Testing & Implementasi Sistem (3 sks)

Black Box Testing (1)

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* 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.

Dekomposisi obyektifitas tes

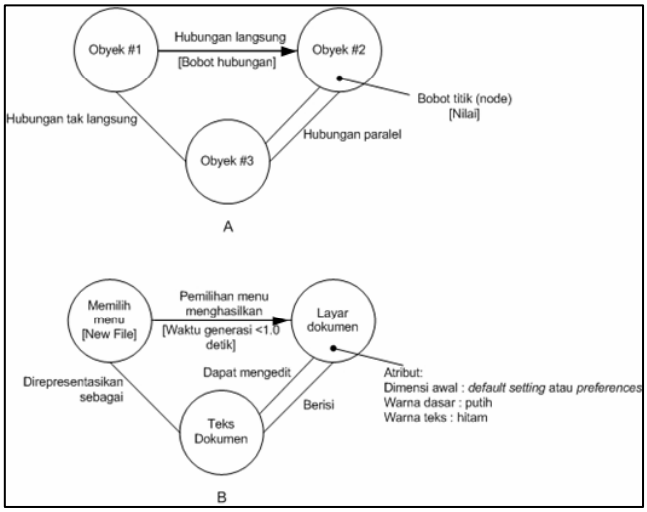
- Disain tes berfokus pada spesifikasi komponen yang dites. Obyektifitas tes tingkat atas disusun berdasarkan pada spesifikasi komponen.
- Tiap obyektifitas tes ini untuk kemudian didekomposisikan ke dalam obyektifitas tes lainnya atau *test cases* menggunakan teknik disain tes.

Jenis teknik disain tes yang dapat dipilih berdasarkan pada tipe testing yang akan digunakan

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

Metode *Graph Based Testing*

- Langkah pertama pada *black box testing* adalah memahami obyek yang dimodelkan dalam *software* dan hubungan koneksi antar obyek, kemudian definisikan serangkaian tes yang merupakan verifikasi bahwa semua obyek telah mempunyai hubungan dengan yang lainnya sesuai yang diharapkan. [BEI95]
- Langkah ini dapat dicapai dengan membuat grafik, dimana berisi kumpulan *node* yang mewakili obyek, penghubung / *link* yang mewakili hubungan antar obyek, bobot *node* yang menjelaskan properti dari suatu obyek, dan bobot penghubung yang menjelaskan beberapa karakteristik dari penghubung / *link.*



- *Nodes* direpresentasikan sebagai lingkaran yang dihubungkan dengan garis penghubung.
- Suatu hubungan langsung (digambarkan dalam bentuk anak panah) mengindikasikan suatu hubungan yang bergerak hanya dalam satu arah.
- Hubungan dua arah, juga disebut sebagai hubungan simetris, menggambarkan hubungan yang dapat bergerak dalam dua arah.
- Hubungan paralel digunakan bila sejumlah hubungan ditetapkan antara dua *nodes*.

Contoh

- Berdasarkan pada gambar, pemilihan menu [New File] akan menghasilkan (*generate*) layar dokumen.
- Bobot *node* dari layar dokumen menyediakan suatu daftar atribut layar yang diharapkan bila layar dibuat (*generated*).
- Bobot hubungan mengindikasikan bahwa layar harus telah dibuat dalam waktu kurang dari 1 detik.
- Suatu hubungan tak langsung ditetapkan sebagai hubungan simetris antara pemilihan menu [New File] dengan teks dokumen, dan hubungan paralel mengindikasikan hubungan layar dokumen dan teks dokumen.

Beizer: Sejumlah metode tingkah laku testing yang dapat menggunakan grafik

- **Pemodelan Alur Transaksi**, dimana *node* mewakili langkah-langkah transaksi (misal langkah-langkah penggunaan jasa reservasi tiket pesawat secara *on-line*), dan penghubung mewakili logika koneksi antar langkah (misal masukan informasi penerbangan diikuti dengan pemrosesan validasi / keberadaan).

