UNIVERSITAS BINA INSANI

TESTING DAN IMPLEMENTASI SISTEM

STRATEGI *TESTING*

Ahmad Chusyairi, M.Kom

Bigger Better Higher

Rencana Kegiatan Pembelajaran Mingguan

UNIVERSITAS BINA INSANI

Capaian Pembelajaran Pertemuan

Mahasiswa mengetahui dan memahami strategi *testing.*

Kemampuan Akhir Capaian Pembelajaran

Mahasiswa mampu menjelaskan dan memahami strategi *testing.*

Bahan Kajian

Strategi *Testing*:

- Pendekatan Strategi *Testing Software*
- Verifikasi dan Validasi
- Pengorganisasian *Testing Software*
- Kriteria Pemenuhan *Testing*

Penilaian

Mahasiswa mampu menjelaskan pendekatan strategi *testing software,* verifikasi dan validasi, pengorganisasian *testing software,* dan kriteria pemenuhan *testing.*

Strategi *Testing*

- **Strategi *Testing*** harus menjadi satu kesatuan dengan:
  - Perencanaan *Test*,
  - Desain *Test Case*,
  - Eksekusi *Test*,
  - Pengumpulan Data Hasil *Testing*, dan
  - Evaluasi *Testing*.

Pendekatan Strategi *Testing*

Pendekatan Strategi *Testing* dilakukan dengan:

- *Testing* dimulai dari tingkat komponen terkecil sampai pada integrasi antar komponen pada keseluruhan sistem komputer tercapai.
- Teknik *testing* berbeda-beda sesuai dengan waktu penggunaannya.
- *Testing* dilakukan oleh pengembang *software* dan (untuk proyek besar) dilakukan oleh satu grup tes yang independen.
- *Testing* dan *debugging* adalah aktifitas yang berlainan, tapi *debugging* harus diakomodasi disetiap strategi *testing*.

Verifikasi dan Validasi

- **Verifikasi** merupakan sekumpulan aktifitas yang memastikan software telah melakukan fungsi tertentu dengan benar.
- **Validasi** merupakan sekumpulan aktifitas berbeda dari verifikasi yang memastikan bahwa software yang dibangun dapat dilacak terhadap kebutuhan atau permintaan pelanggan.
- Pengujian software sebagai Verifikasi dan Validasi (V&V) meliputi banyak aktifitas *Software Quality Assurance* (SQA), termasuk review teknis formal, kualitas dan audit konfigurasi, monitor *performance*.

