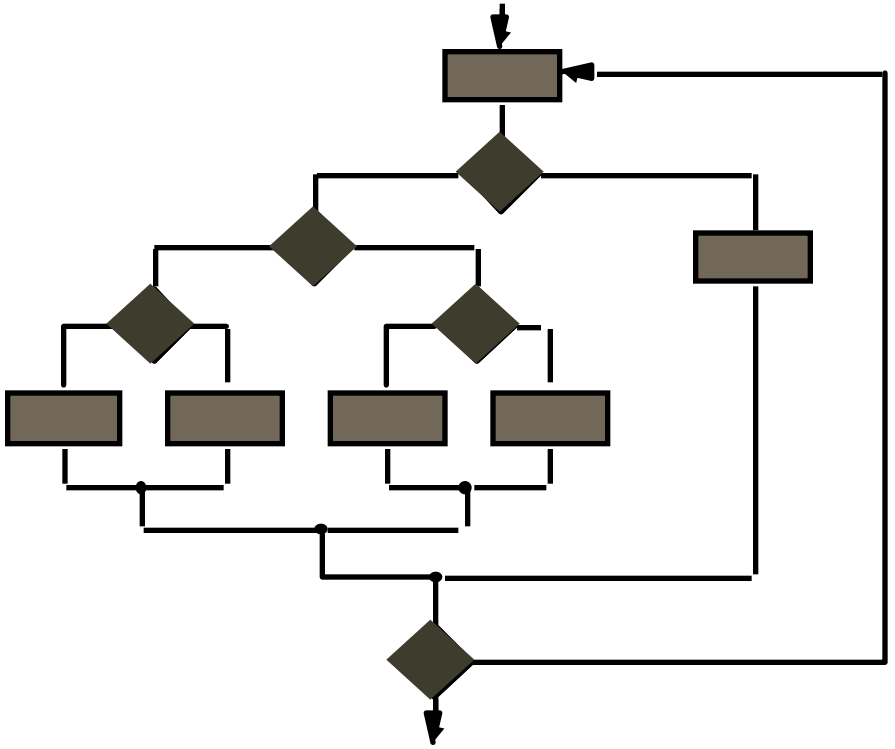
"Bugs lurk in corners and congregate at boundaries ..."

Boris Beizer



OBJECTIVE	to uncover errors
CRITERIA	in a complete manner
CONSTRAINT	with a minimum of effort and time

