Blackbox Testing

- **Metode Graph Based Testing**
- **Metode Equivalence Partitioning**
- **Boundary Value Analysis**
- **Cause-Effect Graphing Techniques**
- **State Transition Testing**
- **Orthogonal Array Testing**
- **Functional Analysis**
- **Use Cases**

Black Box Testing

- *Black box testing*, dilakukan tanpa pengetahuan detil struktur internal dari sistem atau komponen yang dites juga disebut sebagai *behavioral testing*, *specification-based testing*, *input/output testing atau functional testing*.

- *Black box testing* berfokus pada kebutuhan fungsional pada *software*, berdasarkan pada spesifikasi kebutuhan dari *software*.

- *Black box testing* digunakan pada tahap akhir dan berfokus pada domain informasi.

Kategori error

- Kategori *error* yang akan diketahui melalui *black box testing*:
  - Fungsi yang hilang atau tak benar
  - *Error* dari antar-muka
  - *Error* dari struktur data atau akses eksternal database
  - *Error* dari kinerja atau tingkah laku
  - *Error* dari inisialisasi dan terminasi

Tujuan Black box Testing

- Tes didisain untuk menjawab pertanyaan sebagai berikut:
  - Bagaimana validasi fungsi yang akan dites?
  - Bagaimana tingkah laku dan kinerja sistem dites?
  - Kategori masukan apa saja yang bagus digunakan untuk test cases?
  - Apakah sebagian sistem sensitif terhadap suatu nilai masukan tertentu?
  - Bagaimana batasan suatu kategori masukan ditetapkan?
  - Sistem mempunyai toleransi jenjang dan volume data apa saja?
  - Apa saja akibat dari kombinasi data tertentu yang akan terjadi pada operasi sistem?

Test case black box

- Dengan menerapkan teknik *black box*, dapat dibuat sekumpulan *test cases* yang memuaskan kriteria-kriteria sebagai berikut :

  - ***Test cases*** yang mengurangi jumlah *test cases* (lebih dari satu) yang didisain untuk mencapai testing yang masuk akal.

  - ***Test cases*** yang dapat memberikan informasi tentang kehadiran kelas-kelas dari *error*.

Dekomposisi kebutuhan untuk dites secara sistematis (1)

- Karakteristik suatu testcase:
  - Dekomposisi dari tugas-tugas testing suatu sistem ke aktivitas-aktivitas yang lebih kecil
  - Dapat dimanajemeni,
  - Tercapai *test case* individual
  - *test case* yang ada telah cukup mencakup semua aspek dari sistem
  - Pendisainan *test case* dilakukan secara manual