Pengorganisasian *Testing Software*

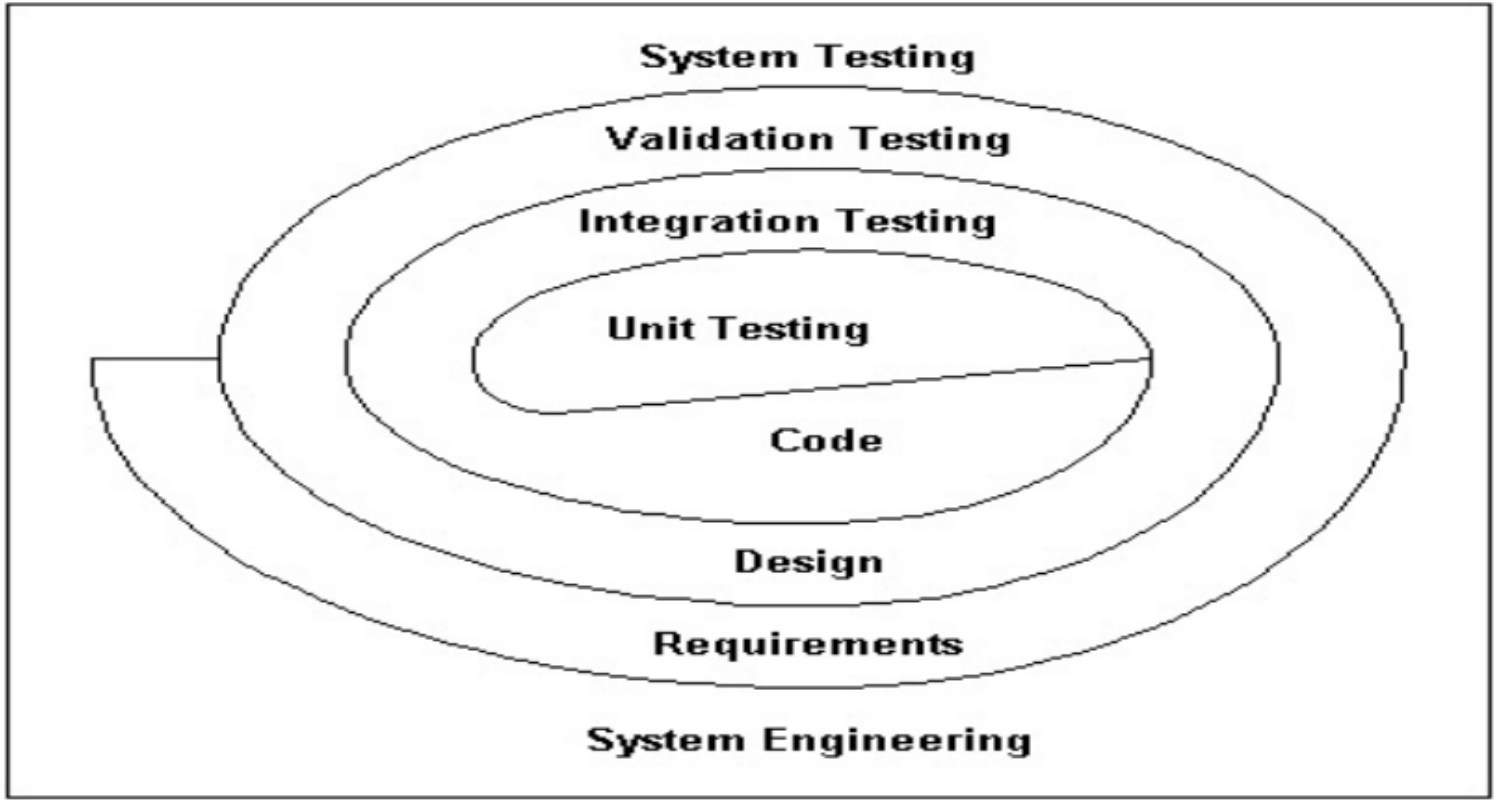
Umumnya testing dilakukan oleh pengembang software, namun diskusi dalam konsepsi *testing* yang salah adalah:

- Pengembang *software* tidak perlu melakukan *testing* sama sekali.
- *Software* diberikan pada orang lain (tidak kenal kredibilitasnya), yang akan melakukan tes pada software tanpa pemahaman dan salah arah.
- *Tester* baru bekerja atau ikut serta dalam proyek, jika tahap *testing* pada proyek tersebut dimulai.

Independent Test Group

- ***Independent Test Group*** adalah grup tes bersifat independen dan ikut serta dalam testing yang terintegrasi dan dilibatkan setelah arsitektur software telah komplit.
- *Testing* Integrasi adalah suatu langkah testing yang mengarahkan pada konstruksi dan tes dari struktur program secara keseluruhan.
- Tujuan dari tes independen adalah untuk menghindari masalah-masalah yang berkaitan dengan membiarkan pembuat melakukan tes terhadap software yang telah dibuatnya sendiri.

