



Katalon Studio For Automated Testing

Sesi 1



Introduction⁺ of Software Testing

Introduction of Software Testing - Sesi 1

Software Testing

Pengujian program komputer (software testing) adalah proses menilai kualitas sebuah sistem komputer, salah satu caranya adalah dengan mencari ketidak-sesuaian program (bugs) dengan harapan pengguna dalam dokumen requirement.

Kenapa Program Perlu Di Uji ?

Pengujian program akan dilakukan dalam fase pengembangan program komputer, karena:

- Mengetahui kualitas program sebelum digunakan pengguna sesungguhnya, apakah sudah memenuhi standar kualitas yang diharapkan
- Menghindari kesalahan program, jangan sampai kesalahan ini menyusahkan pengguna, karena akan berimbas pada reputasi perusahaan
- Menghadirkan rasa aman, dan percaya diri untuk melepaskan fitur/program baru untuk dikonsumsi khalayak
- Menemukan kesalahan di fase awal tentu akan menghemat beban perbaikan daripada ketika ditemukannya kesalahan di fase production
- Tim penguji akan memberikan sudut pandang yang berbeda dalam penggunaan program komputer

Siapa yang Melakukan Pengujian ?

Setiap anggota tim pengembangan memiliki peran dalam meningkatkan dan menjaga mutu program yang baik, pengujian ini beraneka ragam sesuai dengan tingkatannya, kurang lebih:

- Product owner: menguji apakah ide fitur pada program sungguh-sungguh berguna bagi user, dan apakah sudah tepat guna.
- Developer: mengerjakan unit testing dalam rangka memastikan setiap kode yang dituliskan bekerja sebagaimana mestinya, dan tahan akan perubahan kode dimasa depan.
- Tester: memastikan program bekerja secara fungsional, kebergunaan, kinerja, dan keamanan program.
- Lead/Manager/Architects: mengawal ataupun merencanakan strategi dan rencana pengujian.



Introduction of Software Testing - Sesi 1

Kapan Seharusnya Pengujian ini dimulai ?

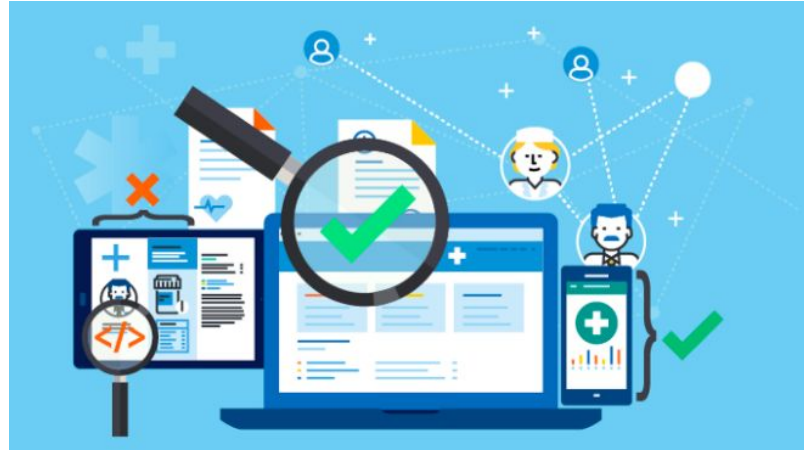
Itu sebenarnya disesuaikan dengan proses SDLC (software development life cycle) model yang dijalankan oleh tim pengembangan, bisa jadi ada fase khusus untuk pengujian, ataupun bisa jadi fase pengujian membaur dengan fase implementasi kode oleh tim pengembang, yang menarik ada satu mitos (yang saya yakini) bahwa katanya pengujian hanya bisa dilakukan ketika implementasi oleh pengembang sudah selesai, tetapi ternyata pengujian itu bisa (harusnya) dimulai bahkan ketika kode belum ditulis.

Pengujian bisa berjalan beriringan (paralel) dengan fase implementasi fitur oleh tim pengembang, contohnya melalui V model.

Fase	Aktifitas pengujian
Perancangan Requirement	Membuat acceptance criteria
Spesifikasi fungsional	Membuat skenario fungsional
Implementasi	Membuat otomatisasi pengujian
Kode selesai	Menjalankan rencana pengujian



Bagaimana Bisa Dikatakan Jika Pengujian Telah Selesai ?



Source : <https://www.guru99.com/software-testing-introduction-importance.html>

Pengujian bisa dijalankan secara manual ataupun automated.

Pengujian manual seperti verifikasi rancangan requirement, mengatur strategi dan rencana pengujian, persiapan data, regression test, dan eksekusi pengujian, sedangkan pengujian otomatisasi seperti membuat script untuk unit testing, integration test, UI test, performance test, bahkan security test dengan bantuan aplikasi dan program khusus sesuai kebutuhan.