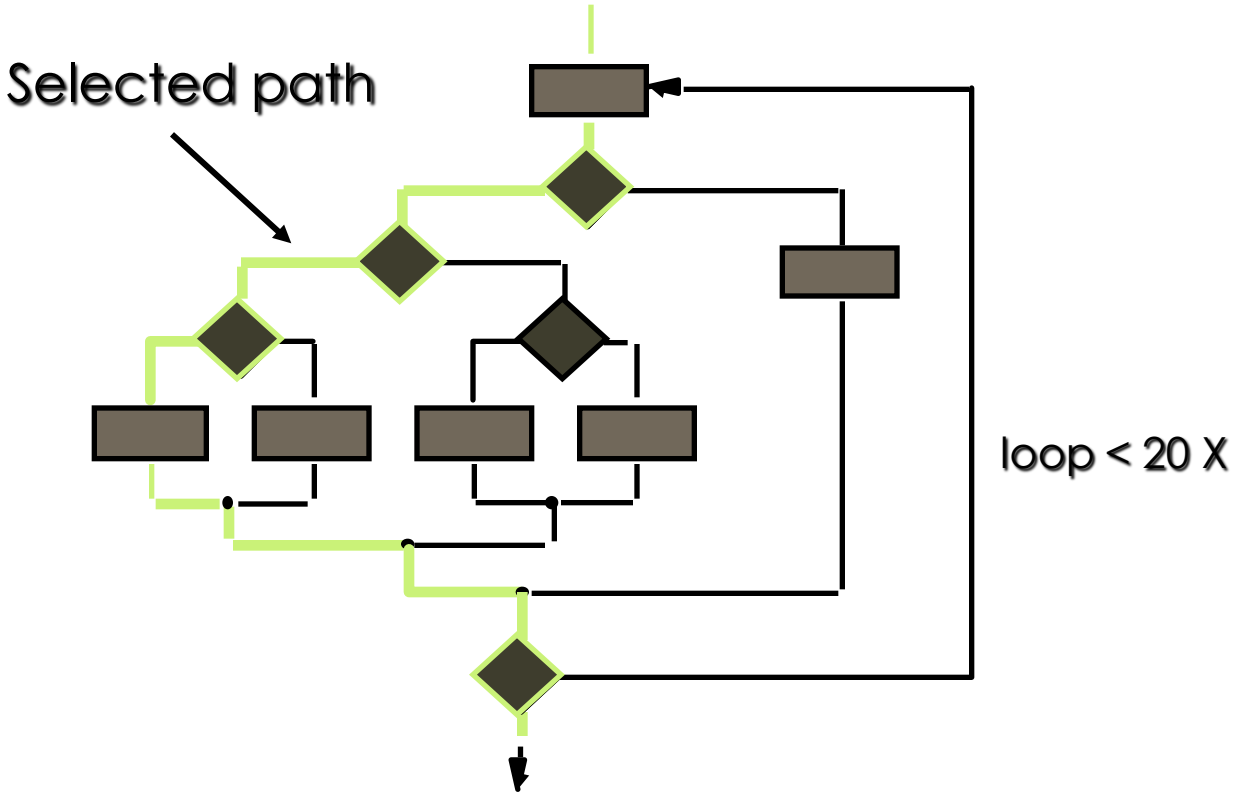
Exhausting Testing



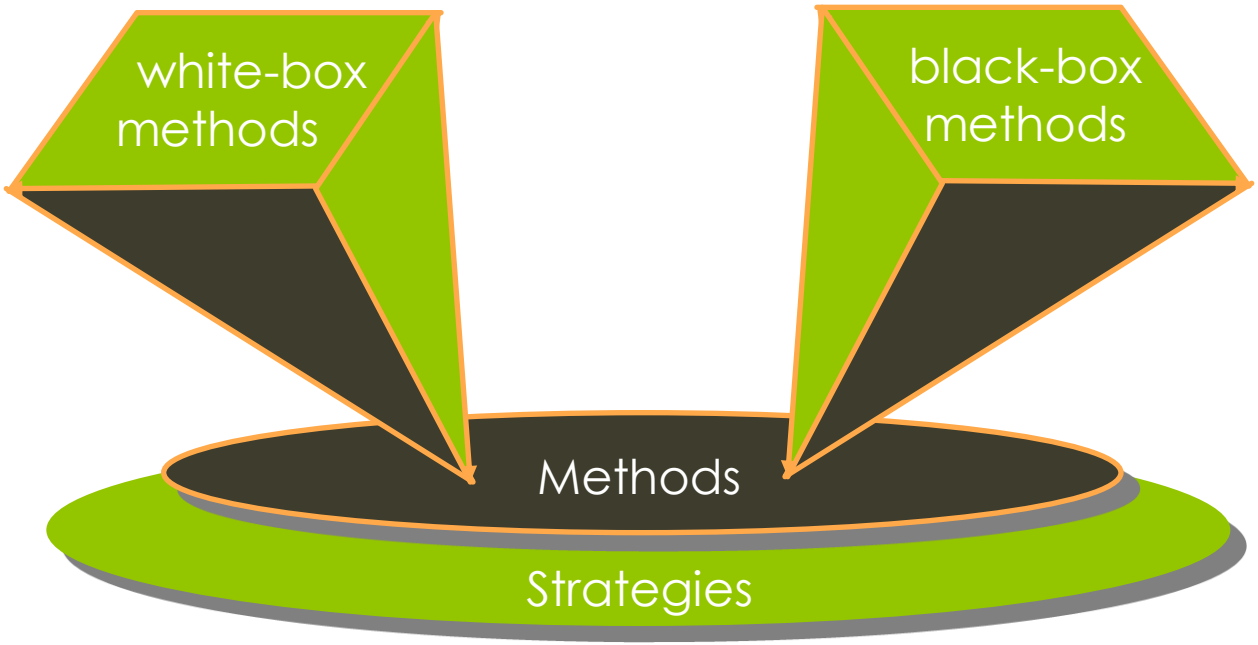
loop < 20 X

There are 10^{14} possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

Selective Testing

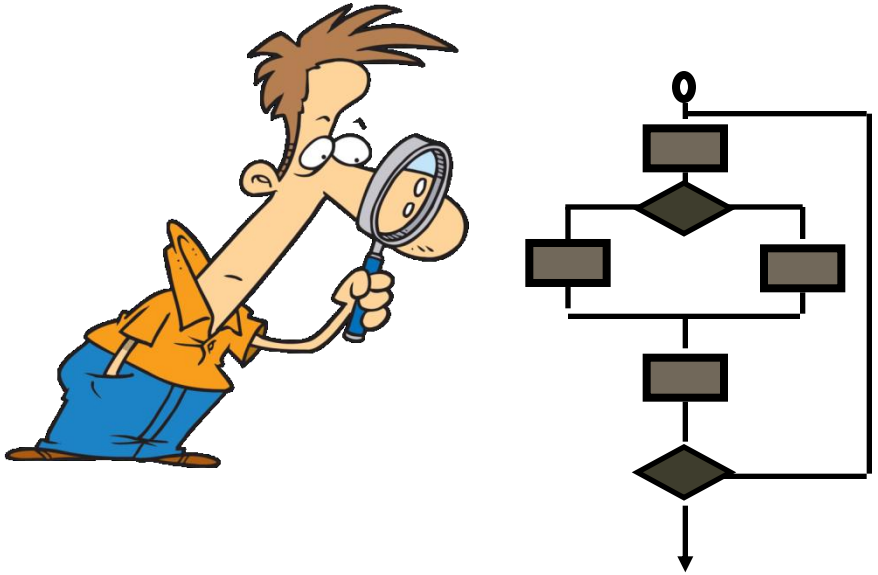


Pengujian Perangkat Lunak



PENGUJIAN WHITE BOX

Pengujian White Box



Tujuannya adalah untuk meyakinkan semua perintah dan kondisi dieksekusi minimal sekali

Kenapa Pengujian White Box?

1. Adanya kesalahan logik dan asumsi yang tidak tepat pada setiap kemungkinan eksekusi.
2. Ada kemungkinan alur program yang tidak tereksekusi.
3. Ada kemungkinan kesalahan typography yang sulit ditemukan kalau tidak dijalankan.

Teknik Pengujian White Box

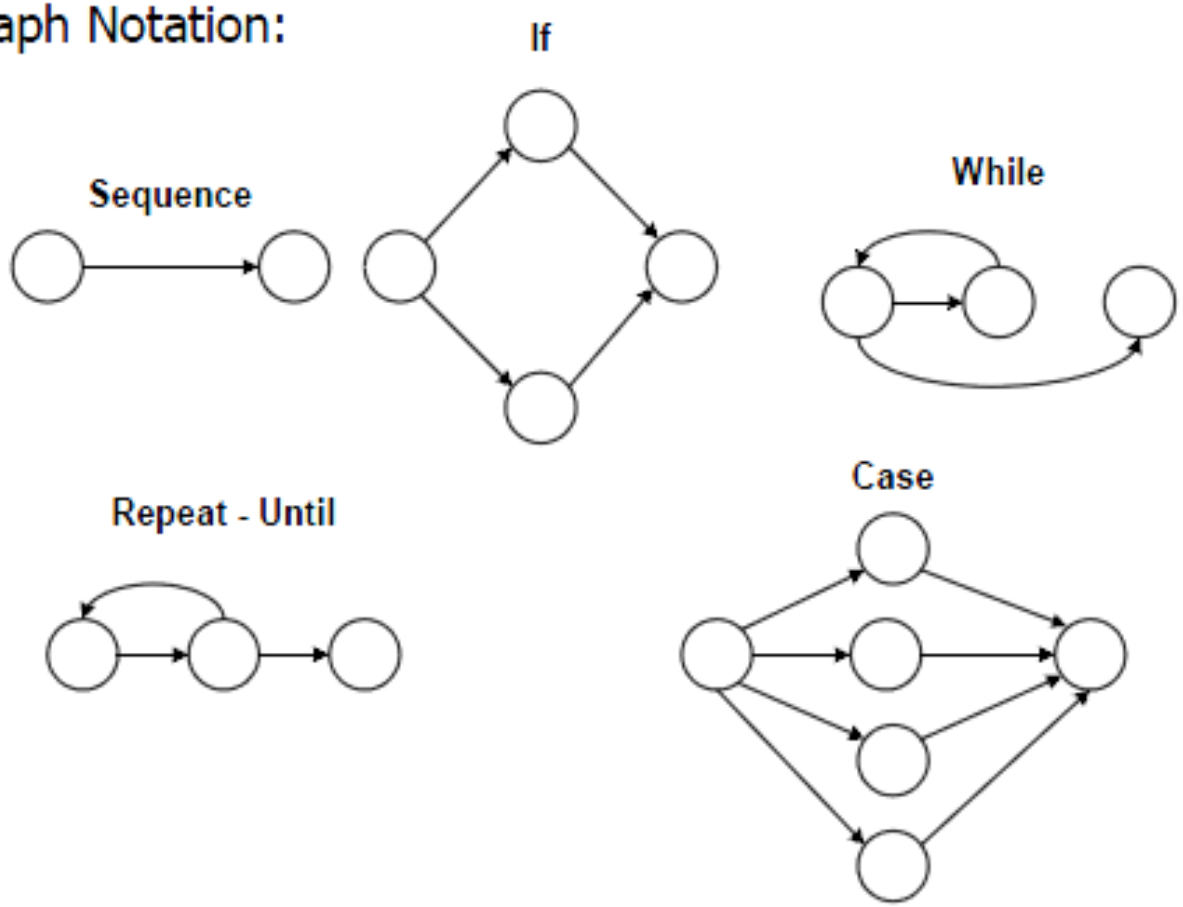
1. Basis Path Testing
2. Control Structure Testing

