



IF2250 Dasar Rekayasa Perangkat Lunak

Pengujian Perangkat Lunak

Oleh: Bayu Hendradjaya

Pengajar:

Bayu Hendradjaya/Christine Suryadi



*Slide kuliah bisa didownload dari
kuliah.itb.ac.id (subyek IF2250)*

Kenapa Perlu Pengujian

- Mariner I space probe (1962)
 - Suatu rumus yang ditulis dengan pensil, akhirnya salah diterjemahkan oleh pemrogram
- NASA's Mars lander: September 1999
 - Jatuh karena ada kesalahan dalam integrasi suatu unit
- THERAC-25 radiation machine (1985) :
 - Pengujian kurang lengkap, hingga sempat menyebabkan 3 pasien meninggal
- Roket Ariane 5 (1996)
 - Meledak karena kesalahan penanganan konversi komponen 16 bit ke 64 bit
- Rudal Patriot (1991)
 - Salah sasaran karena kesalahan pembulatan (round off error)
- Millenium Bug (Y2K Bug)
 - Penghilangan digit dua digit awal pada data tahun menjadi masalah ketika pergantian menuju tahun 2000
- Heartbleed (2014)
 - Kerentanan pada OpenSSL
- Video Game 'Pac Man' (2012)
 - Split screen di Level 256

http://en.wikipedia.org/wiki/List_of_software_bugs



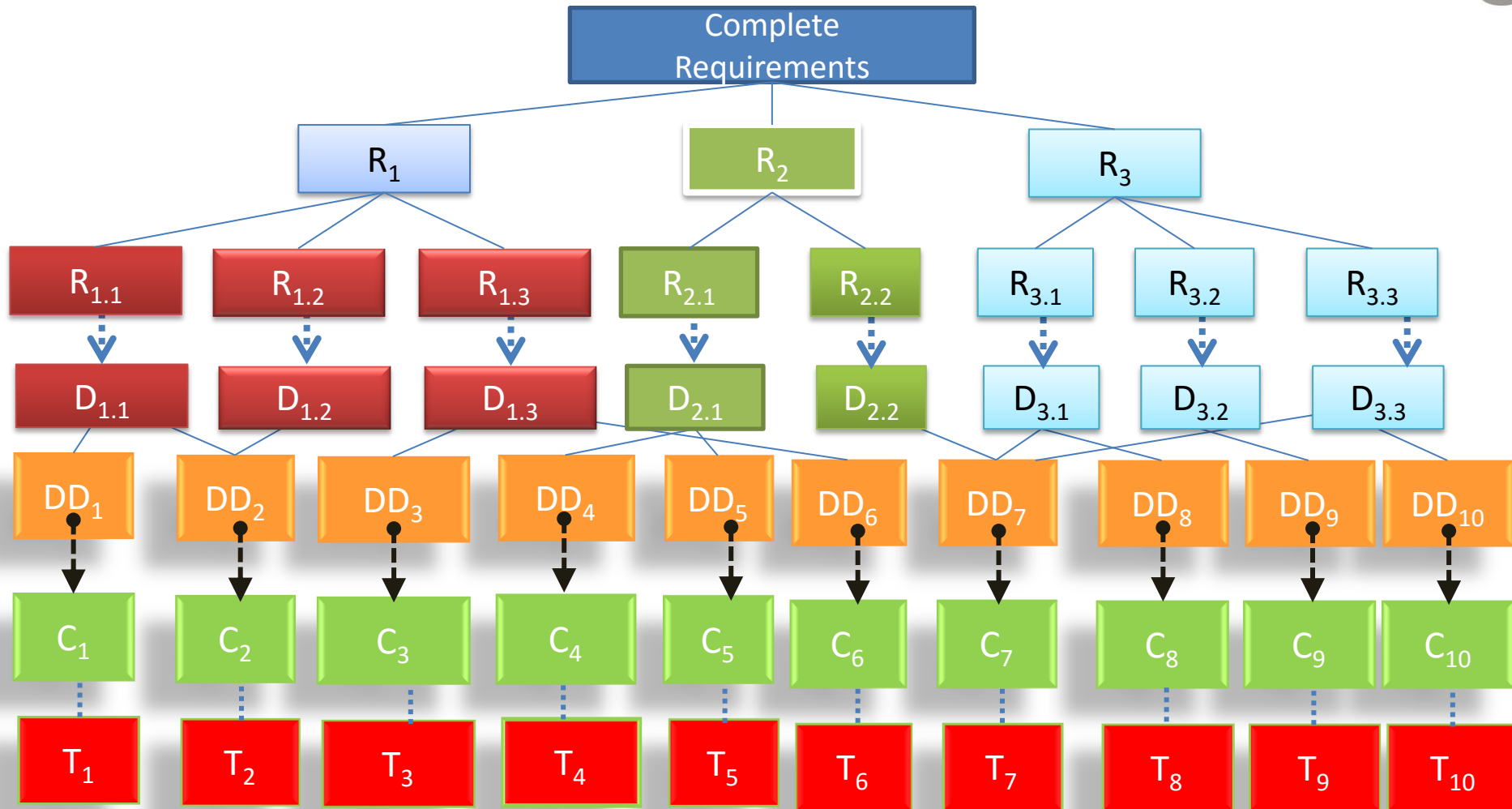


Pengujian Perangkat Lunak

Untuk “Penguji” (bukan pemrogram),
Pengujian adalah proses mencoba
program dengan tujuan mencari kesalahan
sebelum akhirnya diberikan ke pengguna

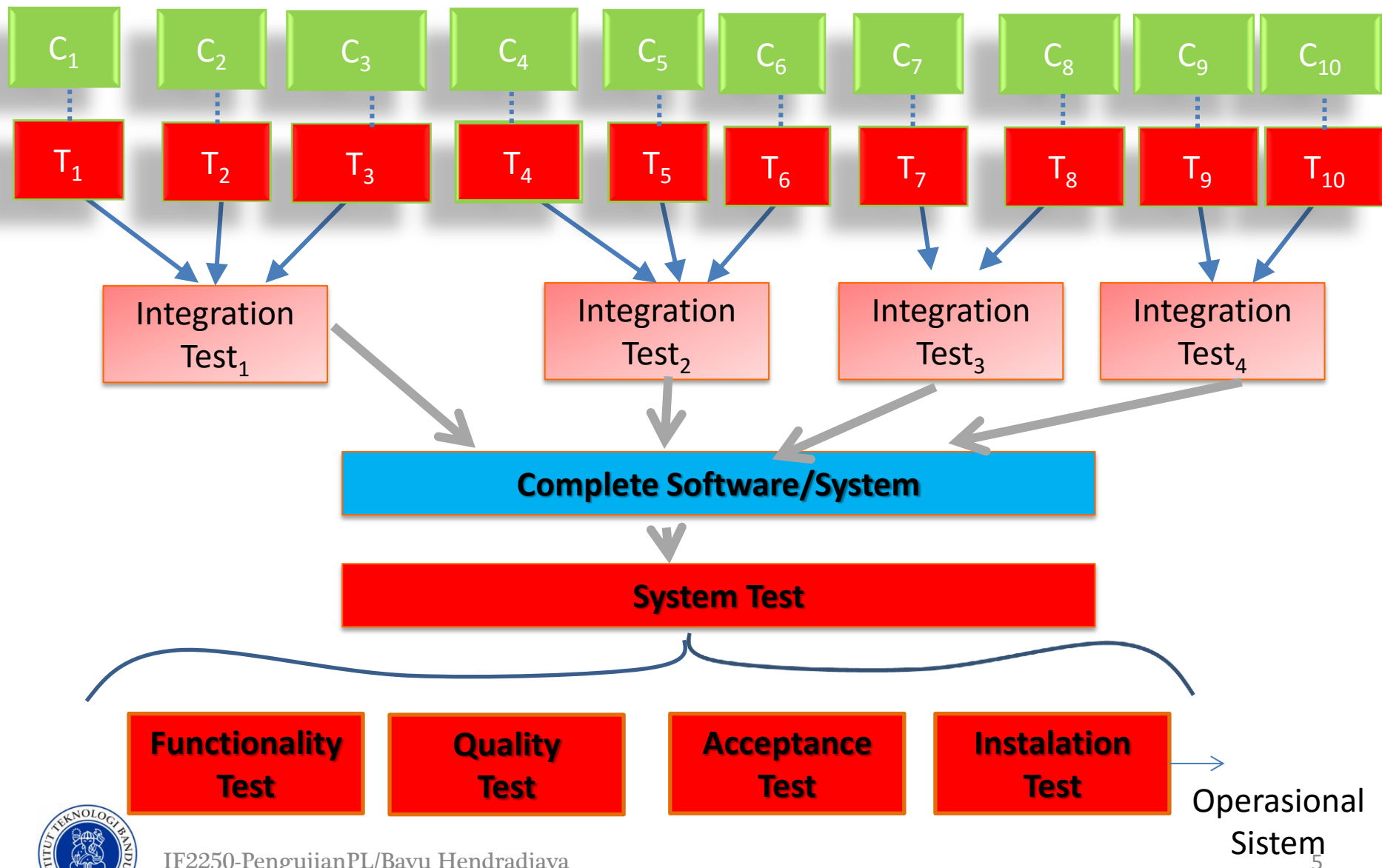
Untuk “Pemrogram” yang menguji program,
Pengujian adalah proses mencoba program
dengan tujuan menunjukkan tidak ada salah
pada programnya