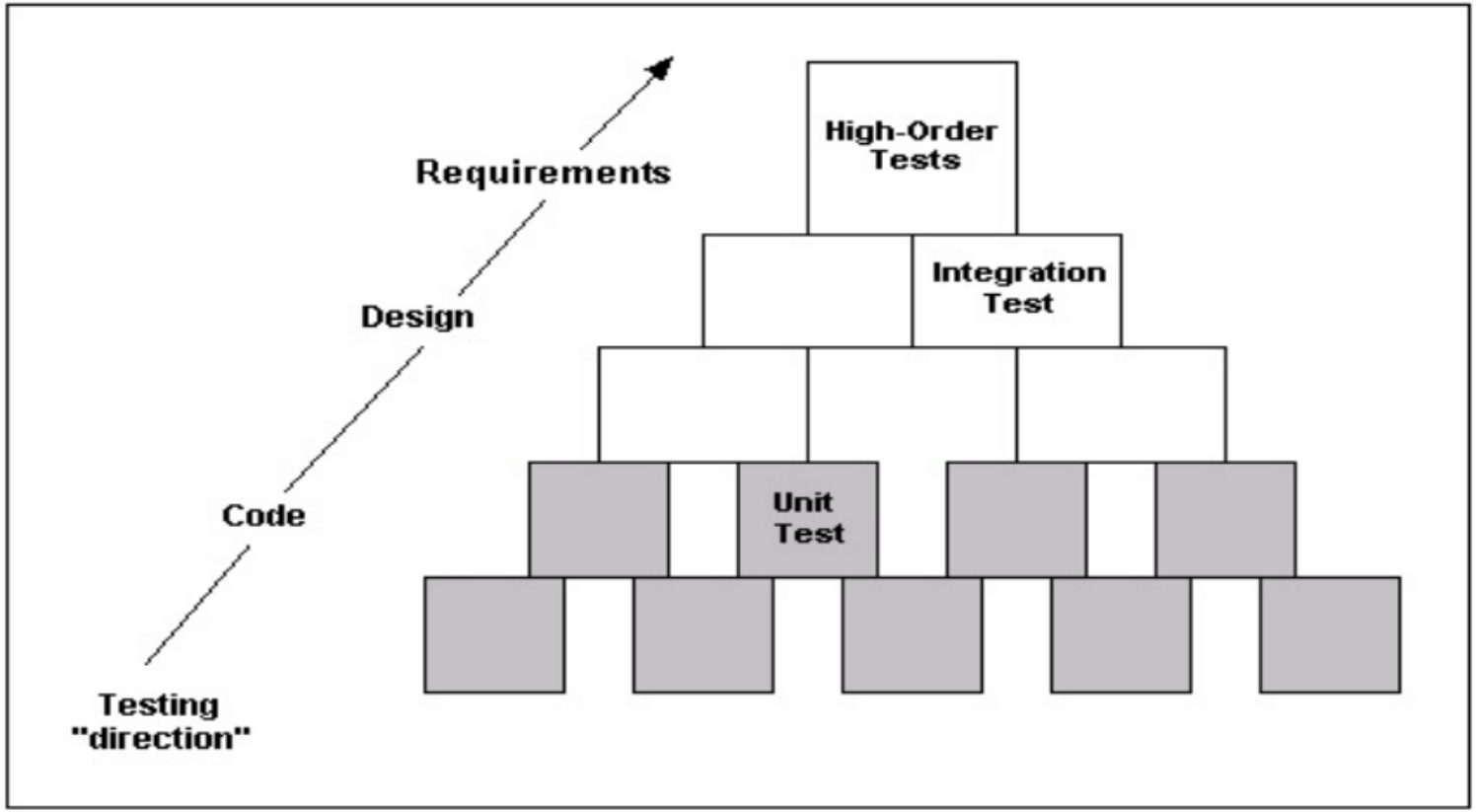
Strategi *Software Testing*



Penjelasan Tahapan *Testing*

- Pada awalnya, fokus tes terletak pada setiap komponen secara individual, memastikan apakah fungsi dari komponen tersebut dapat dikategorikan sebagai suatu unit.
- *Unit Testing,* sangat banyak menggunakan teknik white box testing dengan memerika jalur tertentu dalam suatu struktur kendali.
- *Integration Testing* berkaitan dengan ha-hal yang berhubungan dengan masalah verifikasi dan konstruksi program.
- Teknik desain *test case black box* lebih dominan dipakai selama integrasi.
- *High-Order Testing* berada di luar daerah rekayasa software dan masuk ke dalam kontek yang lebih luas dari rekayasa sistem komputer.

Tahapan *Testing*



Integration Testing

- ***Integration Testing*** adalah suatu teknik yang sistematis untuk pembangunan struktur program, dimana pada saat yang bersamaan melakukan testing untuk mendapatkan *error* yang diasosikan dengan antarmuka.
- Pendekatan *Big Bang* adalah pendekatan ini menggabungkan komponen-komponen secara bersamaan, sekumpulan *error* yang akan diperoleh, dan perbaikan sulit dilakukan, karena terjadi kompilasi saat melakukan isolasi terhadap penyebab masalah.