- **Pemodelan *Finite State***, dimana *node* mewakili status software yang dapat diobservasi (misal tiap layar yang muncul sebagai masukan order ketika kasir menerimaa order), dan penghubung mewakili transisi yang terjadi antar status (misal informasi order diverifikasi dengan menampilkan keberadaan inventori dan diikuti dengan masukan informasi penagihan pelanggan).

- **Pemodelan Alur Data**, dimana *node* mewakili obyek data (misal data Pajak dan Gaji Bersih), dan penghubung mewakili transformasi untuk me-translasikan antar obyek data (misal Pajak = 0.15 x Gaji Bersih).
- **Pemodelan Waktu / *Timing*,** dimana *node* mewakili obyek program dan penghubung mewakili sekuensial koneksi antar obyek tersebut. Bobot penghubung digunakan untuk spesifikasi waktu eksekusi yang dibutuhkan.

- Testing berbasis grafik (*graph based testing*) dimulai dengan mendefinisikan semua *nodes* dan bobot *nodes*.
- Dalam hal ini dapat diartikan bahwa obyek dan atribut didefinisikan terlebih dahulu.
- *Data model* dapat digunakan sebagai titik awal untuk memulai, namun perlu diingat bahwa kebanyakan *nodes* merupakan obyek dari program (yang tidak secara eksplisit direpresentasikan dalam *data model*).
- Agar dapat mengetahui indikasi dari titik mulai dan akhir grafik, akan sangat berguna bila dilakukan pendefinisian dari masukan dan keluaran *nodes.*

- Bila *nodes* telah diidentifikasi, hubungan dan bobot hubungan akan dapat ditetapkan.
- Hubungan harus diberi nama, walaupun hubungan yang merepresentasikan alur kendali antar obyek program sebenarnya tidak butuh diberi nama.
- Pada banyak kasus, model grafik mungkin mempunyai *loops* (yaitu, jalur pada grafik yang terdiri dari satu atau lebih *nodes*, dan diakses lebih dari satu kali iterasi).
- *Loop testing* dapat diterapkan pada tingkat *black box*. Grafik akan menuntun dalam mengidentifikasi *loops* yang perlu dites.

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 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.

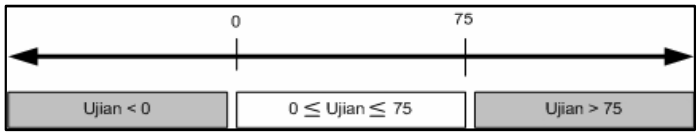
Analisa partisi

- 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.

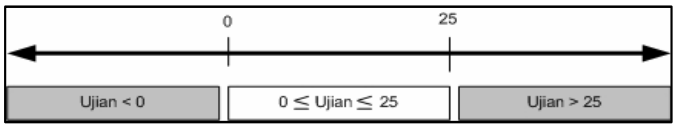
Contoh ilustrasi

- 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*.**

- “Ujian”

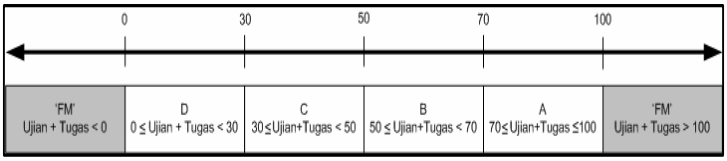


- “Tugas”



- 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*



- **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.**

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	$0 \le e \le 75$	$e < 0$	$e > 75$
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	$0 \le c \le 25$	$c < 0$	$c > 25$
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	I1	I2	I3
Masukan Ujian	-10	12	32
Masukan Tugas	-10	5	13
Total Nilai	-20	17	45
Partisi yang dites	t < 0	0 ≤ t ≤ 30	30 ≤ t ≤ 50
Keluaran yang diharapkan	FM	D	C