Aktivitas Pengujian

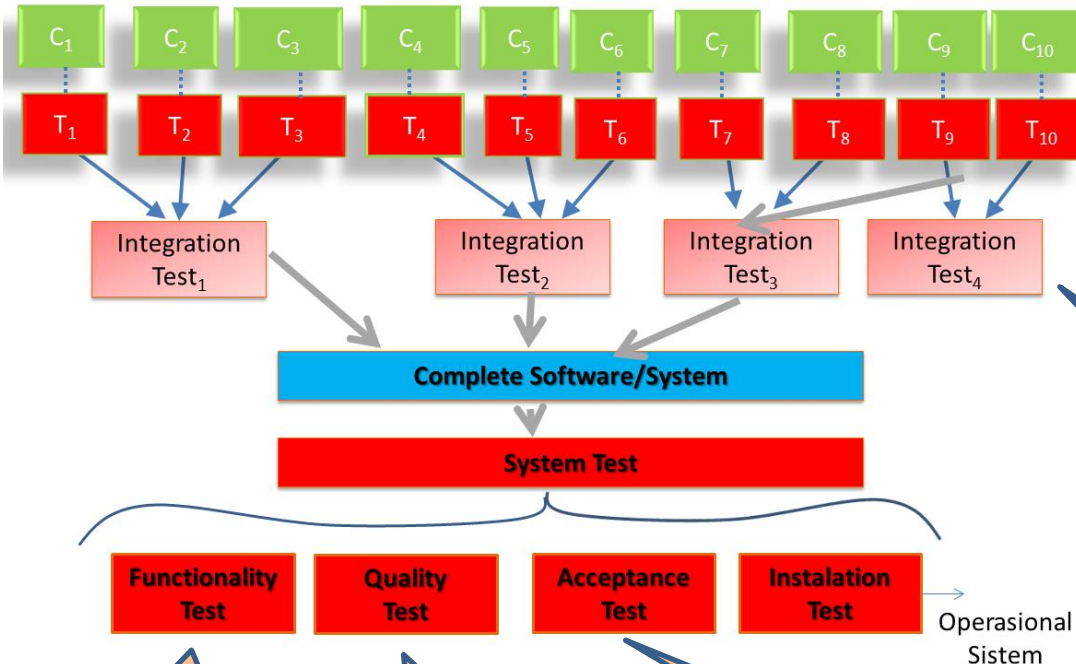


Pemrogram harus melakukan
Pengujian terhadap kode programnya

Setiap unit program harus digabung dan hasil penggabungannya diuji kembali (Integration Testing)



Peran tiap pengujian



Memastikan tiap Komponen/Unit harus sesuai spesifikasi

Memastikan tiap Komponen/Unit harus dapat bekerjasama

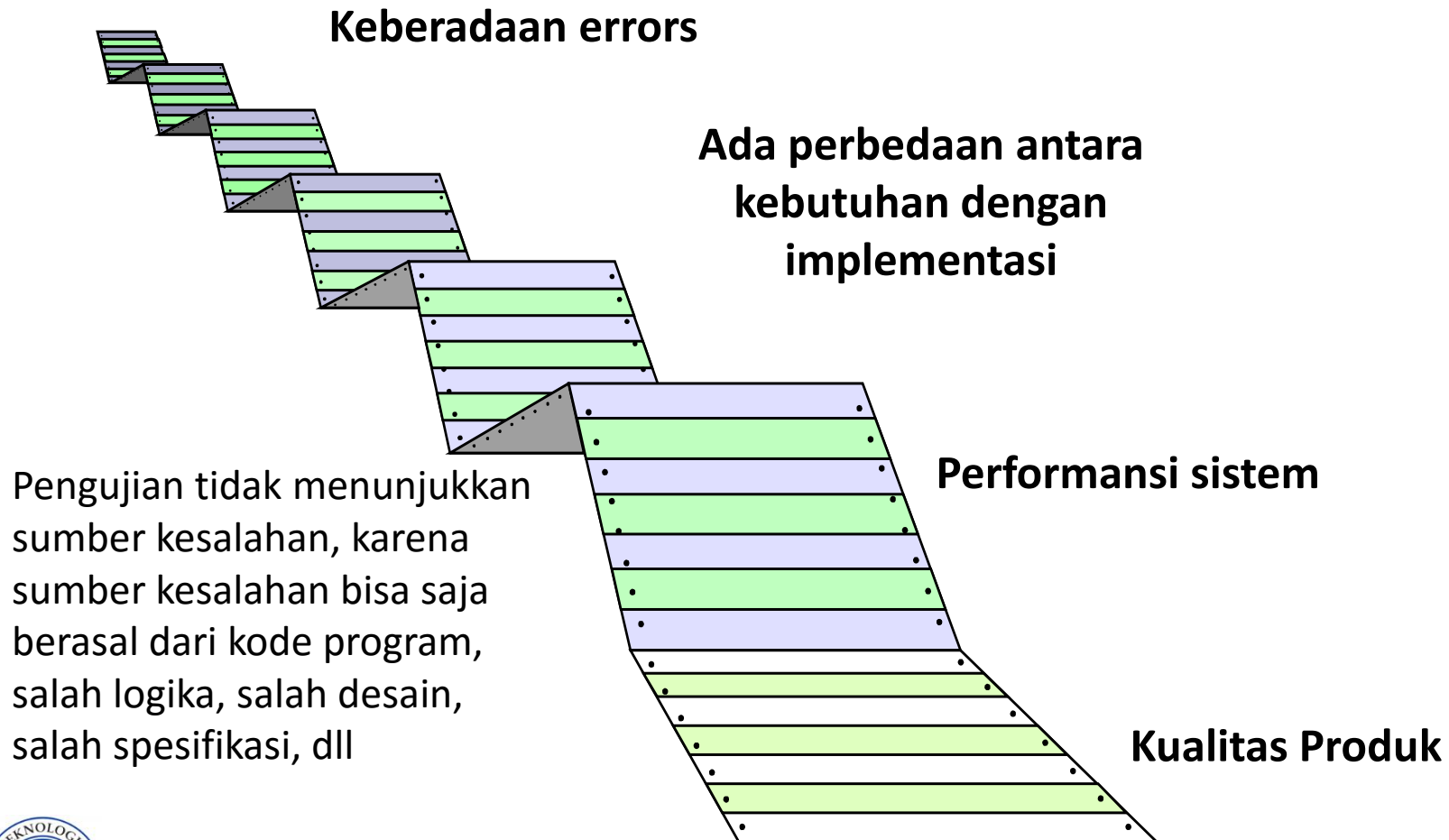
Verifikasi tiap Kebutuhan Fungsional sudah terpenuhi

