Software Quality Assurance

Pertemuan 5
Software Testing
Pengujian Perangkat Lunak

Tim Dosen PMPL INF UAJY

Materi

- Definisi Pengujian Perangkat Lunak
- Pengujian Perangkat Lunak Effective vs Exhaustive
- Evolusi Pengujian Perangkat Lunak
- Beberapa Terminologi
- Verifikasi dan Validasi
- Aktivitas Validasi
- Teknik Pengujian Perangkat Lunak
- Pengujian otomatis / automation testing

Definisi Pengujian Perangkat Lunak

Definisi Software Testing (1)

… the psychology of testing – pandangan 1 …

- Pengujian adalah proses menunjukkan bahwa kesalahan (error) tidak ada.
- The purpose of testing is to show that a program performs its intended function correctly

- bahwa ‘pengujian adalah menunjukkan perangkat bekerja dengan benar

Definisi Software Testing (2)

… the psychology of testing – pandangan 2 …

- Memulai dengan sebuah asumsi bahwa perangkat lunak memiliki error

- Lalu melakukan pengujian untuk menemukan error tersebut

Definisi Software Testing (3)

… the psychology of testing – pandangan 2 …

- Pengujian adalah proses menjalankan program dengan maksud untuk menemukan kesalahan (Myers)
- Pengujian adalah proses menjalankan program dengan maksud khusus untuk menemukan kesalahan sebelum dikirimkan ke pengguna akhir (Pressman)

Definisi Software Testing (3)

… the psychology of testing …

- Apa perbedaan pandangan 1 dan pandangan 2 ?
- Pandangan 1 : testing bersifat ‘konstruktif’
- Pandangan 2 : testing bersifat ‘destruktif’

Tujuan Software Testing

- Tujuan jangka pendek: hasil langsung setelah melakukan pengujian
  - Penemuan bug, Pencegahan bug
- Tujuan jangka panjang: mempengaruhi kualitas produk dalam jangka panjang, ketika satu siklus SDLC berakhir
  - Reliability, Kualitas, Kepuasan konsumen, Manajemen risiko
- Tujuan pasca implementasi : Tujuan ini penting setelah produk dirilis
  - Mengurangi biaya perawatan (*maintenance*), Meningkatkan proses pengujian perangkat lunak

Pengujian Perangkat Lunak
Effective vs Exhaustive
(Efektif vs Lengkap)

Pengujian Perangkat Lunak Exhaustive/Lengkap (1)

- Kita ingin menguji everything sebelum memberikan perangkat lunak kepada pelanggan
- Apa arti “everything”
  - Eksekusi setiap statemen program
  - Eksekusi setiap kondisional (true dan false)
  - Eksekusi setiap kondisi pada suatu decision node (percabangan)
  - Eksekusi setiap jalur program yang ada
  - Eksekusi program dengan seluruh input yang valid.
  - Eksekusi program dengan seluruh input yang tidak valid.

Pengujian Perangkat Lunak Exhaustive/ Lengkap (2)

- Ada pertanyaan-pertanyaan:
  - Kapan kita selesai dengan pengujian?
  - Bagaimana kita tahu bahwa sudah cukup menguji?
- Pengujian secara menyeluruh tidak mungkin untuk dilaksanakan
- Tidak mungkin untuk menemukan semua error yang ada.
- Kombinasi dari kemungkinan pengujian ini tidak terbatas dalam arti bahwa sumber daya dan waktu tidak cukup untuk melakukan pengujian ini.
- Terdapat batasan-batasan sehubungan dengan waktu, biaya, SDM, pelanggan, dll.

Pengujian Perangkat Lunak Effective/Efektif

- Kita harus berkonsentrasi pada pengujian efektif yang menekankan teknik yang efisien untuk menguji perangkat lunak sehingga fitur-fitur penting akan diuji dalam sumber daya yang terbatas.
- Domain dari pengujian mungkin menjadi tidak terbatas, karena kita tidak dapat menguji setiap kombinasi yang mungkin.
- Pengujian harus dilakukan pada himpunan bagian terpilih yang dapat dilakukan dalam sumber daya yang dibatasi.
- Kelompok himpunan bagian yang dipilih ini, tetapi bukan seluruh domain pengujian, membuat pengujian perangkat lunak yang efektif.