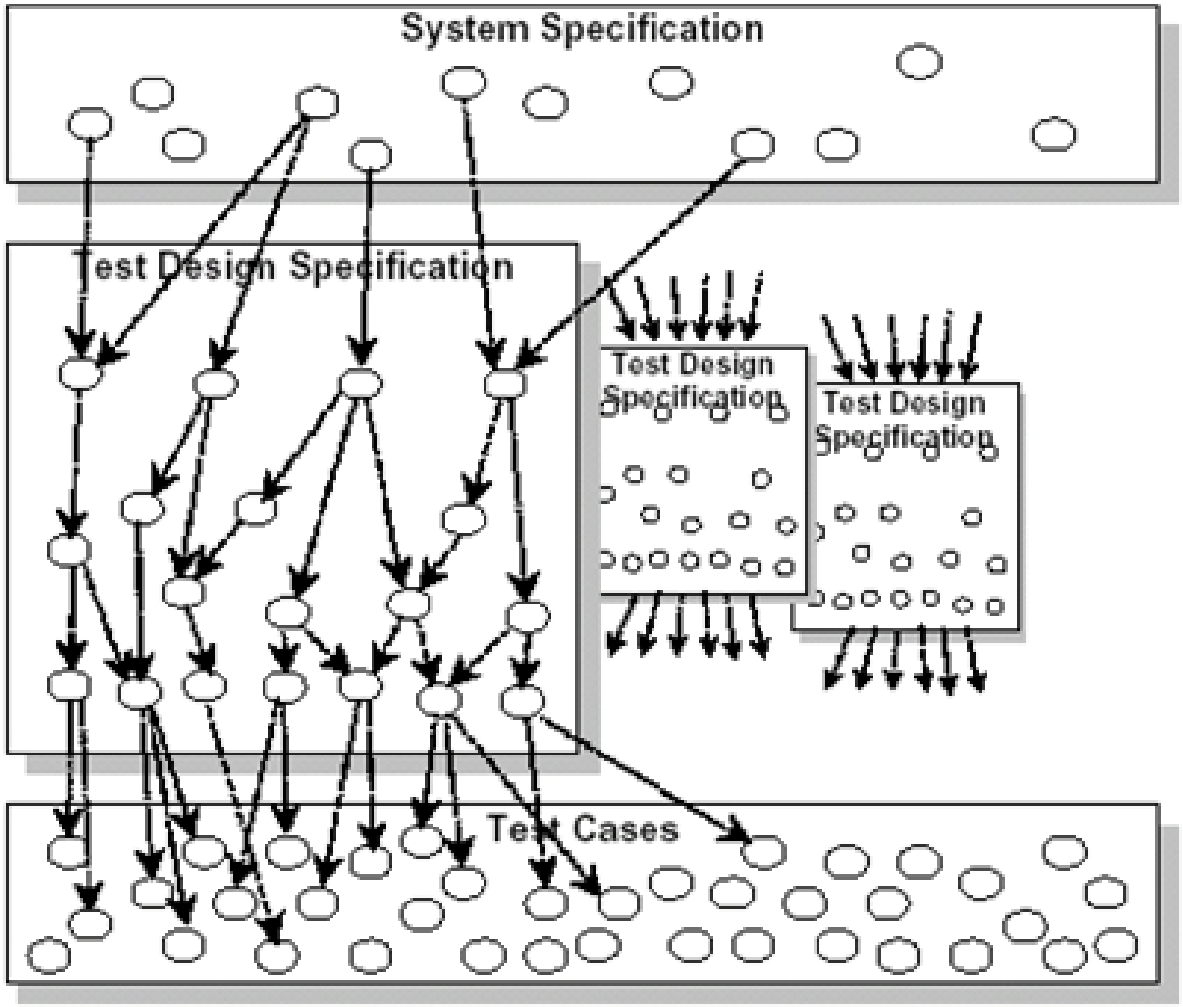
Dekomposisi kebutuhan untuk dites secara sistematis (2)

- **Spesifikasi sebagai tuntunan testing**
  - Spesifikasi atau model sistem adalah titik awal dalam memulai disain tes.
  - Spesifikasi atau model sistem dapat berupa spesifikasi fungsional, spesifikasi kinerja atau keamanan, spesifikasi skenario pengguna, atau spesifikasi berdasarkan pada resiko sistem.
  - Spesifikasi menggambarkan kriteria yang digunakan untuk menentukan operasi yang benar atau dapat diterima, sebagai acuan pelaksanaan tes.

Dekomposisi kebutuhan untuk dites secara sistematis (3)

- **Dekomposisi obyektifitas tes**
  - Disain tes berfokus pas tda spesifikasi komponen yang dites.
  - Obyektifitas tes tingkat atas disusun berdasarkan pada spesifikasi komponen.
  - Tiap obyektifitas tes ini untuk kemudian didekomposisikan ke dalam obyektifitas tes lainnya atau *test cases* menggunakan teknik disain tes.

Dekomposisi obyektifitas tes



Dekomposisi tugas testing adalah memecah-mecah sistem ke obyektifitas-obyektifitas tes yang lebih kecil cakupannya hingga sekumpulan *test cases* teridentifikasi.

Tipe testing Blackbox

Terdapat banyak jenis teknik disain tes yang dapat dipilih berdasarkan pada tipe testing yang akan digunakan yaitu:

- *Equivalence Class Partitioning*
- *Boundary Value Analysis*
- *State Transitions Testing*
- *Cause-Effect Graphing*

II. Metode Equivalence Partitioning

- Adalah metode *black box testing* yang membagi domain masukan dari suatu program ke dalam kelas-kelas data, dimana *test cases* dapat diturunkan.

- *Equivalence partitioning* berdasarkan pada premis masukan dan keluaran dari suatu komponen yang dipartisi ke dalam kelas-kelas, menurut spesifikasi dari komponen tersebut, yang akan diperlakukan sama (ekuivalen) oleh komponen tersebut.

