BAB 18

STRATEGI *TESTING* PERANGKAT LUNAK

A. Pendahuluan

Strategi *testing* perangkat lunak adalah rencana menyeluruh yang menetapkan pendekatan dan teknik yang akan digunakan untuk menguji perangkat lunak guna memastikan bahwa perangkat lunak tersebut bebas dari cacat, memenuhi kebutuhan pengguna, dan sesuai dengan spesifikasi yang telah ditentukan. Strategi *testing* yang baik mencakup berbagai jenis pengujian, alat, dan teknik yang akan diterapkan pada berbagai tahap siklus hidup pengembangan perangkat lunak. Pengujian merupakan serangkaian kegiatan yang dapat direncanakan terlebih dahulu dan dilakukan secara sistematis. Oleh karena itu, template untuk pengujian perangkat lunak seperangkat langkah yang harus Anda lakukan dapat menempatkan teknik desain kasus uji dan metode pengujian tertentu seharusnya ditentukan untuk proses perangkat lunak. Sejumlah strategi pengujian perangkat lunak telah diusulkan dalam literatur. Semua memberi Anda templat untuk pengujian dan semuanya memiliki generik berikut karakteristik. Untuk melakukan pengujian yang efektif, Anda harus melakukan tinjauan teknis yang efektif. Dengan melakukan ini, banyak kesalahan akan dihilangkan sebelum pengujian dimulai.

1. Pengujian dimulai pada tingkat komponen dan berjalan "ke luar" menuju ke tingkat komponen integrasi seluruh sistem berbasis komputer.

2. Teknik pengujian yang berbeda sesuai untuk pendekatan rekayasa perangkat lunak yang berbeda dan pada waktu yang berbeda.
3. Pengujian dilakukan oleh pengembang perangkat lunak dan (untuk proyek besar) kelompok uji independen.
4. Pengujian dan debugging adalah aktivitas yang berbeda, namun debugging harus diakomodasi dalam strategi pengujian apa pun.

Strategi pengujian perangkat lunak harus mengakomodasi pengujian tingkat rendah yang diperlukan untuk memverifikasi bahwa sebagian kecil kode sumber telah diimplementasikan dengan benar. serta pengujian tingkat tinggi yang memvalidasi fungsi sistem utama terhadap pelanggan persyaratan. Sebuah strategi harus memberikan panduan bagi praktisi dan serangkaian tindakan tonggak sejarah bagi manajer. Karena langkah-langkah strategi pengujian terjadi pada satu waktu ketika tekanan tenggat waktu mulai meningkat, kemajuan harus dapat diukur dan ada masalah harus muncul sedini mungkin.

B. Komponen Utama Strategi *Testing* Perangkat Lunak

Tujuan Pengujianadalah menetapkan tujuan utama dari proses pengujian, seperti memastikan keandalan, keamanan, dan kinerja perangkat lunak. Lingkup Pengujian meliputi mendefinisikan ruang lingkup pengujian, termasuk fitur yang akan diuji, jenis pengujian yang akan dilakukan, dan batasan pengujian. Model Pengembangan digunakan untuk menentukan model pengembangan seperti (misalnya, Waterfall, *Agile*, DevOps) dan bagaimana pengujian akan diintegrasikan ke dalam siklus pengembangan tersebut.

Strategi *testing* perangkat lunak yang komprehensif mencakup berbagai komponen utama yang dirancang untuk memastikan kualitas perangkat lunak dari berbagai aspek. Berikut adalah penjelasan rinci tentang komponen utama strategi *testing* perangkat lunak:

Gambar 8. 1 *Testing* Perangkat Lunak

Pendekatan Pengujian

Model Pengembangan: Menentukan model pengembangan perangkat lunak yang digunakan (misalnya, *Agile*, *Waterfal*l, DevOps) dan bagaimana pengujian akan diintegrasikan ke dalam siklus pengembangan tersebut.

Teknik Pengujian: Memilih teknik pengujian yang sesuai seperti pengujian manual, pengujian otomatis, pengujian berbasis risiko, dan pengujian berbasis spesifikasi.

Jenis Pengujian

1. Pengujian Unit (*Unit Testing*): Menguji bagian terkecil dari perangkat lunak (unit atau komponen) secara individual untuk memastikan bahwa setiap unit berfungsi dengan benar.
2. Pengujian Integrasi (*Integration Testing*): Menguji kombinasi unit atau modul untuk memastikan bahwa mereka bekerja bersama-sama sesuai dengan yang diharapkan.
3. Pengujian Sistem (*System Testing*): Menguji sistem perangkat lunak secara keseluruhan untuk memastikan bahwa semua komponen berfungsi sesuai spesifikasi.
4. Pengujian Akseptansi (*Acceptance Testing*): Dilakukan oleh pengguna akhir atau pemangku kepentingan untuk memastikan bahwa perangkat lunak memenuhi kebutuhan dan persyaratan mereka.