Quality Assurance (QA) Behaviour

Quality assurance memiliki peran untuk memastikan kualitas dari sebuah produk, sama halnya di dalam *software development*. Untuk memastikan kualitas tersebut, maka dilakukan proses-proses yang menghasilkan *output* seperti dokumentasi, laporan dan atau metrik kualitas dari sebuah produk.

quality assurance adalah **software tester** tetapi **software tester** belum tentu **quality assurance**.

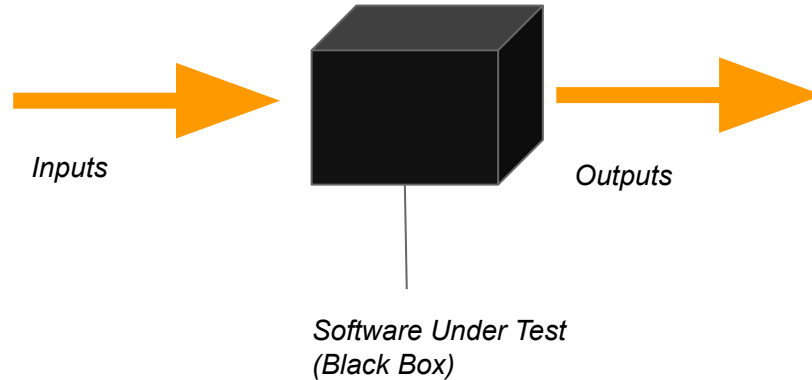
Dimana yang menjadi perbedaan adalah :

"CUSTOMER EXPERIENCE"

SEORANG **QA** MEMPUNYAI PERAN UNTUK MEMASTIKAN ASPEK/PERSPEKTIF TERHADAP **CUSTOMER EXPERIENCE**.

Jenis Testing

1. BLACK BOX TESTING = FUNCTIONAL TESTING



Metode pengujian Perangkat Lunak yang digunakan untuk menguji perangkat lunak tanpa mengetahui struktur internal kode atau Program.

Introduction of Software Testing - Sesi 1

Cara Kerja Black Box Testing

Pengujian Black Box dilakukan dengan cara yang relative bertentangan dengan kebutuhan yang ada dan memastikan sistem dapat menangani semua masukan yang tidak sesuai. Contoh :

- A. Fungsi-fungsi yang tidak benar/hilang yang terdapat pada software
- B. Interface yang salah
- C. Kesalahan dalam struktur data/akses database
- D. Permasalahan dalam kinerja perangkat lunak
- E. Inisialisasi dan kesalahan terminasi perangkat lunak.

Skenario Test	Test Case	Expected Result	Results	Remarks
Tambah Data Barang	Masukkan semua data barang, kemudian klik "Tombol" tambah barang.	Data Barang Berhasil Di Tambah	Sesuai	Normal
Hapus Data Barang	Klik barang yang mau dihapus kemudian, klik "Tombol" Hapus	Data Barang Berhasil di Hapus	Sesuai	Normal
Update Jumlah Barang	Klik Jumlah barang yang mau diupdate kemudian, klik "Tombol" Update	Data Barang Berhasil di Update	Sesuai	Normal



Why Black Box Testing ?

KEUNTUNGAN BLACK-BOX TESTING

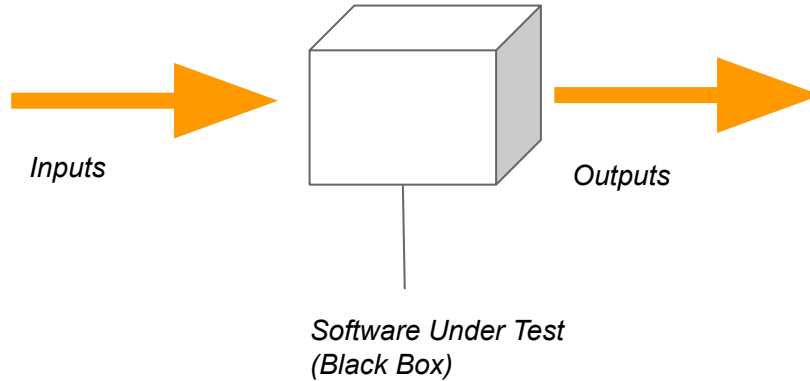
- Penguji tidak perlu memiliki pengetahuan tentang bahasa pemrograman tertentu
- Pengujian yang dilakukan berdasarkan sudut pandang *user* agar dapat mengungkapkan konsistensi atau ambiguitas dalam spesifikasi.
- Programmer dan tester memiliki ketergantungan satu sama lain
- Efisien untuk segmen kode besar
- Akses kode tidak diperlukan
- Pemisahan antara perspektif pengguna dan pengembang