Pendekatan *IntegrationTesting*

Pendekatan *Integration Testing*

- *Top-Down Integration*
- *Bottom-Up Testing*
- *Regression Testing*
- *Smoke Testing*

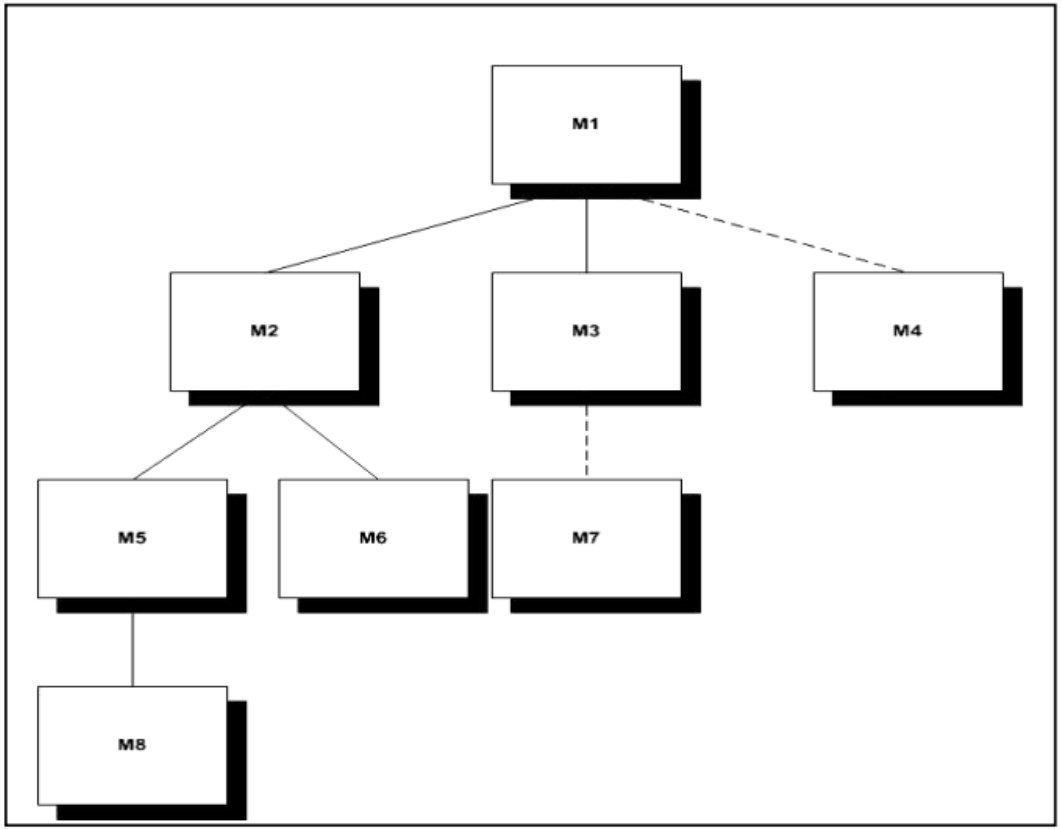
Top-Down Integration

- ***Top-Down Integration*** adalah pendekatan bertahap untuk menyusun struktur program.
- Modul-modul diintegrasikan dari atas ke bawah dalam suatu hirarki kendali, dimulai dari modul kendali utama (program utama).
- Modul sub-ordinat dari modul kendali utama dihubungkan ke struktur yang paling dalam (*depth-first integration*) atau yang paling luas (*breadth-first integration*).

Depth-First Integration & Breadth-First Integration

- ***Depth-First Integration***, akan mengintegrasikan semua komponen-komponen pada struktur jalur kendali mayor, misal dipilih sisi kiri terlebih dahulu, maka komponen M1, M2, M5 akan diintegrasikan dahulu, baru kemudian M8 atau M6 akan diintegrasikan.
- ***Breadth-First Integration***, akan mengintegrasikan semua komponen secara langsung ke tiap tingkat, bergerak secara horisontal. Contoh komponen M2, M3, dan M4 akan diintegrasikan dahulu, kemudian baru M5 dan M6 dan seterusnya.

Depth-First Integration & Breadth-First Integration (2)



Langkah Integrasi

Lima langkah proses integrasi:

- Modul kendali utama digunakan sebagai driver tes dan *stubs* tes disubtitusikan bagi semua komponen yang secara langsung menjadi sub-ordinat modul kendali utama.
- Tergantung pada pendekatan integrasi yang dipilih, *stubs* sub-ordinat digantikan dengan komponen lainnya.
- Tes dilakukan saat tiap komponen diintegrasikan.
- Saat pemenuhan tiap tes, *stubs* lainnya digantikan dengan komponen sebenarnya.
- *Testing* regresi dilakukan untuk memastikan kesalahan baru tidak terjadi lagi.

