BAB IV

PENGENALAN *TESTING*

A. Pengetahuan yang diperlukan dalam melakukan *software testing*

1. Mengidentifikasi pentingnya *software testing* dalam proses pengembangan perangkat lunak.
2. Mengidentifikasi tipe-tipe *software testing,* dampak, kelebihan, dan kekurangan dari implementasi suatu tipe *software testing.*
3. Memahami alur pelaksanaan *software testing*, menyusun *test case*, dan *test report.*
4. Memahami pendekatan *Test Driven Development* dalam *software testing.*
5. Mengindentifikasi macam-macam *tools* dalam penerapan *Test Driven Development.*
6. Memahami implementasi *software testing* di lingkungan DevOps.
7. Mengidentifikasi macam-macam *tools* DevOps untuk *software testing.*

B. Keterampilan yang diperlukan dalam melakukan *software testing*

1. Keterampilan memilih tipe *software testing* sesuai dengan kebutuhan.
2. Keterampilan menyusun *test plan* dan membuat *test report*.
3. Keterampilan menggunakan *tools* yang membantu pelaksanaan *software testing*.

C. Sikap yang diperlukan dalam melakukan *software testing*

1. Disiplin
2. Teliti
3. Objektif
4. Bertanggung jawab

D. Pengenalan *Testing*

1. Konsep Dasar *Software Testing*

Software Testing atau Pengujian Perangkat Lunak / Aplikasi adalah proses memverifikasi dan memvalidasi untuk menentukan kesesuaian *software* dengan persyaratan atau kebutuhan yang telah ditentukan.

Pengujian aplikasi tidak selalu dilakukan di akhir, namun juga dapat dilakukan secara iteratif selama proses pengembangan aplikasi. Tujuan dilakukan pengujian perangkat lunak adalah:

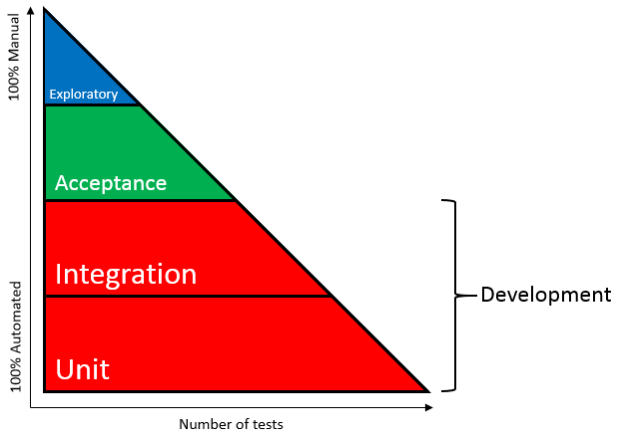
- Mengurangi biaya pengembangan perangkat lunak
- Mengkonfirmasi kesesuaian aplikasi yang dikembangkan dengan persyaratan konsumen/klien atau tidak.
- Menemukan kecacatan *(defect)* dalam perangkat lunak.
- Memberikan produk yang berkualitas dan produk bebas risiko ke konsumen/klien.

Lalu, apa saja yang dilakukan dalam aktivitas *testing* ini? Terdapat dua aktivitas yang biasa dilakukan dalam proses *software testing*, yakni verifikasi dan validasi. Verifikasi adalah sebuah aktivitas untuk memastikan perangkat lunak dikembangkan dengan benar *(well-engineered)*. Aktivitas ini sering disebut dengan *static testing.* Proses verifikasi meliputi pengecekan dokumen, desain, kode, dan program. Sementara, validasi adalah sebuah aktivitas yang dilakukan untuk memastikan perangkat lunak yang dikembangkan sesuai dengan kebutuhan calon pengguna. Aktivitas ini sering disebut dengan *dynamic testing*. Berikut ini merupakan perbedaan antara aktivitas verifikasi dan validasi secara terperinci (Guru99, 2020):

Verifikasi	**Validasi**
Proses verifikasi meliputi pengecekan dokumen, desain, kode, dan program.	Mekanisme dinamis untuk menguji dan memvalidasi produk yang sebenarnya
Tidak melibatkan eksekusi kode	Selalu melibatkan eksekusi kode
Verifikasi menggunakan metode seperti ulasan, penelusuran, inspeksi, dan pemeriksaan meja, dll.	Menggunakan metode seperti *Black Box Testing, White Box Testing*, dan pengujian non-fungsional.
Memeriksa apakah perangkat lunak sesuai dengan spesifikasi	Memeriksa apakah perangkat lunak memenuhi persyaratan dan harapan pelanggan
Menemukan bug di awal siklus	Dapat menemukan bug yang tidak