KEKURANGAN BLACK-BOX TESTING

- Uji kasus sulit di desain tanpa spesifikasi yang jelas
- Kemungkinan memiliki pengulangan tes yang sudah dilakukan oleh *programmer*
- Beberapa bagian back end tidak diuji sama sekali.
- Cakupan terbatas karena hanya sebagian kecil dari skenario pengujian yang dilakukan
- Pengujian tidak efisien karena keberuntungan *tester* dari pengetahuan tentang perangkat lunak internal

Jenis Testing

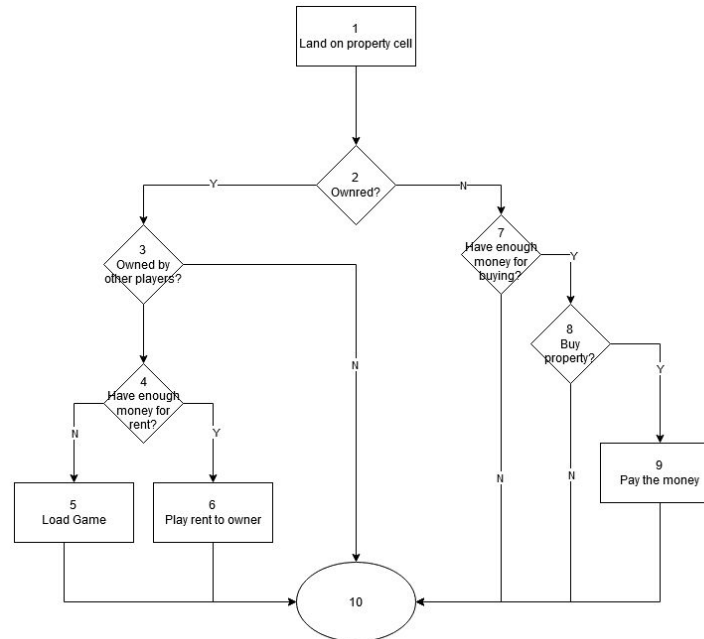
2. WHITE BOX TESTING = FUNCTIONAL TESTING



Metode pengujian Perangkat Lunak yang digunakan untuk menguji perangkat lunak dengan cara melihat modul untuk dapat meneliti dan menganalisa kode dari program yang dibuat ada yang salah atau tidak.

Cara Kerja White-Box Testing

Pengujian ke *white box testing* adalah menguji yang didasarkan kepada pengecekan ke dalam detail rancangan, penggunaan yang dilakukan struktur kontrol dari suatu desain pemrograman untuk dapat membagi pengujian ke beberapa kasus pengujian. Dan didapat bahwa *white box testing* menggunakan petunjuk untuk menghasilkan program yang diharapkan dan efisien.



Pengujian ke *white box testing* selalu merujuk ke tampilan grafik / flowchart, dimana kita dapat menghitung jumlah jalur/journey melalui kode.

Biasanya pada White Box Testing menggunakan metrik yang dikenal dengan penomoran **cyclomatic**, apa itu ?

“perangkat lunak pengukuran yang memungkinkan pengukuran kuantitatif dari kompleksitas logika program”

Cara mudah menghitung jumlah cyclomatic adalah menghitung jumlah conditional / logic dan kita tambahkan 1 (+1),

Nah lihat pada flowchart diatas dari game diatas :

Ada 6 conditional.

Kita akan menghitung jalur journey user :