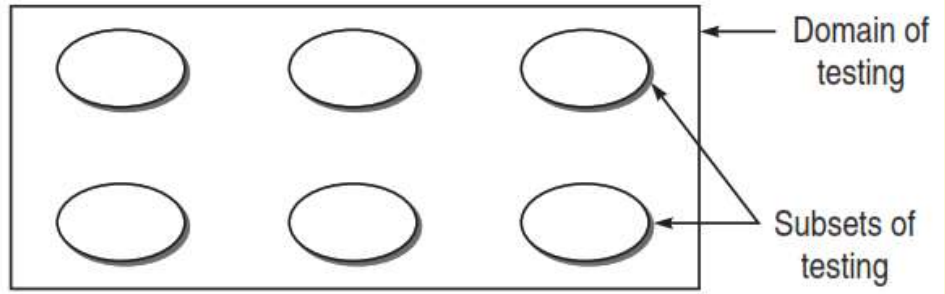


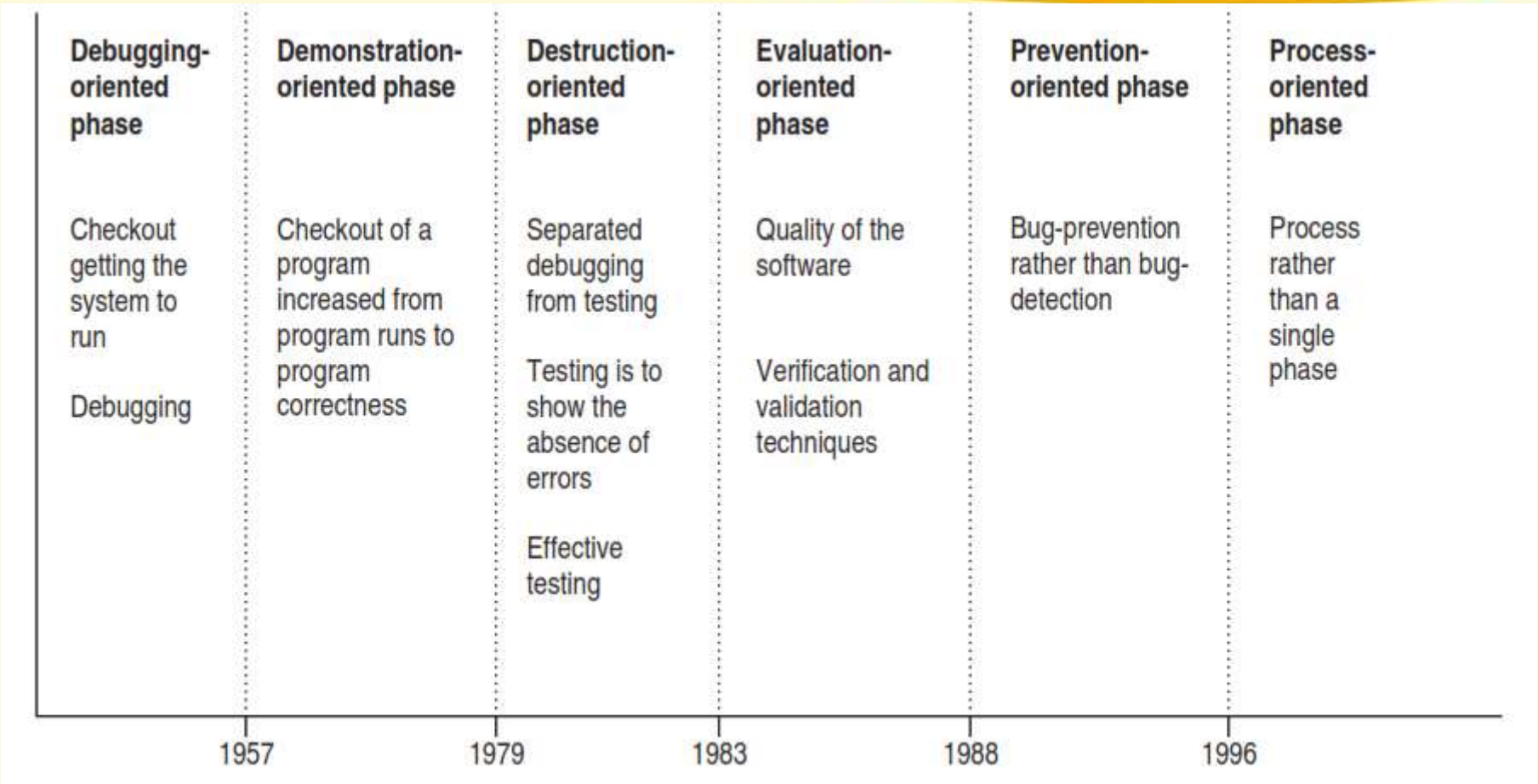
Figure 1.7 Testing domain

Kapan Pengujian Berakhir

- Beberapa aspek yang dapat menjadi pertimbangan selesainya pengujian
  - Deadline pengujian
  - Selesainya eksekusi test case
  - Bug rate sudah mencapaikan level tertentu yang telah ditetapkan
  - Keputusan manajemen

Evolusi Pengujian Perangkat Lunak

Evolusi Software Testing (1)



Evolusi Software Testing (2)

Software Testing 1.0

- Pengujian perangkat lunak hanya dianggap sebagai satu fase yang akan dilakukan setelah pengkodean perangkat lunak di SDLC.
- Tidak ada organisasi pengujian.
- Ada beberapa alat pengujian tetapi penggunaannya terbatas karena biayanya yang tinggi.
- Manajemen tidak peduli dengan pengujian, karena tidak ada sasaran kualitas.

Evolusi Software Testing (3)

Software Testing 2.0

- Pengujian perangkat lunak menjadi penting dalam SDLC dan konsep pengujian awal juga dimulai.
- Pengujian berkembang ke arah perencanaan sumber daya pengujian.
- Banyak alat pengujian juga tersedia dalam fase ini.

Evolusi Software Testing (4)

Software Testing 3.0

- Pengujian perangkat lunak dikembangkan dalam bentuk proses yang didasarkan pada upaya strategis. Artinya harus ada proses yang memberi kita *roadmap* dari keseluruhan proses pengujian.
- Harus didorong oleh tujuan kualitas sehingga semua kegiatan pengendalian dan pemantauan dapat dilakukan oleh para manajer. Dengan demikian, manajemen terlibat secara aktif dalam fase ini.

Beberapa Terminologi

Terminologi

… beberapa istilah dalam pengujian …

Software Errors, Faults dan Failures

… beberapa ungkapan …

- “Kami telah menggunakan Simplex HR di Departemen Sumber Daya Manusia selama tiga tahun dan kami tidak pernah mengalami sekalipun kegagalan (software failure).”

- “Saya mulai menggunakan Simplex HR dua bulan yang lalu. Kami menemukan banyak sekali kegagalan (failures) sehingga kami mempertimbangkan untuk menggantikan dengan perangkat lunak tersebut.”

Software Errors, Faults dan Failures

- Apakah mungkin beberapa pihak mengungkapkan pernyataan atau keluhan yang berbeda, terhadap perangkat lunak yang sama ?

- Apakah mungkin perangkat lunak yang selama bertahun-tahun memberikan layanan yang baik-baik, “tiba-tiba” berubah menjadi perangkat lunak yang memiliki bug ?
- **… YES …**

Software Errors, Faults dan Failures

… software errors …

- Software error merupakan bagian dari kode program (sebagian atau secara keseluruhan) yang tidak benar (incorrect), yang disebabkan karena kesalahan gramatikal, kesalahan logikal atau kesalahan lainnya yang dibuat oleh analis sistem, programmer atau anggota tim pengembang lainnya.
  - Merupakan faktor manusia yang menghasilkan hasil yang tidak benar (incorrect results)
- Contoh: kesalahan spesifikasi, kesalahpahaman tentang apa yang dilakukan subrutin, dan seterusnya.

Software Errors, Faults dan Failures

… software faults …

- Fault identik dengan kata defect atau bug.
- Manifestasi error dalam software.
- Fault adalah suatu kondisi yang secara nyata (dituliskan dalam software) menyebabkan suatu sistem menghasilkan kegagalan.
- Fault adalah kesalahan yang tertanam dalam fase-fase SDLC dan menghasilkan failure.