5. Pengujian Kinerja (*Performance Testing*): Menguji kinerja perangkat lunak di bawah berbagai kondisi beban untuk memastikan bahwa perangkat lunak berfungsi dengan baik dalam situasi dunia nyata.
6. Pengujian Keamanan (*Security Testing*): Menguji kerentanan perangkat lunak terhadap ancaman keamanan untuk memastikan bahwa perangkat lunak aman dari serangan.
7. Pengujian Kegunaan (*Usability Testing*): Menguji kemudahan penggunaan perangkat lunak untuk memastikan bahwa pengguna dapat mengoperasikan perangkat lunak dengan efisien dan efektif.

Alat dan Lingkungan Pengujian

1. Alat Pengujian: Memilih dan menggunakan alat pengujian yang sesuai untuk mengotomatisasi dan mempercepat proses pengujian. Contoh alat pengujian termasuk Selenium, JUnit, TestNG, JIRA, Bugzilla, dan JMeter.
2. Lingkungan Pengujian: Menyiapkan lingkungan pengujian yang mereplikasi kondisi produksi termasuk perangkat keras, perangkat lunak, jaringan, dan konfigurasi sistem untuk memastikan bahwa hasil pengujian representatif.

Sumber Daya dan Jadwal

1. Sumber Daya: Menentukan tim pengujian dan sumber daya yang diperlukan termasuk penguji, alat, perangkat keras, dan perangkat lunak.
2. Jadwal Pengujian: Membuat jadwal pengujian yang rinci mencakup waktu untuk persiapan, pelaksanaan, pengujian ulang, dan analisis hasil pengujian. Menetapkan tenggat waktu dan milestone yang jelas.

Metodologi dan Teknik Pengujian

1. Pengujian Manual vs. Otomatis: Menentukan kapan dan bagaimana menggunakan pengujian manual dan otomatis berdasarkan kebutuhan dan konteks proyek.
2. Pengujian Berbasis Risiko: Mengidentifikasi area kritis yang memerlukan pengujian lebih intensif berdasarkan analisis risiko kegagalan perangkat lunak dan dampaknya.

Pelaporan dan Pelacakan

1. Pelaporan Cacat: Menggunakan alat manajemen cacat untuk mencatat, melacak, dan mengelola cacat yang ditemukan selama pengujian.
2. Laporan Pengujian: Menyusun laporan pengujian yang memberikan gambaran menyeluruh tentang hasil pengujian, termasuk temuan, analisis, dan rekomendasi. Laporan ini harus disajikan secara teratur kepada tim pengembangan dan manajemen.

Evaluasi dan Peningkatan Berkelanjutan

1. Evaluasi Hasil Pengujian: Menganalisis hasil pengujian untuk menentukan efektivitas strategi pengujian dan mengidentifikasi area untuk perbaikan.
2. Peningkatan Berkelanjutan: Melakukan peningkatan berkelanjutan pada strategi pengujian berdasarkan umpan balik dan temuan dari siklus pengujian sebelumnya. Mengadakan pertemuan retrospektif dan menyusun rencana aksi untuk peningkatan.

Strategi *testing* perangkat lunak yang efektif dan efisien mencakup berbagai komponen utama yang bekerja bersama untuk memastikan kualitas perangkat lunak. Dengan perencanaan yang matang, penggunaan alat yang tepat, dan pendekatan yang terstruktur, tim pengembangan dapat mengidentifikasi dan memperbaiki cacat sejak dini, memastikan bahwa perangkat lunak memenuhi standar kualitas yang tinggi, dan siap untuk digunakan oleh pengguna akhir. Strategi *testing* yang baik juga mencakup evaluasi dan peningkatan berkelanjutan untuk memastikan bahwa proses pengujian terus berkembang dan beradaptasi dengan kebutuhan proyek dan perubahan teknologi.

C. Strategi Uji untuk Perangkat Lunak Konvensional

Strategi uji untuk perangkat lunak konvensional melibatkan serangkaian pendekatan dan metode yang bertujuan untuk memastikan kualitas, keandalan, dan fungsionalitas perangkat lunak. Perangkat lunak konvensional umumnya

dikembangkan menggunakan model pengembangan yang lebih linier seperti Waterfall, tetapi juga bisa menggunakan model iteratif seperti Incremental atau Spiral.