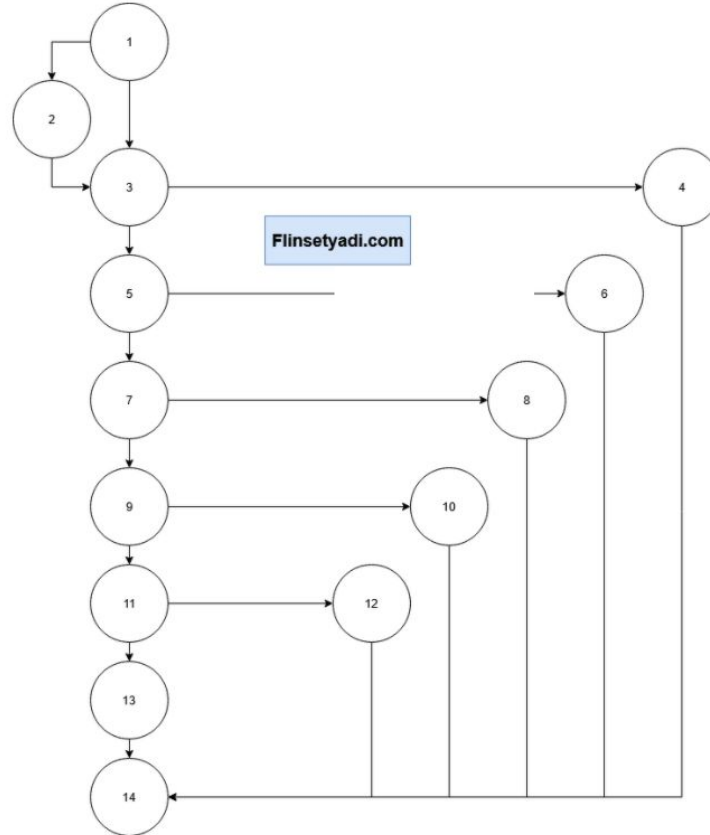
- A. 1-2-3-4-5-10 => properti yang dimiliki oleh orang lain, tapi user tidak memiliki uang untuk nyewa
- B. 1-2-3-4-6-10 => properti yang dimiliki oleh orang lain akan tetapi membayar sewa
- C. 1-2-3-10 => merupakan properti yang dimiliki oleh pemain.
- D. 1-2-7-10 => merupakan properti yang tersedia, punya uang dan tidak ingin membelinya
- E. 1-2-7-8-10 => merupakan properti yang tersedia, punya uang dan tidak ingin membelinya
- F. 1-2-7-8-9-10 => properti yang tersedia, punya uang dan membelinya

Sekarang bagaimana laporan algoritma dipetakan dalam node grafik, dengan aturan :
Node grafik - nomor disebelah kiri.

```
1  public double calculate(int amount)
2  {
3      -1- double rushCharge = 0;
4      -1- if (nextday.equals("yes") )
5      {
6          -2- rushCharge = 14.50;
7      }
8      -3- double tax = amount * .0725;
9      -3- if (amount >= 1000)
10     {
11         -4- shipcharge = amount * .06 + rushCharge;
12     }
13     -5- else if (amount >= 200)
14     {
15         -6- shipcharge = amount * .08 + rushCharge;
16     }
17     -7- else if (amount >= 100)
18     {
19         -8- shipcharge = 13.25 + rushCharge;
20     }
21     -9- else if (amount >= 50)
22     {
23         -10- shipcharge = 9.95 + rushCharge;
24     }
25     -11- else if (amount >= 25)
26     {
27         -12- shipcharge = 7.25 + rushCharge;
28     }
29     else
30     {
31         -13- shipcharge = 5.25 + rushCharge;
32     }
33     -14- total = amount + tax + shipcharge;
34     -14- return total;
35
36 }
37
38 //end calculate
```



Hasil Flowchart Testing



MENENTUKAN KOMPLEKSITAS CYCLOMATIC DARI FLOWCHART

$$\begin{aligned}V(G) &= E - N + 2 \\&= 19 - 14 + 2 \\&= 7\end{aligned}$$

Dibawah ini merupakan keterangan dari rumus diatas :
E: Jumlah Busur / Link
N: Jumlah Simpul

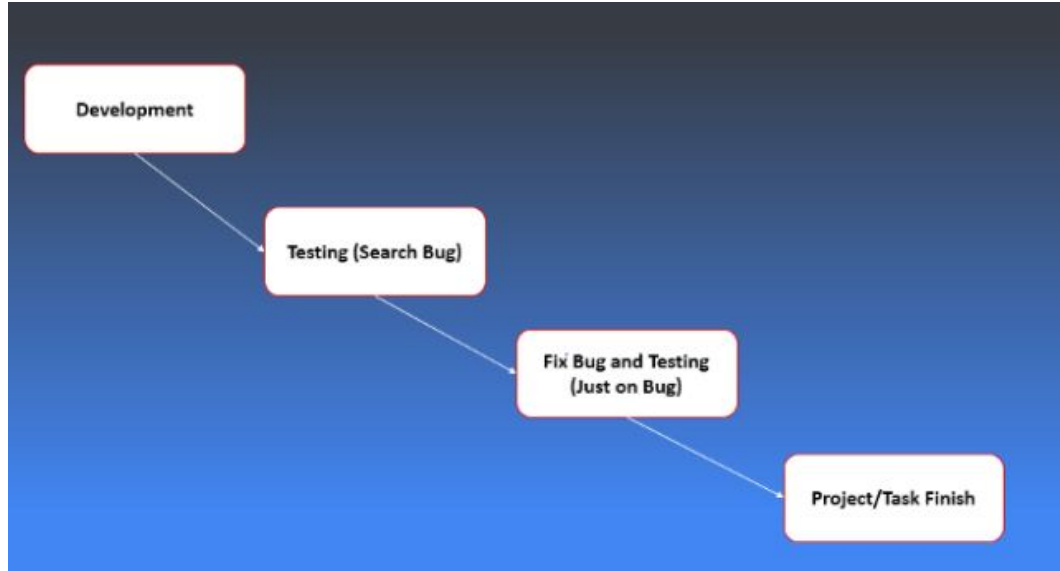
Ini menjelaskan bahwa batas atas pada ukuran basis set.
Artinya memberikan batas atas pada ukuran basis set yang memberikan alur journey user yang perlu kita cari.

