Strategi Testing

Dr. Karmilasari

Strategi Testing

- Pengujian PL dilakukan untuk tujuan **menemukan kesalahan yang dibuat secara tidak sengaja saat PL tersebut dirancang dan dibangun**
- Strategi pengujian PL menyediakan petunjuk yang menjelaskan langkah-langkah yang harus dilaksanakan sebagai bagian dari pengujian, kapan langkah-langkah ini direncanakan dan kemudian dilakukan dan berapa banyak saha, waktu serta sumber daya yang harus disertakan dalam pengujian tersebut.
- Strategi pengujian harus menyertakan : perencanaan pengujian, perancangan kasus pengujian, pelakasanaan pengujian dan evaluasi serta pengumpulan data hasil pengujian.

Strategi Testing

Pola baku untuk pengujian PL memiliki karakteristik umum :

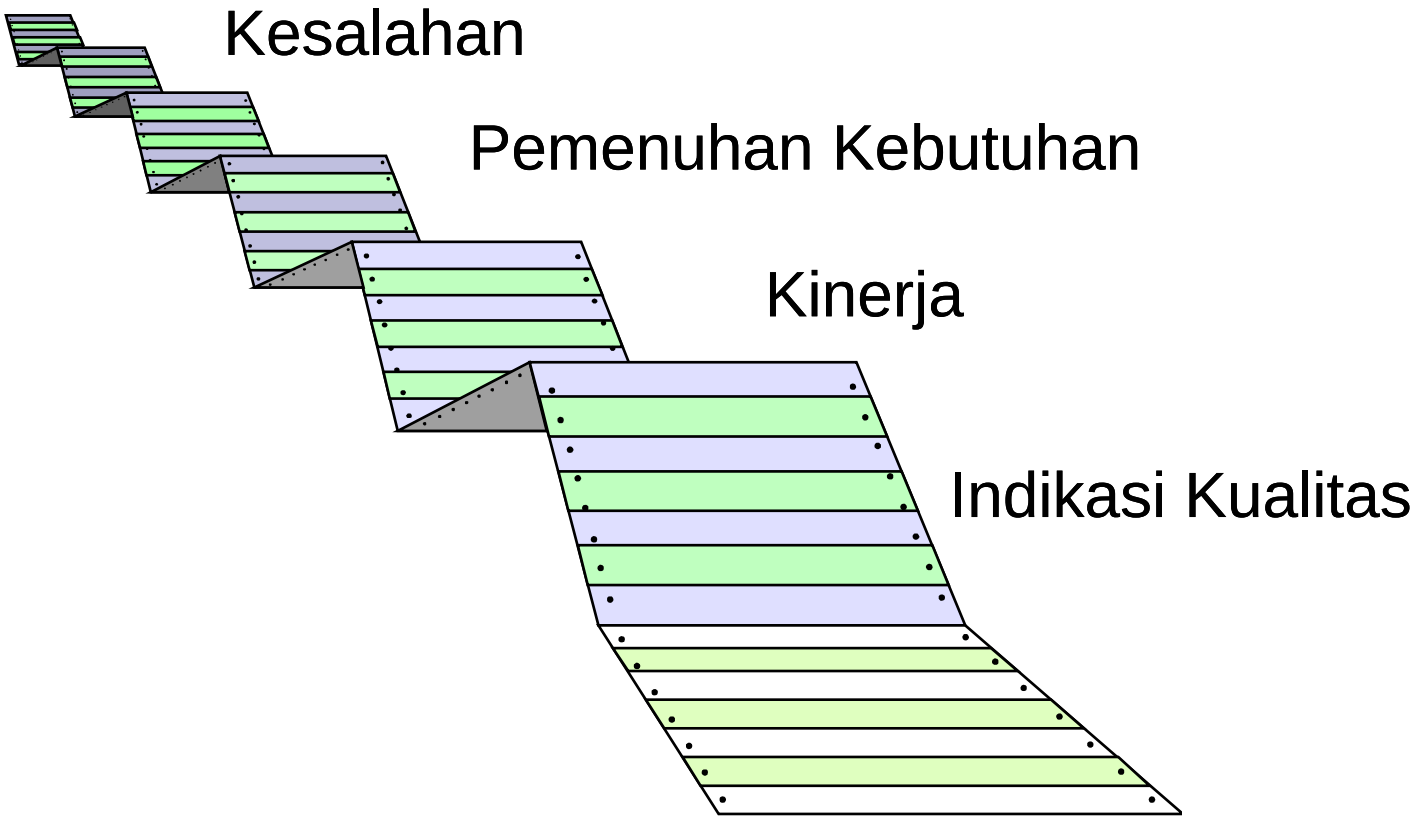
- Pengujian yang efektif harus dilakukan melalui tinjauan teknis yang efektif pula, dengan demikian banyak kesalahan akan dihilangkan sebelum pengujian dilakukan
- Pengujian dimulai pada tingkat komponen dan bekerja ke arah "luar" menuju integrasi sistem berbasis komputer secara menyeluruh
- Teknik pengujian yang berbeda tepat untuk pendekatan rekayawan PL yang berbeda pula dan waktu yang berbeda
- Pengujian dilakukan oleh pengembang PL (untuk proyek besar) dan kelompok penguji independen
- Pengujian dan pelacakan kesalahan (*debuging)* adalah aktivitas yang berbeda, namun *debuging* harus terakomodasi dalam setiap strategi pengujian.