Verifikasi tiap Kebutuhan Non-Fungsional sudah terpenuhi

Verifikasi dari Calon Pengguna/Customer

Pengujian di lingkungan user

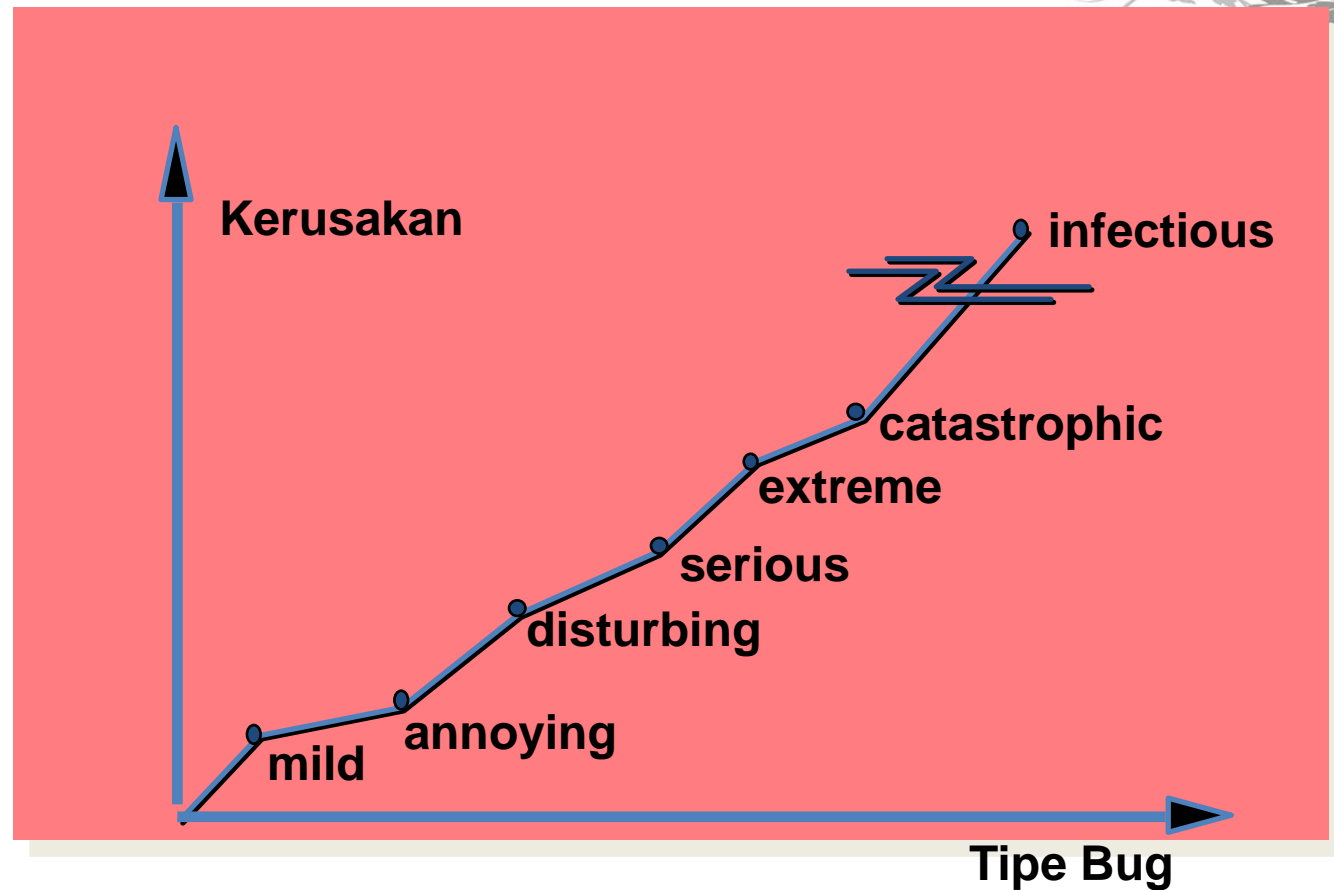
Pengujian memberikan indikasi:



Failure, Error, Fault and Defect

- **Failure**
 - Kegagalan (failure) terjadi kalau ada perilaku dari sistem yang tidak sesuai dengan permintaan di spesifikasi sistem
- **Error**
 - Error adalah status dari sistem
 - Status error ini bisa menyebabkan kegagalan jika tidak ada perbaikan
- **Fault**
 - Fault adalah sumber dari error
- **Defect**
 - Sinonim dengan **Fault**
 - Sering disebut juga **bug**

Akibat dari Bug



Kategori Bug

- Bug di Fungsi
- Bug di Sistem
- Data Bugs
- Coding Bugs
- Coding Bugs
- Design Bugs
- Documentation Bugs
- Standard Violations

Pengujian yang efektif



- Lakukan **Technical Review**
 - Banyak 'error' akan sudah ditemukan sebelum pengujian benar-benar dilakukan
- Pengujian dimulai dari elemen yang paling dasar hingga integrasi antar modul dan integrasi dengan sistem
- Teknik pengujian yang dipilih disesuaikan dengan teknik pendekatan software engineering
 - Konvensional? OO ?
- Pengujian dilakukan oleh developer
 - Untuk skala besar dilakukan oleh kelompok pengujian yang independen dari pengembang



Testing vs. Debugging



- Pengujian (testing) berbeda dengan debugging
 - Debugging dilakukan bila sudah ditemukan suatu kesalahan, dan tujuannya mencari sumber kesalahan (fault atau defect)
 - Debugging tetap menjadi bagian dari strategi pengujian

V & V



- *Verifikasi*

- Aktivitas untuk menjamin bahwa **proses implementasi** suatu fungsi sudah diimplementasikan dengan cara yang benar

- *Validasi*

- Sekumpulan aktivitas untuk menjamin bahwa **fungsi sudah dibuat dengan benar** sesuai dengan kebutuhan pengguna
 - Boehm [Boe81]:
 - *Verification*: "Are we building the product right?"
 - *Validation*: "Are we building the right product?"

Siapa yang menguji?



Pengembang

Mengerti system, tapi akan menguji untuk membuktikan bahwa programnya sudah benar



Penguji Independen

Harus belajar sistemnya, tapi akan menguji untuk membuktikan bahwa program masih ada yang salah

Aktivitas Pengujian

- Perencanaan Pengujian
- Perancangan Pengujian
- Eksekusi Pengujian
- Laporan Pengujian



Aktivitas Pengujian



- Perencanaan Pengujian
- Perancangan Pengujian
- Eksekusi Pengujian
- Laporan Pengujian

- Perencanaan **apa** saja yang akan diuji
- Perencanaan **waktu**
- Perencanaan **sumberdaya** yang diperlukan
 - Orang
 - Perangkat lunak pendukung
 - Alat/perangkat keras
 - Lingkungan pengujian
 - dan lain-lain
- Perencanaan **prosedur umum** pengujian

Aktivitas Pengujian



- Perencanaan Pengujian
- Perancangan Pengujian
- Eksekusi Pengujian
- Laporan Pengujian

- Untuk apapun yang akan diuji, maka perlu dirancang **Kasus Pengujian (Test Case)**
- Test Case berisi:
 - **Masukan yang diuji**
 - **Harapan Pengujian**
 - **Cara menilai hasil eksekusi pengujian**

Aktivitas Pengujian



- Perencanaan Pengujian
- Perancangan Pengujian
- Eksekusi Pengujian
- Laporan Pengujian

- Dilakukan sesudah kode program selesai dikerjakan.
- Eksekusi program dilakukan dengan menggunakan masukan data yang sudah di rancang

Aktivitas Pengujian



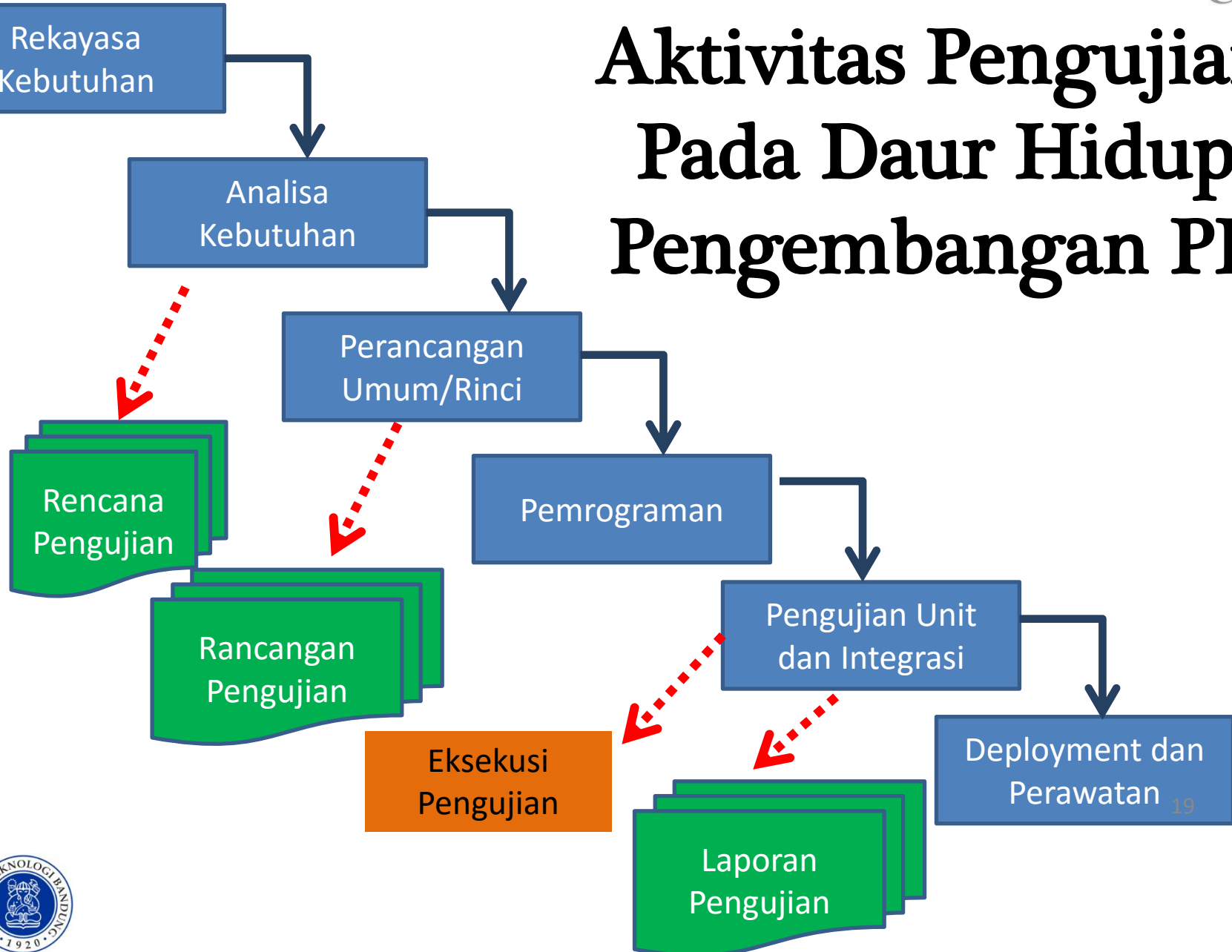
- Perencanaan Pengujian
- Perancangan Pengujian
- Eksekusi Pengujian
- Laporan Pengujian