Path	Journey User
1	1-2-3-5-7-9-11-13-14
2	1-3-4-14
3	1-3-5-6-14
4	1-3-5-7-8-14
6	1-3-5-7-9-10-14
7	1-3-5-7-9-11-12-14
8	1-3-5-7-9-11-13-14

Path	Next Day	Amount	Expected Results
1	yes	10	30.48
2	no	1500	?????.??
3	no	300	345.75
4	no	150	174.125
6	no	75	90.3875
7	no	30	39.425
8	no	10	15.975



Software tester pada umumnya bekerja apabila sebuah modul atau project telah dikerjakan atau dianggap tuntas oleh developer.



<https://www.guru99.com/software-testing-life-cycle.html>

Methodologi Dunia Tester

Metode Scrum sebagai metode development sehingga modul akan selesai bisa dalam waktu singkat *per-Task* atau *per-Sprint*.

Unit Testing

Unit test merupakan *testing level* untuk menguji bagian *software* terkecil.

Tujuan dilakukannya *unit test* ini yaitu untuk menemukan *error* lebih cepat sehingga dapat mengantisipasi *error* yang lebih kompleks dikemudian hari.

Unit test dilakukan dengan cara kolaborasi antara tim *developer* dan tim *quality assurance*.

Dari tim *developer* yaitu melakukan pengujian terhadap prosedur atau fungsi dalam sebuah *program code*.
Sedangkan

dari tim *QA* yaitu melakukan pengujian terhadap *parameter* dan *response API*, *field data* pada *database*, atau terhadap *UI elements* yang disesuaikan dengan kebutuhan *development* dan *software requirement*.



Introduction of Software Testing - Sesi 1

Integration Testing

Integration test yaitu pengujian dari gabungan atau integrasi antar *unit* code untuk menghasilkan *flow scenario* fungsi, fitur dan atau *module* tertentu.

Contohnya *register*, *login*, *search*, dan lainnya. *Integration testing* dapat dilakukan dengan metode *end-to-end testing*, dimana *end-to-end testing* merupakan *testing* alur dari awal sampai selesai.

Hal yang harus diperhatikan dan dipastikan dari *testing level* ini yaitu bagaimana kesesuaian alur data pada *database* dengan *response* API dan tampilan di-UI, atau bagaimana alur integrasi antar API bekerja.

Selain menguji integrasi antar *unit*, contoh lain untuk *integration testing* yaitu menguji integrasi antar *software*.

Integration testing juga dapat dikombinasikan dengan pendekatan *exploratory testing* yaitu proses *testing* tanpa ada panduan atau *test cases* atau *requirement* tertentu.



Introduction of Software Testing - Sesi 1

System Test

System test adalah pengujian yang dilakukan untuk cakupan yang lebih besar. Pengujian ini dilakukan ketika *integration test* selesai.

System test merupakan pengujian yang dilakukan secara keseluruhan terhadap sebuah *software*.

Misalnya dalam satu *development life-cycle* akan membuat fitur transaksi seperti pembelian atau pembayaran, maka yang harus diperhatikan dalam pengujian ini adalah *module* atau fitur lainnya tidak boleh ada *error* dikarenakan *impact* dari *development* fitur baru tersebut.

Salah satu jenis pengujian yang dapat dilakukan pada *system test* ini adalah *regression test*.

regression test adalah pengujian *software* untuk memastikan bahwa penambahan atau perubahan code baru tidak mempengaruhi code yang sudah ada.

Panduan :

- Dilakukan dengan mengeksekusi semua *test case* yang ada, mencakup *software* secara keseluruhan
- Dilakukan dengan mengeksekusi sebagian *test case* yang berkaitan dengan penambahan fitur
- Dilakukan dengan mengeksekusi semua *test case* yang mempunyai prioritas/*level* tertentu,

Introduction of Software Testing - Sesi 1

Acceptance Test

Acceptance test adalah tahap uji terakhir pada *functional testing level*. Tahap ini QA secara umum tidak terlibat langsung, karena yang bertanggung jawab pada tahap *acceptance test* adalah *product manager* atau tim yang berhubungan langsung dengan *user* yang akan menggunakan *software*.