Basic Path Testing (1)

1. Dibuat oleh Tom McCabe
2. Metode basic path memungkinkan perancang uji kasus untuk memperoleh ukuran kompleksitas logis dari sebuah perancangan prosedural dan menggunakan ukuran ini sebagai sebuah panduan dalam mendefinisikan sekumpulan dasar dari alur eksekusi.

Basic Path Testing (2)

Flow Graph Notation:

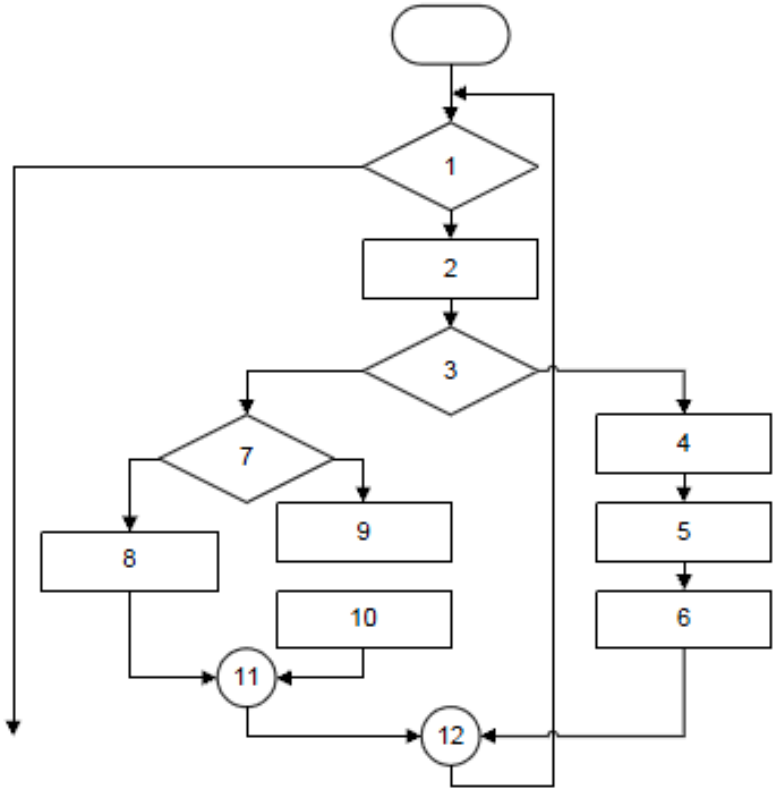


Basic Path Testing (3)

- Procedure Sort

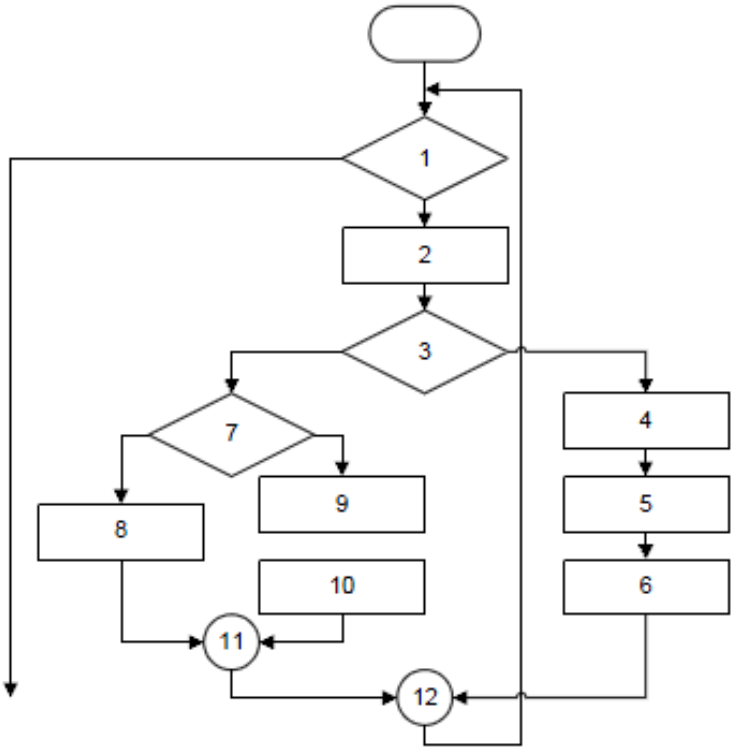
```
   Procedure Sort
1. do while not eof
2.    Read Record
3.    if record field 1 = 0
4.    then process record
5.         store in buffer;
6.         increment counter
7.    else if record field 2 = 0
8.         then reset counter
9.         else process record
10.             store in file
11.        endif
12.   endif
13. enddo
```

