SOFTWARE TESTING AND QUALITY ASSURANCE A

DOSEN PENGAMPU :

Abdul Kadir Jailani, S.Kom., M.T.

DISUSUN OLEH :

Saldan Rama (H071221071)

PROGRAM STUDI SISTEM INFORMASI

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS HASANUDDIN

KOTA MAKASSAR

2024

1. Tulis penjelasan, Langkah/proses dan contoh kasus dari:

a. White-box testing

b. Black-box testing

c. Unit testing

d. Integration testing

e. System testing

f. Regression testing

A. White-box Testing

White-box testing adalah metode pengujian di mana penguji memiliki akses penuh terhadap kode sumber dan struktur internal aplikasi. Tujuan utamanya adalah untuk mengevaluasi jalur logika dan struktur data di dalam program. Dengan metode ini, penguji dapat mengidentifikasi dan memperbaiki bug yang mungkin tersembunyi dalam kode yang tidak diuji oleh metode pengujian lainnya. Pengujian ini juga sering kali mencakup analisis kompleksitas kode untuk memastikan bahwa semua cabang dan jalur logika telah diuji.

Langkah Proses:

1. **Memahami Struktur dan Logika Kode:** Penguji mulai dengan mempelajari struktur kode secara menyeluruh untuk memahami logika di balik program, mencakup alur data, cabang kondisi, dan jalur logika.
2. **Membuat Skenario Pengujian untuk Setiap Jalur Kode:** Setelah memahami kode, penguji menyusun skenario pengujian yang mencakup semua jalur logika yang mungkin ditempuh dalam program, termasuk jalur alternatif dan pengecualian.
3. **Melakukan Pengujian pada Setiap Jalur Logika:** Pengujian dilakukan dengan menjalankan setiap skenario untuk memastikan bahwa seluruh jalur logika dalam kode diuji secara menyeluruh, dan alat bantu pengujian dapat digunakan untuk otomatisasi.
4. **Menganalisis Hasil Pengujian dan Memperbaiki Bug yang Ditemukan:** Setelah pengujian selesai, hasilnya dianalisis untuk mengidentifikasi bug yang muncul, dan pengembang memperbaiki bug tersebut sebelum melakukan pengujian ulang untuk memverifikasi perbaikannya.

Studi Kasus: Pada sistem pemesanan tiket bioskop online, white-box testing diterapkan pada modul pemrosesan pembayaran. Penguji memeriksa apakah algoritma memproses semua jenis metode pembayaran (kartu kredit, e-wallet) dengan benar. Misalnya, penguji mengevaluasi jalur kode untuk menangani pembayaran yang gagal, seperti ketika saldo e-wallet tidak mencukupi, untuk memastikan aplikasi memberikan umpan balik yang tepat kepada pengguna.

B. Black-box Testing

Black-box testing adalah metode pengujian yang berfokus pada fungsionalitas perangkat lunak tanpa mempertimbangkan struktur internal atau kode sumber aplikasi. Dalam pengujian ini, penguji hanya berinteraksi dengan aplikasi melalui antarmuka pengguna, menggunakan spesifikasi fungsional untuk mendefinisikan test case. Black-box testing sangat berguna untuk mengidentifikasi masalah pada antarmuka pengguna dan perilaku aplikasi.

Langkah Proses:

1. **Mempelajari Spesifikasi Fungsional Aplikasi:** Penguji memulai dengan memahami spesifikasi fungsional dari aplikasi berdasarkan kebutuhan dan ekspektasi pengguna, tanpa memperhatikan kode atau logika di baliknya.
2. **Membuat Skenario Pengujian Berdasarkan Spesifikasi Pengguna:** Berdasarkan spesifikasi tersebut, penguji menyusun skenario pengujian yang menguji apakah sistem berfungsi sesuai dengan kebutuhan pengguna, dengan fokus pada input dan output.
3. **Memberikan Input dan Memeriksa Output Tanpa Mengetahui Kode di Balik Layar:** Penguji memasukkan data ke dalam sistem dan memeriksa output yang dihasilkan tanpa memperhatikan bagaimana kode diproses di balik layar, sehingga hanya fungsionalitas yang diuji.
4. **Menganalisis Hasil Pengujian untuk Memastikan Fungsionalitas Sesuai dengan Spesifikasi:** Setelah pengujian selesai, penguji mengevaluasi apakah output yang dihasilkan sesuai dengan spesifikasi fungsional dan melakukan perbaikan jika diperlukan.