Hal yang perlu diperhatikan dalam *acceptance test* ini yaitu *software* yang akan diuji sudah melewati tahap uji sebelumnya, kemudian mempersiapkan dokumentasi serta alur *test* yang akan dieksekusi menyesuaikan dengan *user story* dan alur bisnis.

Ketika sedang melaksanakan *acceptance test*, *behavior user* dalam menggunakan *software* juga perlu diperhatikan. Tujuannya adalah untuk mengevaluasi apakah fitur tersebut sudah nyaman digunakan atau masih butuh improvisasi.

Tinggal bagaimana QA tersebut dapat mengedukasi dirinya sendiri dan tim dalam mengembangkan dan memastikan kualitas dari sebuah *software*.

Introduction of Software Testing - Sesi 1

Teknik Testing

1. Manual Testing

Langkah untuk mencari cacat atau bug pada program perangkat lunak,

pada metode ini tester/penguji memiliki peran penting sebagai pengguna akhir untuk pengecekan semua fitur aplikasi bekerja dengan benar sehingga bisa memiliki SKEMA TESTING.

Apa yang diharapkan dari SKEMA TESTING ?

mempermudah Penguji/Tester untuk melakukan automation testing baik dari penyiapan sisi script dan urutan testing serta membagi testing dalam format POSITIF & NEGATIF TESTING.



Skema Testing : LOGIN

POSITIF TESTING

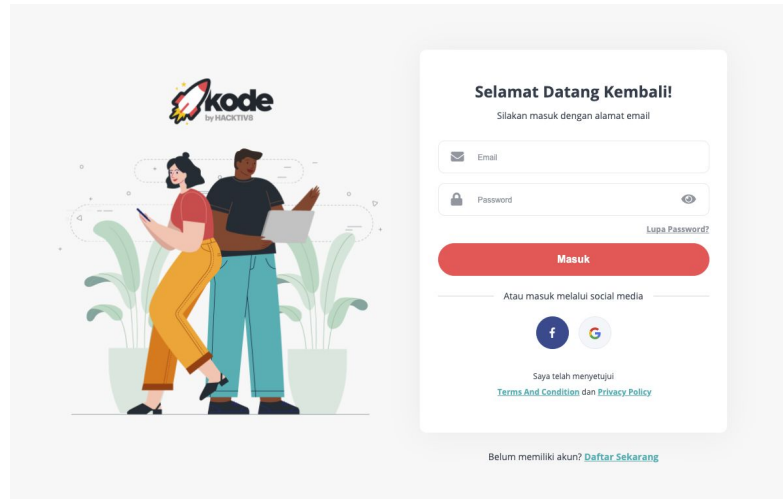
“Login dimulai dari memasukkan Username/Email (example@domain.com)+ Password (12345678)”

“Setelah itu klik Button Login. Inilah yang sering dilakukan oleh user pada umumnya yang mengerti atau sudah terbiasa dengan sistem”

NEGATIF TESTING

“Login dimulai dari memasukan Username/Email (**eXaMpD*&le@domain.om**)+Password(12345678)”

“Setelah itu user akan klik Button Login. Akan tetapi sistem akan memberikan notifikasi bahwa **Email/Password anda Salah**”



Analisa

Ketika user menerima notifikasi tersebut dalam satu informasi tentu akan **membuat user bingung manakah yang salah ? Apakah email yang digunakan atau password yang di Inputkan?**

Hal inilah yang akan menjadi laporan bug tester,

dimana Developer harus memperbaiki notifikasi tersebut terbagi menjadi **dua yaitu pada kolom Email dan Password secara terpisah** sehingga ketika user melakukan kesalahan pada salah satu kolom ia dapat mengetahui secara pas mana yang harus diperbaiki,

contoh apabila password yang di Inputkan lah yang salah maka user tinggal klik button Lupa Password (Forgot Password) untuk melakukan reset password yang telah ia lupakan.

Setelah membuat skema seperti contoh tadi, hasil dari temuan bug atau bahkan saran penggunaan maka saya akan mengklasifikasikan bug tersebut lalu membuat Task pada Trello/Asana untuk hasil testing yang telah saya temukan dengan memberikan informasi Hasil Temuan Bug tersebut kepada Developer yang bersangkutan sehingga segera dapat untuk diPerbaiki dalam waktu yang tepat.



Then.