Pilihan *Tester*

Tester hanya mempunyai 3 pilihan:

- Tunda kebanyakan tes sampai stubs digantikan dengan modul sebenarnya, hal ini menyebabkan hilangnya beberapa kendali yang berhubungan antar tes tertentu dan modul tertentu.
- Kembangkan *stubs* yang mempunyai fungsi terbatas untuk mensimulasikan modul sebenarnya, mungkin dapat dilakukan, namun akan menambah biaya overhead dengan semakin kompleknya *stubs*.
- Integrasikan software dari bawah ke atas dalam hirarki, disebut sebagai *bottom-up integration*.

Bottom-Up Testing

- ***Bottom-Up Testing***, integrasi ini dimulai dari modul terkecil, karena komponen-komponen diintegrasikan dari bawah ke atas, sub-ordinat untuk tingkat bersangkutan dari komponen selalu diperlukan untuk proses, dan kebutuhan terhadap *stubs* dapat dihilangkan.
- Langkah-langkah strategi ini adalah:
  - Komponen level bawah dikombinasikan dalam cluster yang mewakili sub-fungsi software tertentu.
  - Driver ditulis untuk koordinasi masukan dan keluaran *test case*.
  - Cluster dites.
  - Driver dihapus dan cluster dikombinasikan, bergerak ke atas di dalam struktur program.

Bottom-Up Integration

- Komponen dikombinasi untuk membentuk cluster 1, 2 dan 3.
- Tiap cluster dites dengan menggunakan driver.
- Komponen pada cluster 1 dan 2 adalah sub ordinat Ma.
- Driver D1 dan D2 dihilangkan dan cluster dihubungkan langsung ke Ma, demikian seterusnya.

Regression Testing

- ***Regression Testing*** adalah eksekusi kembali dari subset dari tes yang telah di lakukan untuk memastikan apakah perubaha-perubahan yang dilakukan telah benar dan tidak menimbulkan efek samping yang tidak diharapkan.
- Sub set test yang dieksekusi dari 3 kelas test case yang berbeda:
  - Representasi dari contoh tes yang akan memeriksa semua fungsi software.
  - Tes tambahan yang berfokus pada fungsi *software* yang mungkin dipengaruhi oleh perubahan.
  - Tes yang berfokus pada komponen *software* yang diubah.