Berikut adalah komponen utama dan langkah-langkah dalam strategi uji untuk perangkat lunak konvensional:

1. **Pendekatan Pengujian**
   a. Model Pengembangan: Menggunakan pendekatan Waterfall atau model linier lainnya yang membagi pengembangan perangkat lunak ke dalam fase-fase yang jelas (analisis, desain, pengkodean, pengujian, dan pemeliharaan).
   b. Teknik Pengujian: Mengadopsi teknik pengujian manual dan otomatis, pengujian berbasis spesifikasi (black-box *testing*), dan pengujian berbasis kode (*White-box testing*).

2. **Jenis Pengujian**
   a. Pengujian Unit (Unit *Testing*): Menguji setiap komponen atau modul perangkat lunak secara individual untuk memastikan bahwa masing-masing berfungsi dengan benar. Biasanya dilakukan oleh pengembang.
   b. Pengujian Integrasi (*Integration Testing*): Menguji kombinasi unit atau modul untuk memastikan bahwa mereka bekerja bersama-sama sesuai dengan yang diharapkan. Ini mencakup pengujian antarmuka dan aliran data antar modul.
   c. Pengujian Sistem (*System Testing*): Menguji seluruh sistem perangkat lunak secara keseluruhan untuk memastikan bahwa semua bagian berfungsi sesuai dengan spesifikasi.
   d. Pengujian Akseptansi (*Acceptance Testing*): Dilakukan oleh pengguna akhir atau pemangku kepentingan untuk memastikan bahwa perangkat lunak memenuhi kebutuhan dan persyaratan mereka. Biasanya mencakup UAT (*User Acceptance Testing*).
   e. Pengujian Regresi (*Regression Testing*): Menguji kembali perangkat lunak setelah perubahan atau pembaruan untuk memastikan bahwa tidak ada fungsi lain yang terpengaruh oleh perubahan tersebut.

3. **Alat dan Lingkungan Pengujian**
   a. Alat Pengujian: Menggunakan alat pengujian seperti JUnit untuk pengujian unit, Selenium untuk pengujian otomatis pada antarmuka pengguna, dan JIRA atau Bugzilla untuk manajemen cacat.
   b. Lingkungan Pengujian: Menyiapkan lingkungan pengujian yang mencerminkan kondisi produksi, termasuk perangkat keras, perangkat lunak, jaringan, dan konfigurasi sistem. Ini memastikan bahwa hasil pengujian representatif dari lingkungan produksi.

4. **Sumber Daya dan Jadwal**
   a. Sumber Daya: Mengidentifikasi tim pengujian yang terdiri dari penguji manual, pengembang, dan spesialis otomatisasi pengujian. Juga termasuk alat dan infrastruktur pengujian.
   b. Jadwal Pengujian: Menyusun jadwal yang rinci untuk seluruh aktivitas pengujian, termasuk waktu untuk persiapan, pelaksanaan, pengujian ulang, dan analisis hasil pengujian. Jadwal harus disinkronkan dengan tahapan pengembangan perangkat lunak.

5. **Metodologi dan Teknik Pengujian**
   a. Pengujian Manual vs. Otomatis: Menentukan kapan menggunakan pengujian manual untuk validasi fitur kompleks atau pengujian ad-hoc, dan pengujian otomatis untuk pengujian regresi dan pengujian fungsional yang berulang.
   b. Pengujian Berbasis Risiko: Mengidentifikasi area perangkat lunak yang memiliki risiko tinggi kegagalan atau dampak yang signifikan, dan memfokuskan upaya pengujian pada area tersebut untuk mengurangi risiko secara efektif.

6. **Pelaporan dan Pelacakan**
   a. Pelaporan Cacat: Menggunakan sistem pelacakan cacat (bug tracking system) untuk mencatat, melacak, dan mengelola cacat yang ditemukan selama pengujian.

b. Laporan Pengujian: Menyusun laporan pengujian yang merangkum hasil pengujian, temuan cacat, analisis risiko, dan rekomendasi. Laporan ini disajikan secara berkala kepada tim pengembangan dan manajemen proyek.

7. **Evaluasi dan Peningkatan Berkelanjutan**
   a. Evaluasi Hasil Pengujian: Menganalisis hasil pengujian untuk menilai efektivitas strategi pengujian, mengidentifikasi tren cacat, dan menentukan area untuk perbaikan.
   b. Peningkatan Berkelanjutan: Mengadakan pertemuan retrospektif untuk mengevaluasi proses pengujian dan membuat rencana tindakan untuk peningkatan berkelanjutan. Mengimplementasikan pembaruan dan perbaikan berdasarkan umpan balik dan analisis hasil pengujian.