Strategi Testing

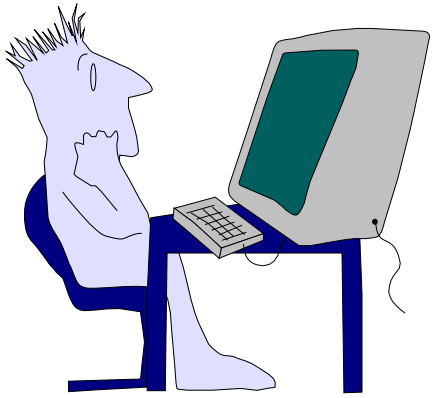
- Pengujian PL adalah salah satu elemen dari topik yang lebih luas yang dikenal dengan Verifikasi dan Validasi (V & V)
- Verifikasi merujuk pada sekumpulan tugas yang memastikan bahwa PL benar menerapkan fungsi yang telah ditentukan
- Validasi merujuk ke sekumpulan tugas yang berbeda yang memastikan bahwa PL yang telah dibagnun dapat dilacak berdasar persyarakatan pelanggan.
- Verifikasi dan validasi meliputi banyak kegiatan jaminan kualitas PL, yaitu : tinjauan teknis, audit konfigurasi dan kualitas, monitoring kinerja, simulasi, pengujian pengembangan, pengujian kegunaan, pengujian kualifikasi, uji penerimaan dan uji instalasi.

Strategi Testing

Apa yang ditunjukan pada saat Pengujian ?

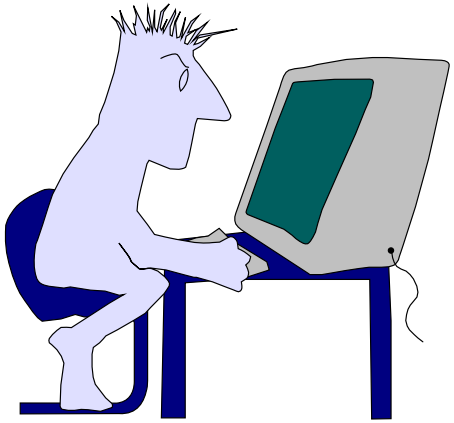


– Siapa yang melakukan Pengujian PL ?



Pengembang

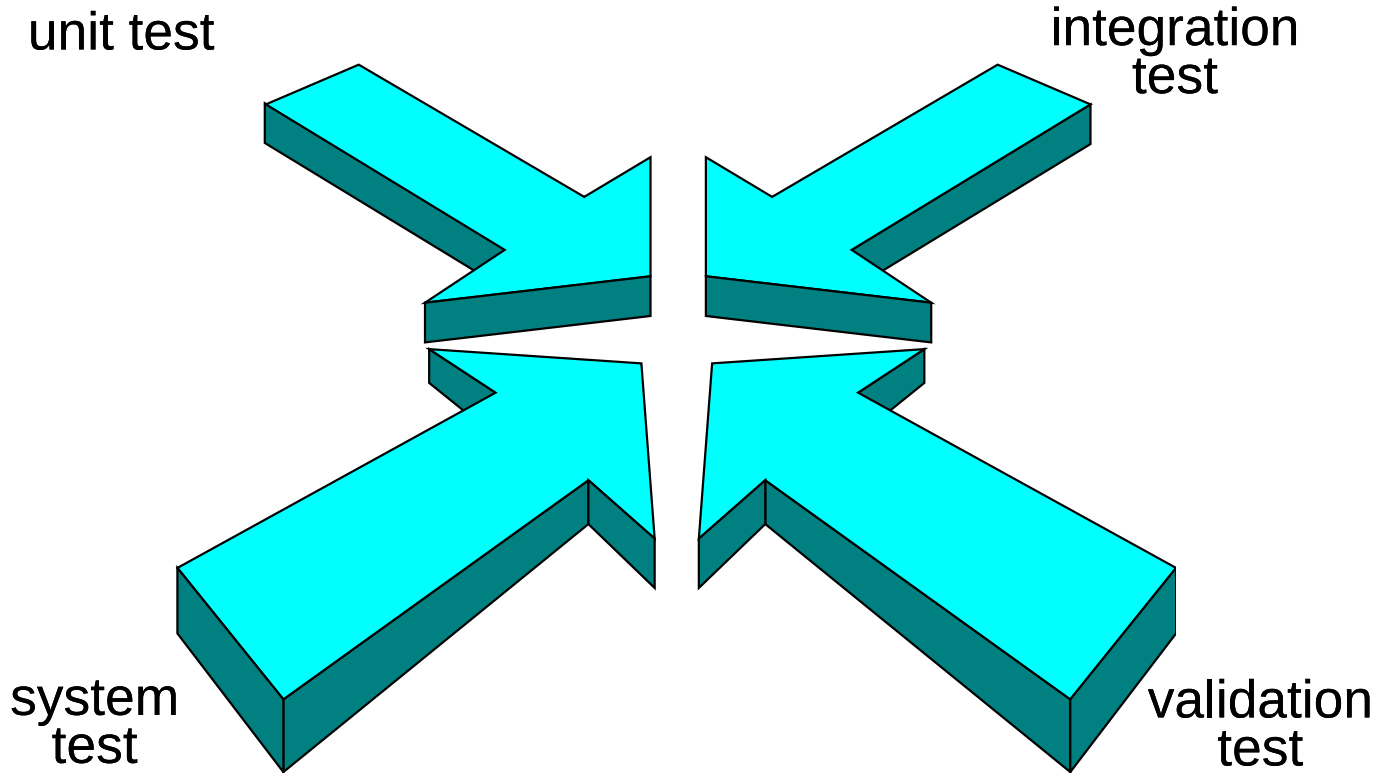
Memahami sistem, namun
pada saat pengujian harus adil,
Karena berpatokan pada penyebaran PL



Penguji Independent

Harus belajar mengenai sistem,
namun saat ditemukan ketidakbenaran
akan menghentikannya, karena
berpatokan pada kualitas