Laporan pengujian berisi laporan untuk setiap hasil eksekusi Test Case:

- **Lulus pengujian** jika hasil eksekusi program memberikan hasil yang sama dengan harapan hasil
- **Tidak lulus pengujian** jika hasil eksekusi memberikan hasil yang berbeda
- Kadang dapat disertai dengan catatan khusus



Aktivitas Pengujian Pada Daur Hidup Pengembangan PL



Strategi Pengujian



- Dimulai dari yang kecil ('testing-in-the-small') hingga bagian yang lebih besar ('testing-in-the-large')
- Untuk software biasa
 - Fokus awal adalah pengujian modul (unit)
 - Diikuti dengan pengujian integrasi modul
- Untuk software OO
 - Fokusnya pada pengujian pada setiap kelas yang melibatkan atribut dan operasinya
 - Berikut komunikasi/kolaborasi antar kelas



Pengujian Unit

atau Pengujian Modul atau Komponen

(Unit Testing, Module Testing or Component Testing)



Proses Pengujian

Pembuatan Kasus Uji

```
1 #include <stdio.h>
2
3 int Kuadrat(int A)
4 {
5     int hasil;
6     hasil = 0;
7     for (int i = 0; i < A; i++)
8     {
9         hasil = hasil + A;
10    }
11    return hasil;
12 }
```

**Unit Program
yang akan diuji**

Masukan	Harapan Hasil
A=0	return 0
A=1	return 1
A=2	return 4
A=3	return 9
A=10	return 100

Perbaiki Kode

Hasil:

- Kasus uji no x tidak lolos; atau
- Semua kasus uji lolos;

Eksekusi Kasus Uji

Selesai



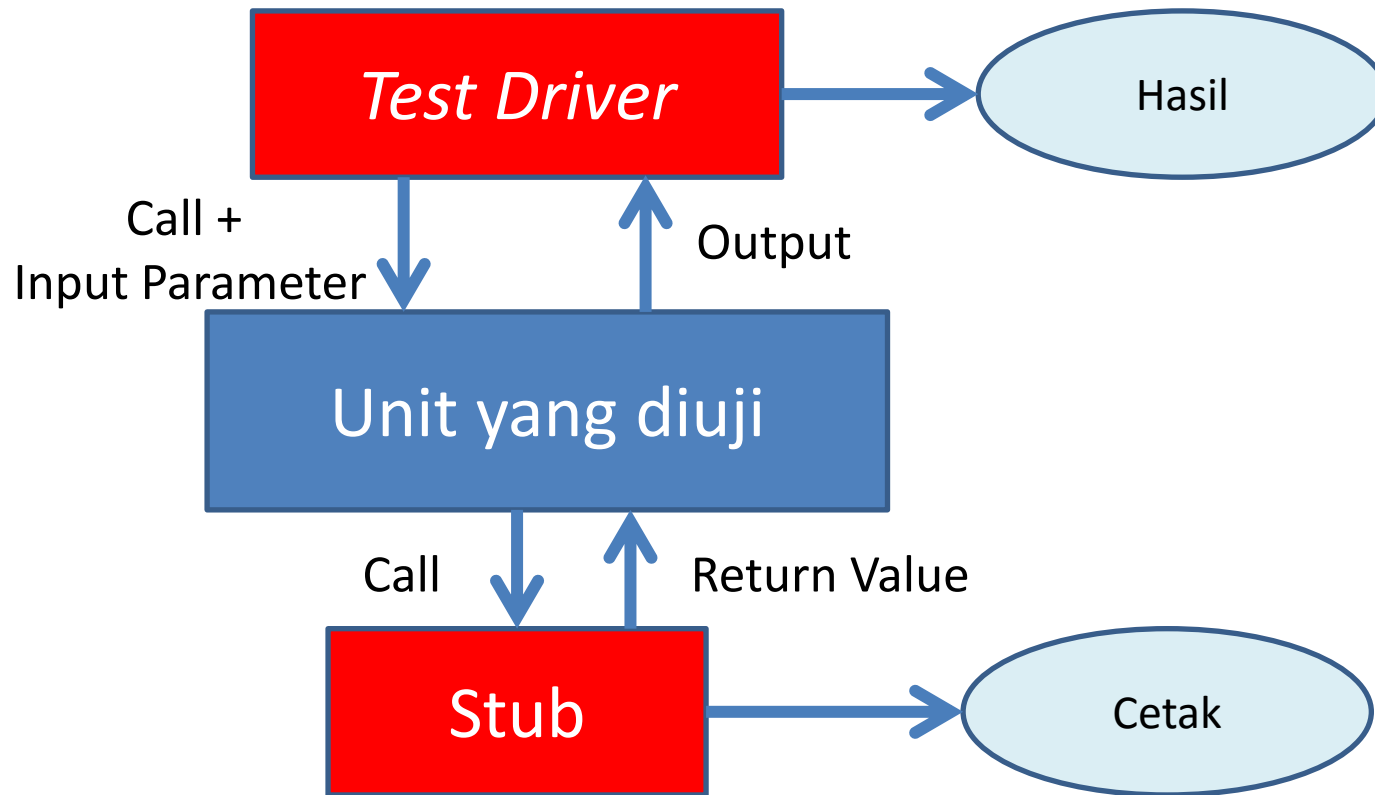
Lingkungan Pengujian



- Lingkungan untuk menguji harus disiapkan
 - Perlu computer, perlu compiler, perlu persiapan jaringan, dan lain-lain
- Untuk pengujian Unit, maka mungkin saja ada bagian program lain yang belum siap, tetapi pengujian unit tetap harus dilakukan
 - Sehingga perlu disiapkan **TEST DRIVER** dan **STUB**
- *Test driver* adalah program yang memanggil unit yang sedang diuji, sekaligus memberikan input data, dan memeriksa hasilnya
- Bila unit yang akan diuji, perlu memanggil unit lain yang belum selesai di program, maka dipersiapkan STUB untuk meng-emulasi peran unit lain tadi.
 - Bagian ini disebut juga sebagai program dummy
- Test driver dan stub secara bersama-sama disebut **scaffolding**



Lingkungan Pengujian Unit



Unit Under Test

```
int HitungGajiPegawai(int NIP)
{
    int GaPok = ReadGajiPokok(NIP);
    int GaTun = ReadGajiTun( NIP);

    return GaPok + GaTun;
}
```

STUB

```
int ReadGajiPokok(int NIP)
{
    // execute sql statement
    // select gaji from Pegawai
    return 2500;
}
```

```
int ReadGajiTun(int NIP)
{
```

```
    execute sql statement
    select tunj from Pegawai
    return 75000;
}
```

Test Driver

```
int main()
{
    assert(HitungGajiPegawai(111), 10000);
    assert(HitungGajiPegawai(200), 5000);
}
```

Unit Under Test



```
int HitungGajiPegawai(int NIP)
{
    int GaPok = ReadGajiPokok(NIP);
    int GaTun = ReadGajiTun( NIP);

    return GaPok + GaTun;
}
```

- Procedure HitungGajiPegawai yang ditulis dalam bahasa C ini adalah bagian yang diberikan oleh user untuk diuji oleh penguji
- Dalam pengembangan software skala menengah, pengujian unit ini dapat dilakukan oleh pengembang sendiri. Hal ini dilakukan untuk meminimalisasi biaya. Tentunya, pengembang biasanya menggunakan alat bantu seperti JUnit (java), NUnit (.NET), atau PHPUnit (PHP).