- Flow Chart

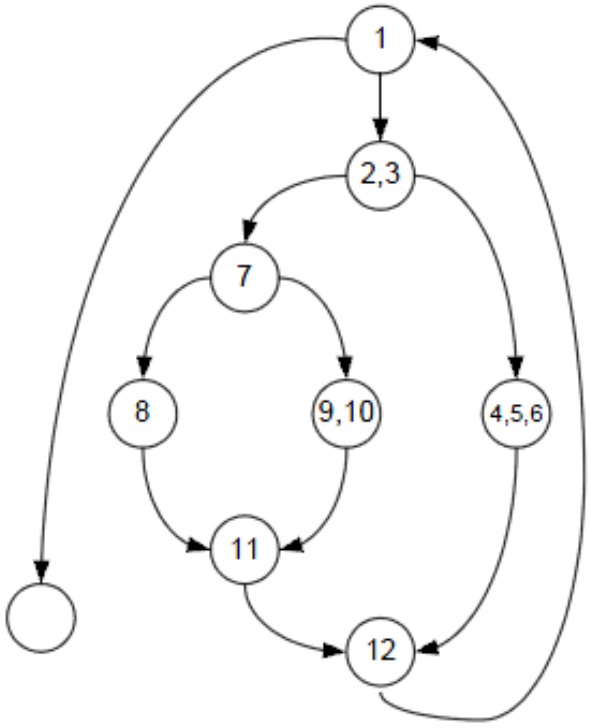


Basic Path Testing (4)

- Flow Chart

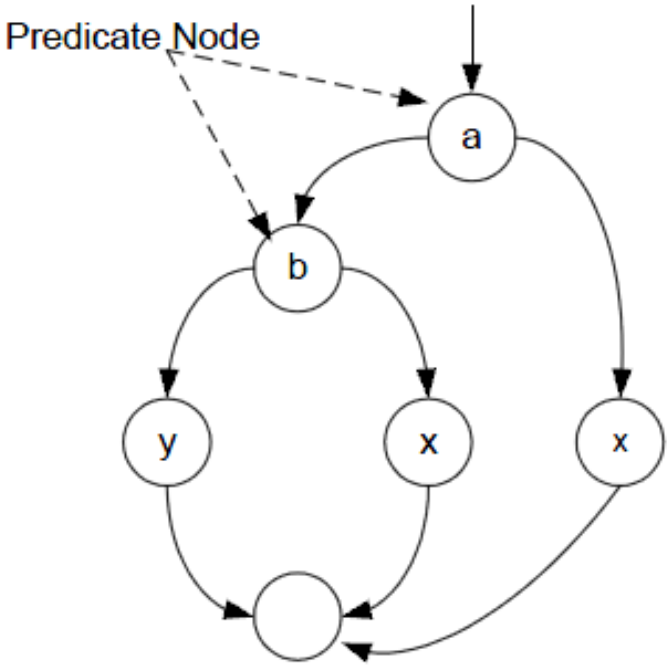


- Flow Graph



Compound Logic

```
IF a or  b
then procedure X
else procedure Y
endif
```



Cyclomatic Complexity (V(G)) (1)

Software metric yang mengembangkan ukuran secara

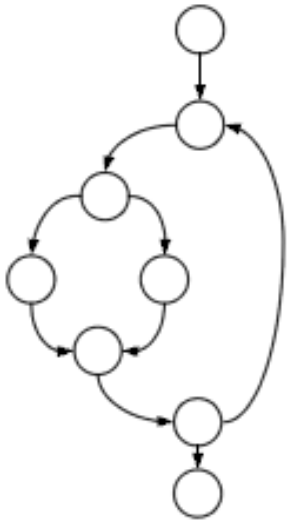
kuantitatif dari sebuah kompleksitas logik program.

V(G) = E – N + 2

V(G) = 9 – 8 + 2 = 3

E = Jumlah busur pada flow graph

N = Jumlah simpul pada flow graph



Cyclomatic Complexity (V(G)) (2)

Atau:

Rumus V(G) = P + 1 dimana P adalah jumlah dari predicate node.

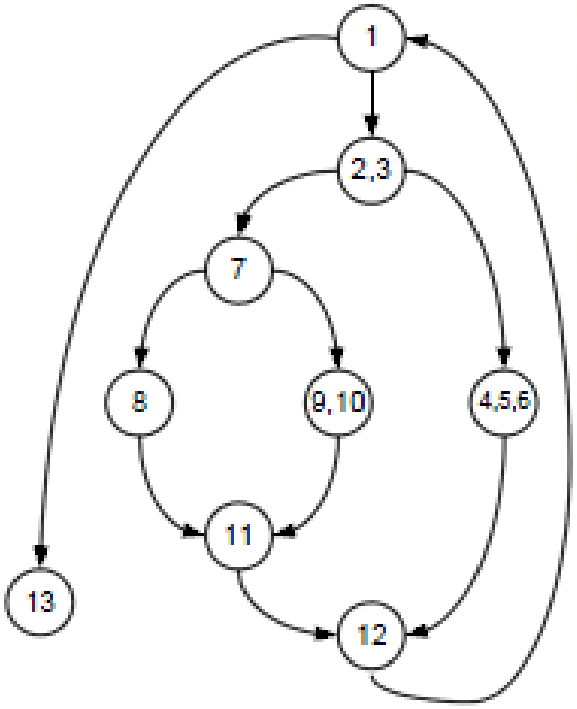
Independent Path (1)

1. Sebuah independent path adalah jalur di dalam program yang memperkenalkan setidaknya satu set pernyataan atau kondisi baru.
2. Sebuah independent path harus bergerak setidaknya sepanjang satu sisi yang belum ditelusuri sebelum jalur tersebut didefinisikan.

Independent Path (1)

1. Sebuah independent path adalah jalur di dalam program yang memperkenalkan setidaknya satu set pernyataan atau kondisi baru.
2. Sebuah independent path harus bergerak setidaknya sepanjang satu sisi yang belum ditelusuri sebelum jalur tersebut didefinisikan.

Independent Path (2)



path 1: 1-13

path 2: 1-2-3-7-8-11-12-1-13

path 3: 1-2-3-7-9-10-11-12-1-13

path 4: 1-2-3-4-5-6-12-1-13

Is the path

1-2-3-4-5-6-12-1-2-3-7-8-11-12-1-13

an independent path ?