–Strategi Pengujian



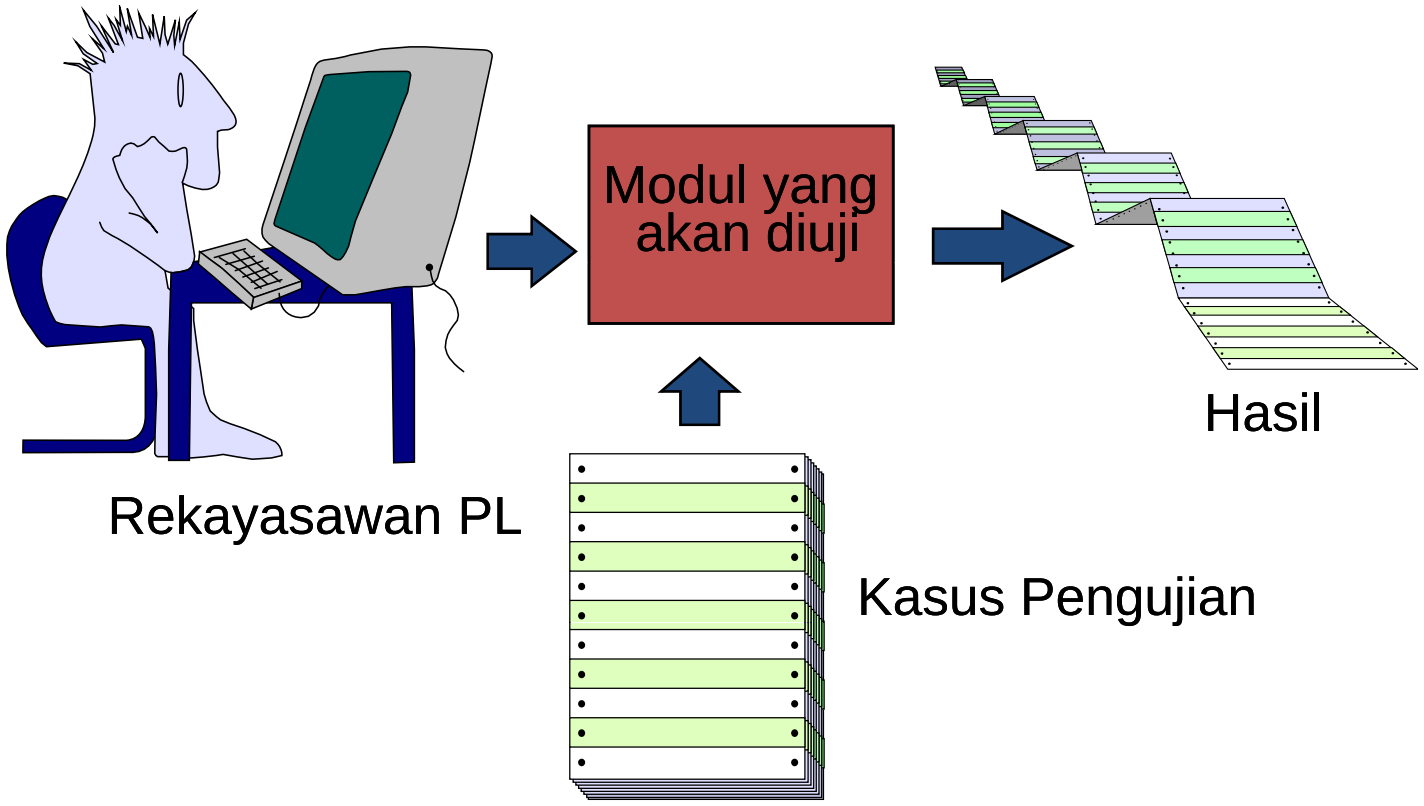
Strategi Pengujian

- Mulailah dari pengujian dengan lingkup kecil kemudian bergerak ke lingkup yang lebih besar
- Untuk PL konvensional :
  - Fokus awal pada pengujian modu/komponen
  - Dilanjutkan pada pengujian integrasi
- Untuk PL berorientasi objek
  - Pengujian lingkup kecil akan berubah dari modul individual (sudut pandang konvensional) menjadi kelas OO yang meliputi atribut dan operasi yang berimplikasi pada komunikasi dan kolaborasi