Smoke Testing

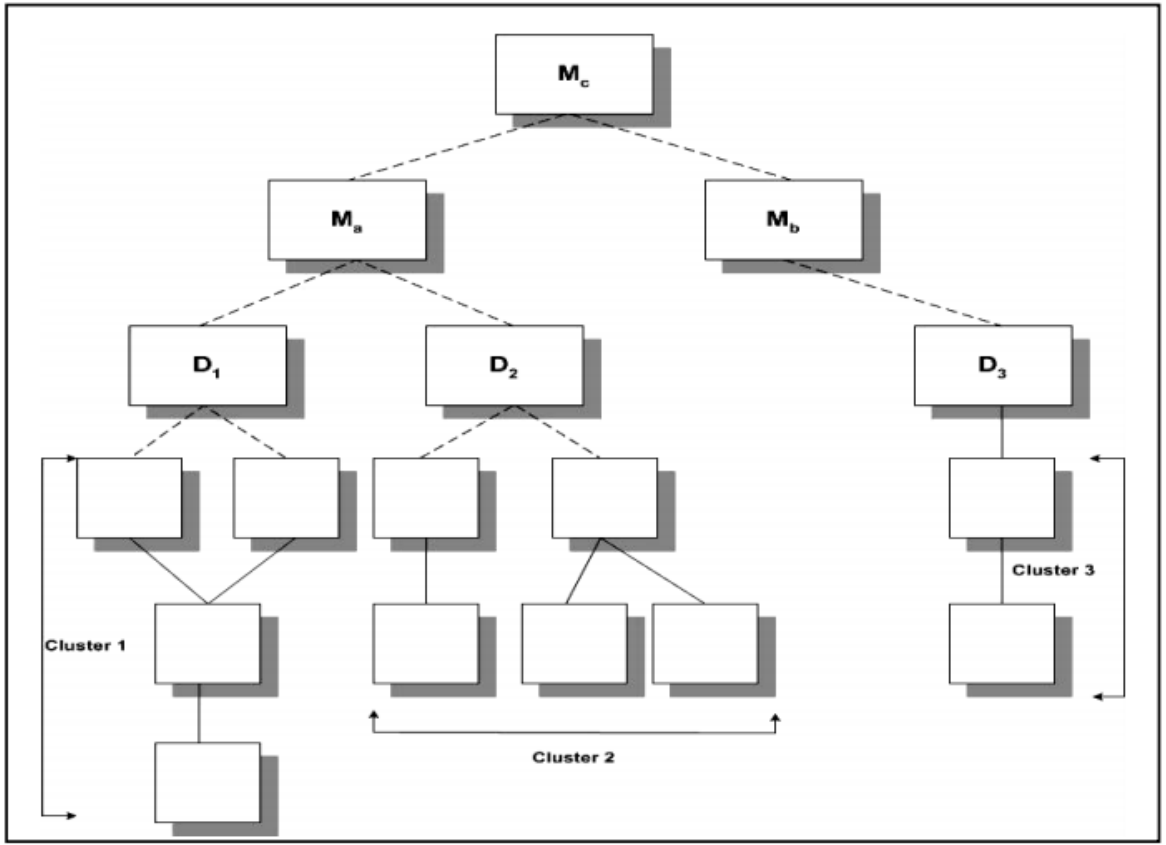
- ***Smoke Testing*** adalah pendekatan integration testing yang sering digunakan ketika produk software "kecil terbatas" dibuat.
- Pendekatan *smoke testing* tediri dari aktivitas-aktivitas berikut:
  - Komponen *software* yang ditranslasikan ke kode, diintegrasikan ke *build* yang terdiri dari semua file data, pustaka, modul yang digunakan lagi, dan komponen yang dibutuhkan untuk menerapkan satu atau lebih fungsi produk.
  - Serangkaian tes didesain untuk menghasilkan kesalahan yang akan membuat buid tetap berfungsi sebagaimana mestinya.
  - *Build* diintegrasikan dengan *build* lainnya dan keseluruhan produk yang dilakukan *smoke* tes harian.

Keuntungan *Smoke Testing*

Keuntungan ***Smoke Testing***, antara lain:

- Meminimalkan resiko integrasi.
- Meningkatnya kualitas produk akhir.
- Diagnosa kesalahan dan koreksi disederhanakan.
- Penilaian proses kerja lebih mudah.

Bottom-Up Integration



Kriteria Pemenuhan *Testing*

- Model kesalahan software (yang didapat selama testing) sebagai fungsi dari waktu eksekusi, dengan berdasarkan pada pemodelan statistik dan teori reliabilitas yang disebut ***Logarithmic Poission Execution-Time Model***, dengan bentuk:

$$f(t) = (1 / p) \ln (I_0 p t + 1)$$

dimana f(t) = Jumlah *error* kumulatif yang diharapkan terjadi saat *software* di tes untuk suatu wantu eksekusi, t.

I_0 = Inisial dari intensitas *error* dari *software* (*error* per unit waktu) saat awal *testing*.

P = Pengurangan secara eksponensial intensitas *error* saat *error* telah diperbaiki.