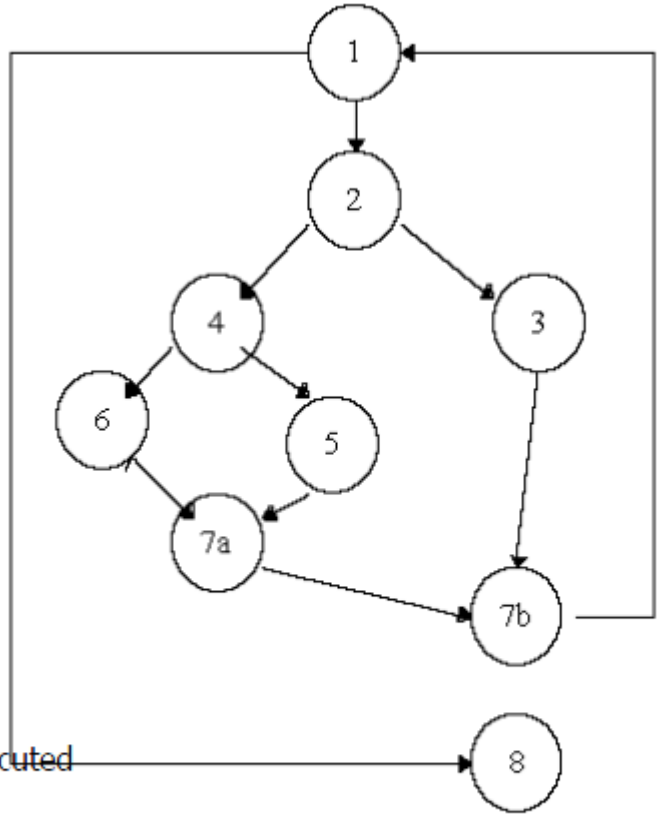
Membuat Uji Kasus

1. Gambar sebuah flowgraph yang diadopsi dari perancangan atau kode sebagai dasar utama.
2. Carilah cyclomatic complexity dari flow graph (V(G)).
3. Carilah sekumpulan dasar dari independent path secara linear.
4. Persiapkan uji kasus yang mengeksekusi path di dalam kumpulan tersebut.
5. Setiap uji kasus mendefinisikan kondisi masukan dan hasil yang diinginkan.

Matriks Graph (2)

	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



Some other interesting link weights:

- Probability that a link (edge) will be executed
- Processing time for traversal of a link
- Memory required during traversal of a link
- Resources required during traversal of a link

Branch Testing

1. Branch testing adalah strategi pengujian yang paling sederhana.
2. Untuk kondisi majemuk C yang benar dan salah, cabang-cabang dari C dan setiap kondisi sederhana dalam C harus dieksekusi minimal sekali.

Control Structure Testing

1. Condition Testing
2. Data Flow Testing
3. Loop Testing

Condition Testing (1)

1. Condition Testing aims to exercise all logical conditions in a program module.
2. Can Define
    a. Relational Expression (E1 op E2) : where E1 and E2 are arithmetic expression
    b. Simple Condition: Boolean variable or relational expression, possibly preceded by a NOT operator
    c. Compound condition: composed of two or more simple conditions, boolean operators and parentheses
    d. Boolean Expression: condition without relational expression

Condition Testing (2)

3. Types of errors in a condition include the following:
    a. Boolean operator error (existence of incorrect/missing/extra boolean operator)
    b. boolean variable error
    c. boolean parenthesis error
    d. relational operator error
    e. arithmetic expression error

Data Flow Testing

Cara menguji berdasarkan lokasi dari pendefinisian dan penggunaan suatu peubah dalam modul program

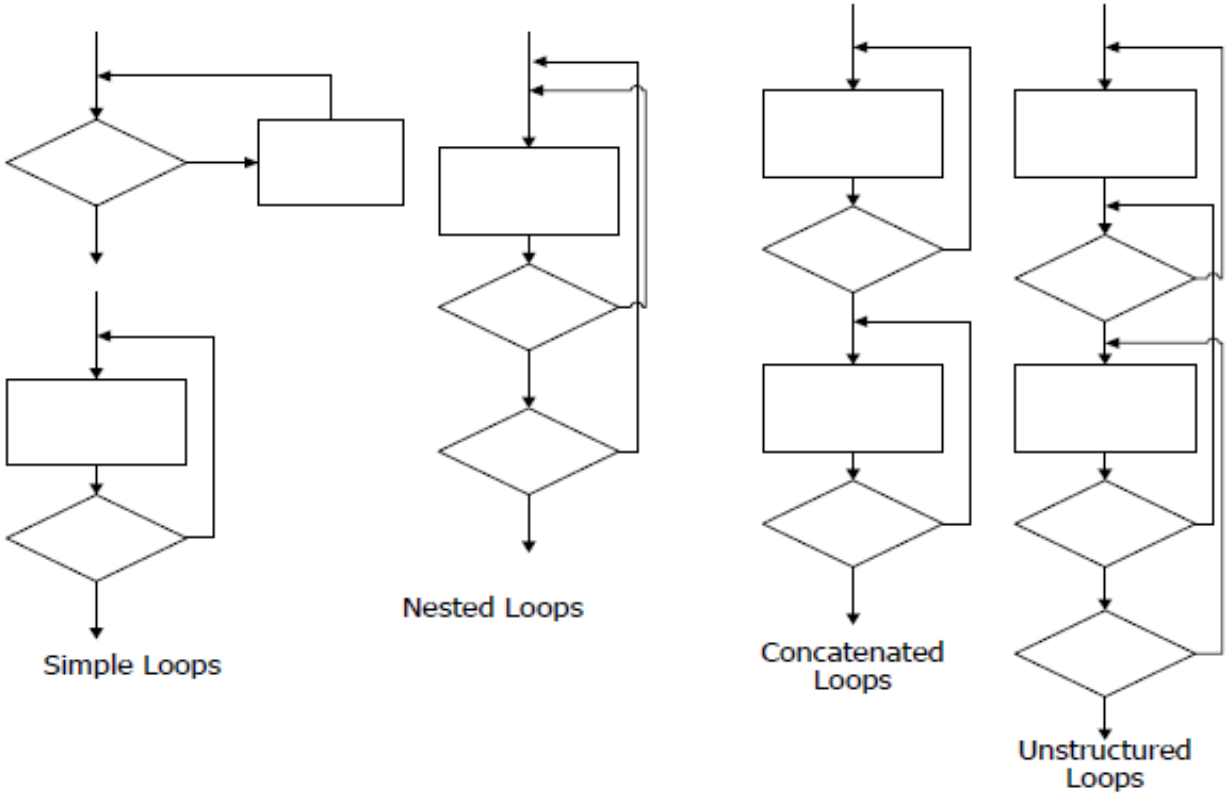
Loop Testing (1)

Cara menguji berdasarkan validitas dari konstruksi pengulangan yang digunakan dalam modul program:

1. Sederhana
2. Bercabang
3. Bersambung (*concatenated*)
4. Tak terstruktur

Loop Testing (2)

- Loop is fundamental to many algorithms.
- Loop can be defined as simple, concatenated, nested, and unstructured.