Verifikasi	Validasi
pengembangan	dapat ditangkap oleh proses verifikasi
Targetnya adalah arsitektur aplikasi dan perangkat lunak, spesifikasi, desain lengkap, level tinggi, dan desain database, dll.	Target adalah produk yang sebenarnya
Tim QA melakukan verifikasi dan memastikan bahwa perangkat lunak sesuai dengan persyaratan dalam dokumen SRS.	Dengan keterlibatan tim validasi, pengujian dijalankan pada kode perangkat lunak.
Dilakukan sebelum aktivitas validasi	Dilakukan setelah aktivitas verifikasi

Pada dasarnya, *testing* mengikuti metode *software* yang digunakan oleh sebuah organisasi (Ferdiana, 2012).



Dari gambar diatas, dapat dilihat bahwa terdapat 4 tahap *software testing* dalam sebuah skema pengembangan software, yaitu *Unit*, *Integration*, *Acceptance*, dan *Exploratory*. Setiap tahap dapat menentukan tipe software testing yang tepat. Contohnya pada tahap Unit, *testing* akan dilakukan pada fungsi *software*, sedangkan tipe *software testing* yang dipilih akan lebih efektif jika *automated testing*, karena jumlah *testing* berjumlah banyak.

a. Tipe *Software Testing* Berdasarkan Fungsionalitas

1) *Functional Testing*

Pengujian aplikasi berdasarkan persyaratan dari klien atau spesifikasi *software,* meliputi:

- *Unit Testing*
- *Integration Testing*
- *User Acceptance*
- *Localization Testing*
- dan lain-lain

2) *Non-Functional Testing*

Pengujian aplikasi berdasarkan kualitas performa *software,* meliputi:

- *Load Testing*
- *Usability Testing*
- *Scalability Testing*
- *Load Time*
- dan lain-lain

Contoh *test case* dengan *Functional Testing* dan *Non-Functional Testing* dapat dilihat pada tabel dibawah ini:

Functional Testing	***Non-Functional Testing***
Ketika input valid, register dan login aplikasi berfungsi	Setelah login, sistem dashboard memuat (*loading*) dalam 3 detik
Ketika notifikasi email menyala → pengguna menerima pesan baru → sebuah notifikasi email terkirim	Notifikasi email terkirim dalam 3 menit
Ketika dokumen PDF kurang dari 5 MB, maka sistem menerima file	Ketika ada lebih dari 1 dokumen terunggah dalam waktu yang sama, maka otomatis akan membentuk antrian (*queue*)

Berikut ini merupakan Tabel perbandingan antara penggunaan *Functional Testing* dan *Non-Functional Testing*:

Functional Testing	***Non-Functional Testing***
Dilakukan berdasarkan kebutuhan bisnis/klien	Dilakukan berdasarkan ekspektasi konsumen dan persyaratan performa
Menguji apakah hasil kerja sistem sesuai dengan hasil yang diharapkan/disyaratkan	Memeriksa perilaku *software*: waktu respons, dan kecepatan perangkat lunak dalam kondisi tertentu.
Jenis pengujian meliputi: *Unit Testing*, *Integration Testing*, *System Testing*, dan *Acceptance Testing*	Jenis pengujian meliputi: *Performance testing, Load Testing, Stress testing, Volume testing, Security testing, Installation testing*, dan *Recovery testing*
Dilakukan secara manual, contohnya dengan metode uji Black Box.	Lebih layak untuk diuji menggunakan *automated tools*, contohnya Loadrunner.

b. Tipe *Software Testing* Berdasarkan Sudut Pandang

Berikut ini merupakan tabel perbandingan *Black Box Testing* dan *White Box Testing*:

Black Box Testing	***White Box Testing***
Tujuan utama dari pengujian ini adalah untuk menguji fungsionalitas / perilaku *software.*	Tujuan utamanya adalah untuk menguji infrastruktur *software.*
Dapat dilakukan oleh tester tanpa ilmu coding AUT (*Application Under Test*).	Penguji harus memiliki pengetahuan tentang struktur internal dan cara kerjanya.
Pengujian dapat dilakukan dengan GUI	Pengujian dapat dilakukan pada tahap