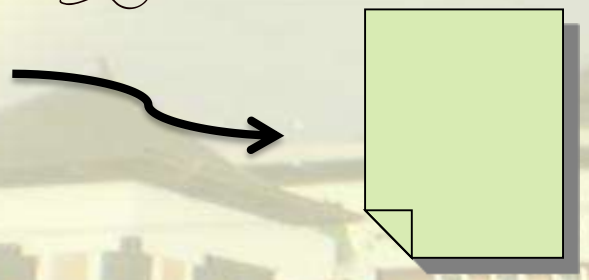
Software Errors, Faults dan Failures

… software failure …

- Ketidakmampuan sistem atau komponen untuk melakukan fungsi sesuai dengan spesifikasinya.
- Ketika hasil atau perilaku sistem yang diuji berbeda dibandingkan dengan harapan yang ditentukan
- Software faults menjadi software failures hanya jika software faults tersebut "di-aktivasi / dieksekusi".
  - Terjadi saat pengguna mencoba menjalankan bagian perangkat lunak yang mengandung fault tersebut. Tidak semua software fault akan menjadi failure.
- Akar dari setiap software failure adalah software error.

Software Errors, Faults dan Failures

Seseorang
membuat error



yang menciptakan fault
pada software

yang dapat menyebabkan
failure

Software Errors, Faults dan Failures

```
1       #include<stdio.h>
2
3       int main ()
4       {
5             int value1, value2, ans;
6
7             value1 = 5;
8             value2 = 3;
9
10            ans = value1 - value2;    // seharusnya '+'
11
12            printf("The addition of 5 + 3 = %d.", ans);
13
14            return 0;
15      }
```

... software fault ...

Software Errors, Faults dan Failures

```
1       #include<stdio.h>
2
3       int main ()
4       {
5             int value1, value2, ans;
6
7             value1 = 5;
8             value2 = 3;
9
10            ans = value1 - value2;    // seharusnya '+'
11
12            printf("The addition of 5 + 3 = %d.", ans);
13
14            return 0;
15      }
```

… dieksekusi …

The addition of 5 + 3 = 2.

… software failure…

Software Errors, Faults dan Failures

Fault: Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Test 1
[2, 7, 0]
Expected: 1
Actual: 1

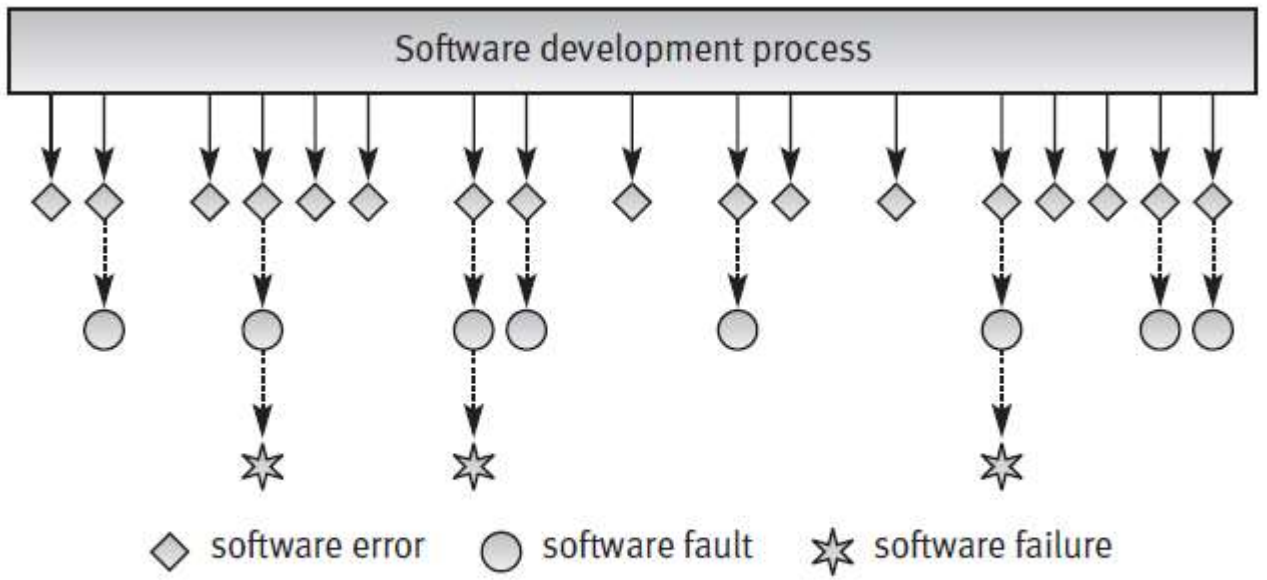
Error: i is 1, not 0, on the first iteration
Failure: none

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error: i is 1, not 0
Error propagates to the variable count
Failure: count is 0 at the return statement

Sumber : Paul Ammann & Jeff Offutt

Software Errors, Faults dan Failures



Software errors, sotware faults and software failures

Terminologi

… beberapa istilah dalam pengujian …

test case **test suite**

Test Case, Test Suite

… test case …

- Test case adalah prosedur yang terdokumentasi dengan baik yang dirancang untuk menguji fungsionalitas fitur dalam sistem.
- Test case dirancang berdasarkan teknik pengujian yang dipilih.
- Test case yang baik memiliki probabilitas tinggi untuk menunjukkan kondisi kegagalan.

Test Case, Test Suite

… test case …

- Setiap test case memiliki nomor identifikasi unik.
- Sebuah test case terdiri dari input yang diberikan ke program dan output yang diharapkan.
- Setelah eksekusi, hasil yang diamati dibandingkan dengan keluaran yang diharapkan yang disebutkan dalam test case.
- Jika keduanya sama, kasus uji berhasil.
- Jika berbeda, itu adalah kondisi kegagalan dan ini harus dicatat dengan benar untuk menemukan penyebab kegagalan.