Contoh Penerapan Strategi Pengujian pada Proyek Perangkat Lunak Konvensional

Kasus Studi: Sistem Pengelolaan Inventaris

Pendahuluan

1. Tujuan Pengujian: Memastikan bahwa sistem pengelolaan inventaris dapat mengelola stok barang, pesanan, dan laporan dengan akurat.
2. Lingkup Pengujian: Meliputi modul manajemen stok, pemesanan, pelaporan, dan antarmuka pengguna.

Pendekatan Pengujian

1. Model Pengembangan: Waterfall, dengan pengujian dilakukan setelah fase pengkodean selesai.
2. Teknik Pengujian: Menggunakan kombinasi pengujian black-box dan white-box.

Jenis Pengujian

1. Pengujian Unit: Menggunakan JUnit untuk menguji modul manajemen stok secara individual.
2. Pengujian Integrasi: Menggunakan Postman untuk menguji API antara modul pemesanan dan manajemen stok.

3. Pengujian Sistem: Menggunakan Selenium untuk mengotomatiskan pengujian end-to-end dari aplikasi web.
4. Pengujian Akseptansi: Melibatkan pengguna akhir dari departemen gudang untuk melakukan pengujian penerimaan.
5. Pengujian Regresi: Melakukan pengujian regresi otomatis setiap kali ada pembaruan kode.

Alat dan Lingkungan Pengujian

1. Alat Pengujian: JUnit, Selenium, Postman, JIRA.
2. Lingkungan Pengujian: Server pengujian yang mencerminkan lingkungan produksi dengan konfigurasi yang sama.

Sumber Daya dan Jadwal

1. Sumber Daya: Tim pengujian terdiri dari dua penguji manual, satu spesialis otomatisasi, dan satu manajer pengujian.
2. Jadwal Pengujian: Pengujian dilakukan selama dua bulan, dengan fase persiapan, pelaksanaan, dan pengujian ulang.

Metodologi dan Teknik Pengujian

1. Pengujian Manual vs. Otomatis: Pengujian otomatis untuk regresi dan pengujian manual untuk validasi fitur baru.
2. Pengujian Berbasis Risiko: Fokus pada modul pemesanan karena memiliki dampak langsung pada operasional bisnis.

Pelaporan dan Pelacakan

1. Pelaporan Cacat: Semua cacat dicatat dan dilacak menggunakan JIRA.
2. Laporan Pengujian: Laporan mingguan disajikan kepada tim pengembang dan manajemen untuk pemantauan kemajuan dan pengambilan keputusan.

Evaluasi dan Peningkatan Berkelanjutan

1. Evaluasi Hasil Pengujian: Melakukan analisis tren cacat setiap dua minggu.

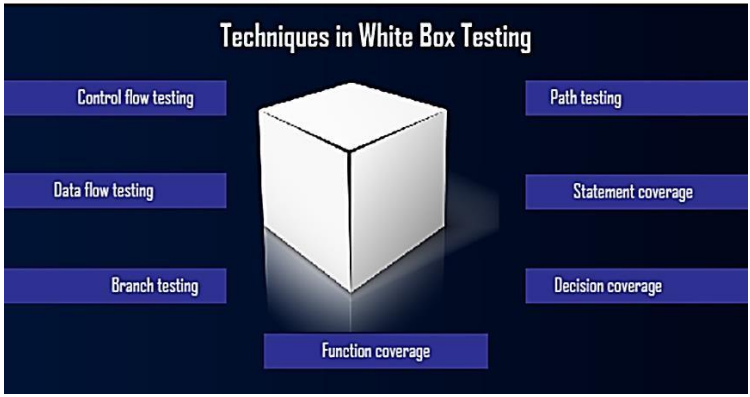
2. Peningkatan Berkelanjutan: Mengadakan retrospektif akhir proyek untuk mengidentifikasi area peningkatan dan memperbarui strategi pengujian untuk proyek mendatang.

Strategi uji yang dirancang dengan baik untuk perangkat lunak konvensional memastikan bahwa produk akhir memenuhi standar kualitas yang tinggi, mengurangi risiko kegagalan, dan memastikan bahwa perangkat lunak berfungsi sesuai dengan kebutuhan pengguna. Dengan mengikuti langkah-langkah yang terstruktur dan menggunakan teknik serta alat pengujian yang tepat, organisasi dapat mengembangkan perangkat lunak yang handal dan berkualitas tinggi.