Test Case	I4	I5	I6
Masukan Ujian	40	60	80
Masukan Tugas	22	20	30
Total Nilai	66	80	110
Partisi yang dites	60 ≤ t ≤ 70	70 ≤ t ≤ 100	t > 100
Keluaran yang diharapkan	B	A	FM

- Dan akhirnya, partisi keluaran tidak valid, adalah

Test Case	I7	I8	I9
Masukan Ujian	-10	100	null
Masukan Tugas	J	10	null
Total Nilai	-10	110	?
Partisi yang dites	Ξ	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:

- $0 \leq$ Ujian ≤ 75
- $0 \leq$ Tugas ≤ 25
- Hasil gradasi = A : $70 \leq$ Ujian + Tugas ≤ 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

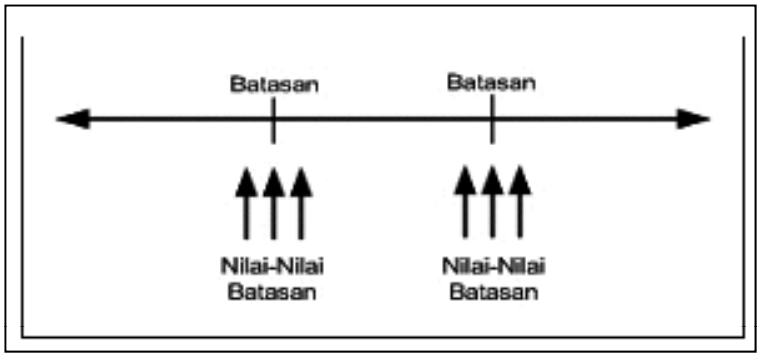
- Kekurangan dari pendekatan *one-to-one* membutuhkan lebih banyak *test cases*.
- Bagaimana juga identifikasi dari partisi memakan waktu lebih lama daripada penurunan dan eksekusi *test cases*. Tiap penghematan untuk mengurangi jumlah *test cases*, relatif kecil dibandingkan dengan biaya pemakaian teknik 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.

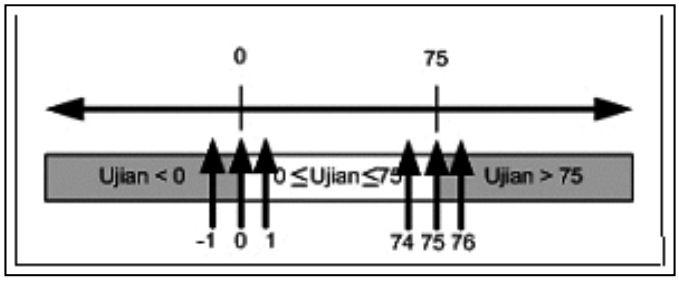
- Teknik *boundary value analysis* merupakan komplemen dari teknik *equivalence partitioning*.
- Setelah dilakukan pemilihan tiap elemen suatu kelas ekuivalensi (menggunakan *equivalence partitioning*), BVA melakukan pemilihan nilai batas-batas dari kelas untuk *test cases*.
- BVA tidak hanya berfokus 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*).



Contoh ilustrasi

- Sebagai contoh, partisi “Ujian” memberikan nilai batasan tes untuk menguji nilai “Ujian” pada –1, 0, 1, 74, 75, dan 76



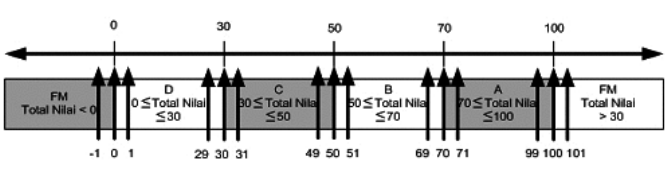
Test Case	1	2	3	4	5	6
Masukan (Ujian)	-1	0	1	74	75	76
Masukan (Tugas)	15	15	15	15	15	15
Total Nilai	14	15	16	89	90	91
Nilai Batasan		0			75	
Keluaran yang Diharapkan	FM	D	D	A	A	FM

Suatu nilai acak 15 digunakan untuk semua masukan nilai Tugas.