Test Case, Test Suite

… contoh test case …

```
1.  int funct1 (int c)
2.  {
3.  int d, flag;
4.  d = c ++ ;
5.  if (d < 20000)
6.              flag = 1 ;
7.  else
8.              flag = 0;
9.  return (flag);
10. }
```

Test case	Input c	Expected output	Actual output
1.	0	1	1
2.	1	1	1
3.	20000	0	0
4.	30000	0	0
5.	-10000	1	1
6.	-20000	1	1
7.	-1	1	1
8.	-16000	1	1
9.	27000	0	0
10.	32000	0	0

Test Case ketika outputnya berbeda

Input c	Expected output	Actual output
19999	0	1
32767	Integer out of specified range	0

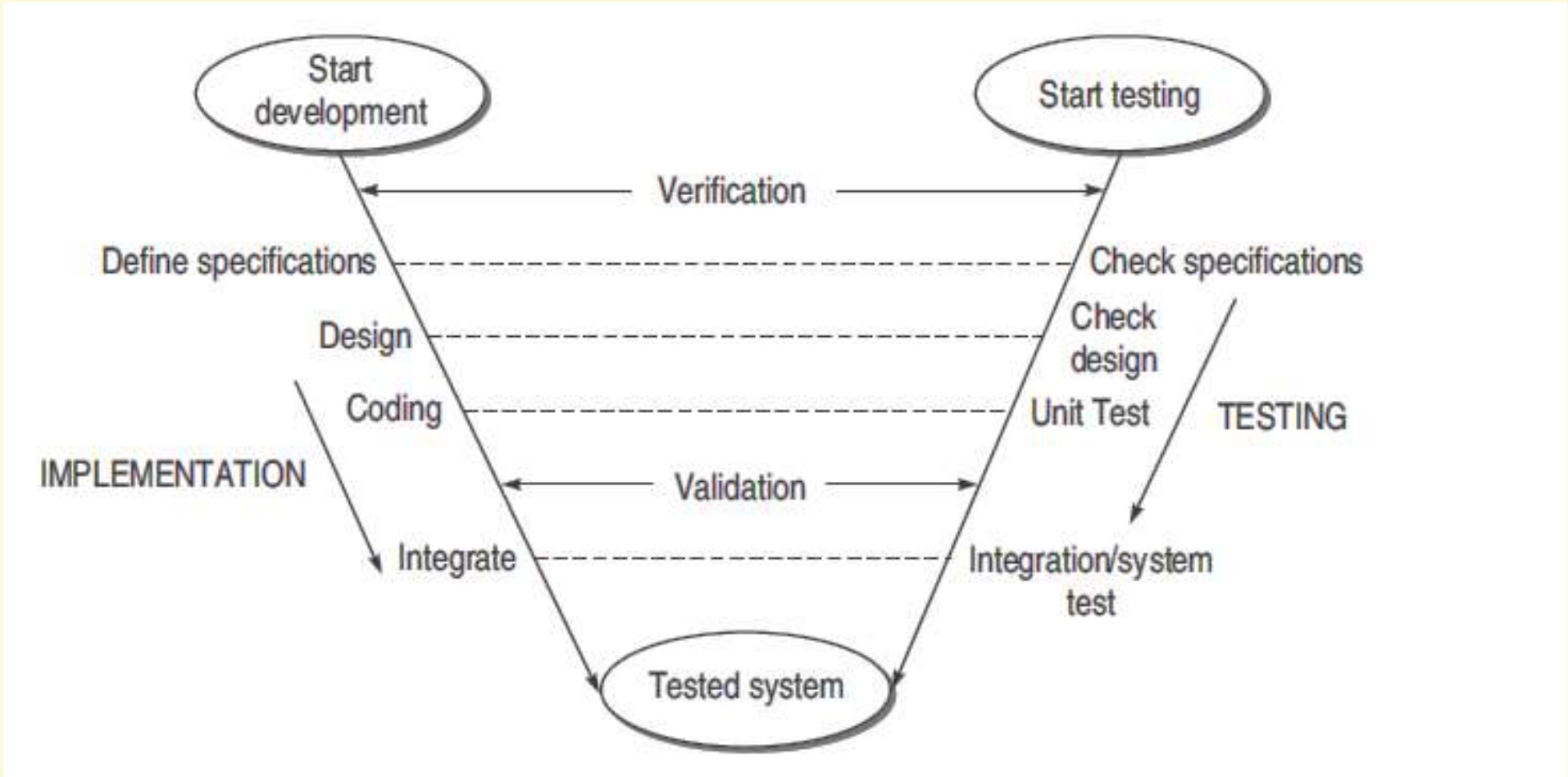
Test Case, Test Suite

… test suite …

- Himpunan test case disebut test suite.
- Mungkin memiliki test suite dari semua test case, test suite dari semua test case yang berhasil, dan test suite dari semua test case yang gagal.

Verifikasi dan Validasi

V-Testing Life Cycle Model (1)



V-Testing Life Cycle Model (2)

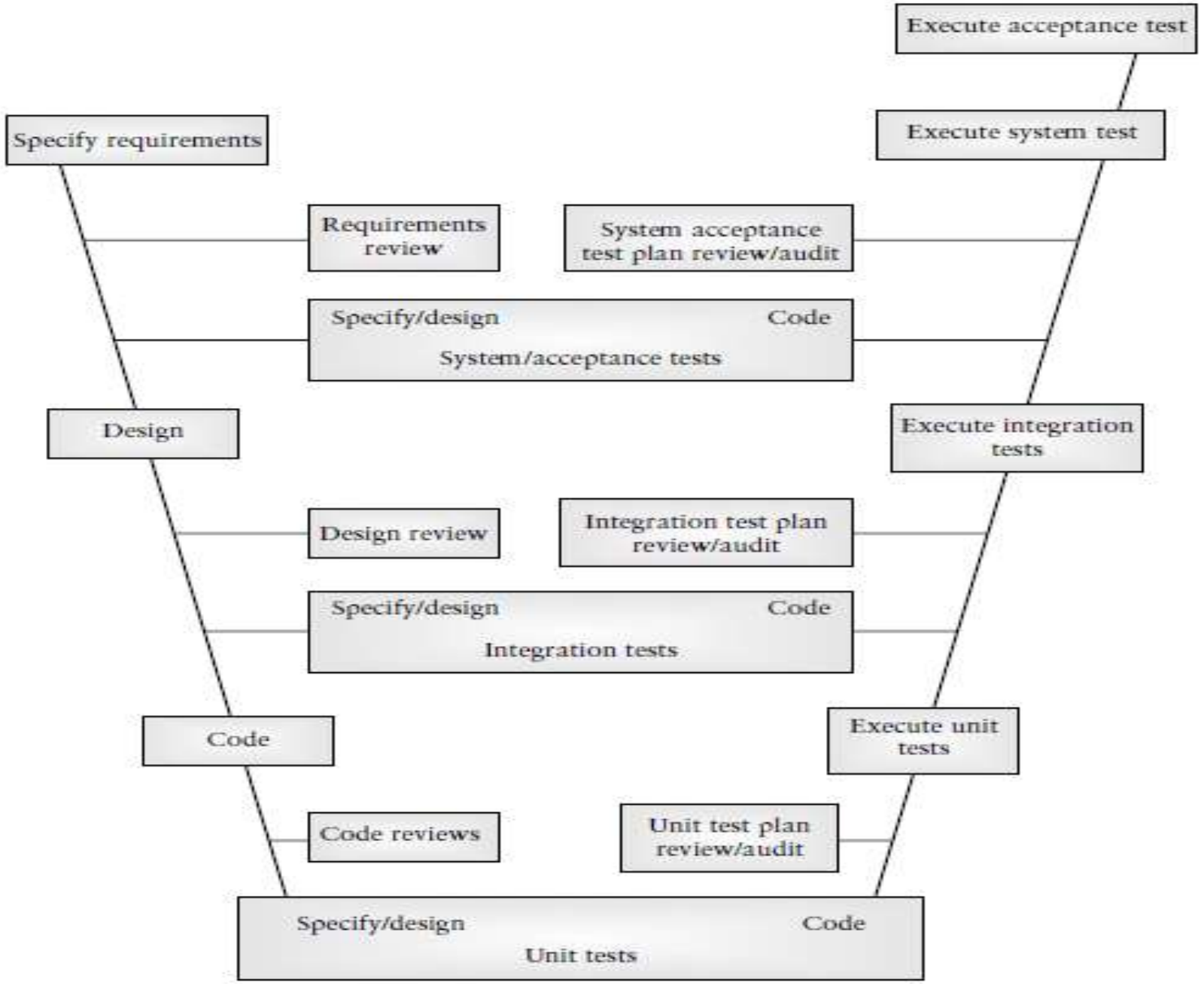
- Pengujian dapat diimplementasikan dalam alur yang sama seperti untuk SDLC
- Pengujian dapat direncanakan secara luas dalam dua kegiatan, yaitu **verifikasi** dan **validasi**
- Pengujian harus dilakukan pada setiap langkah SDLC
- V-diagram mendukung konsep pengujian awal.
- V-diagram mendukung paralelisme dalam aktivitas developer dan tester
- Tester harus terlibat dalam proses pengembangan