- Dapat juga diasumsikan bahwa masukan yang sama akan menghasilkan respon yang sama pula

Petunjuk pelaksanaan dalam melakukan *equivalence partitioning*

- Jika masukan mempunyai jenjang tertentu, maka definisikan kategori valid dan tak valid terhadap jenjang masukan tersebut.
- Jika masukan membutuhkan nilai tertentu, definisikan kategori valid dan tak valid.
- Jika masukan membutuhkan himpunan masukan tertentu, definisikan kategori valid dan tak valid.
- Jika masukan adalah *boolean*, definisikan kategori valid dan tak valid.

Kombinasi partisi ekuivalensi

- Nilai masukan yang valid atau tak valid.
- Nilai numerik yang negatif, positif atau nol.
- *String* yang kosong atau tidak kosong.
- Daftar (*list*) yang kosong atau tidak kosong.
- *File* data yang ada dan tidak, yang dapat dibaca / ditulis atau tidak.
- Tanggal yang berada setelah tahun 2000 atau sebelum tahun 2000, tahun kabisat atau bukan tahun kabisat (terutama tanggal 29 Pebruari 2000 yangg mempunyai proses tersendiri).
- Tanggal yang berada di bulan yang berjumlah 28, 29, 30, atau 31 hari.
- Hari pada hari kerja atau liburan akhir pekan.
- Waktu di dalam atau di luar jam kerja kantor.
- Tipe *file* data, seperti: teks, data berformat, grafik, video, atau suara.
- Sumber atau tujuan *file*, seperti *hard drive, floppy drive, CD-ROM,* jaringan.

Contoh Ilustrasi

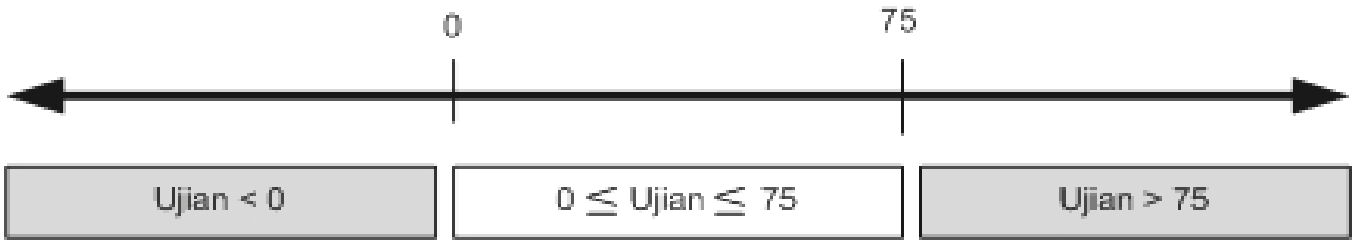
Suatu fungsi, *generate_grading*, dengan spesifikasi sebagai berikut:

- Fungsi mempunyai dua penanda, yaitu “Ujian” (di atas 75) dan “Tugas” (di atas 25).
- Fungsi melakukan gradasi nilai kursus dalam rentang ‘A’ sampai ‘D’. Tingkat gradasi dihitung dari kedua penanda, yang dihitung sebagai total penjumlahan nilai “Ujian” dan nilai “Tugas”, sebagaimana dinyatakan berikut ini:
  - Lebih besar dari atau sama dengan 70 – ‘A’
  - Lebih besar dari atau sama dengan 50, tapi lebih kecil dari 70 – ‘B’
  - Lebih besar dari atau sama dengan 30, tapi lebih kecil dari 50 – ‘C’
  - Lebih kecil dari 30 – ‘D’
- Dimana bila nilai berada di luar rentang yang diharapkan akan muncul pesan kesalahan (‘FM’). Semua masukan berupa *integer*.