Ketika sudah di lakukan perbaikan oleh Developer, Seorang Penguji wajib melakukan uji testing lagi sesuai skema testing sebelumnya,

artinya akan ada pengulangan aktifitas yang sama dengan ideal waktu setiap saat , bagaimana solusi untuk hal ini ?

Perlu adanya :

2. Automation Testing..

adl langkah untuk mencari cacat atau bug pada program perangkat lunak, pada metode ini tester/penguji memiliki peran penting seperti : Menulis script test dan melakukan penjadwalan testing setiap saat *untuk case diatas* dan menerima report setiap saat atau menerima report JIKA HANYA ada status error yang disebabkan salah satunya:

Developer melakukan Update Script baru di Source Code Aplikasi.

Nah Automate Testing lumrah dilakukan oleh tools support seperti :



SUGGESTION.

Ketahui dulu Goals Testing yang ingin anda lakukan :

Jika goalsnya berhubungan dengan VISUAL Feedback maka gunakan Manual Testing. sedangkan

jika goalsnya berhubungan dengan Finding Bug dan dilakukan secara berkala penerapan AUTOMATE TESTING harus dilakukan.

Kapan menggunakan Manual Testing ?

Untuk menggunakan Manual testing akan baik digunakan pada area ataupun skenario berikut :

- **Exploratory Testing.** Sangat membutuhkan pengetahuan seorang tester analitikal/logika skil, kreatifitas, dan intuisi.
- **Usability Testing.** Area ini dibutuhkan untuk melakukan pengecekan user-friendly, efisien, ataupun kenyamanan untuk software atau produk untuk end users.
- **Ad-hoc Testing.** Skenario ini dilakukan tanpa persiapan atau tidak menuliskan test case, sehingga QA secara acak melakukan tes pada fungsi di sistem. Dimana tujuannya adalah untuk secara kreatif untuk **"Merusak"** sistem dan mencari kesalahan.



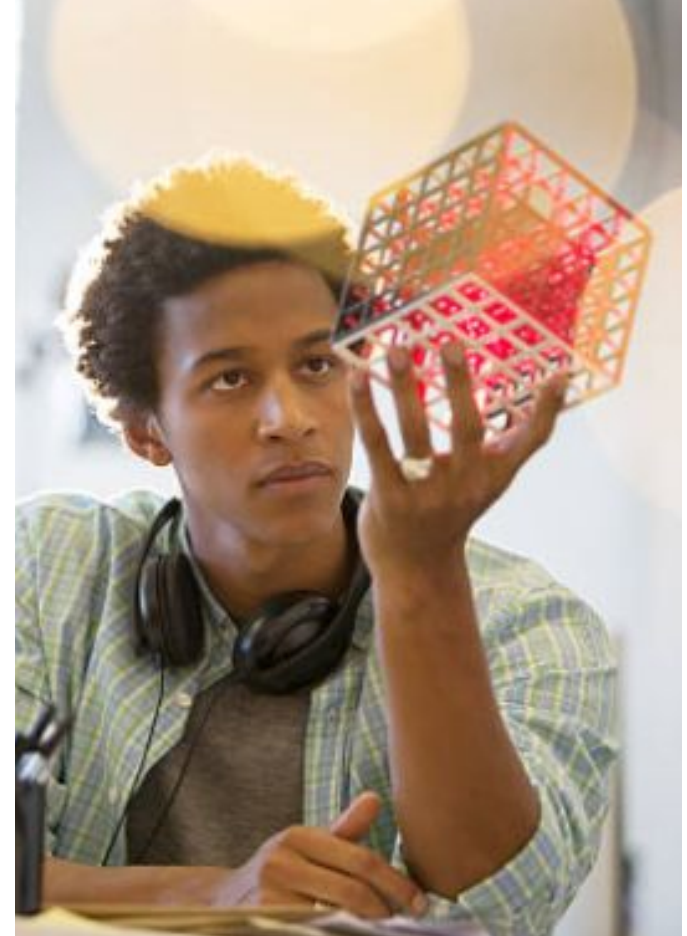
Kapan menggunakan Automated Testing ?

Untuk menggunakan Automated testing akan baik digunakan pada area ataupun skenario berikut :

- **Regression Testing.** Automated testing sangat cocok jika banyaknya perubahan pada koding dan abiliti untuk mengerjakan secara tepat waktu.
- **Load Testing.** Sangat membutuhkan automated testing untuk load test, seperti penggunaan untuk tes respon API.
- **Repeated Execution.** Untuk tes yang berulang lebih baik menggunakan Automated testing.
- **Performance Testing.** Pengujian yang membutuhkan simulasi ribuan pengguna sangat membutuhkan Automated testing.