Verifikasi

- Verifikasi / pengujian statis / defect removal non tes
- Menerapkan kegiatan verifikasi dari fase awal pengembangan perangkat lunak dan melakukan review dokumen yang dihasilkan setelah penyelesaian setiap fase.
- Proses review dokumen kebutuhan, dokumen desain, kode sumber, dan dokumen lainnya secara manual, tanpa melakukan eksekusi kode program.
- Dapat dilakukan pada SETIAP tahap pada SDLC
- Review, inspeksi, walkthrough, dll…

Validasi

- Validasi / pengujian dinamis / defect removal tes
- Ini adalah proses mengevaluasi sistem atau komponen selama atau pada akhir proses pengembangan untuk menentukan apakah memenuhi spesifikasi yang ditentukan.
- Mengeksekusi kode sumber dan melihat caranya bekerja sesuai dengan input tertentu.
- Membutuhkan eksekusi program yang sebenarnya, membutuhkan komputer untuk menjalankan program.



The Extended/Modified V-model.

Aktivitas Validasi

Unit Testing

- Upaya validasi yang dilakukan pada modul terkecil dari sistem, dapat disebut komponen, modul, prosedur, fungsi, dll.
- Sebuah unit atau modul harus divalidasi sebelum mengintegrasikannya dengan modul lain.
- Validasi unit adalah kegiatan validasi pertama setelah pengkodean satu modul selesai.
- Yang bertanggungjawab adalah developer dari modul tersebut.

Integration Testing

- Proses menggabungkan dan menguji beberapa komponen atau modul bersama-sama.
- Modul yang diuji secara individual, ketika digabungkan dengan modul lain, tidak diuji untuk antarmukanya. Ketika satu modul digabungkan dengan yang lain dalam lingkungan yang terintegrasi, antarmuka antar unit harus diuji.
- Struktur data, pesan, atau hal lain di antara beberapa modul, maka format standar antarmuka ini harus diperiksa selama pengujian integrasi.
- Antarmuka antar modul diwakili oleh desain sistem, mengintegrasikan unit sesuai dengan desain. Penguji harus mengetahui desain sistem.
- Yang bertanggungjawab adalah tester dan developer.

System Testing

- Pengujian ini berfokus pada pengujian keseluruhan sistem terintegrasi.
- Tujuannya adalah untuk menguji validitas untuk pengguna dan lingkungan tertentu.
- Menjamin bahwa seluruh sistem diperiksa sesuai dengan spesifikasi kebutuhan (kebutuhan fungsional dan non fungsional sistem).
- Functional testing, performance testing, stress testing, security testing, compatibility testing, recovery testing, usability testing.
- Yang bertanggungjawab adalah tester dan developer

Acceptance Testing (1)

- Perangkat lunak yang dibangun harus memenuhi kebutuhan pengguna dan tidak peduli seberapa elegan desainnya, tidak akan diterima oleh pengguna kecuali membantu mencapai tujuan.
- Proses menentukan apakah sistem akhir sudah sesuai dengan kebutuhan pengguna, sudah memenuhi kriteria penerimaan (acceptance) yang disepakati dalam kontrak.
- Memberikan kesempatan kepada pengguna akhir untuk memberikan umpan balik kepada tim pengembangan.
- Memungkinkan pengguna menentukan apakah akan menerima sistem atau tidak.
- Yang bertanggungjawab adalah tester dan end user.

Acceptance Testing (2)

- Alpha testing :
  - Client menggunakan perangkat lunak di tempat pengembang (development environment).
  - Perangkat lunak digunakan dalam setting terkendali, pengembang selalu siap untuk memperbaiki kesalahan yang terjadi.
- Beta testing :
  - Dilakukan tempat pengguna (lingkungan yang sebenarnya).
  - Pengembang tidak ada
  - Perangkat lunak digunakan dalam lingkungan yang sebenarnya (target environment).

Installation Testing

- Setelah tim penguji memberikan sinyal hijau, perangkat lunak ditempatkan ke dalam status operasional dimana ia diinstal.
- Pengujian instalasi tidak menguji sistem, tetapi menguji proses pengoperasian sistem perangkat lunak.
- Proses instalasi harus mengidentifikasi langkah-langkah yang diperlukan untuk menginstal sistem.