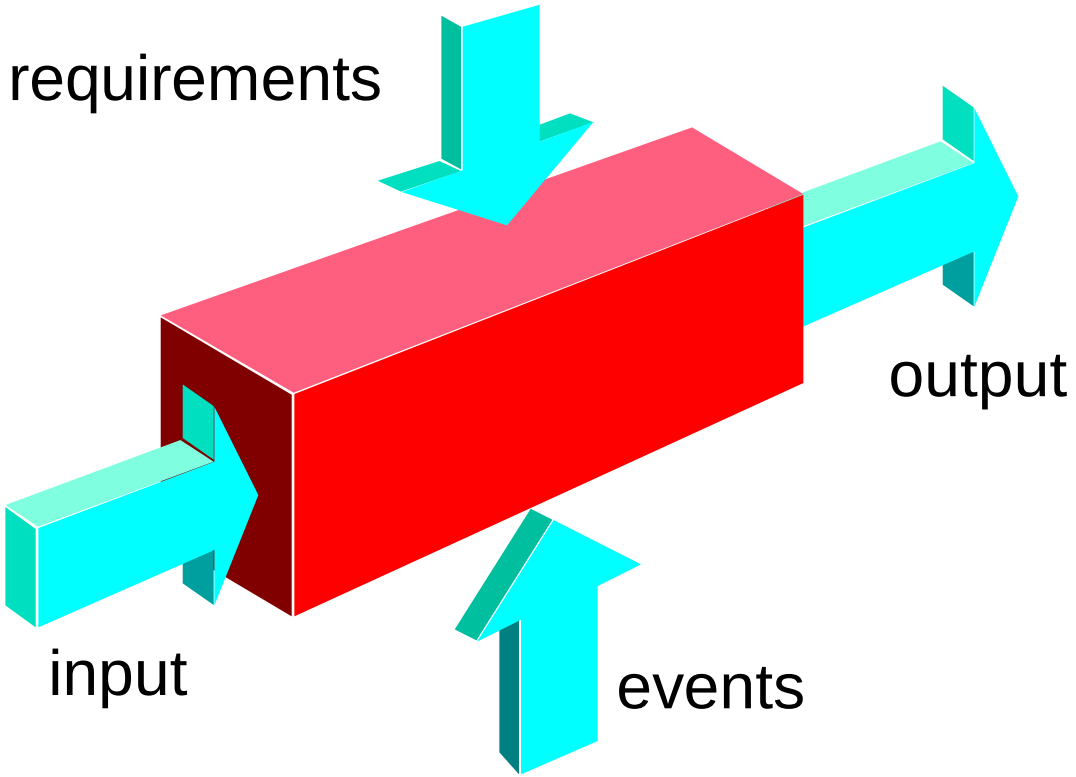


Loop Testing (3)

- To test:
  - **Simple Loops** of size n:
    - Skip loop entirely
    - Only one pass through loop
    - Two passes through loop
    - m passes through loop where m<n.
    - (n-1), n, and (n+1) passes through the loop.
  - **Nested Loops**
    - Start with inner loop. Set all other loops to minimum values.
    - Conduct simple loop testing on inner loop.
    - Work outwards
    - Continue until all loops tested.
  - **Concatenated Loops**
    - If independent loops, use simple loop testing.
    - If dependent, treat as nested loops.
  - **Unstructured loops**
    - Don't test - redesign.

PENGUJIAN black BOX

Pengujian Black Box (1)



Pengujian Black Box (2)

1. Digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang.
2. Kebenaran perangkat lunak yang diuji hanya dilihat berdasarkan keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut.
3. Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi kebutuhan pemakai dapat diukur sekaligus dapat diiketahui kesalahan-kesalahannya

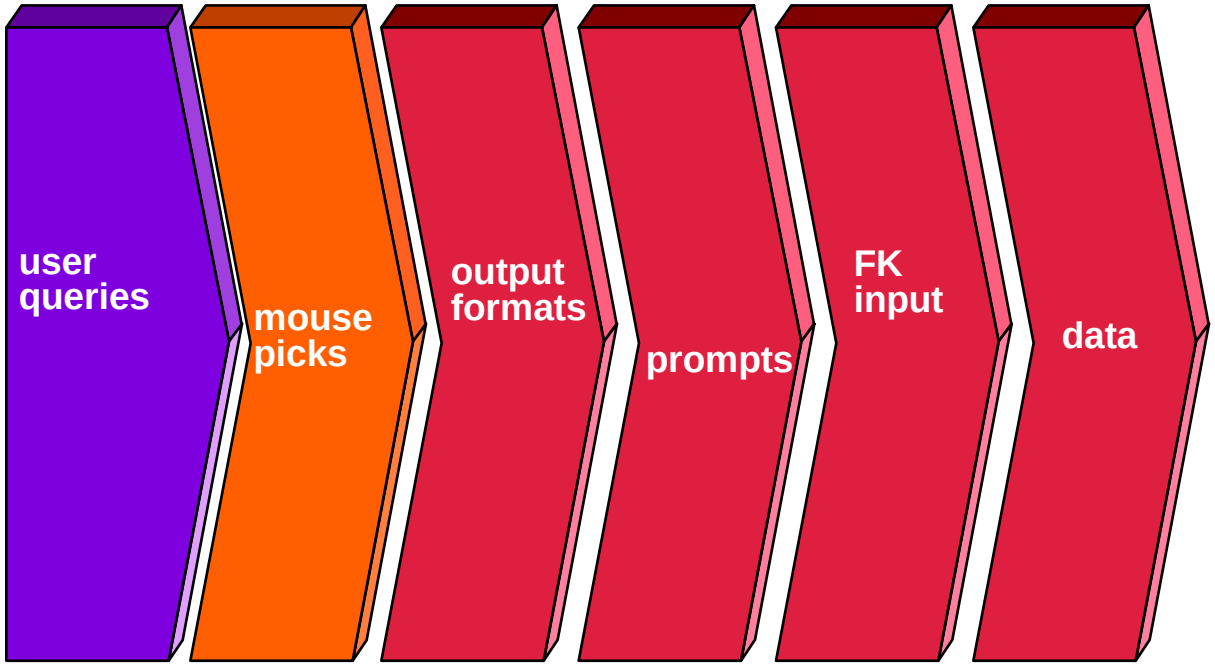
Pengujian Black Box (3)

1. Fungsi tidak benar atau hilang
2. Kesalahan antar muka
3. Kesalahan pada struktur data (pengaksesan basis data)
4. Kesalahan inisialisasi dan akhir program
5. Kesalahan performasi.