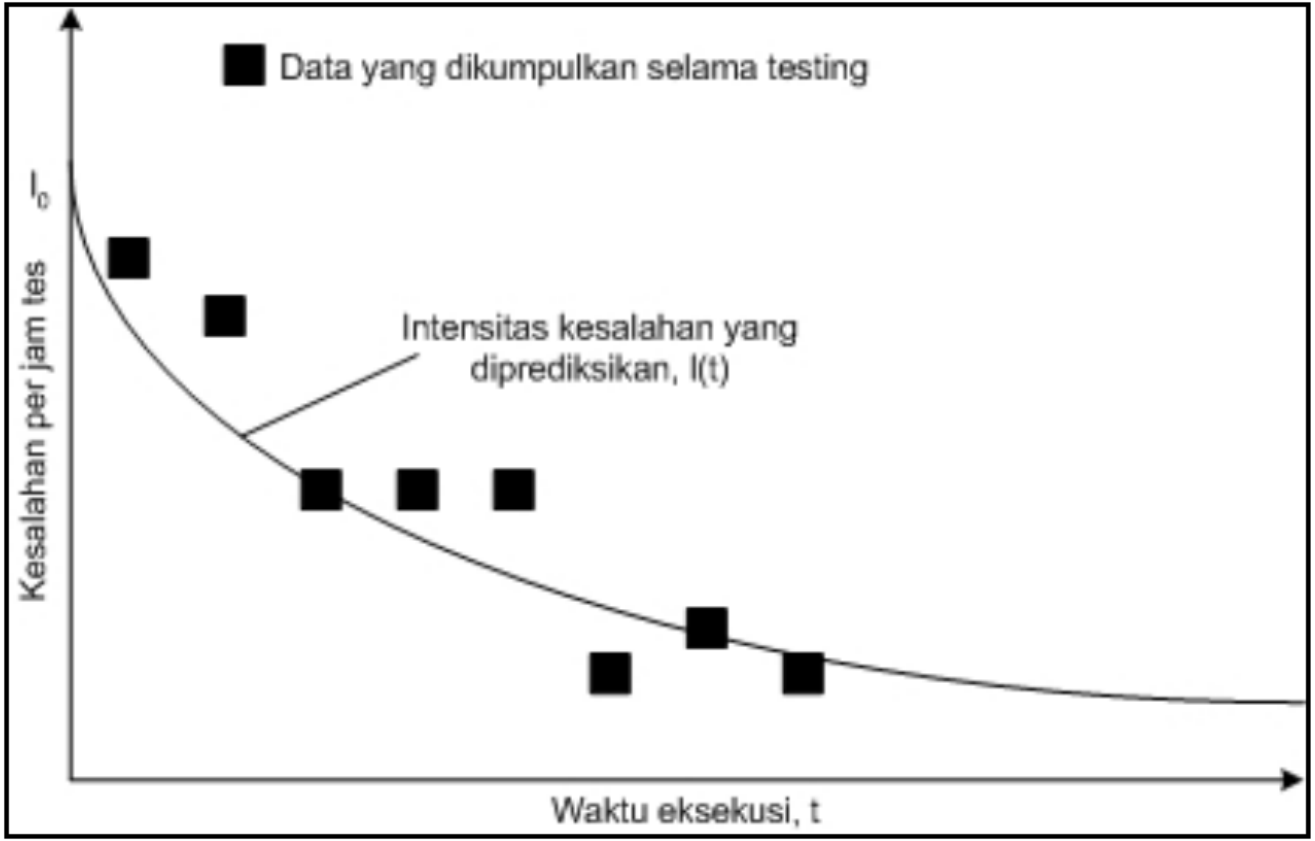
Logarithmic Poisson Execution-Time Model

- Intensitas *error*, I(t) dapat diturunkan dengan menurunkan derivasi dari f(t):

$$I(t) = I_0 / (I_0 \; p \; t + 1)$$

- Dengan menggunakan persamaan I(t), *tester* dapat memprediksi turunnya *error* dari hasil kinerja proses *testing*.

Intensitas Kesalahan



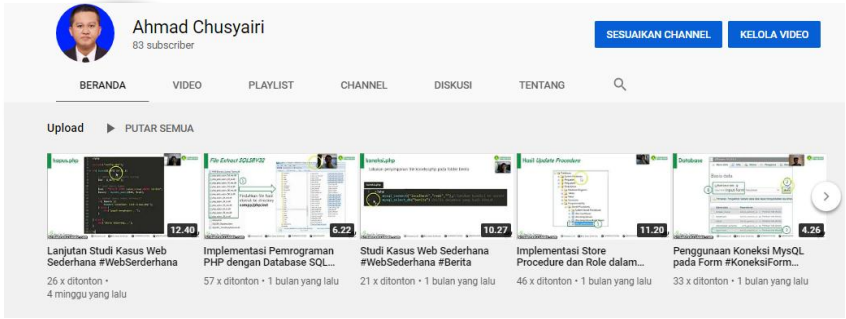
Referensi

Romeo. 2003. Testing dan Implementasi Sistem Edisi Pertama. STIKOM Surabaya.

Mustaqbal. 2015. Pengujian Aplikasi menggunakan Black Box Testing Boundari Value Analysis (Studi Kasus: Aplikasi Prediksi Kelulusan SNMPTN). Jurnal Ilmiah Teknologi Informasi Terapan, 1(3), pp.31-36.

Terima Kasih

ahmad chusyairi



Bigger Better Higher