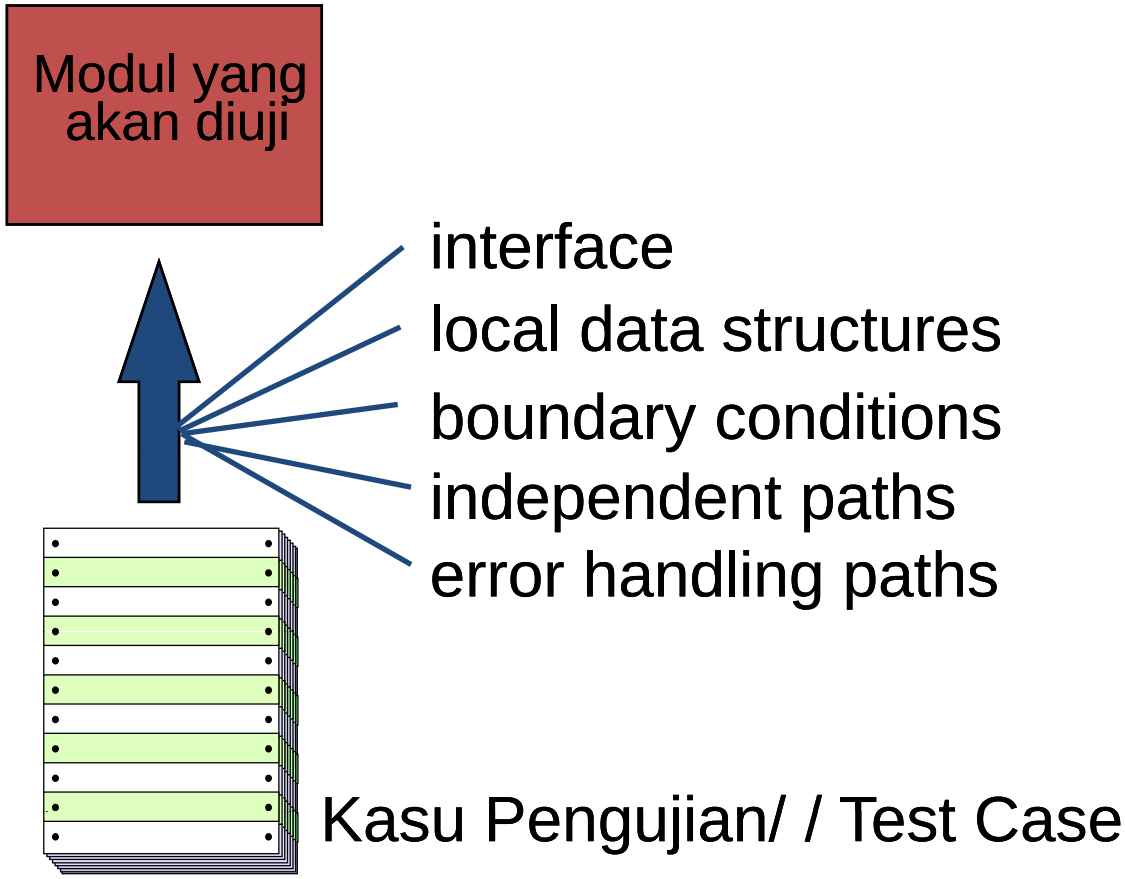
Isu Strategi Pengujian

- Tahapan pengujian harus diungkapkan secara eksplisit
- Memahami user sebagai pengguna PL dan membangun profile untuk setiap kategori user
- Membuat perencanaan pengujian yang dapat mengakomodir "siklus cepat pengujian"
- Menggunakan tinjauan teknis formal yang efektif sebagai filter awal pengujjian
- Mengelola tinjauan teknis formal untuk menilai strategi dan kasus pengujian
- Membangun pendekatan berkelanjutan untuk pengujian proses