D. *White-box Testing*

Pengujian *White Box*, terkadang disebut pengujian kotak kaca, adalah filosofi desain kasus uji yang menggunakan struktur kontrol yang dijelaskan sebagai bagian dari desain tingkat komponen untuk mendapatkan kasus uji. Dengan menggunakan metode pengujian kotak putih, Anda dapat memperoleh kasus uji tersebut menjamin bahwa semua jalur independen dalam suatu modul telah dilaksanakan setidaknya sekali, melaksanakan semua keputusan logis berdasarkan sisi benar dan salahnya, melaksanakan semua loop pada batas-batasnya dan dalam batas-batas operasionalnya, dan latihan struktur data internal untuk memastikan validitasnya.

White-box testing, juga dikenal sebagai clear-box *testing*, glass-box *testing*, atau open-box *testing*, adalah metode pengujian perangkat lunak di mana penguji memiliki pengetahuan tentang struktur internal atau kode dari aplikasi yang diuji. Berbeda dengan black-box *testing* yang hanya menguji fungsionalitas eksternal tanpa melihat ke dalam struktur internal, *White-box testing* melibatkan analisis alur kontrol dan alur data dalam aplikasi.



Gambar 8. 2 ***White-box Testing***

Komponen dan Teknik ***White-box Testing***

1. **Pengujian Aliran Kontrol (*Control Flow Testing*)**

   Pengujian aliran kontrol fokus pada logika yang menentukan jalur eksekusi dalam perangkat lunak. Teknik ini memastikan bahwa semua jalur yang mungkin dijalankan setidaknya sekali selama pengujian. Beberapa teknik utama termasuk:

   a. Pengujian Pernyataan (*Statement Coverage*): Memastikan bahwa setiap pernyataan dalam kode dieksekusi setidaknya sekali.
   b. Pengujian Cabang (*Branch Coverage*): Memastikan bahwa setiap cabang dari setiap keputusan (*if, for, while, switch*) diuji untuk true dan false.
   c. Pengujian Jalur (*Path Coverage*): Memastikan bahwa semua jalur eksekusi yang mungkin dari awal sampai akhir program diuji.

2. **Pengujian Aliran Data (*Data Flow Testing*)**

   Pengujian aliran data fokus pada variabel dan nilai mereka dalam program. Ini memastikan bahwa variabel digunakan dan dimanipulasi dengan benar. Teknik ini meliputi:

a. *Def-Use* Pair *Testing*: Menguji hubungan antara definisi dan penggunaan variabel untuk memastikan bahwa variabel diinisialisasi sebelum digunakan.
b. Pengujian Aliran Nilai (*Value Flow Testing*): Menguji aliran nilai variabel melalui program untuk menemukan anomali seperti nilai yang tidak pernah digunakan atau variabel yang tidak diinisialisasi.

3. Pengujian Loop (*Loop Testing*)

Pengujian loop memastikan bahwa semua loop dalam program diuji dengan berbagai kondisi untuk memastikan kinerja dan stabilitas. Teknik ini meliputi:

a. Pengujian Loop Sederhana (*Simple Loop Testing*): Menguji loop dengan 0, 1, dan beberapa iterasi untuk memeriksa perilaku loop.
b. Pengujian Loop Terkait (*Nested Loop Testing*): Menguji loop bersarang dengan berbagai skenario untuk memeriksa interaksi antara loop.
c. Pengujian Loop Tak Berhingga (*Unstructured Loop Testing*): Menguji loop yang tidak memiliki struktur terdefinisi dengan baik untuk memeriksa ketahanan program.

Langkah-langkah *White-box Testing*

1. Memahami Kode

a. Membaca dan Memahami Kode: Penguji harus memahami kode sumber, logika program, dan struktur data yang digunakan.
b. Identifikasi Jalur Uji: Menentukan jalur eksekusi yang mungkin dan memilih jalur yang perlu diuji.

2. Merancang Kasus Uji

a. Kasus Uji Pernyataan: Membuat kasus uji yang memastikan setiap pernyataan dalam kode dieksekusi.
b. Kasus Uji Cabang: Membuat kasus uji untuk setiap cabang logika dalam kode.
c. Kasus Uji Jalur: Merancang kasus uji untuk setiap jalur eksekusi dalam program.

3. **Pelaksanaan Pengujian**
   a. Menjalankan Kasus Uji: Menjalankan kasus uji yang dirancang pada kode.
   b. Memantau Output: Memeriksa output program untuk memastikan bahwa ia berfungsi sesuai dengan yang diharapkan.