- Stub dibuat oleh penguji
 - Jika pengujian unit dilakukan oleh programmer, maka stub dibuat oleh programmer
- Stub dibuat karena bagian program itu belum siap atau belum selesai dikerjakan
 - Biasanya Stub dibuat untuk bagian yang ber-interface ke sistem lain.
 - Pada contoh ini, kedua stub seharusnya memanggil suatu perintah SQL ke basisdata, tetapi pada saat itu, pengembangan basisdata masih belum siap/selesai.
- Stub dibuat agar memungkinkan program yang diuji dapat di-compile dan melakukan fungsi yang diharapkan
 - Sehingga menghindari ketergantungan pada modul/unit lain yang belum selesai

STUB

```
int ReadGajiPokok(int NIP)
{
    // execute sql statement
    // select gaji from Pegawai
    return 2500;
}
```

```
int ReadGajiTun(int NIP)
{
    // execute sql statement
    // select tunj from Pegawai
    // return 75000;
}
```





- Test driver dikembangkan oleh penguji
 - Dan kalau program cukup kecil, maka programmer bertanggung jawab untuk mengembangkan test driver ini
- Test driver akan memanggil unit yang akan diuji
 - Test driver ini biasanya berisi beberapa kasus uji yang terkait dengan modul yang akan diuji
 - Test driver juga bisa berisi urutan skenario yang menggunakan unit yang diuji
- Test driver juga perlu memperhatikan penggunaan atau penyiapan stub
 - test driver menyesuaikan pemakaian stub sesuai kebutuhan, atau
 - Stub menyesuaikan dengan kebutuhan test driver
- Pada contoh dibawah ini, test driver menggunakan perintah 'assert' untuk melihat apakah fungsi HitungGajiPegawai mengembalikan suatu nilai
 - Dengan parameter 111, diharapkan mengembalikan nilai 10000

Test Driver

```
int main()
{
    assert(HitungGajiPegawai(111), 10000);
    assert(HitungGajiPegawai(200), 5000);
}
```



Pengujian yang Baik

- Memiliki kemungkinan tinggi untuk menemukan error
- Tidak redundan (duplikasi yang tidak perlu)
- Pilih teknik yang terbaik
 - Atau tepat sesuai dengan karakteristik software yang diuji
- Tidak terlalu sederhana juga tidak terlalu kompleks

Kasus Uji



Ada masukan data

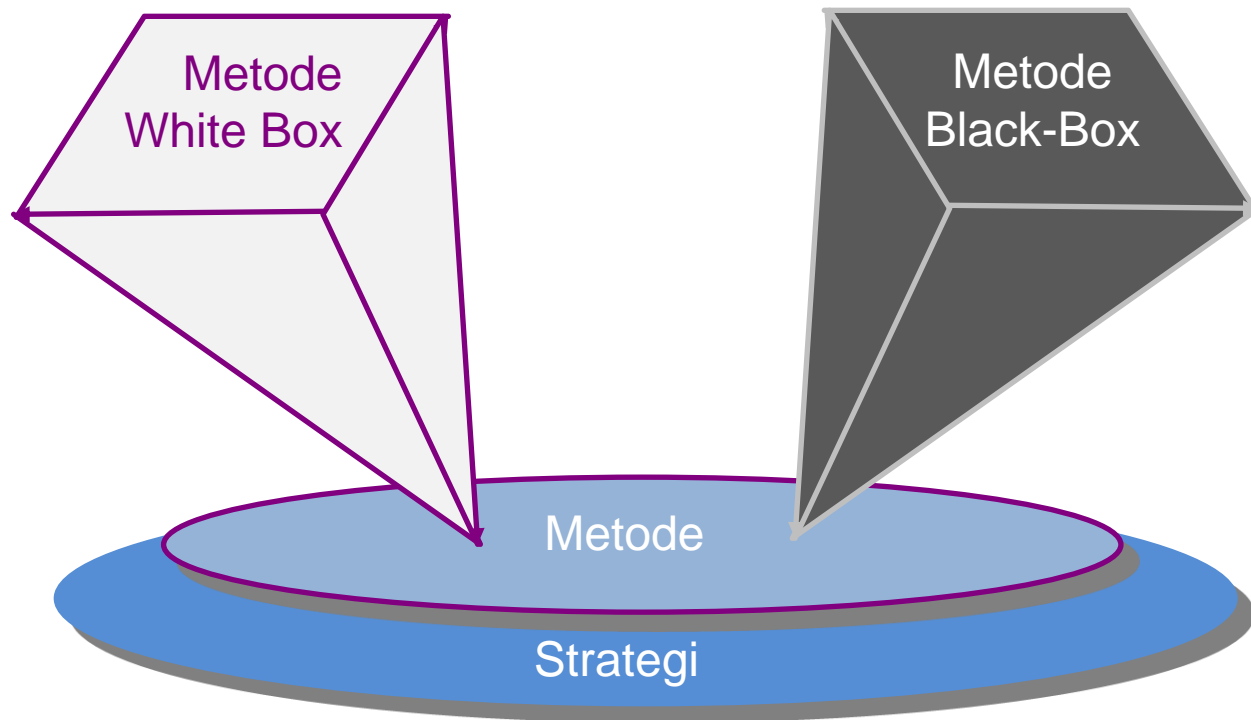
Ada harapan hasil
(expected result/outcome)

Masukan	Harapan Hasil
A=0	return 0
A=1	return 1
A=2	return 4

Contoh perhitungan saldo Bank

Masukan	Harapan Hasil
1) Saldo 50000 2) Ambil uang 20000	Saldo akhir=30000

Metode/Strategi Pengujian



White Box Testing

Penguji bisa membaca kode program(bisa melihat bagian pengulangan, statement kondisi, pemanggilan prosedur/fungsi, dan lain-lain), sehingga kasus uji dibuat berdasarkan informasi dari kode program ini

Black Box Testing

Penguji tidak membaca source-code, sehingga pengembangan kasus uji berdasarkan spesifikasi fungsional dari program. Sering disebut juga sebagai pengujian fungsional



WHITE BOX TESTING

Contoh:

- Control Flow Testing
- Data Flow Testing
- Mutation Testing
- dan lain-lain



Control Flow Testing

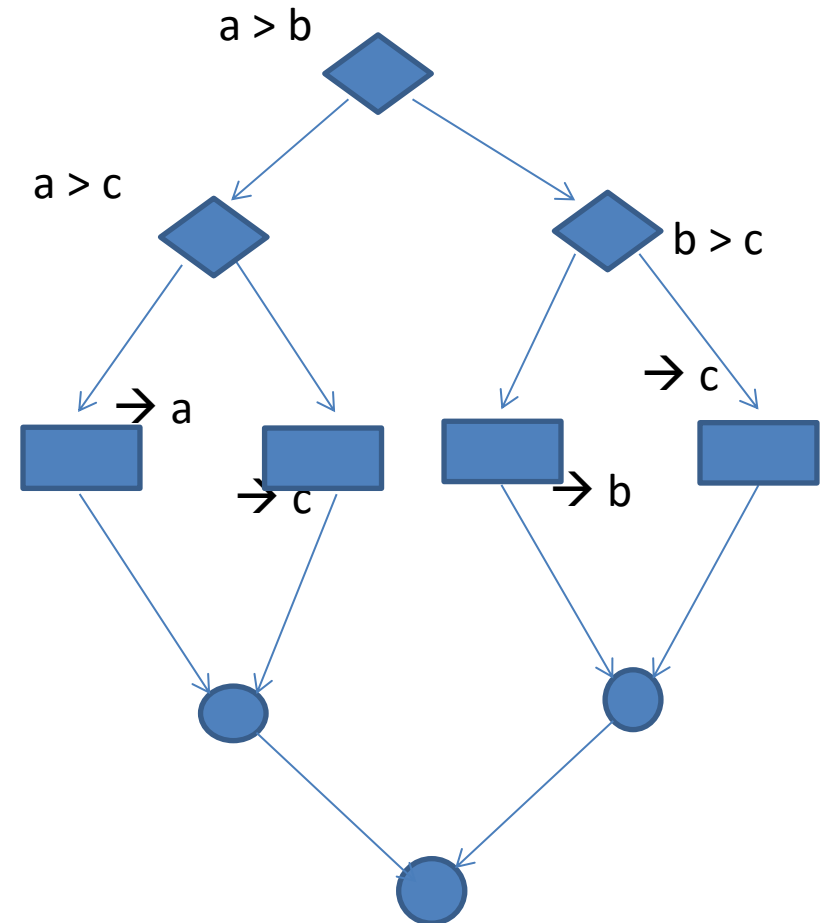
- Langkah
 - Kode Program diubah menjadi Control Flow Graph
 - Dilihat jalur eksekusinya
 - Setiap jalur eksekusi harus dilewati oleh kasus uji