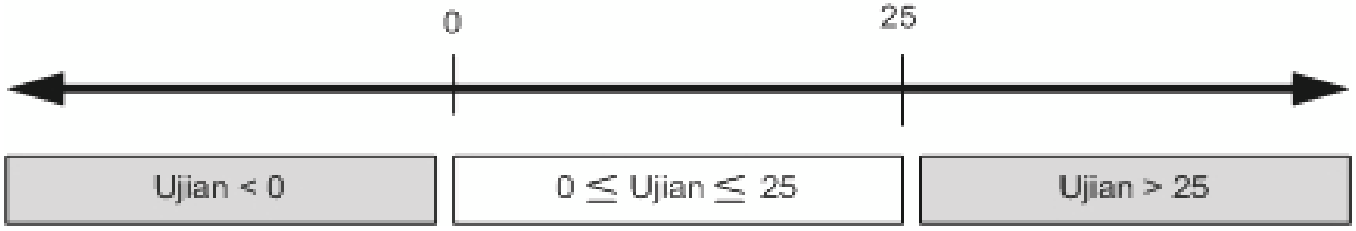
Analisa partisi

Partisi untuk nilai valid dan tidak valid harus ditentukan.

- “Ujian”



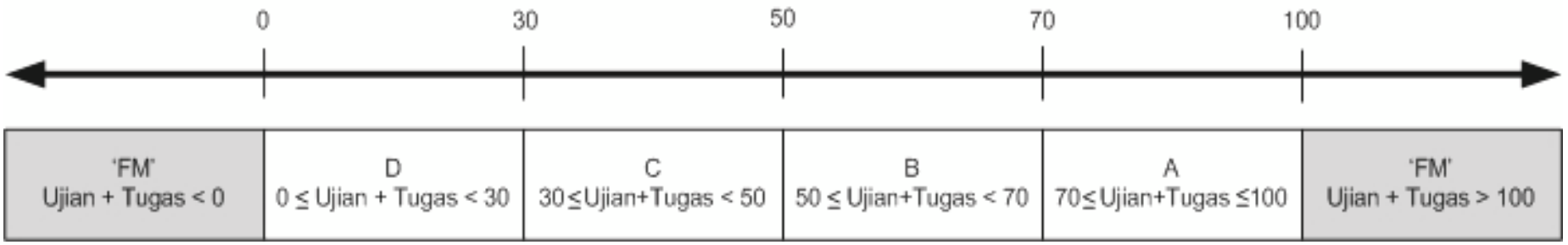
- “Tugas”



Nilai masukan dapat berupa nilai bukan *integer*. Sebagai contoh:

- Ujian = real number
- Ujian = alphabetic
- Tugas = real number
- Tugas = alphabetic

- Berikutnya, keluaran dari fungsi *generate-grading*, yaitu:



- Partisi ekuivalensi juga termasuk nilai yang tidak valid.
- Sulit untuk mengidentifikasi keluaran yang tidak dispesifikasikan, tapi harus tetap dipertimbangkan, seolah-olah dapat dihasilkan / terjadi, misal:
  - Gradasi = E
  - Gradasi = A+
  - Gradasi = null

- Pada contoh ini, didapatkan 19 partisi ekuivalensi.

- Dalam pembuatan partisi ekuivalensi, tester harus melakukan pemilihan secara subyektif. Contohnya, penambahan masukan dan keluaran tidak valid.
- Karena subyektifitas ini, maka partisi ekuivalensi dapat berbeda-beda untuk tester yang berbeda.

Disain *test cases*

- *Test cases* didisain untuk menguji partisi.

- Suatu *test case* menyederhanakan hal-hal berikut:
  - Masukan komponen.
  - Partisi yang diuji.
  - Keluaran yangg diharapkan dari *test case*.

- Dua pendekatan pembuatan *test case* untuk menguji partisi, adalah:

  1. *Test cases* terpisah dibuat untuk tiap partisi dengan ***one-to-one basis***.
  2. Sekumpulan kecil *test cases* dibuat untuk mencakup semua partisi. *Test case* yang sama dapat diulang untuk *test cases* yang lain.