Teknik Pengujian Perangkat Lunak

Teknik Pengujian Perangkat Lunak

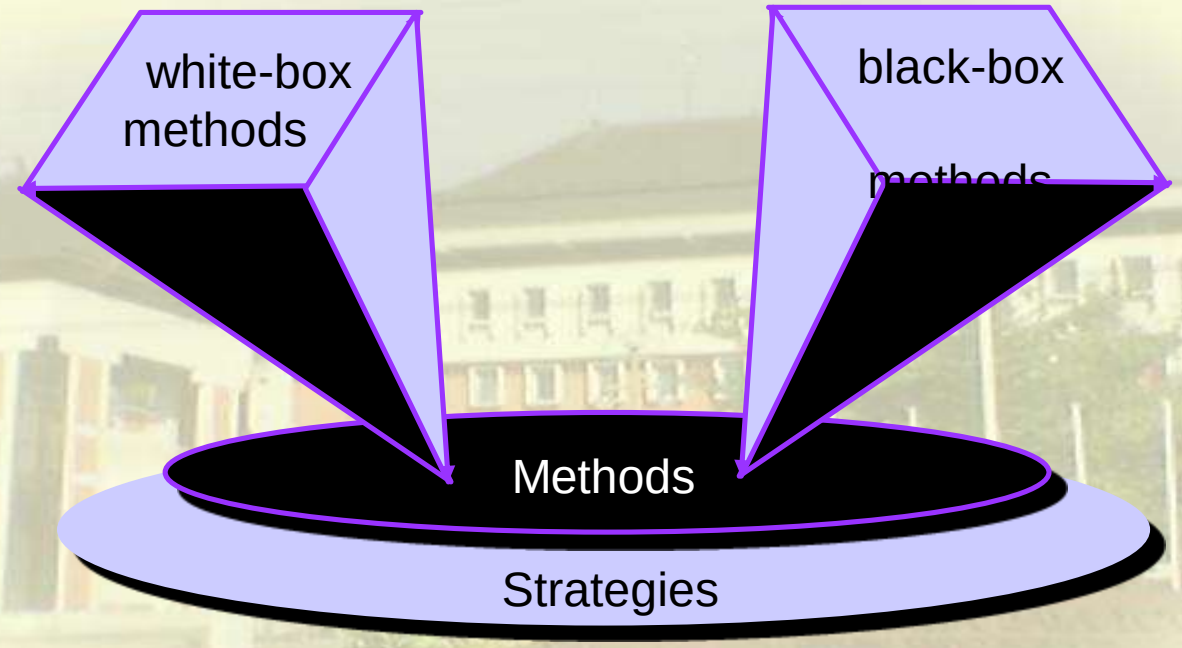
- **Pengujian statik :**

  Aktivitas pengujian tanpa mengeksekusi kode sumber. Semua aktivitas verifikasi seperti inspeksi, walkthroughs, reviews, dll. termasuk dalam kategori pengujian ini.

- **Pengujian dinamis :**

  Mengeksekusi kode sumber dan melihat bagaimana kinerjanya dengan input tertentu. Semua aktivitas validasi masuk dalam kategori ini dimana eksekusi program sangat penting.

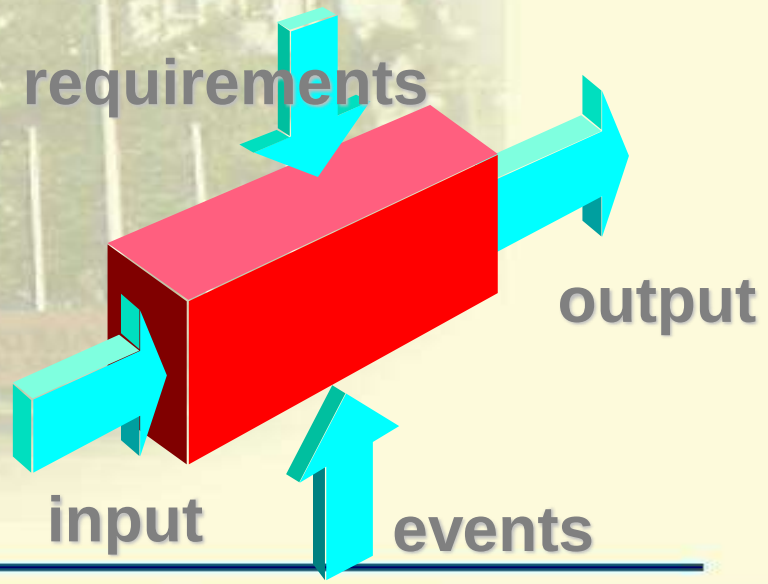
Pengujian Dinamis (1)



Sumber : Software Engineering: A Practitioner's Approach, 5/e R.S. Pressman 2005

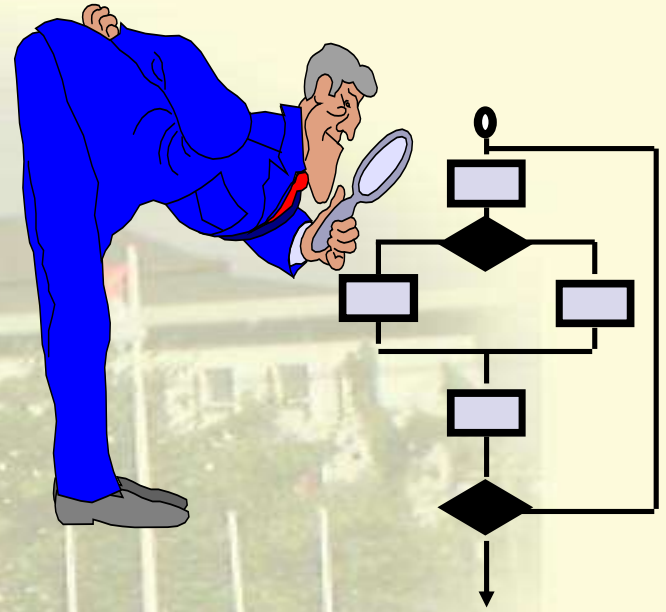
Pengujian Dinamis (2)

- Pengujian Fungsional (Black Box Testing)
  - Fokus pada output yang dihasilkan dengan memberikan input dan kondisi eksekusi
  - Membandingkan kesesuaian output dengan spesifikasi kebutuhan fungsional



Pengujian Dinamis (3)

- Pengujian Struktural (White Box Testing)
  - Menguji dengan memperhatikan mekanisme internal sistem
  - Menguji untuk memastikan operasi internal berjalan sesuai spesifikasi
  - Semua komponen diuji, memastikan semua statemen dan kondisi diuji paling tidak satu kali