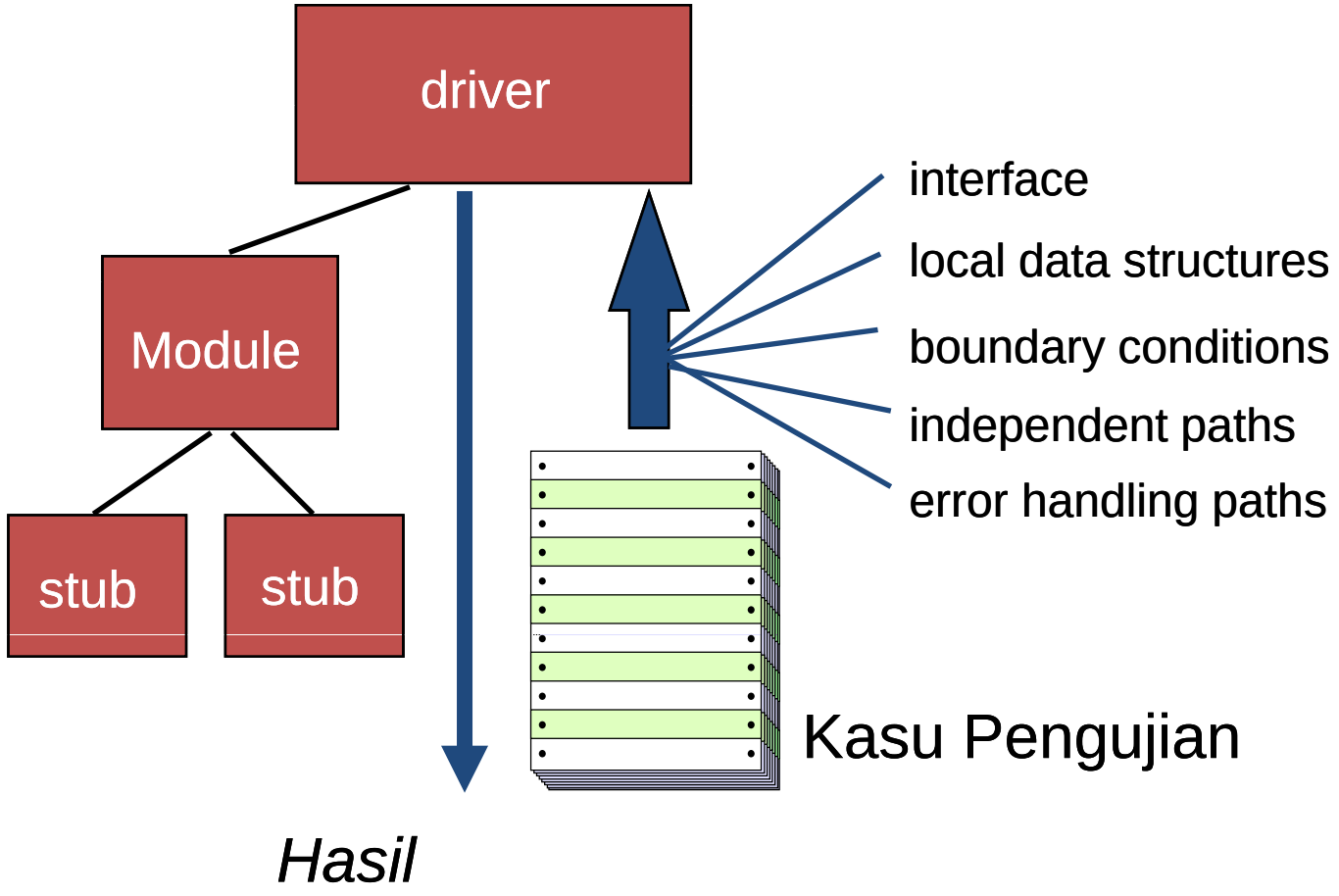
Pengujian Unit/Unit Testing



Pengujian Unit/Unit Testing



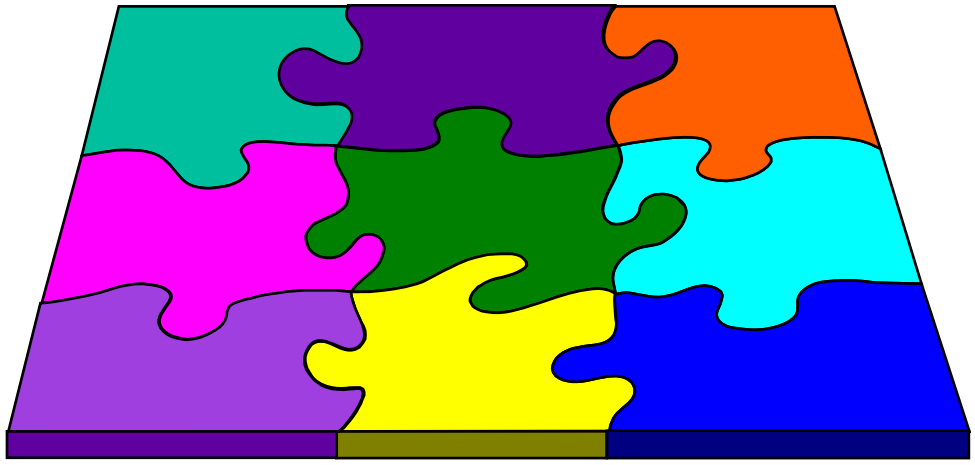
Lingkungan Pengujian Unit



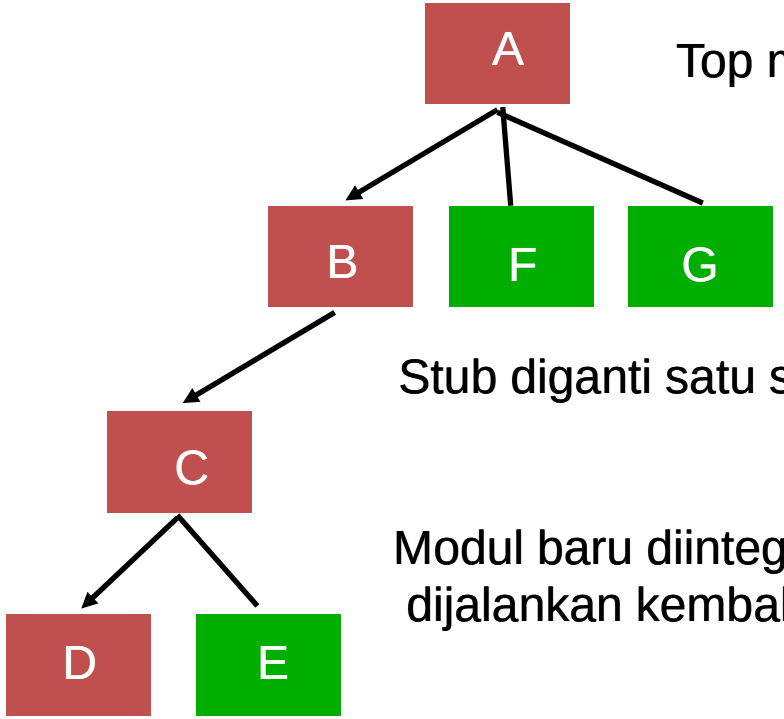
Strategi Pengujian Integrasi

Opsi : - Pendekatan “big bang”

- Strategi pengembangan inkremental



Integrasi TOP-DOWN

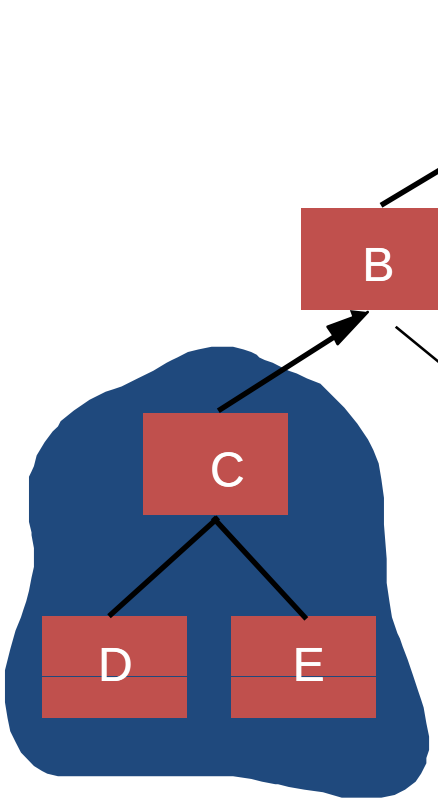


Top modul diuji dengan stub

Stub diganti satu setiap waktu, dengan “depth first”

Modul baru diintegrasikan, beberapa subset uji dijalankan kembali

Integrasi BOTTOM-UP

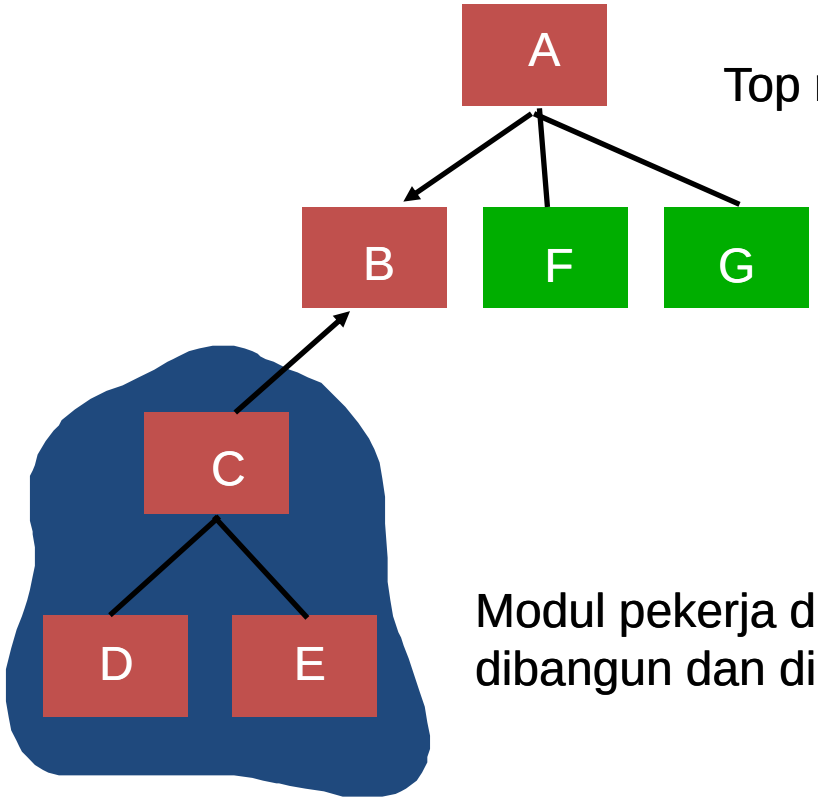


Driver diganti satu setiap waktu, dengan “depth first”