Contoh Mencari Bilangan Terbesar

```
1 Function Largest( a, b, c : integer) : integer
2 Kamus:-
3 Algoritma
4
5     if a > b then
6         if a > c then
7             return a
8         else
9             return c
10    else
11        if b > c then
12            return b
13        else
14            return c
```

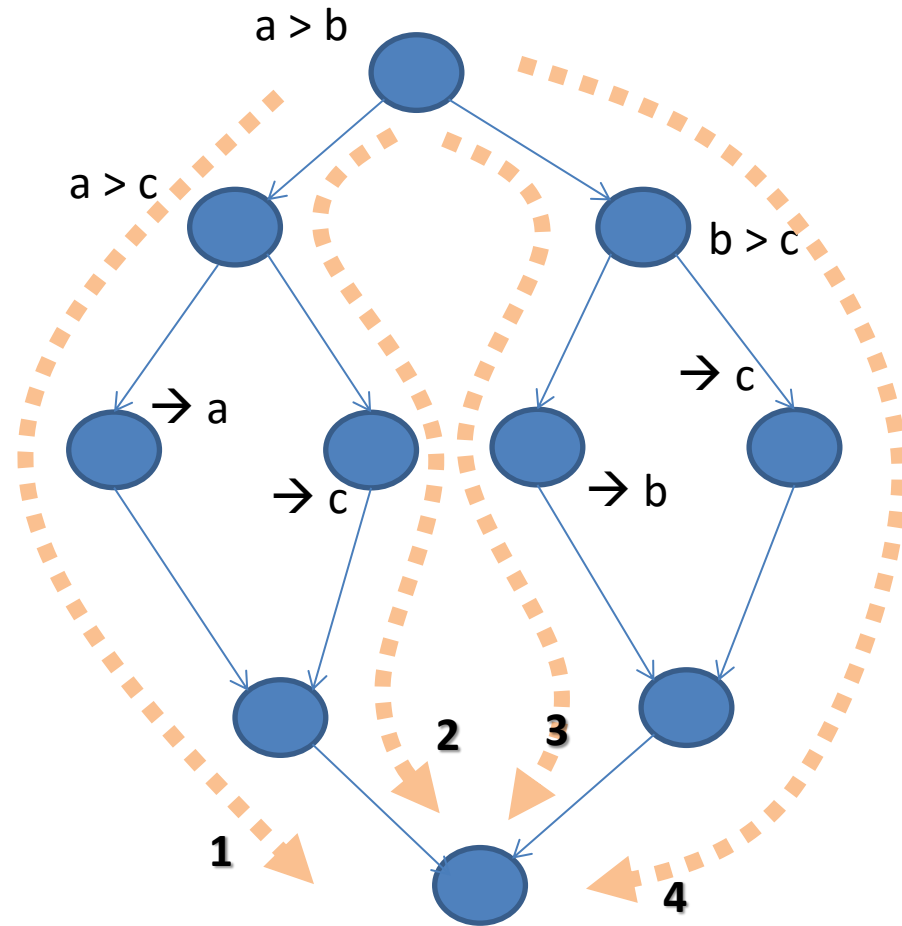
CFG


```
if a > b then
  if a > c then
    return a
  else
    return c
else
  if b > c then
    return b
  else
    return c
```



CFG dapat disederhanakan menjadi kumpulan nodes saja

```
if a > b then
  if a > c then
    return a
  else
    return c
else
  if b > c then
    return b
  else
    return c
```





Cari semua masukan yang
memungkinkan semua jalur di
lewati

Jalur	Input (a, b, c)	Hasil yang diharapkan
Jalur 1	(10, 3, 2)	10
Jalur 2	(25, 15, 40)	25
Jalur 3	(10, 15, 6)	15
Jalur 4	(10, 15, 19)	19

Buat CFG!

Dari C code berikut

```
#include <stdio.h>

int main()
{
    int x;
    a = 5;
    x = 0;
    while (x < 10)
    {
        printf( "%d %d\n", x, a );
        x = x + 1;
    }
}
```

```
#include <stdio.h>

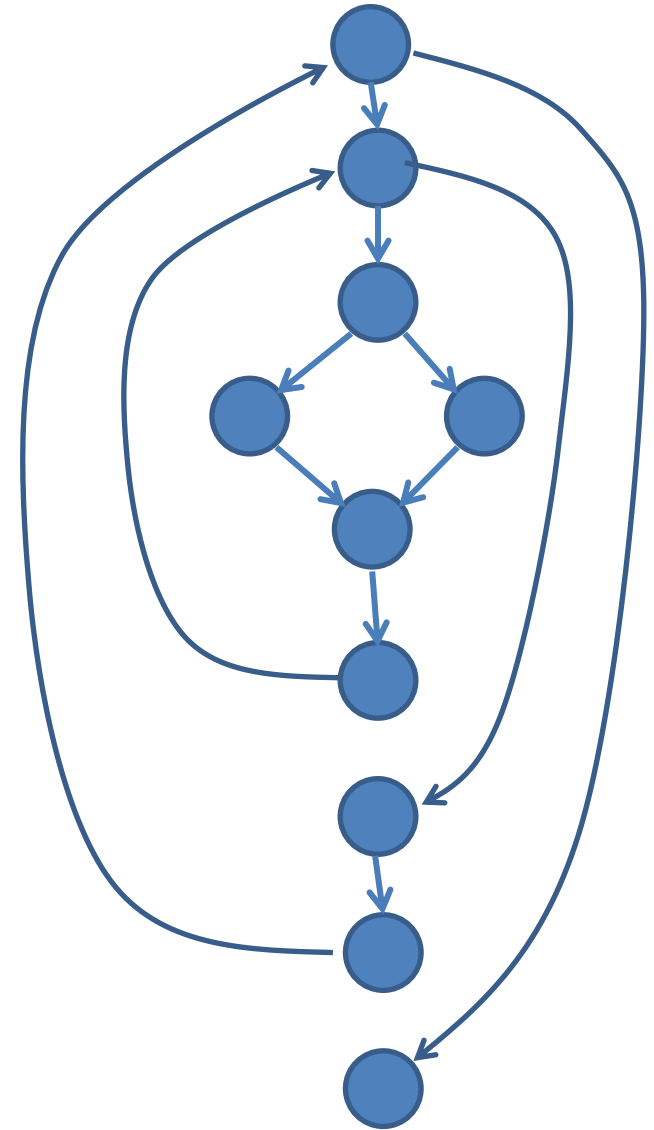
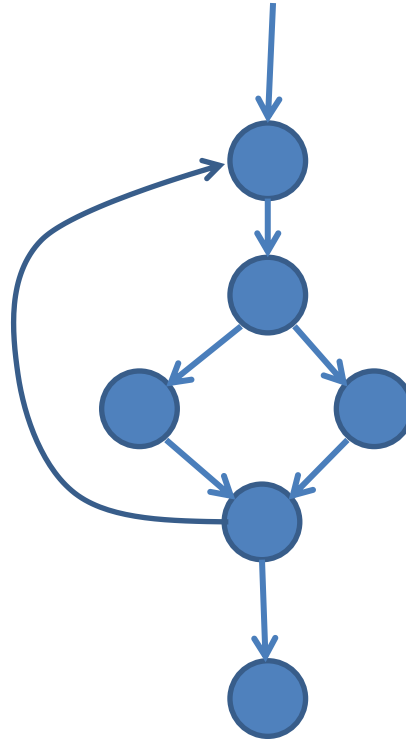
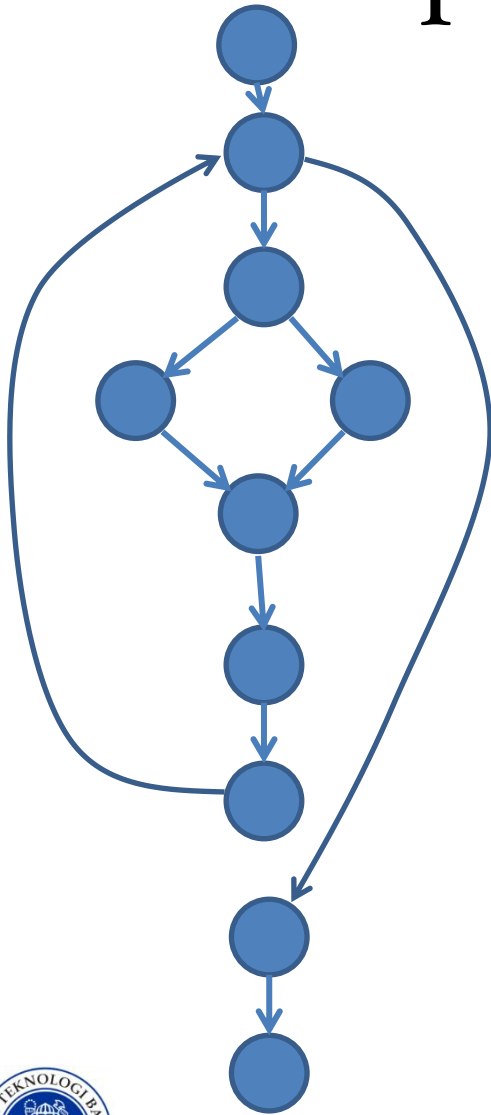
int main()
{
    int x;
    X = 0;
    do {
        if (x < 1)
            printf(“%d”, x);
        else
            printf(“%d”, x * 2);
        x = x + 1;
    } while x < 10;
}
```

```
#include <stdio.h>

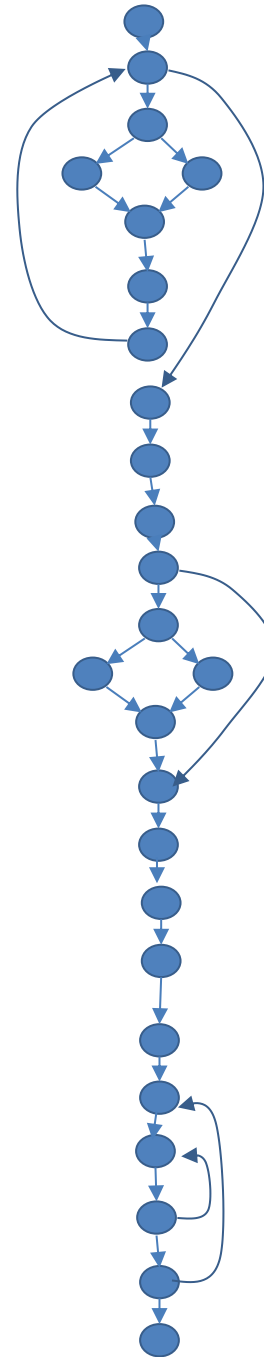
int main()
{
    int x;
    for ( x = 0; x < 10; x++ )
    {
        for (y = 10; y > 1; y--)
            printf(“%d, %d”, x, y );
    }
}
```