Partisi *one-to-one test cases (1)*

- *Test cases* untuk partisi masukan “Ujian”, adalah sebagai berikut:

Test Case	1	2	3
Masukan Ujian	44	-10	93
Masukan Tugas	15	15	15
Total Nilai	59	5	108
Partisi yang dites	$0 \leq e \leq 75$	$e < 0$	$e > 75$
Keluaran yang diharapkan	B	FM	FM

Partisi *one-to-one test cases (2)*

- Suatu nilai acak 15 digunakan untuk masukan "Tugas".
- *Test cases* untuk partisi masukan "Tugas", adalah sebagai berikut:

Test Case	4	5	6
Masukan Ujian	44	40	40
Masukan Tugas	8	-15	47
Total Nilai	48	25	87
Partisi yang dites	$0 \leq c \leq 25$	$c < 0$	$c > 25$
Keluaran yang diharapkan	C	FM	FM

Partisi *one-to-one test cases (3)*

- Suatu nilai acak 40 digunakan untuk masukan "Ujian".
- *Test cases* untuk partisi masukan tidak valid lainnya, adalah sebagai berikut:

Test Case	7	8	9	10
Masukan Ujian	48.7	'q'	40	40
Masukan Tugas	15	15	12.76	'g'
Total Nilai	63.7	?	52.76	?
Partisi yang dites	real	alpha	real	alpha
Keluaran yang diharapkan	FM	FM	FM	FM

Partisi *one-to-one test cases (4)*

- *Test cases untuk partisi keluaran valid, adalah sebagai berikut:*

Test Case	11	12	13
Masukan Ujian	-10	12	32
Masukan Tugas	-10	5	13
Total Nilai	-20	17	45
Partisi yang dites	$t < 0$	$0 \leq t \leq 30$	$30 \leq t \leq 50$
Keluaran yang diharapkan	FM	D	C

Partisi *one-to-one test cases (5)*

- *Test cases untuk partisi keluaran valid, adalah sebagai berikut:*

Test Case	14	15	16
Masukan Ujian	40	60	80
Masukan Tugas	22	20	30
Total Nilai	66	80	110
Partisi yang dites	$50 \leq t \leq 70$	$70 \leq t \leq 100$	$t > 100$
Keluaran yang diharapkan	B	A	FM

Partisi *one-to-one test cases (6)*

- Nilai masukan “Ujian” dan “Tugas” diambil dari total nilai “Ujian” dengan nilai “Tugas”.
- Dan akhirnya, partisi keluaran tidak valid, adalah:

Test Case	17	18	19
Masukan Ujian	-10	100	null
Masukan Tugas	0	10	null
Total Nilai	-10	110	?
Partisi yang dites	E	A+	null
Keluaran yang diharapkan	FM	FM	FM

Test cases minimal untuk multi partisi (1)

- Pada kasus *test cases* di atas banyak yang mirip, tapi mempunyai target partisi ekuivalensi yang berlainan.
- Hal ini memungkinkan untuk mengembangkan *test cases* tunggal yang menguji multi partisi dalam satu waktu.
- Pendekatan ini memungkinkan tester untuk mengurangi jumlah *test cases* yang dibutuhkan untuk mencakup semua partisi ekuivalensi.

Test cases minimal untuk multi partisi (2)

- Contoh:

Test Case	1
Masukan Ujian	60
Masukan Tugas	20
Total Nilai	80
Keluaran yang diharapkan	A

Test case di atas menguji tiga partisi:

- 0 ≤ Ujian ≤ 75
- 0 ≤ Tugas ≤ 25
- Hasil gradasi = A : 70 ≤ Ujian + Tugas ≤ 100

Test cases minimal untuk multi partisi (3)

- Hal yang sama, *test cases* dapat dibuat untuk menguji multi partisi untuk nilai tidak valid:

Test Case	2
Masukan Ujian	-10
Masukan Tugas	-15
Total Nilai	-25
Keluaran yang diharapkan	FM

Test case di atas menguji tiga partisi:

- Ujian < 0
- Tugas < 0
- Hasil gradasi = FM : Ujian + Tugas < 0

Perbandingan pendekatan *one-to-one* dengan minimalisasi

One-to-one	Minimalisasi
Pendekatan *one-to-one* membutuhkan lebih banyak *test cases*. Bagaimana juga identifikasi dari partisi memakan waktu lebih lama daripada penurunan dan eksekusi *test cases*. Tiap penghematan untuk mengurangi jumlah *test cases*, relatif kecil dibandingkan dengan biaya pemakaian teknik dalam menghasilkan partisi.	Pendekatan minimalisasi sulit menentukan penyebab dari terjadinya kesalahan. Hal ini akan menyebabkan *debugging* menjadi lebih menyulitkan, daripada pelaksanaan proses testingnya sendiri.