Testing Implementasi Sistem

Black Box Testing

Equivalence Partitioning & Boundary
Value Analysis

Black Box Testing

- *Black box testing*, dilakukan tanpa pengetahuan detail 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* bukan teknik alternatif daripada *white box testing*. Lebih daripada itu, ia merupakan pendekatan pelengkap dalam mencakup *error* dengan kelas yang berbeda dari metode *white box testing*.

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

Dekomposisi kebutuhan untuk dites secara sistematis

- Untuk dapat membuat *test cases* yang efektif, harus dilakukan dekomposisi dari tugas-tugas testing suatu sistem ke aktivitas-aktivitas yang lebih kecil dan dapat dimanajemeni, hingga tercapai *test case* individual.
- Tentunya, dalam disain *test case* juga digunakan mekanisme untuk memastikan bahwa *test case* yang ada telah cukup mencakup semua aspek dari sistem.

- Pendisainan *test case* dilakukan secara manual, tidak ada alat bantu otomasi guna menentukan *test cases* yang dibutuhkan oleh sistem, karena tiap sistem berbeda, dan alat bantu tes tak dapat mengetahui aturan benar-salah dari suatu operasi.
- Disain tes membutuhkan pengalaman, penalaran dan intuisi dari seorang tester.

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.

- Banyak kasus, biasanya berhubungan dengan sistem lama, hanya terdapat sedikit atau bahkan tidak ada dokumentasi dari spesifikasi sistem.
- Dalam hal ini sangat dibutuhkan peran dari pengguna akhir yang mengetahui sistem untuk diikutsertakan ke dalam disain tes, sebagai ganti dari dokumen spesifikasi sistem.
- Walaupun demikian, harus tetap ada dokumentasi spesifikasi, yang bisa saja dibuat dalam bentuk sederhana, yang berisi sekumpulan obyektifitas tes di level atas.

Equivalence Partitioning

- Adalah metode *black box testing* yang membagi domain masukan dari suatu program ke dalam kelas-kelas data, dimana *test cases* dapat diturunkan [BCS97a].
- *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.

- Nilai tunggal pada suatu partisi ekuivalensi diasumsikan sebagai representasi dari semua nilai dalam partisi.
- Hal ini digunakan untuk mengurangi masalah yang tidak mungkin untuk testing terhadap tiap nilai masukan (lihat prinsip testing: testing yang komplit tidak mungkin).

Kombinasi yang mungkin dalam 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 yang 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.

Analisa partisi

- Tester menyediakan suatu model komponen yang dites yang merupakan partisi dari nilai masukan dan keluaran komponen.
- Masukan dan keluaran dibuat dari spesifikasi dari tingkah laku komponen.
- Partisi adalah sekumpulan nilai, yang dipilih dengan suatu cara dimana semua nilai di dalam partisi, diharapkan untuk diperlakukan dengan cara yang sama oleh komponen (seperti mempunyai proses yang sama).
- Partisi untuk nilai valid dan tidak valid harus ditentukan.

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*.

Pendisainan *test cases*

- *Test cases* didisain untuk menguji partisi.
- Suatu *test case* menyederhanakan hal-hal berikut:
 - Masukan komponen.
 - Partisi yang diuji.
 - Keluaran yang diharapkan dari *test case*.
- Dua pendekatan pembuatan *test case* untuk menguji partisi, adalah:
 - *Test cases* terpisah dibuat untuk tiap partisi dengan *one-to-one basis*.
 - 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*

- *Test cases* untuk partisi masukan “Ujian”, dengan Suatu nilai acak 15 digunakan untuk masukan “Tugas”, 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

- *Test cases* untuk partisi masukan “Tugas”, dengan Suatu nilai acak 40 digunakan untuk masukan “Ujian”, 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

- *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

- *Test cases* untuk partisi keluaran valid, dengan Nilai masukan “Ujian” dan “Tugas” diambil dari total nilai “Ujian” dengan nilai “Tugas”, 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

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

- 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

- 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.

Contoh:

Test case menguji tiga partisi:

- $0 \leq \text{Ujian} \leq 75$
- $0 \leq \text{Tugas} \leq 25$
- Hasil gradasi = A : $70 \leq \text{Ujian} + \text{Tugas} \leq 100$

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

- Hal yang sama, *test cases* dapat dibuat untuk menguji multi partisi untuk nilai tidak valid
- *Test case* menguji tiga partisi:
 - **Ujian < 0**
 - **Tugas < 0**
 - **Hasil gradasi = FM : Ujian + Tugas < 0**

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

One-to-one VS minimalisasi

- Kekurangan dari 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.
- Kekurangan dari pendekatan minimalisasi adalah sulitnya menentukan penyebab dari terjadinya kesalahan. Hal ini akan menyebabkan *debugging* menjadi lebih menyulitkan, daripada pelaksanaan proses testingnya sendiri.

Boundary Value Analysis

- Untuk suatu alasan yang tidak dapat sepenuhnya dijelaskan, sebagian besar jumlah *errors* cenderung terjadi di sekitar batasan dari domain masukan daripada di “pusat”nya.
- Karena alasan inilah *boundary value analysis* (BVA) dikembangkan sebagai salah satu teknik testing.
- *Boundary value analysis* adalah suatu teknik disain *test cases* yang berguna untuk melakukan pengujian terhadap nilai sekitar dari pusat domain masukan.

- Teknik *boundary value analysis* merupakan komplement dari teknik *equivalence partitioning*.
- Setelah dilakukan pemilihan tiap elemen suatu kelas ekuivalensi (menggunakan *equivalence partitioning*), BVA melakukan pemilihan nilai batas-batas dari kelas untuk *test cases*.
- BVA tidak hanya berfokus pada kondisi masukan, BVA membuat *test cases* dari domain keluaran juga.

- *Boundary-values* merupakan nilai batasan dari kelas-kelas ekuivalensi. Contoh:
 - Senin dan Minggu untuk hari.
 - Januari dan Desember untuk bulan.
 - (-32767) dan 32767 untuk *16-bit integers*.
 - Satu karakter *string* dan maksimum panjang *string*.
- *Test cases* dilakukan untuk menguji nilai-nilai di kedua sisi dari batasan.
- Nilai tiap sisi dari batasan yang dipilih, diusahakan mempunyai selisih sekecil mungkin dengan nilai batasan (misal: selisih 1 untuk bilangan *integers*).

Sebagai catatan, ada banyak partisi teridentifikasi yang terikat hanya pada satu sisi, seperti:

- ❑ Nilai Ujian > 75
- ❑ Nilai Ujian < 0
- ❑ Nilai Tugas > 25
- ❑ Nilai Tugas < 0
- ❑ Total Nilai (Nilai Ujian + Nilai Tugas) > 100
- ❑ Total Nilai (Nilai Ujian + Nilai Tugas) < 0

Partisi-partisi ini akan diasumsikan terikat oleh tipe data yang digunakan sebagai masukan atau keluaran. Contoh, 16 *bit integers* mempunyai batasan 32767 dan -32768.

Maka dapat dibuat *test cases* untuk nilai Ujian sebagai berikut:

Test Case	28	29	30	31	32	33
Masukan (Ujian)	32766	32767	32768	-32769	-32768	-32767
Masukan (Tugas)	15	15	15	15	15	15
Nilai Batasan	32767			-32768		
Keluaran yang Diharapkan	FM	FM	FM	FM	FM	FM