(*Graphical User Interface*).	awal sebelum GUI bersiap-siap.
Dilakukan oleh *Software Tester*	Dilakukan oleh *Software Developer* terkait
Kasus uji akan memiliki rincian lebih detail tentang kondisi input, langkah uji, hasil yang diharapkan dan data uji.	Kasus uji akan sederhana dengan perincian konsep teknis seperti pernyataan, cakupan kode, dan lain-lain.
Fokus utama pada bagaimana *software* bekerja	Fokus utama pada bagaimana *software* dikembangkan

c. Tipe *Software Testing* Berdasarkan Eksekusi

1) Manual Testing

Membuat sebuah *test case*, eksekusi langkah demi langkah, catat hasil-hasilnya. Dikenal juga sebagai *Acceptance Testing*. Karakteristik:

- Cepat untuk dieksekusi
- Proses di awal pengembangan
- Review Model
- Subjektif pada hasil pengujian

Merupakan *Static Testing* yaitu pengujian terhadap *code* atau dokumentasi dari *software*, tanpa melakukan eksekusi *code*-nya.

2) Automated Testing

Menggunakan sebuah software untuk membantu menguji kode-kode *software*. Karakteristik:

- Lambat dalam penyusunan
- Cepat dalam pengujian
- Mudah untuk diulang
- Kesalahan lebih rendah daripada *manual testing*

Merupakan *Automated Testing* yaitu pengujian perilaku kode software, contohnya *unit testing*.

d. Tipe *Software* Testing Lainnya

1) *Regression*

   Ada perubahan, aplikasi harus mengevaluasi melalui pengujian regresi.

2) *Stress / Load Testing*

   Menekan Batasan fungsional sistem, beberapa pengguna.

3) *Performance Testing*

   Mengukur responsivitas, kecepatan, alokasi sumber daya.

4) *Security Testing*

   Memvalidasi keamanan sebuah aplikasi.

5) *Usability Testing*

   Memvalidasi pengalaman pengguna.

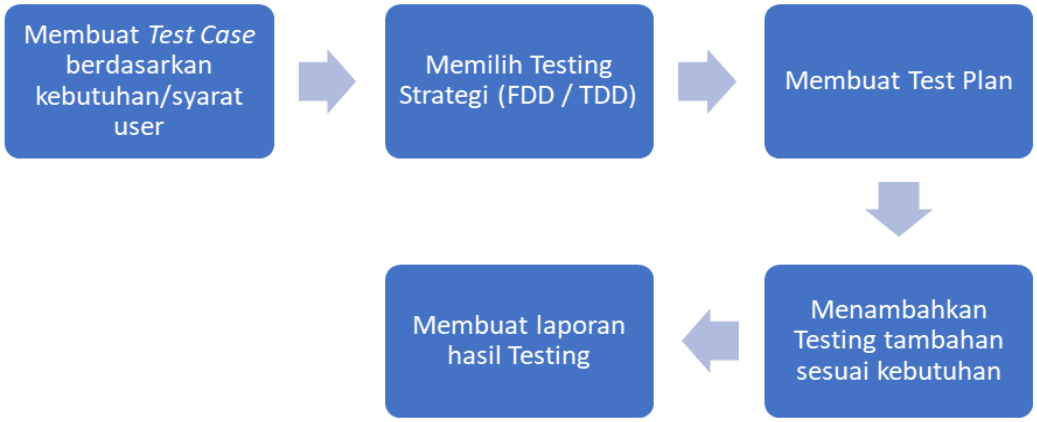
6) Localization Testing

   Fokus pada UI dan konten (*daylight*, *region*, *time*).

7) Accessibility

   Aplikasi ramah disabilitas.