4. **Analisis Hasil**
   a. Menganalisis Hasil: Memeriksa hasil pengujian untuk menemukan cacat atau kesalahan dalam kode.
   b. Perbaikan: Melakukan perbaikan berdasarkan hasil pengujian dan menjalankan kembali pengujian untuk memastikan bahwa perbaikan berhasil.

Alat dan Lingkungan White-box *Testing*

1. Alat Statis: Alat yang menganalisis kode sumber tanpa mengeksekusinya, seperti linters dan analisis statis (misalnya, SonarQube, Checkmarx).
2. Alat Dinamis: Alat yang menganalisis kode selama eksekusi, termasuk debugger dan profiler (misalnya, JUnit untuk pengujian unit di Java, NUnit untuk .NET).
3. IDE: Lingkungan pengembangan terintegrasi (misalnya, IntelliJ IDEA, Visual Studio) yang menyediakan alat bantu untuk *White-box testing* seperti debugging dan profiling.

Keuntungan dan Keterbatasan White-box *Testing*

Keuntungan :

1. Deteksi Dini Cacat: Memungkinkan deteksi dan perbaikan kesalahan logika atau kesalahan lainnya pada tahap awal pengembangan.
2. Pengujian Menyeluruh: Menyediakan pengujian menyeluruh pada semua jalur logika dan kondisi.
3. Optimalisasi Kode: Membantu dalam identifikasi dan pengoptimalan bagian-bagian kode yang tidak efisien.

Keterbatasan :

1. Keterampilan Tinggi Diperlukan: Membutuhkan pemahaman mendalam tentang kode dan keterampilan pemrograman yang tinggi.

2. Waktu dan Biaya: Pengujian yang mendalam dapat memakan waktu dan biaya yang signifikan.
3. Tidak Fokus pada Perspektif Pengguna: Lebih berfokus pada bagaimana sistem bekerja secara internal daripada pada bagaimana sistem digunakan oleh pengguna akhir.

Contoh Penerapan White-box *Testing*

1. Kasus Studi: Sistem Manajemen Inventaris
2. Memahami Kode
3. Modul yang Diuji: Fungsi untuk menambah stok barang ke dalam inventaris.
4. Logika Program: Mengecek apakah barang sudah ada, menambahkan jumlah jika sudah ada, atau menambah entri baru jika belum ada.

Merancang Kasus Uji

1. Kasus Uji Pernyataan: Pastikan setiap pernyataan dalam fungsi dieksekusi.
2. Kasus Uji Cabang: Uji kondisi apakah barang sudah ada atau belum.
3. Kasus Uji Jalur: Uji semua jalur dari fungsi menambah barang, termasuk jalur yang menghasilkan kesalahan.

Pelaksanaan Pengujian

1. Menjalankan Kasus Uji: Jalankan semua kasus uji pada fungsi.
2. Memantau Output: Memeriksa apakah barang ditambahkan dengan benar dan kesalahan ditangani dengan baik.

Analisis Hasil

1. Menganalisis Hasil: Memeriksa apakah semua jalur dieksekusi dan hasilnya sesuai dengan yang diharapkan.
2. Perbaikan: Melakukan perbaikan jika ditemukan cacat dan mengulang pengujian.

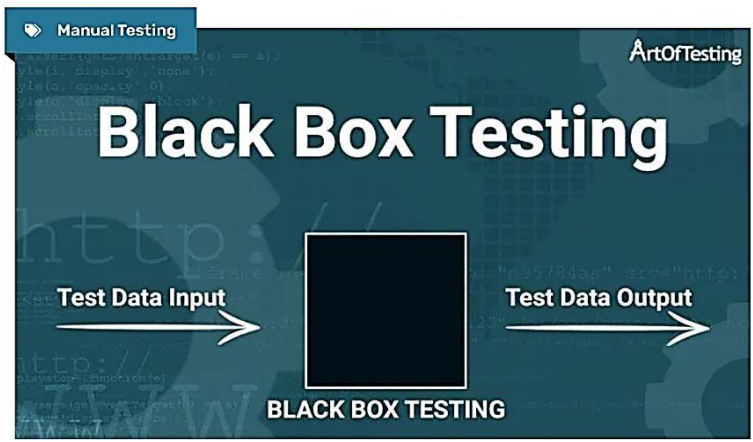
White-box testing adalah metode pengujian perangkat lunak yang kritis dan efektif untuk memastikan bahwa kode berfungsi sesuai dengan yang diharapkan. Dengan mengidentifikasi dan menguji semua jalur eksekusi, aliran

kontrol, dan aliran data, *White-box testing* membantu menemukan cacat yang mungkin tidak terdeteksi oleh pengujian black-box. Meskipun memerlukan keterampilan dan pengetahuan mendalam tentang kode, manfaatnya dalam meningkatkan kualitas perangkat lunak sangat signifikan.