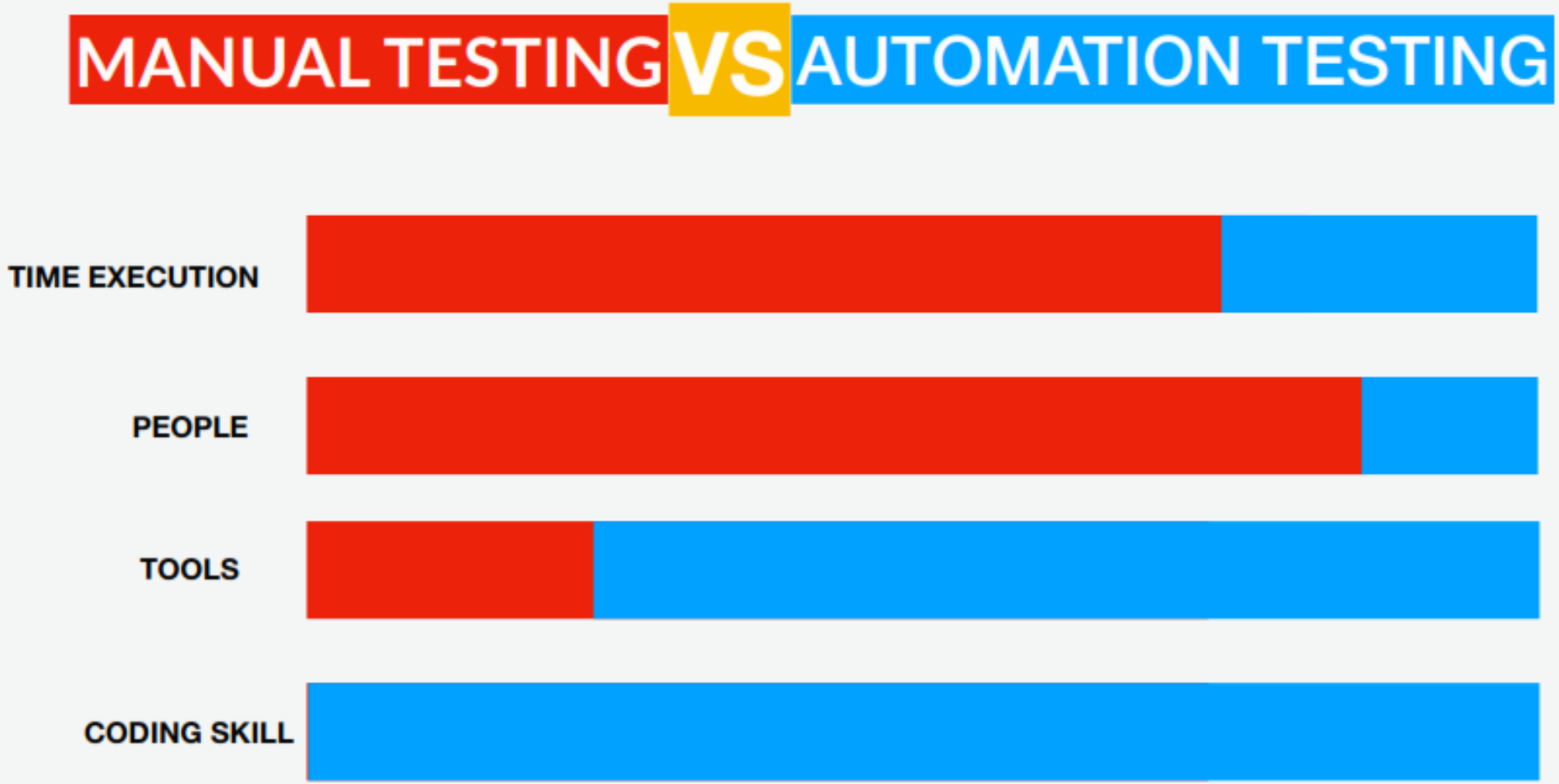
... our goal is to ensure that all statements and conditions have been executed at least once ...

Sumber : Pressmann (2005)

Pengujian Otomatis / Automation Testing

Automation Testing

- Teknik pengujian, baik statis atau dinamis, dapat diotomatisasi dengan bantuan perangkat lunak yang dapat sangat meningkatkan proses pengujian perangkat lunak.
- Automation testing efektif sehingga segala jenis aktivitas berulang dan membosankan dapat dilakukan oleh mesin, dan penguji dapat memanfaatkan waktu mereka dalam pekerjaan yang lebih kreatif dan kritis.
- Pengujian otomatis tidak boleh dilihat sebagai pengganti pengujian manual. Ada banyak aktivitas dalam siklus hidup pengujian yang tidak dapat diotomatisasi dan diperlukan upaya manual. Jadi, alat otomatis hanyalah bagian dari solusi, bukan jawaban ajaib untuk masalah pengujian.



Keuntungan Automation Testing

- Mengurangi upaya pengujian: Eksekusi test case dalam jumlah banyak menggunakan tools sangat mengurangi waktu yang dibutuhkan.
- Mengurangi keterlibatan penguji dalam melaksanakan tes: Mengotomatiskan proses eksekusi test case akan membebaskan penguji untuk melakukan pekerjaan lain
- Memfasilitasi pengujian regresi: Jika mengotomatiskan pengujian regresi, maka upaya pengujian serta waktu yang dibutuhkan akan berkurang dibandingkan dengan pengujian manual
- Menghindari kesalahan manusia
- Mengurangi biaya keseluruhan perangkat lunak: Jika waktu pengujian menurun, biaya perangkat lunak juga menurun
- Pengujian simulasi: Load performance testing, dapat membuat jutaan data virtual secara bersamaan dan secara efektif.

Automation Testing Tools (1)

Selenium

cucumber

appium

Katalon

Kobiton

LAMBDATEST

cypress

TestProject

IBM Rational software Performance Tester

APACHE JMeter™

Ranorex®

eggplant™

WORKSOFT®

Virtuoso

Cerberus

Automation Testing Tools (2)

- Selenium
  - open-source, automation testing tool no 1 untuk aplikasi web, dapat dijalankan di beberapa browser dan sistem operasi, kompatibel dengan beberapa bahasa pemrograman, memiliki pengalaman dan keterampilan dalam pemrograman dan script
- Cucumber
  - open-source Behavior Driven Development (BDD) tool, kode tes ditulis dalam bahasa Inggris sederhana yang disebut Gherkin, dapat dieksekusi pada kerangka kerja yang berbeda seperti Selenium, Ruby, dll
- Appium
  - open-source, terutama untuk aplikasi mobile (IOS & android), bisa dijalankan baik pada simulator ataupun real devices

Automation Testing Tools (3)

- Katalon Studio
  - open-source, mencakup pengujian API, Web, Desktop hingga mobile, bekerja di atas Selenium dan Appium, dapat digunakan dengan script code dan tanpa script, bisa digunakan untuk pemula ataupun yang sudah ahli.