Modul pekerja dikelompokan untuk dibangun dan diintegrasikan

cluster

Pengujian SANDWICH



Top modul diujui dengan stub

Modul pekerja dikelompokan untuk dibangun dan diintegrasikan

Pengujian berorientasi Objek

- Tujuan pengujian tetap yaitu menemukan kesalahan dalam selang waktu yang realistik
- Dimulai dengan mengevaluasi kebenaran dan konsistensi model OOA dan OOD
- Melakukan perubahan strategi uji
  - Konsep ‘unit’ setara dengan encapsulasi
  - Fokus integrasi pada kelas dan persilangan eksekusi “thread” atau dalam konteks penggunaan skenario
  - Validasi menggunakan metode konvensional black box
- Rancangan kasus uji/test case digambarkan dengan metode konvensional tetapi melingkupi fitur spesial.

Perluasan Sudut Pandang Pengujian (Berorientasi Objek)

- Kesalahan pendefinisian atribut kelas yang ditemukan pada tahap analisis akan menghilangkan pengaruh yang dapat muncul.
- Contoh : Sebuah kelas dengan sejumlah atribut didefinisikan pada tahap analisis. Sebuah atribut yang tidak berhubungan dan dua operasi yang memanipulasi atribut tersebut terdefinisi.
- – Jika atribut yang tidak berhubungan dihilangkan pada tahap analisis, dapat mengurangi beberapa masalah dan usaha sbb :
  - Pembuatan subclass yang khusus untuk mengakomodasi atribut tersebut
  - Pembuatan relasi antar kelas yang salah
  - Kelakuan dari sistem dapat menjadi tidak tepat

Perluasan Sudut Pandang Pengujian (Berorientasi Objek)

- Jika kesalahan tidak ditemukan, masalah yang dapat muncul pada tahap perancangan :
  - penempatan kelas yang tidak tepat pada subsistem
  - perancangan kerja yang tidak perlu
  - model *messaging (message connection) yang tidak tepat*
- Jika kesalahan tetap ada sampai pada tahap pengkodean akan menghabiskan banyak waktu dan usaha untuk
  - membuat kode dari atribut dan dua operasi yang tidak diperlukan,
  - membuat *message untuk komunikasi antar objek*

Pengujian Model OOA dan OOD

Langkah :

- Lakukan pemeriksaan silang antara model CRC dengan model *object relationship* untuk memastikan semua kolaborasi yang dinyatakan dalam OOA direfleksikan dengan tepat dalam kedua model
- Periksa deskripsi dari setiap CRC index card untuk menentukan apakah suatu tanggung jawab merupakan bagian dari definisi *collaborator*
- Periksa hubungan balik untuk memastikan bahwa setiap *collaborator* menerima permintaan dari sumber yang tepat.
- Periksa hubungan balik untuk memastikan apakah kelas lain diperlukan sebagai *collaborator*
- Tentukan apakah beberapa tanggung jawab dapat digabungkan menjadi tanggung jawab
- Ke lima langkah di atas diterapkan untuk setiap kelas dan setiap evolusi dari model OOA

Strategi Pengujian Berorientasi Objek (OOT)

- Strategi : pengujian semua unit program terkecil, pengujian integritas dari modul, dan pengujian keseluruhan sistem
- **Pengujian Unit dalam konteks berorientasi objek**
  - Unit terkecil -> Kelas atau objek
  - Setiap operasi yang diturunkan pada kelas turunan harus diperiksa
- **Pengujian Integritas dalam konteks berorientasi objek**
  - *Thread-based testing ,* mengintegrasikan sekumpulan kelas suatu input atau kejadian dalam sistem. Setiap *thread diintegrasikan dan diuji secara individual.*
- **Pengujian regresi** diterapkan untuk memastikan tidak ada efek samping yang muncul.

Strategi Pengujian Untuk Aplikasi Web

1. Model isi untuk aplikasi web ditinjau untuk menemukan kesalahan
2. Model antarmuka ditinjau untuk memastikan bahwa semua kasus yang digunakan dapat diakomodasi
3. Model perancangan untuk aplikasi web ditinjau untuk menemukan kesalahan navigasi
4. Antarmuka pengguna diuji untuk menemmukan kesalahan dalam presentasi dan/atau mekanik navigasi
5. Setiap komponen fungsional diterapkan pengujian –unit
6. Navigasi seluruh arsitektur diuji
7. Aplikasi web diimplementasikan dalam berbagai konfigurasi lingkungan yng bereda dan diuji kompatibilitasnya dengan setiap konfigurasi
8. Uji keamanan dilakukan dalam upaya mengeksploitasi kelemahan-kelemahan dalam aplikasi web atau dalam lingkungannya
9. Kinerja pengujian dikontrol
10. Aplikasi web diuji oleh populasi yang dikendalikan dan dipantau oleh pengguna akhir . Hasil interaksi mereka dengan sistem dievaluasi, yakni dalam hal kesalahan isi dan navigasi, kegunaan,kompatibilitas, dan keandalan serta kinerja aplikasi web

Pengujian Validasi

- Pengujian validasi dimulai di titik puncak pengujian integrasi ketika komponen individu telah dieksekusi, PL sudah benar-benar dirakit sebagai sebuah paket dan kesalahan antarmuka telah ditemukan dan diperbaiki.
- Pada level validasi atau sistem, perbedaan antara PL konvensional, PL berorientasi objek dan aplikasi web menghilang
- Fokus pengujian pada tindakan pengguna yang terlihat dan output dari sistem yang dikenali pengguna
- Validasi berhasil jika PL berfungsi dengan cara yang diharapkan oleh pengguna

Kriteria Pengujian Validasi

Rencana pengujian menguraikan :