Teknik Pengujian Black Box

1. Equivalence Partitioning
2. Boundary Value Analysis/Limit Testing
3. Comparison Testing
4. Sample Testing
5. Robustness Testing
6. Behavior Testing
7. Requirement Testing
8. Performance Testing
9. Endurance Testing
10. Cause-Effect Relationship Testing

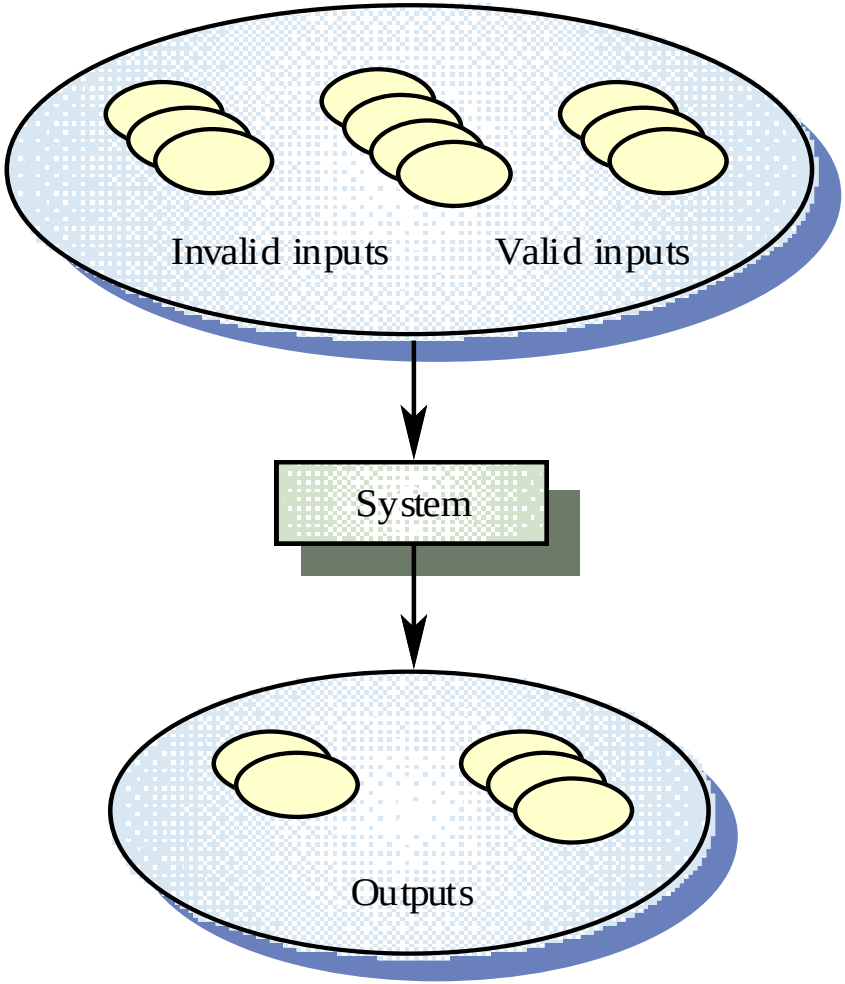
Equivalence Partitioning (1)



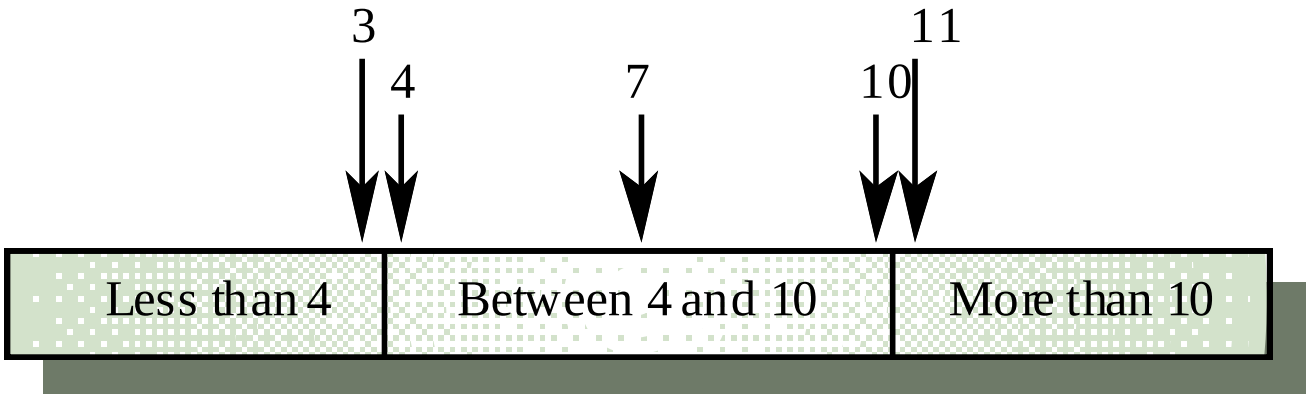
Equivalence Partitioning (2)

1. Input data dan output hasil terdapat di kelas yang berbeda yang sesuai dengan kelas inputnya
2. Masing-masing kelas equivalensi partition diproses dimana program akan memproses anggota kelas-kelas tersebut secara equivalen.
3. Test cases dipilih dari masing-masing partisi.

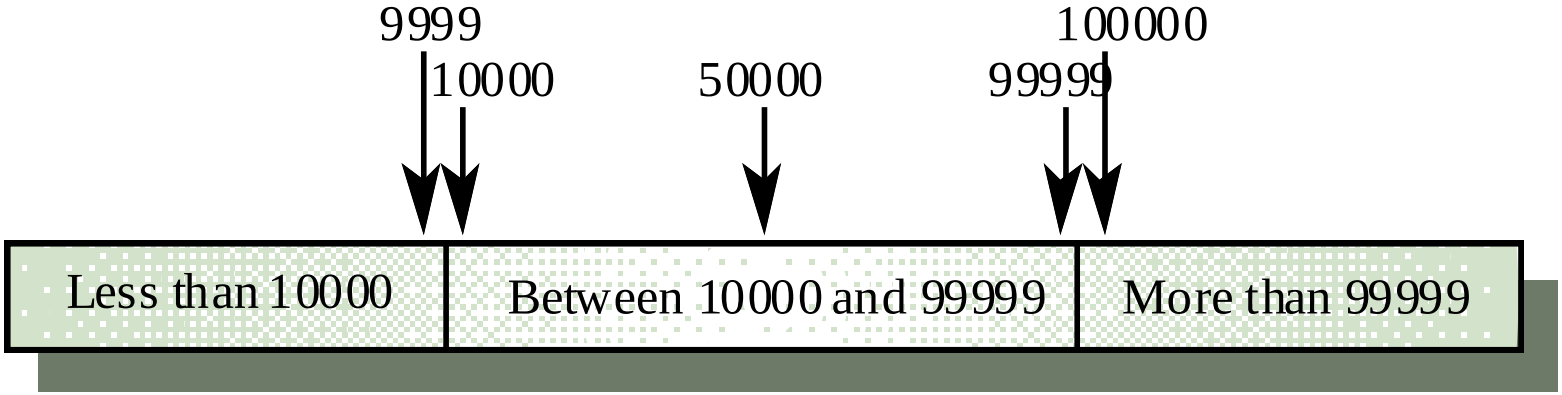
Equivalence Partitioning (3)



Equivalence Partitioning (4)



Number of input values



Input values

Boundary Value Analysis/Limit Testing

1. Menguji untuk input di sekitar batas atas maupun bawah sebuah range nilai yang valid.
2. Menguji nilai maksimal dan minimal.
3. Menerapkan (1 & 2) untuk output.
4. Menguji batas struktur data yang dipakai. Misal ukuran array.