- Kobiton
  - Pengujian pada aplikasi mobile, memiliki kemampuan otomatisasi pengujian tanpa skrip, dapat membuat tes otomatis dari tes manual, memfasilitasi pengujian pada real device, mendukung perangkat iOS dan Android terbaru, mendukung Appium, Selenium, XCUI, Expresso, dll.

Contoh Selenium & Cucumber

Steps

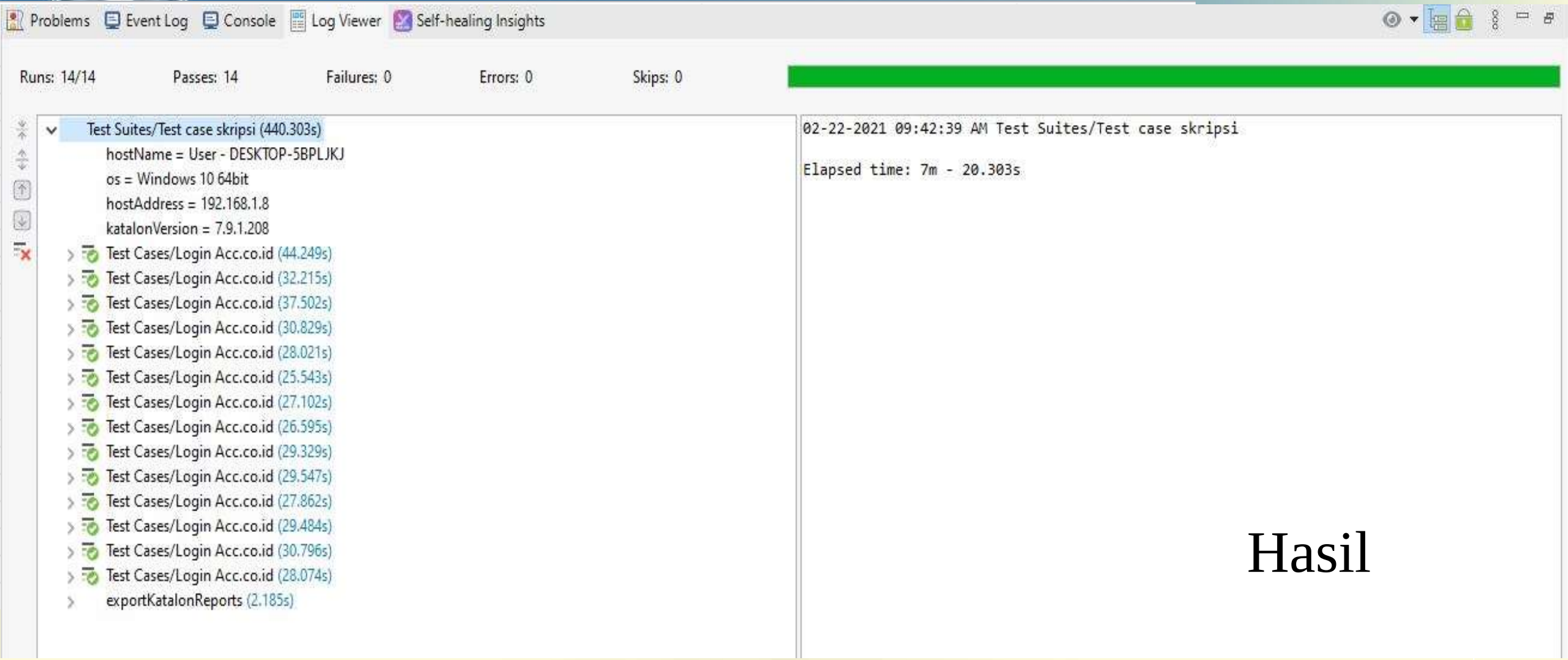
Scenario:

```
Feature: Check LMS

Scenario: Sign In LMS
    Given I sign in LMS
```

```
require 'selenium-webdriver'
require 'rubygems'
require 'rspec/expectations'

# Create new profile for Firefox
profile = Selenium::WebDriver::Firefox::Profile.new
profile.secure_ssl = true
driver =
Selenium::WebDriver::Firefox::Options#profile=profile

# Make firefox trust our web
capabilities =
Selenium::WebDriver::Remote::Capabilities.firefox(accep
t_insecure_certs: true)
driver = ''
```

```
# Scenario: Sign In LMS
Given("I sign in LMS") do
  driver = Selenium::WebDriver.for :firefox,
desired_capabilities: capabilities

driver.get('https://kuliah.uajy.ac.id/login/index.php')
  driver.find_element(id: 'username').send_keys('npm')
  driver.find_element(id:
'password').send_keys('password')

  driver.find_element(id: 'loginbtn').click
end
```

Contoh Katalon Studio



Hasil

Mitos – Mitos Software Testing

- Pengujian adalah fase tunggal dalam SDLC.
- Pengujian dimulai setelah pengembangan program.
- Pengembangan perangkat lunak lebih berharga daripada pengujian.
- Tujuan pengujian adalah untuk memeriksa fungsionalitas perangkat lunak.
- Pengujian lengkap dimungkinkan.
- Pengujian itu mudah.
- Siapapun bisa menjadi penguji.

Prinsip – Prinsip Pengujian

- Effective testing, bukan exhaustive testing
- Pengujian bukan hanya satu fase yang dilakukan di SDLC
- Pendekatan destruktif untuk pengujian konstruktif
- Pengujian awal adalah kebijakan terbaik
- Strategi pengujian harus dimulai pada tingkat modul terkecil dan diperluas ke seluruh program
- Pengujian juga harus dilakukan oleh tim independen
- Semuanya harus dicatat dalam pengujian perangkat lunak
- Input yang tidak valid dan perilaku yang tidak diharapkan memiliki kemungkinan besar untuk menemukan kesalahan
- Penguji harus berpartisipasi dalam review spesifikasi dan desain

Referensi

- Yoges Singh; **Software Testing**, Cambridge Univerity Press; 2011
- Naresh Chauhan; **Software Testing – Principles and Practices**, Oxford University Press; 2010