Sedangkan untuk nilai Tugas mempunyai nilai batasan resiko 0 dan 25, yang menghasilkan *test cases* sebagai berikut:

Test Case	7	8	9	10	11	12
Masukan (Ujian)	40	40	40	40	40	40
Masukan (Tugas)	-1	0	1	24	25	26
Total Nilai	39	40	41	64	65	66
Nilai Batasan	0			25		
Keluaran yang Diharapkan	FM	C	C	B	B	FM

Partisi ekuivalensi untuk hasil gradasi nilai juga dipertimbangkan. Batasan nilai dari partisi hasil gradasi nilai adalah 0, 30, 50, 70, dan 100.



Test cases berdasarkan pada nila batasan dari keluaran nilai gradasi tersebut di atas, adalah sebagai berikut

Test Case	13	14	15	16	17	18
Masukan (Ujian)	-1	0	0	29	15	6
Masukan (Tugas)	0	0	1	0	15	25
Total Nilai	-1	0	1	29	30	31
Nilai Batasan		0			30	
Keluaran yang Diharapkan	FM	C	D	D	C	C

Test Case	19	20	21	22	23	24
Masukan (Ujian)	24	50	26	49	45	71
Masukan (Tugas)	25	0	25	20	25	0
Total Nilai	49	50	51	69	70	71
Nilai Batasan		50			70	
Keluaran yang Diharapkan	C	E	B	B	A	A

Test Case	25	26	27
Masukan (Ujian)	74	75	75
Masukan (Tugas)	25	25	26
Total Nilai	99	100	101
Nilai Batasan		100	
Keluaran yang Diharapkan	A	A	FM

Partisi salah dari hasil nilai gradasi yang digunakan pada contoh *equivalence partitioning*, (seperti E, A+ dan *null*), tidak mempunyai batasan yang dapat diidentifikasikan, sehingga tidak dapat dibuatkan *test cases* yang berdasarkan nilai batasan-batasannya.

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 Nila (Nlai Jian + Nilai Tugas) > 100
- Total Nila (Nlai Jian + Nilai Tugas) < 0

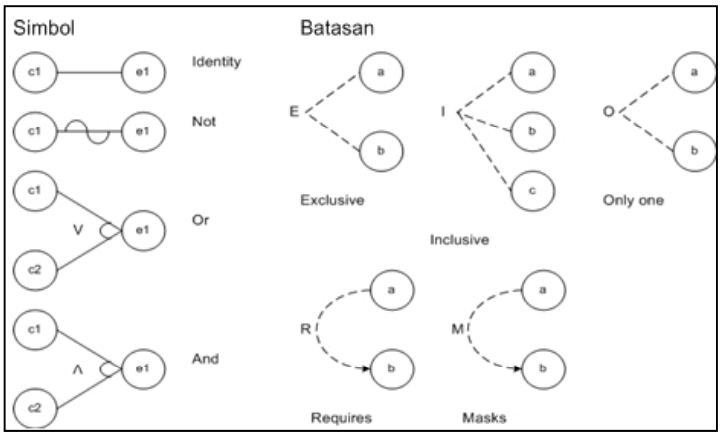
Partisi-partisi ini akan d asumsikan terikat oleh tipe data yang digunakan sebagai masukan atau keluaran. Contoh, 16 *bit integers* mempunyai batasan 32737 dan –32768.

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

Cause-Effect Graphing Techniques

- Merupakan teknik disain *test cases* yang menggambarkan logika dari kondisi terhadap aksi yang dilakukan.
- Terdapat empat langkah, yaitu:
  - Tiap penyebab (kondisi masukan) dan akibat (aksi) yang ada pada suatu modul didaftarkan.
  - Gambar sebab-akibat (cause-effect graph) dibuat.
  - Gambar di konversikan ke tabel keputusan.
  - Aturan-aturan yang ada di tabel keputusan di konversikan ke *test cases*.