Tahapan melakukan *software testing* dapat dilihat pada skema dibawah ini:



Test Plan adalah suatu dokumen yang berisi kumpulan *test case.* Sedangkan *test report* adalah suatu dokumen yang berisi hasil dari *testing* sesuai dengan *test case*. Berikut ini merupakan macam test case yang perlu dipertimbangkan untuk dilakukan:

- *Test Case Positif:* tes sistem dengan input benar
- *Test Case Negatif:* tes sistem dengan input salah
- *Test Case Terbatas:* tes sistem dengan input dalam batas

- *Test Case Kosong, Null,* atau Nol: tes sistem dengan input kosong

2. Pendekatan *Test Driven Development*

Test Driven Development (TDD) adalah skema pengembangan perangkat lunak / aplikasi dengan melakukan unit test terlebih dahulu sebelum melakukan pemrograman lebih lanjut atau *production*. TDD merupakan pendekatan yang penting untuk diterapkan karena:

- Pendeteksian masalah / *issue* lebih cepat
- *Feedback* yang lebih cepat, sedikit demi sedikit namun terus-menerus
- *Refactor code* untuk menurunkan risiko dan masalah kedepannya
- Pengujian sebagai sumber informasi dengan dokumentasi berbagai keputusan

Berikut ini tahapan dalam siklus *Test Driven Development*:

1. RED

Merupakan tanda test gagal, karena memang pada tes TDD pada tahap ini akan di *setup* gagal untuk menandakan bahwa *tester* baru membuat sebuah skema *testing* dan belum membuat implementasi dari *testing*-nya.

2. GREEN

Merupakan tanda *tester* sudah membuat kode yang fungsional dan sudah berhasil melewati fase *testing*.

3. REFACTOR

Merupakan proses mengubah bagian dalam kode tanpa mengubah *behavior*-nya. *Refactor* bertujuan agar kode yang di tulis lebih efisien.

Pengembang bisa melakukan *Test Driven Development* dengan bahasa pemrograman apapun, seperti Golang, Python, dan lain sebagainya. Berikut ini merupakan contoh tools untuk *Test Driven Development*:

- JUnit untuk *Unit Testing*
- Jmeter untuk *Load / Performance Testing*
- Mockito untuk *Rest API Testing*

Untuk mempelajari lebih lanjut terkait pendekatan *Test Driven Development*, pengembang dapat mengakses beberapa sumber di https://www.guru99.com/test-driven-development.html dan http://www.agiledata.org/essays/tdd.html.

3. Implementasi Software Testing dalam Lingkungan DevOps

DevOps *(Development & Operations)* adalah sebuah metodologi yang bertujuan mengintegrasi dan mengotomatisasi seluruh fungsi pada pengembangan perangkat lunak dari pengembangan hingga operasional di dalam satu alur. DevOps memiliki beberapa komponen, salah satu komponennya adalah *testing*. Testing yang dilakukan di dalam lingkungan DevOps dijalankan dengan prinsip otomasi, sehingga waktu pengujian akan lebih efisien. Cakupan pengujian yang dilakukan di lingkungan DevOps dapat beranekaragam, di antaranya:

- *Integration Testing*, jenis pengujian di mana modul perangkat lunak diintegrasikan secara logis dan diuji sebagai suatu kelompok. Proyek perangkat lunak tipikal terdiri dari beberapa modul perangkat lunak, dikodekan oleh pemrogram yang berbeda. Tujuan dari level pengujian ini adalah untuk mengekspos cacat dalam interaksi antara modul perangkat lunak ini ketika diintegrasikan.
- *Functional Testing*, jenis pengujian perangkat lunak yang memvalidasi sistem perangkat lunak terhadap persyaratan / spesifikasi fungsional. Tujuan dari *functional testing* adalah untuk menguji setiap fungsi aplikasi perangkat lunak, dengan memberikan masukan yang sesuai, memverifikasi keluaran terhadap persyaratan fungsional.
- *API Testing,* jenis pengujian perangkat lunak yang memvalidasi antarmuka pemrograman Aaplikasi. Tujuan *API Testing* adalah untuk memeriksa fungsionalitas, keandalan, kinerja, dan keamanan antarmuka pemrograman. Dalam pengujian API, alih-alih menggunakan masukan (keyboard) dan keluaran pengguna standar, pengembang menggunakan perangkat lunak untuk mengirim panggilan ke API, mendapatkan keluaran, dan mencatat respons sistem. *API Testing* sangat berbeda dari tes GUI dan tidak akan berkonsentrasi pada tampilan dan nuansa aplikasi. Ini terutama berkonsentrasi pada lapisan logika bisnis dari arsitektur perangkat lunak.
- *Exploratory Testing*, jenis pengujian perangkat lunak di mana kasus uji tidak dibuat sebelumnya tetapi penguji memeriksa sistem dengan cepat. *Tester* mungkin mencatat ide tentang apa yang akan diuji sebelum pelaksanaan tes.