- Kelas-kelas pengujian yang akan dilakukan
- Prosedur pengujian yang mendefinisikan kasus pengujian tertentu yang dirancang untuk memastikan bahwa semua fungsional memenuhi persyaratan yang diminta, karakteristik tercapai, pendokumentasian benar, dan kegunaan dan persyaratan lainnya dipenuhi (misal : transportability, kompatibilitas, perbaikan kesalahan, pemeliharaan)

Setelah setiap kasus pengujian validasi ditemukan, ditemukan salah satu dari kondisi berikut :

- Karakteristik fungsi atau kinerja sesuai dengan spesifikasi dan diterima
- Penyimpangan ditemukan pada tahap ini

Pengujian Tingkat Tinggi

- **Pengujian Validasi**, fokus pada kebutuhan PL, akan dilihat kesesuaian antara rencana (kelas pengujian dan prosedur) dengan kondisi akhir yang dihasilkan (sesuai atau menyimpang)
- **Pengujian Sistem**, fokus pada integrasi sistem, untuk memverifikasi bahwa semua elemen sistem telah terintegrasi dengan baik dan menjalankan fungsinya.
- **Pengujian Alpha/Bheta**, fokus pada penggunaan oleh pengguna
  - **Pengujian Alpha**, dilakukan di sisi pengembang oleh sekelompok perwakilan dari pengguna akhir. PL digunakan dalam kondisi natural dimana pengembang “melihat dengan kacamata” pengguna dan mencatat kesalahan dan masalah penggunaan. Pengujian alpha dilakukan dalam lingkungan yang dikendalikan
  - **Pengujian Bheta**, dilakukan oleh satu atau lebih pengguna akhir. Pengujian ini adalah "aplikasi hidup” dari PL dalam sebuah lingkungan yang tidak dapat dikendalikan oleh pengembang

Pengujian Tingkat Tinggi

- **Pengujian Pemulihan/Recovery**, pengujian yang memaksa PL untuk gagal dalam berbagai cara dan memverifikasi bahwa pemulihan dilakukan dengan benar. Jika pemulihan dilakukan dengan otomatis, maka inisialisasi kembali,mekanisme checkpointing, pemulihan data dan restart dievaluasi untuk mengethui apakah itu semua berjalan benar. Jika pemulihan membutuhkan intervensi manusia, maka rata-rata waktu untuk perbaikan dievaluasi untuk menentukan apakah masih dalam batas yang dapat diterima
- **Pengujian Keamanan/Security**, memverifikasi mekanisme perlindungan yang dibangun ke dalam sistem untuk melindunginya dari penetrasi yang tidak benar.

Pengujian Tingkat Tinggi

– **Pengujian Tekanan/Stress**, menajlankan sistem dengan cara yang meminta sumber daya dalam jumlah, frekuensi atau volume abnormal, misal :

- Pengujian khusus yang dirancang untuk menghasilkan sepuluh sela per detik, ketika rata-ratanya adalah satu atau dua sela
- Kecepatan indput data dapat ditingkatkan untuk menentukan bagaiman fungsi-fungsi input akan menanggapi
- Kasus pengujian yang membutuhkan memori maksimum atau sumberd daya lain untuk dijalankan
- Kasus pengujian yang dapat menyebabkan tabrakan di dalam sistem operasi virtual yang dirancang
- Kasus pengujianyang dapat menyebabkan junting yang berlebihan atas data yang ada pada disk yang dapat dibuatnya.

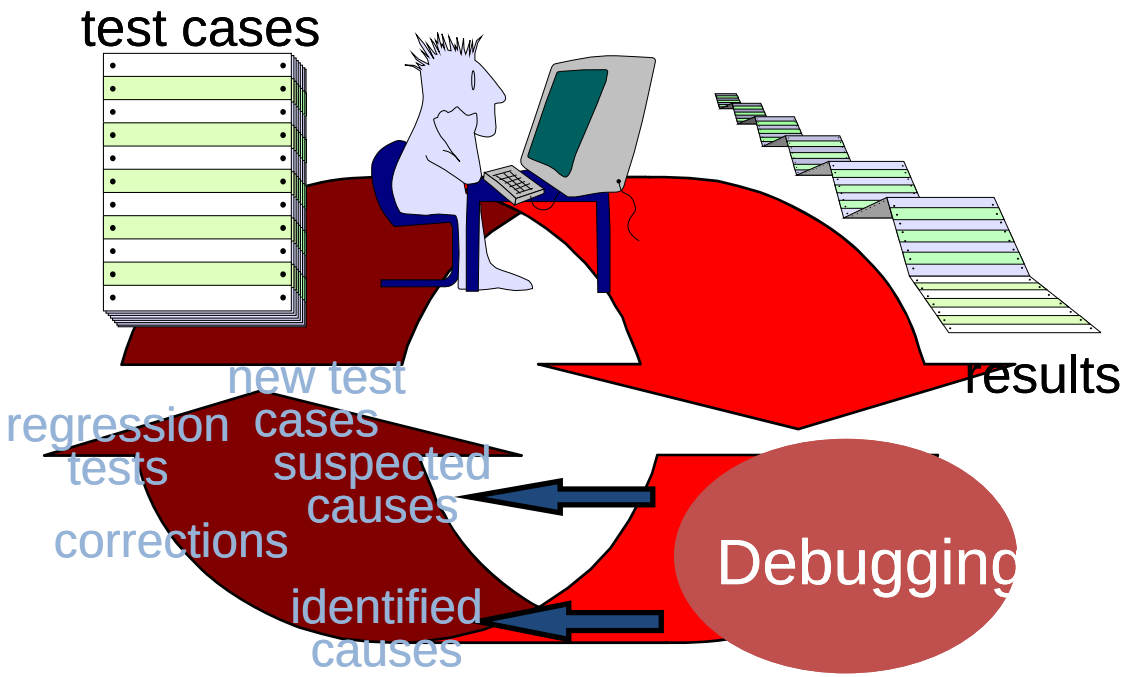
– **Pengujian Kinerja/Performance**, dilakukan untuk menguji kinerja run-time dari PL dalam konteks sebagai sistem yang terintegrasi. Pengujian dilakukan di seluruh langkah dalam proses. Sering pula digabungkan dengan pengujian stress