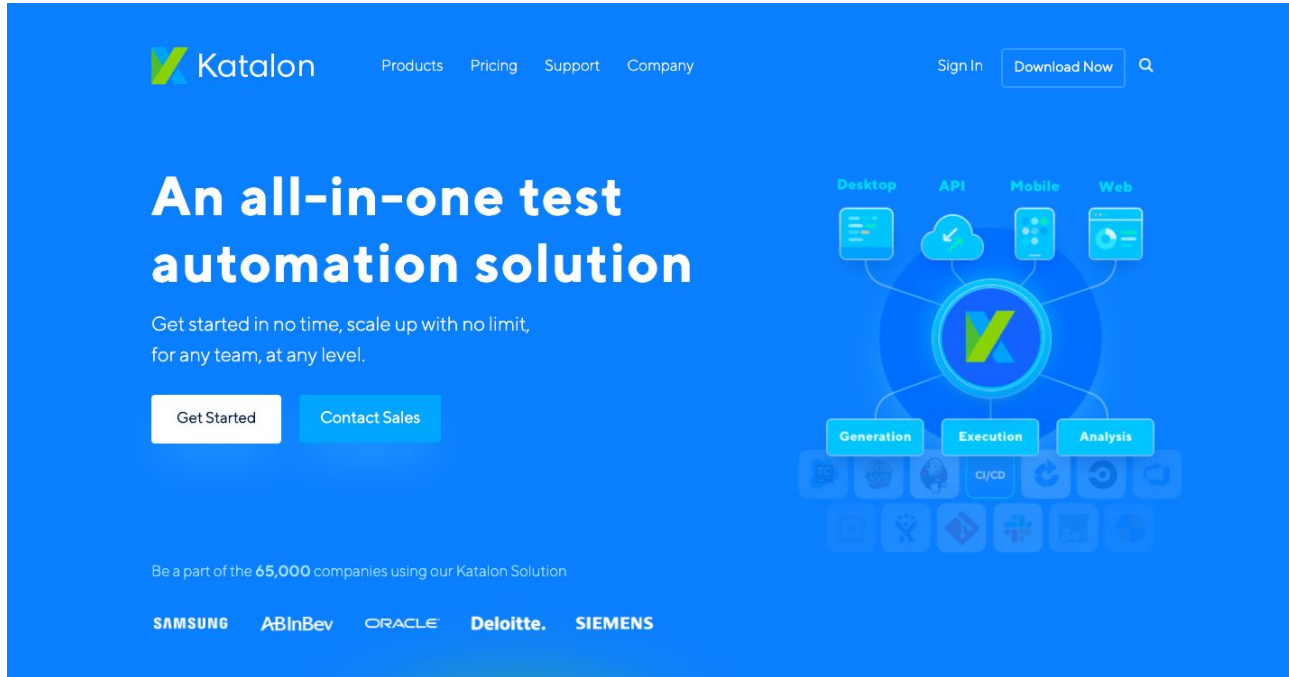
1. *Recognize Industry Changes*
2. *Start Small Thing but Successful*
3. *Practice*
4. *Learn Coding*
5. *Make Developers your Partner*
6. *Friends Network*
7. *Take Opportunities*
8. *Decide the Project Structure*



Introduction of Software Testing - Sesi 1

Katalon Studio Download

Download Katalon Studio on :

The image is a screenshot of the Katalon Studio website's homepage. The background is a solid blue color. At the top left is the Katalon logo, which consists of a stylized 'K' made of two overlapping shapes (one green, one blue) followed by the word 'Katalon' in white. To the right of the logo are navigation links: 'Products', 'Pricing', 'Support', and 'Company'. Further right are 'Sign In' and a 'Download Now' button. A search icon is also present. The main heading in the center-left is 'An all-in-one test automation solution' in large white font. Below it is a sub-headline: 'Get started in no time, scale up with no limit, for any team, at any level.' There are two buttons: 'Get Started' (white with blue text) and 'Contact Sales' (blue with white text). To the right of the text is a diagram showing a central Katalon logo connected to four boxes labeled 'Desktop', 'API', 'Mobile', and 'Web'. Below these are three boxes labeled 'Generation', 'Execution', and 'Analysis'. At the bottom of the diagram is a 'CI/CD' box. The bottom of the page features a statement 'Be a part of the 65,000 companies using our Katalon Solution' and a row of logos for 'SAMSUNG', 'ABInBev', 'ORACLE', 'Deloitte.', and 'SIEMENS'.

https://www.katalon.com/?pk_abe=AB_testing_Homepage_08_2020&pk_abv=layout1

Next. Silahkan kerjakan Quiz di sesi 1 ini. Sampai Jumpa di sesi selanjutnya..