Fokus *exploratory testing* lebih pada pengujian sebagai aktivitas "berpikir". *Exploratory testing* banyak digunakan dalam model Agile dan semuanya tentang penemuan, investigasi, dan pembelajaran. Ini menekankan kebebasan pribadi dan tanggung jawab penguji individu.

- *Regression Testing*, jenis pengujian perangkat lunak untuk memastikan bahwa program atau perubahan kode terbaru tidak mempengaruhi fitur yang ada. Pengujian Regresi tidak lain adalah pilihan penuh atau sebagian dari kasus uji yang sudah dieksekusi yang dieksekusi ulang untuk memastikan fungsionalitas yang ada berfungsi dengan baik.
- *Compatibility Testing,* jenis pengujian perangkat lunak untuk memeriksa apakah perangkat lunak yang dikembangkan dapat berjalan pada perangkat keras, sistem operasi, aplikasi, lingkungan jaringan, atau perangkat seluler yang berbeda.
- *Security Testing,* jenis pengujian perangkat lunak yang mengungkap kerentanan, ancaman, risiko dalam aplikasi perangkat lunak dan mencegah serangan jahat dari penyusup. Tujuan *security testing* adalah untuk mengidentifikasi semua kemungkinan celah dan kelemahan dari sistem perangkat lunak yang dapat mengakibatkan hilangnya informasi, pendapatan, reputasi di tangan karyawan atau pihak luar Organisasi.
- *Acceptance Testing,* jenis pengujian yang dilakukan oleh pengguna akhir atau klien untuk memverifikasi / menerima sistem perangkat lunak sebelum memindahkan aplikasi perangkat lunak ke lingkungan produksi. *Acceptance testing* dilakukan pada pengujian tahap akhir setelah pengujian fungsional, integrasi dan sistem dilakukan.
- *Deployment Testing,* jenis pengujian yang dilakukan setelah rilis produk dengan memanfaatkan sumber daya dan pengaturan yang tersedia di lokasi pelanggan, ini adalah upaya gabungan oleh organisasi pengembangan produk dan organisasi yang mencoba menggunakan produk.

Hal-hal yang terjadi pada *Software Testing* di lingkungan DevOps antara lain:

- Tim Penguji atau QA (*Quality Assurance*) akan secara aktif terlibat selama siklus pengembangan *software*.

- Pengujian software terjadi di setiap tahap DevOps, termasuk pengujian integrasi software dan *release*-nya. Karena itu perlu dilakukan otomatisasi *software testing*.
- Dengan testing yang dilakukan secara kontinyu, maka tim QA dapat melacak *defects* dan *bugs* lebih awal sekaligus melacak performa produk selama proses pengembangannya.

Strategi-strategi yang dapat diterapkan pada *software testing* dalam DevOps:

- Melakukan *automated test* sebanyak mungkin.
- Tim QA dan Pengembang perlu berdiskusi bersama mengenai area yang terpengaruh karena pengembangan tertentu.
- Dengan menggunakan berbagai teknik otomatisasi, QA perlu bisa menjalankan *automated testing* di berbagai lingkungan lintas platform.
- *Bugs* dan *Defects* yang ditemukan harus dilaporkan dan diperbaiki melewati proses yang sama sebelum kode tersebut digunakan di lingkungan produksi.

Untuk pembelajaran lebih lanjut terkait *software testing* pembaca dapat mengakses sumber daya belajar di https://www.guru99.com/software-testing.html.

DAFTAR PUSTAKA

Atlassian. (2020). *The different types of software testing*. Diambil dari https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Atlassian. (2020). *Managing Your DevOps Lifecycle with Atlassian*. Diambil dari https://www.atlassian.com/devops/start-your-journey

Khanam, Zeba., & Ahsan, Mohammed Najeeb. (2017). *Evaluating the Effectiveness of Test Driven Development: Advantages and Pitfalls*. India: Research India Publications.

Microsoft. (2020). *DevOps solutions on Azure*. Diambil dari https://azure.microsoft.com/en-us/solutions/devops/

Guru99. (2020). *Difference Between Verification and Validation with Example*. Dipetik 8 11, 2020, dari https://www.guru99.com/verification-v-s-validation-in-a-software-testing.html