E. *Black-Box Testing*

Black box testing atau dapat disebut juga Behavioral *Testing* adalah pengujian yang dilakukan untuk mengamati hasil input dan output dari perangkat lunak tanpa mengetahui struktur kode dari perangkat lunak. Pengujian ini dilakukan di akhir pembuatan perangkat lunak untuk mengetahui apakah perangkat lunak dapat berfungsi dengan baik. Untuk melakukan pengujian, penguji tidak harus memiliki kemampuan menulis kode program. Pengujian ini dapat dilakukan oleh siapa saja.

Black-box testing, juga dikenal sebagai pengujian berbasis spesifikasi atau pengujian perilaku, adalah metode pengujian perangkat lunak di mana penguji tidak mengetahui struktur internal atau kode dari aplikasi yang diuji. Fokus utama dari black-box *testing* adalah pada fungsionalitas perangkat lunak sesuai dengan persyaratan dan spesifikasi yang telah ditentukan. Pengujian ini melihat perangkat lunak dari sudut pandang pengguna akhir.



Gambar 8. 3 *White-box Testing*

Komponen dan Teknik *Black-Box Testing*

1. **Pengujian Fungsional (*Functional Testing*)**

   Pengujian fungsional mengevaluasi perangkat lunak berdasarkan fungsionalitas yang ditentukan dalam spesifikasi. Teknik yang umum digunakan meliputi:

   a. Pengujian Kasus Uji (*Test Case*): Membuat dan menjalankan kasus uji yang mewakili skenario penggunaan nyata.
   b. Pengujian Input-Output (*Input-Output Testing*): Memeriksa apakah output yang dihasilkan sesuai dengan input yang diberikan dan spesifikasi yang ditentukan.
   c. Pengujian Ekivalensi (*Equivalence Partitioning*): Membagi input ke dalam kelas-kelas ekivalen dan menguji satu kasus dari setiap kelas untuk mengurangi jumlah pengujian yang diperlukan.
   d. Pengujian Nilai Batas (*Boundary Value Analysis*): Menguji batas-batas nilai input untuk menemukan kesalahan pada ekstrem nilai.

2. **Pengujian Non-Fungsional (*Non-Functional Testing*)**

   Pengujian non-fungsional mengevaluasi aspek perangkat lunak yang tidak terkait langsung dengan fungsionalitas, seperti kinerja, keamanan, dan kegunaan. Teknik yang digunakan meliputi:

   a. Pengujian Kinerja (*Performance Testing*): Mengukur bagaimana sistem berperforma di bawah beban tertentu. Subkategori termasuk pengujian beban (*load testing*), pengujian stres (*stress testing*), dan pengujian daya tahan (*endurance testing*).
   b. Pengujian Keamanan (*Security Testing*): Mengidentifikasi kerentanan keamanan dalam perangkat lunak.
   c. Pengujian Kegunaan (*Usability Testing*): Mengevaluasi kemudahan penggunaan dan interaksi pengguna dengan perangkat lunak.

d. Pengujian Kompatibilitas (*Compatibility Testing*): Memastikan perangkat lunak bekerja pada berbagai lingkungan, sistem operasi, dan perangkat keras yang berbeda.

Langkah-langkah *Black-Box Testing*

1. **Memahami Persyaratan**
   a. Analisis Spesifikasi: Memahami spesifikasi dan persyaratan perangkat lunak untuk menentukan fitur dan fungsionalitas yang akan diuji.
   b. Identifikasi Kasus Uji: Menentukan skenario uji yang relevan berdasarkan spesifikasi.
2. **Merancang Kasus Uji**
   a. Pengembangan Kasus Uji: Membuat kasus uji yang mencakup semua fitur dan fungsionalitas yang ditentukan dalam spesifikasi.
   b. Penentuan Data Uji: Menentukan input data yang akan digunakan untuk menguji setiap kasus uji.
3. **Pelaksanaan Pengujian**
   a. Menjalankan Kasus Uji: Melaksanakan kasus uji pada perangkat lunak dan mencatat hasilnya.
   b. Memantau Output: Membandingkan output yang dihasilkan dengan output yang diharapkan sesuai dengan spesifikasi.
4. **Analisis Hasil**
   a. Evaluasi Hasil Pengujian: Menganalisis hasil pengujian untuk menemukan cacat atau ketidaksesuaian dengan spesifikasi.
   b. Pelaporan Cacat: Mendokumentasikan cacat yang ditemukan dan melaporkannya kepada tim pengembang.