Bagaimana bentuk program-program ini?



...atau yang ini?





BLACK BOX TESTING

- **Equivalence Class Partitioning**
- **Boundary Value Analysis**
- Graph Based method
- Comparison Testing
- Orthogonal Array Testing
- Model Based Testing
- Dan lain-lain

Permasalahan Black-Box Testing

- Mencari masukan apa yang bagus untuk kasus uji
- Apakah suatu masukan data akan memungkinkan unit program yang diuji menjadi salah?
- Berapa jumlah masukan data yang perlu dicoba?
 - Mungkin perlu banyak, tetapi seberapa banyak?
- Ada nilai batas?
 - Apakah nilai-nilai batas sudah diperiksa pada kode program



Beberapa Teknik Black Box



- **Equivalence Class Partitioning**
- **Boundary Value Analysis**
- Graph Based method
- Comparison Testing
- Orthogonal Array Testing
- Model Based Testing
- Dan lain-lain

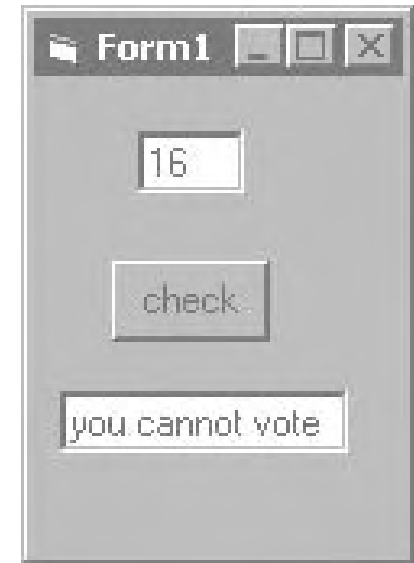


Equivalence Class Partitioning dan Boundary Value Analysis (ECP/BVA)



Contoh


- Sebuah program digunakan untuk Pemilu. Program akan menerima data masukan umur. Umur yang dibolehkan adalah 17 tahun ke atas agar bisa memilih.
 - Ada dua kelas ekivalen (ada dua partisi)



0	16	17	Tak hingga?
---	----	----	-------------

- Jadi akan hanya diperlukan dua kasus uji

Nomor Test	Data	Hasil Yang Diharapkan
1	12	You cannot vote
2	21	You can vote

- 
- Kedua test itu tentunya belum cukup, karena kita tahu bahwa test tersebut belum mencakup perbedaan batas yang penting yaitu batas umur 17 tahun. Jadi perlu diperiksa yang yang berumur 16, 17 dan 18.
 - Kenapa? Karena pemrogram bisa saja melakukan kesalahan memasukkan kondisi umur tersebut.
 - If (umur >17) atau if (umur >= 17) atau if (umur <= 16) ?
 - Jadi kita buat dua test lagi menggunakan ide Boundary Value Analysis.

Nomor Test	Data	Hasil Yang Diharapkan
3	16	You cannot vote
4	17	You can vote

- Kenapa 18 tidak perlu diperiksa lagi?
- Apakah perlu memeriksa umur 0 dan umur 100 misalnya?

Contoh lain



- Test program yang menampilkan bilangan terbesar antara dua input. Kedua input selalu dalam range 0-10000. Jika nilainya sama, maka akan ditampilkan nilainya yang sama.
 - Maka kita bisa mengambil nilai sembarang antara 0-10000 untuk keduanya:
 - Misalnya untuk input 1 diambil 500 dan dan kedua 1000
 - Kemudian diambil nilai batas, 0 dan 10000
 - Catatan: untuk kasus ini, jika range sudah ditentukan 0 sampai 10000, artinya masukan tidak mungkin kurang dari 0, atau lebih dari 10000

Kasus Uji



Nomor Test	Angka 1	Angka 2	Hasil yang diharapkan
1	0	0	0
2	0	1000	1000
3	0	10000	10000
4	500	0	500
5	500	1000	1000
6	500	10000	10000
7	10000	0	10000
8	10000	1000	10000
9	10000	10000	10000



Pengujian Kebutuhan Fungsional *atau* Pengujian Fungsional *(Functional Testing)*



Functional Testing



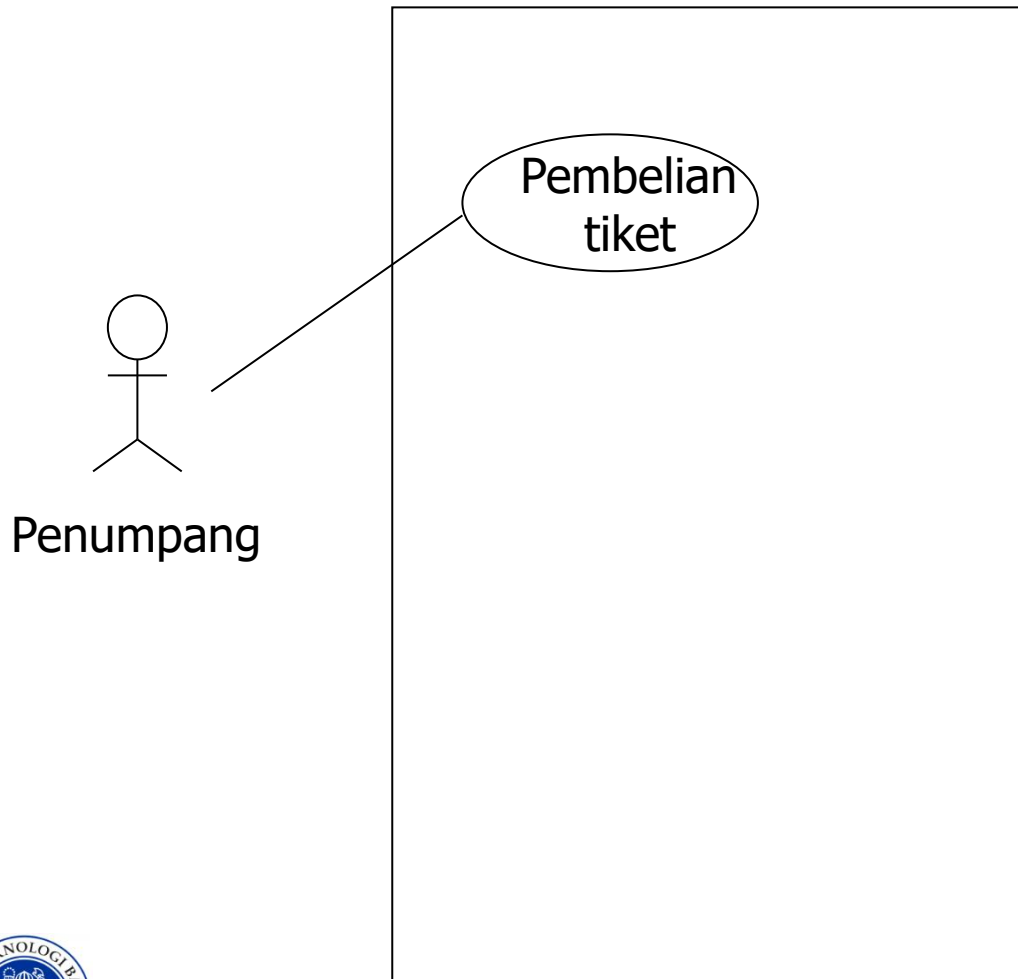
- Functional Testing adalah bagian dari System Testing
 - Contoh system testing lain: performance testing, security testing, recovery testing, acceptance testing, pilot testing dan installation testing
- Fokus dari Pengujian fungsional adalah menjamin semua kebutuhan fungsional sudah diuji
 - Hasil implementasi program harus sudah mencakup semua yang dituliskan dalam spesifikasi kebutuhan perangkat lunak
- Pengujian ini sifatnya adalah pengujian **Black-Box**
 - Memanfaatkan model use case
 - Dipilih suatu instansiasi dari use case yang kemungkinan bisa menyebabkan error
 - Bisa memanfaatkan Equivalence Class Partitioning/Boundary Testing
 - Test case dibuat berdasarkan skenario normal dan juga skenario alternatif