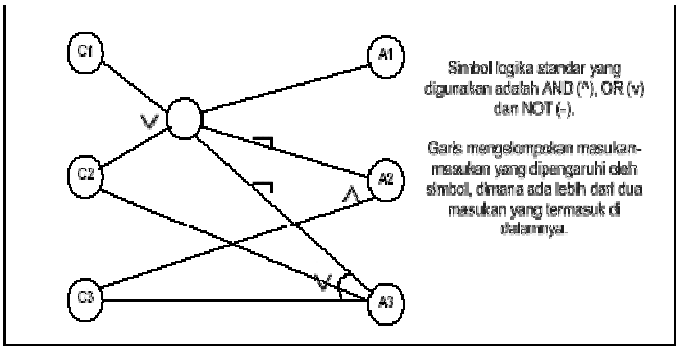
Contoh

Sebagai ilustrasi, diberikan beberapa kordisi dar aksi dari suatu fungsi debit cek, sebagai berikut:

- Kondisi:
  - C1 – transaksi jurnal kredit baru.
  - C2 – transaksi jurnal penarikan baru, namun dalam batas penarikan tertentu.
  - C3 – perkiraan mempunyai pos jurnal.
- Aksi:
  - A1 – pemrosesan debit.
  - A2 – penundaan jurnal perkiraan.
  - A3 – pengiriman surat.

Dimana spesifikasi fungsi debit cek:

- Jika dana mencukupi pada perkiraan atau transaksi jurnal baru berada di dalam batas penarikan dana yang diperkenankan, maka proses debit dilakukan.
- Jika transaksi jurnal baru berada di luar batas penarikan yang diperkenankan, maka proses debit tidak dilakukan.
- Jika merupakan perkiraan yang mempunyai pos jurnal maka proses debit ditunda. Surat akan dikirim untuk semua transaksi perkiraan mempunyai pos jurnal dan untuk perkiraan yang tidak mempunyai pos jurnal, bila dana tidak mencukupi.



Grafik *cause-effect* kemudian diformulakan dalam suatu tabel keputusan. Semua kombinasi benar (*true*) dan salah (*false*) untuk kondisi masukan dimasukan, dan nilai benar dan salah dialokasikan terhadap aksi-aksi (tanda * digunakan untuk kombinasi dari kondisi masukan yang tidak fisibel dan secara konsekuen tidak mempunyai aksi yang memungkinkan).

Aturan	1	2	3	4	5	6	7	8
C1: transaksi jurnal kredit baru	F	F	F	F	T	T	T	T
C2: transaksi jurnal penarikan baru, tapi dengan batas penarikan tertentu.	F	F	T	T	F	F	T	T
C3: jurnal yang mempunyai pos perkiraan.	F	T	F	T	F	T	F	T
A1: pemrosesan debit.	F	F	T	T	T	T	*	*
A2: penundaan jurnal perkiraan.	F	T	F	F	F	F	*	*
A3: pengiriman surat.	T	T	T	T	F	T	*	*

Kombinasi masukan yang fisibel can umuk kemudian dicakup oleh test cases, adalah sebagai berikut:

Test case	Sebab (cause)				Akibat (effect)	
	Tipe perkiraan	Batas penarikan	Nilai perkiraan saat ini	Jumlah debit	Nilai perkiraan baru	Kode aksi
1	kredit	100	-70	50	-70	L
2	tunda	1500	420	2000	420	S & L
3	kredit	250	650	800	-150	D & L
4	tunda	750	-500	200	-700	D & L
5	kredit	1000	2100	1200	900	D
6	tunda	500	250	150	100	D & L

Terimakasih