Alat dan Lingkungan *Black-Box Testing*

1. Alat Pengujian Otomatis: Alat yang membantu mengotomatisasi pengujian fungsional dan non-fungsional, seperti Selenium untuk pengujian web, JMeter untuk pengujian kinerja, dan QTP (Quick Test Professional) untuk pengujian fungsional.

2. Sistem Manajemen Pengujian: Alat untuk mengelola dan melacak kasus uji, hasil pengujian, dan cacat, seperti TestRail, Zephyr, dan ALM (Application Lifecycle Management).
3. Lingkungan Pengujian: Mengatur lingkungan pengujian yang mencerminkan lingkungan produksi untuk memastikan hasil pengujian representatif.

Keuntungan dan Keterbatasan *Black-Box Testing*
Keuntungan

1. Tidak Memerlukan Pengetahuan Kode: Penguji tidak perlu memahami kode atau struktur internal perangkat lunak, sehingga memungkinkan pengujian dilakukan oleh individu dengan keahlian non-teknis.
2. Pendekatan Pengguna Akhir: Menguji perangkat lunak dari perspektif pengguna akhir, memastikan bahwa perangkat lunak berfungsi sesuai dengan kebutuhan pengguna.
3. Deteksi Cacat Fungsional: Efektif dalam menemukan cacat yang terkait dengan fungsionalitas perangkat lunak.

Keterbatasan

1. Cakupan Kode Terbatas: Tidak menjamin bahwa semua jalur atau pernyataan dalam kode diuji, sehingga beberapa cacat internal mungkin tidak terdeteksi.
2. Pengujian Ekstensif: Memerlukan sejumlah besar kasus uji untuk mencakup semua kemungkinan input dan kondisi.
3. Tidak Menguji Struktur Internal: Fokus hanya pada output dan perilaku eksternal, tanpa memeriksa struktur internal atau logika program.

Contoh Penerapan *Black-Box Testing*
Kasus Studi: Sistem Pemesanan Online

Memahami Persyaratan

1. Fitur yang Diuji: Proses pendaftaran, pemesanan produk, pembayaran, dan konfirmasi pesanan.
2. Spesifikasi: Dokumentasi spesifikasi yang menjelaskan alur proses pemesanan, validasi input, dan respons sistem.

Merancang Kasus Uji

1. Kasus Uji Pendaftaran: Menguji berbagai skenario pendaftaran seperti pendaftaran dengan informasi yang valid, pendaftaran dengan email yang sudah ada, dan pendaftaran dengan informasi yang tidak lengkap.
2. Kasus Uji Pemesanan Produk: Menguji skenario pemesanan produk dengan jumlah stok yang mencukupi dan tidak mencukupi.
3. Kasus Uji Pembayaran: Menguji berbagai metode pembayaran dan validasi informasi pembayaran.
4. Kasus Uji Konfirmasi Pesanan: Menguji alur konfirmasi pesanan termasuk pengiriman email konfirmasi dan pemberitahuan kepada pengguna.

Pelaksanaan Pengujian

1. Menjalankan Kasus Uji: Melakukan pengujian sesuai dengan skenario yang telah dirancang, mencatat hasilnya, dan membandingkan dengan hasil yang diharapkan.
2. Memantau Output: Memastikan bahwa sistem berperilaku sesuai dengan spesifikasi yang telah ditentukan.

Analisis Hasil

1. Evaluasi Hasil Pengujian: Meninjau hasil pengujian untuk menemukan cacat atau ketidaksesuaian dengan spesifikasi.
2. Pelaporan Cacat: Mencatat dan melaporkan cacat yang ditemukan kepada tim pengembang untuk diperbaiki.

Black-box *testing* adalah pendekatan pengujian yang penting untuk memastikan bahwa perangkat lunak berfungsi sesuai dengan spesifikasi dan memenuhi kebutuhan pengguna akhir. Dengan berfokus pada input dan output tanpa memperhatikan struktur internal, black-box *testing* memberikan perspektif pengguna yang nyata. Meskipun memiliki keterbatasan dalam hal cakupan kode, black-box *testing* tetap merupakan komponen esensial dalam strategi pengujian perangkat lunak yang komprehensif.