Studi Kasus: Mesin Tiket Kereta Api



Contoh use case pembelian karcis kereta lewat mesin tiket



Diketahui ada 5 stasiun tujuan, dengan biaya masing-masing

- stasiun 1: 1000,
- stasiun 2: 1500 ,
- stasiun 3: 2500,
- stasiun 4: 3000,
- stasiun 5: 4000

Use case: Pembelian Tiket

Kondisi Awal: Penumpang berdiri di depan mesin tiket dengan uang yang cukup

Skenario Normal:

- Penumpang memilih tujuan stasiun, dengan menekan tombol tujuan, bila ada lebih dari satu yang ditekan, maka pilihan terakhir yang diambil
- Mesin akan menampilkan biaya yang harus dibayar
- Penumpang memasukkan koin ke slot mesin
 - Alternatif skenario
 - Jika penumpang memasukkan stasiun baru sebelum uang yang dimasukkan cukup, maka mesin akan mengembalikan seluruh koin
 - Jika penumpang memasukkan jumlah koin lebih, maka pengembalian akan diberikan oleh mesin
- Mesin akan mencetak tiket
- Penumpang akan mengambil tiket dan kembalian jika ada

Kondisi akhir: Tiket sudah ditangan penumpang



Ide pengujian

- Dari use-case tersebut, maka ada beberapa fitur dari mesin tiket tersebut yang harus diuji
 - Jika penumpang menekan tombol dan memasukkan jumlah koin yang tepat, maka ia akan mendapatkan tiket saja.
 - Jika penumpang menekan tombol untuk beberapa stasiun, maka mesin harus menampilkan harga tiket stasiun terakhir
 - Jika penumpang menekan tombol stasiun lain, selagi memasukkan uang ke mesin, maka uang otomatis dikembalikan dulu
 - Jika penumpang memasukkan uang yang melebihi yang harus dibayar, mesin harus melakukan pengembalian

Test Case 1



Nama Test case	TC1: Uang Pas
Kondisi awal	Penumpang di depan mesin, dan memiliki koin 5 koin 1000, 2 koin 500
Skenario	<ol style="list-style-type: none">1. Penumpang menekan tombol stasiun 3 dan mesin akan menampilkan 25002. Penumpang memasukkan dua koin 1000 dan satu koin 500
Kondisi akhir	Penumpang mendapatkan tiket untuk stasiun 3 dan tidak ada koin kembalian

Test Case 2



Nama Test case	TC2: Penekanan tombol berbeda
Kondisi awal	Penumpang di depan mesin, dan memiliki koin 5 koin 1000, 2 koin 500
Skenario	<ol style="list-style-type: none">1. Penumpang menekan tombol stasiun 5, 3, 1, 2, pada saat yang bersamaan setiap kali mesin akan menampilkan 4000, 2500, 1000, lalu 15002. Penumpang memasukkan dua koin 10003. Mesin harus mengembalikan koin sejumlah 500
Kondisi akhir	Penumpang mendapat tiket ke stasiun 2 dan mendapatkan pengembalian koin 500

Test Case 3



Nama Test case	TC3: pemilihan stasiun lain saat koin sudah dimasukkan
Kondisi awal	Penumpang di depan mesin, dan memiliki koin 5 koin 1000, 2 koin 500
Skenario	<ol style="list-style-type: none">1. Penumpang menekan tombol stasiun 5 dan mesin akan menampilkan 40002. Penumpang memasukkan dua koin 10003. Lalu penumpang menekan tombol 34. Mesin harus langsung mengembalikan koin sejumlah 2000
Kondisi akhir	Penumpang mendapat Koin kembali sejumlah 2000

Test Case 4



Nama Test case	TC4: Pengembalian Koin
Kondisi awal	Penumpang di depan mesin, dan memiliki koin 5 koin 1000, 2 koin 500
Skenario	<ol style="list-style-type: none">1. Penumpang menekan tombol stasiun 3 dan mesin akan menampilkan 25002. Penumpang memasukkan tiga koin 10003. Mesin harus mengembalikan koin sejumlah 500
Kondisi akhir	Penumpang mendapatkan tiket dan koin kembali sejumlah 500

Pengujian System



- Validation testing
 - Fokus pada kebutuhan perangkat lunak (software requirements)
- System testing
 - Fokus pada integrasi sistem
- Alpha/Beta testing
 - Fokus pada pengguna (bagaimana mereka menggunakan)
- Recovery testing
 - Software dibuat 'gagal' dan melihat apakah proses recovery sudah dilakukan.
 - Contoh: software yang memanfaatkan internet dilihat perilaku recovery-nya bila internet tiba-tiba putus
- Security testing
 - Memverifikasi apakah mekanisme proteksi sudah melindungi suatu 'penetrasi'
- Stress testing
 - Sistem diuji bagaimana jika menghadapi permintaan pemakaian sumberdaya yang melebihi jumlah normal, atau frekuensi atau volume normal
- Performance Testing
 - Menguji performansi software saat dieksekusi pada suatu konteks tertentu.

Tugas 9 (Tugas Terakhir)

- Dari Usecase yang sudah dibuat, buatlah pengujian fungsional berdasarkan scenario yang ada pada use case
 - Satu scenario mungkin memiliki satu atau beberapa fungsi yang perlu diuji
 - Satu fungsi yang diuji memiliki satu atau lebih kasus uji (test case)
 - Setiap kasus uji harus dilengkapi data masukan, dan hasil yang diharapkan
 - Untuk tugas 9 ini, anda hanya melakukan perencanaan dan pembuatan deskripsi pengujian (kasus uji), tidak perlu melakukan eksekusi pengujian juga tidak perlu membuat laporan hasil pengujian.
 - Penulisan laporan menggunakan dokumen GL03 (PDHUPL) yang bisa didownload dari situs kuliah di <http://kuliah.itb.ac.id>
 - Bab 1 akan mirip dengan deskripsi di SKPL
 - Bab 2 akan berisi persiapan lingkungan pengujian (anda harus memberikan usulan lingkungan persiapan pengujian yang 'sewajar' nya.
 - Bab 3 akan dimulai dengan penjelasan terkait Use Case yang sudah dibuat di tugas sebelumnya, pada sub bab 3 akan diberikan deskripsi dari pengujian.
 - Perhatikan nanti catatan yang ada di dokumen GL03.



Pengiriman Tugas



- Tugas dikirim online ke:
informatikaitb@gmail.com
- Subjek: **[if2250] Tugas 9 kelompok “n” kelas “m”**
- Message: Nama-nama anggota kelompok
- Attach dokumen tugas (format PDF)
 - Nama file: IF2250-Tugas09 KelompokXXKelasYY.pdf
 - 2 digit untuk nomor kelompok dan kelas (didahului dengan 0 untuk kelompok 1-9)
 - Contoh: IF2250-Tugas09-Kelompok15Kelas04.pdf (adalah nama file tugas ketujuh dari kelompok 15, dan kelas 4).
- Tugas sudah di terima di email sebelum jam 19:00wib Kamis 28 April 2016
 - Ganti “n” dengan nomor kelompok hasil pembagian tugas
 - Ganti “m” dengan nomor kelas: 1 atau 2
 - Bila tidak ikut ketentuan di atas, nilai akhir tugas akan dikurangi 25%