Pengujian Tingkat Tinggi

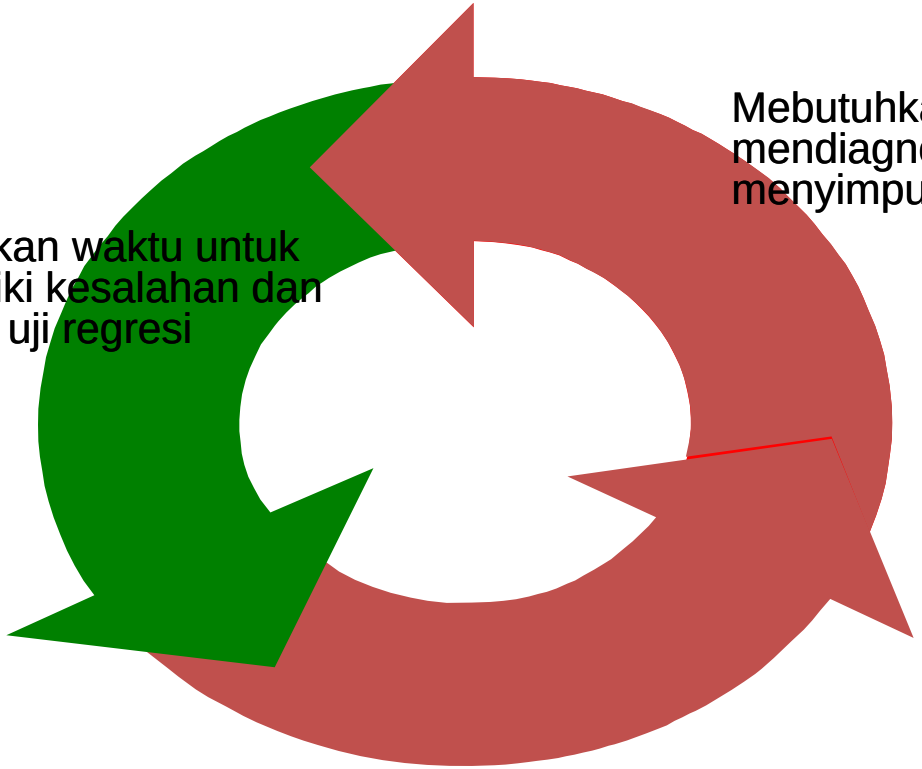
– **Pengujian Deployment**, disebut juga pengujian konfigurasi. Untuk menguji PL dalam berbagai lingkungan dimana PL tersebut dioperasikan. Menguji pula semua prosedur instalasi (instaler) yang akan digunakan oleh pelanggan) dan semua dokumentasi yang akan diperkenalkan pada pengguna akhir

DEBUGGING / Pelacakan Kesalahan

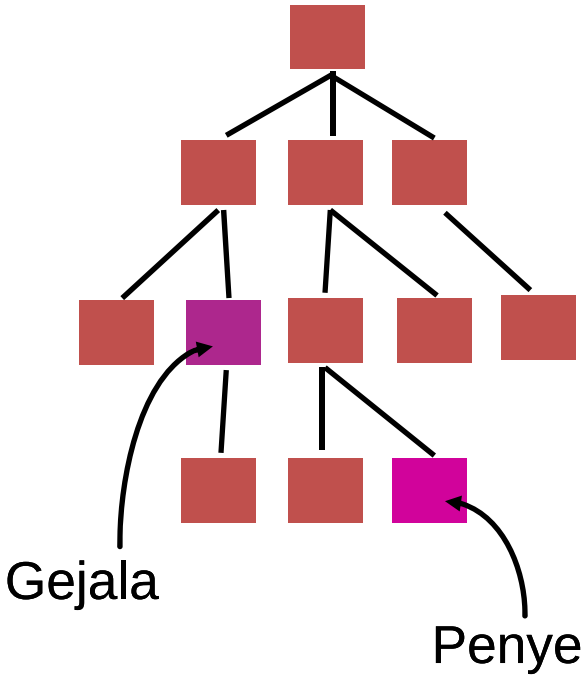
– Debugging terjadi sebagai akibat pengujian yang berhasil (pada saat kasus pengujian menangkap kesalahan), dimana debugging merupakan proses yang menghasilkan penghapusan kesalahan.



– Upaya Debugging



– Gejala dan Penyebabnya



•Gejala dan penyebabnya mungkin secara geografis jauh. Gejala dapatmuncul di salah satu bagian program,tapi penyebabnya mungkin terletak di tempat jauh

• Gejala mungkin hilang sementara saat kesalahan lain dikoreksi

•Gejala ini sebenarnya disebabkan oleh non error, misal ketidakakuratan

• Gejala dapat disebabkanoleh kesalahan manusia yang tidak mudah dilacak

• Gejala mungkin akibat masalah waktudaripada masalah pemrosesan

•Gejala dapat berselang, terutam untuk sistem tertanam

•Gejala dikarenakan penyebab didistribusikan ke sejumlah tugas berjalan pada prosesor ang berbeda

– Strategi Debugging

1. Brute force, filosofinya “Biarkan komputer menemukan kesalahan”, maka akan sampah memori diambil, bekas run time dipanggi dan program sarat dengan laporan keluaran.
2. Backtracking, umumnya digunakan untuk program kecil. Mulai dari gejala, kode program dilacak ke belakang/manual sampai penyebabnya diketahui.
3. Menyingkirkan penyebab/cause elimination, ditunjukan oleh induksi atau deduksi dan memperkenalkan konsep partisi biner. Data yang terkait dengan terjadinya kesalahan dikelola untuk diisolasi penyebab potensial.

– Memperbaiki Kesalahan

Pertanyaan dasar sebelum membuat koreksi menghilangkan penyebab kesalahan :

1. Apakah penyebab bug dibuat ulang di bagian lain dari program ini ?
2. Apakah bug selanjutnya apat terjadi akibat perbaikan yang sedang dibuat ?
3. Apakah yang bisa dilakukan untuk mencegah bug ini di tempat pertama ?

Pertanyaan

Dan

Diskusi

.....