Comparison Testing

1. Spesifikasi kebutuhan yang sama dimungkinkan menghasilkan aplikasi/ perangkat lunak yang berbeda.
2. Skenario pengujian pada aplikasi yang demikian bisa digunakan untuk skenario pengujian aplikasi serupa yang lain.

Sample Testing

1. Pengujian Sampel melibatkan pemilihan sejumlah nilai dari kelas kesetaraan input data
2. Mengintegrasikan nilai-nilai ke uji kasus
3. Nilai-nilai ini dapat dipilih pada konstan atau variabel interval

Behaviour Testing

Pengujian yang hasilnya baru terlihat setelah sekumpulan data diinputkan dalam rangka memanggil sub program yang ada.

Sebagai contoh pengujian pada struktur data stack (tumpukan).

Requirement Testing

1. Kebutuhan yang diasosiasikan dengan perangkat lunak (input/output/function/perfomance) diidentifikasi selama aktifitas spesifikasi perangkat lunak dan perancangan.
2. Untuk memfasilitasi pengujiannya, setiap kebutuhan ditelusuri dengan menggunakan matriks keterhubungan.

Req.	Spec.	Pre-Design	Det-Design	CSU	Test Proc.	Test Rpt

Perfomance Testing

1. Pengujian ini digunakan untuk mengukur dan mengeksplorasi batas perfomansi dari sebuah kinerja perangkat lunak.
2. Paremeter yang dinilai antara lain:
    a. Aliran data
    b. Ukuran memori yang digunakan
    c. Waktu eksekusi yang digunakan.

Endurance Testing

1. Endurance testing menggunakan uji kasus yang berulang-ulang dalam rangka mengevaluasi kemampuan perangkat lunak dalam memenuhi kebutuhan yang ada.
2. Sebagai contoh:
   a. Pengujian terhadap ketepatan perhitungan floating point.
   b. Pengujian terhadap manajemen sumberdaya sistem
   c. Pengujian input dan output dengan menggunakan framework untuk memnvalidasi input dan output layer.

Cause-effect Relationship Testing (1)

1. Teknik ini menghasilkan pengujian yang ekuivalen dengan cara mendeterminasi dan memilih kombinasi dari data input.
2. Langkah-langkah:
    a. Pecah kebutuhan menjadi beberapa subset yang masih mungkin bekerja.
    b. Definisikan sebab dan akibat berdasarkan kebutuhan.
    c. Analisis kebutuhan untuk membuat relasi logis
    d. Tandai graph, ketidakmungkinan dari kombinasi dari sebab-akibat dikarenakan batasan dari kebutuhan
    e. Konversi graph menjadi decision table
    f. Kolom → Uji kasus
    g. Baris → sebab-akibat
    h. Konversi kolom-kolom tersebut ke dalam uji kasus.

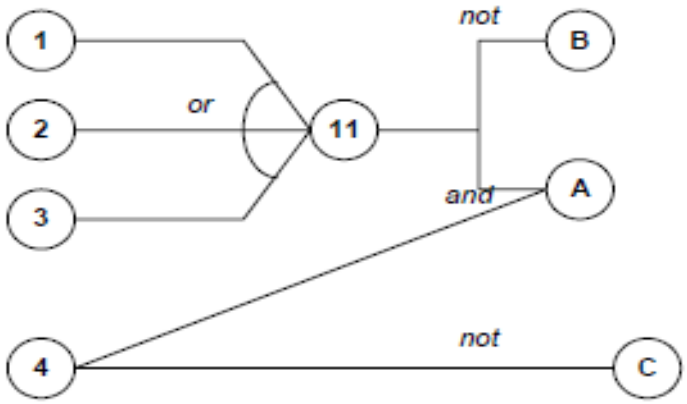
Cause-effect Relationship Testing (2)

A CITROEN car will be identified by a letter A, B or C, and have the letter X as the second character. Messages M1 and M2 must be sent in the event of an error in the first or second character respectively. If the identifier is correct, it is inserted in the database.

The the input state (the causes) and the output states (the effect) can be determined:

Input states:
1. 1st character: A
2. 1st character: B
3. 1st character: C
4. 2nd character: X

Output states:
A. database insertion
B. message M1
C. message M2



	1	2	3	4	5
1	0	1	0	0	
2	0	0	1	0	
3	0	0	0	1	
4		1	1	1	0
A		1	1	1	
B	1				
C					1

Alpha dan Beta Testing

1. Secara virtual, mustahil untuk seorang software developer untuk memperkirakan bagaimana customer akan melihat dan menggunakan softwarenya.
2. Instruksi yang disediakan mungkin saja disalah artikan
3. Kombinasi data yang aneh mungkin bisa digunakan oleh customer.
4. Keluaran dari sistem mungkin sudah jelas bagi tester akan tetapi belum tentu untuk user di dunia nyata.
5. Alpha dan beta testing memungkinkan untuk membuka kesalahan yang mungkin terjadi pada end user.

Alpha Testing

1. Pengujian alpha diadakan di lingkungan developer oleh sekumpulan end user yang akan menggunakan perangkat lunaknya.
2. Pihak developer mendampingi serta mencatat kesalahan-kesalahan maupun permasalahan dalam hal usability yang dirasakan oleh end user.

Beta Testing

1. Pengujian beta dilakukan di lingkungan end user tanpa kehadiran pihak developer.
2. Pengujian ini merupakan pengujian yang bersifat ‘live’ di lingkungan yang sebenarnya.
3. End user mencatat kesalahan yang terjadi kemudian menyampaikannya kepada pihak developer untuk diperbaiki.