Studi Kasus: Penguji mencoba memesan tiket bioskop dengan berbagai kombinasi pilihan kursi dan jadwal film untuk memverifikasi apakah aplikasi memproses pemesanan dengan benar. Penguji memberikan input yang tidak valid (seperti memilih lebih dari satu kursi untuk satu pengguna) dan memeriksa apakah aplikasi memberikan pesan kesalahan yang sesuai.

C. Unit Testing

Unit testing adalah jenis pengujian perangkat lunak yang berfokus pada pengujian bagian terkecil dari perangkat lunak, biasanya fungsi atau metode, secara individual. Tujuan dari unit testing adalah untuk memastikan bahwa setiap unit berfungsi dengan baik sebelum unit-unit tersebut digabungkan ke dalam sistem yang lebih besar. Pengujian ini sering dilakukan oleh pengembang saat mereka menulis kode.

Langkah Proses:

1. **Mengidentifikasi Unit yang Akan Diuji:** Penguji memilih unit terkecil dari perangkat lunak, seperti fungsi atau metode, yang akan diuji secara mandiri dan terisolasi dari bagian sistem lainnya.
2. **Membuat Skenario Pengujian untuk Setiap Unit:** Penguji menyusun skenario pengujian yang menguji perilaku unit dalam kondisi normal maupun ekstrem, termasuk skenario positif dan negatif.
3. **Melakukan Pengujian pada Setiap Unit Secara Terisolasi:** Setiap unit diuji secara terpisah untuk memastikan bahwa ia berfungsi dengan benar secara mandiri, tanpa dipengaruhi oleh modul atau unit lainnya.
4. **Memperbaiki Bug yang Ditemukan Selama Pengujian:** Jika ditemukan bug, penguji segera memperbaiki masalah tersebut dan melakukan pengujian ulang untuk memastikan bahwa unit telah berfungsi dengan benar setelah perbaikan.

Studi Kasus: Pada aplikasi pemesanan tiket, fungsi untuk memilih kursi diuji secara terpisah. Penguji memastikan bahwa pengguna tidak dapat memilih kursi yang sudah dipesan. Jika penguji menemukan bug yang memungkinkan pengguna memilih kursi yang telah dipesan, bug tersebut harus diperbaiki sebelum melanjutkan ke tahap pengujian berikutnya.

D. Integration Testing

Integration testing adalah proses pengujian yang dilakukan untuk memastikan bahwa modul atau unit yang sudah diuji secara terpisah dapat berfungsi dengan baik saat digabungkan. Pengujian ini bertujuan untuk mengidentifikasi masalah yang muncul saat unit-unit yang terpisah diintegrasikan, yang mungkin tidak terlihat saat pengujian unit dilakukan.

Langkah Proses:

1. **Mengidentifikasi Modul yang Akan Diintegrasikan:** Penguji menentukan modul-modul atau unit-unit yang telah diuji secara terpisah dan akan diintegrasikan untuk memastikan fungsionalitas mereka bekerja bersama.
2. **Membuat Skenario Pengujian Interaksi Antar Modul:** Penguji menyusun skenario pengujian yang menguji interaksi antara modul-modul yang terintegrasi, termasuk alur data antar modul.
3. **Melakukan Pengujian Integrasi:** Pengujian dilakukan dengan menjalankan skenario pengujian yang menguji komunikasi dan interaksi antara modul, memastikan bahwa data dapat mengalir dengan benar antar modul.
4. **Menganalisis Hasil dan Memperbaiki Masalah yang Ditemukan:** Setelah pengujian integrasi selesai, hasilnya dianalisis untuk menemukan masalah yang timbul akibat interaksi antar modul, dan perbaikan dilakukan jika diperlukan.

Studi Kasus: Penguji memeriksa apakah sistem pemesanan kursi di bioskop dapat bekerja dengan baik bersama dengan modul pembayaran, memastikan bahwa setelah pemesanan kursi selesai, sistem dapat langsung memproses pembayaran. Jika ada masalah dalam aliran data antara modul pemesanan dan pembayaran, maka perlu dilakukan perbaikan pada komunikasi antara kedua modul tersebut.

E. System Testing

System testing adalah jenis pengujian yang dilakukan pada keseluruhan sistem setelah semua unit dan modul diintegrasikan. Tujuannya adalah untuk memastikan bahwa semua fungsionalitas aplikasi bekerja secara baik dan memenuhi spesifikasi yang ditetapkan. Pengujian ini sering kali mencakup pengujian fungsional dan non-fungsional, termasuk performa, keamanan, dan kompatibilitas.

Langkah Proses:

1. **Menyusun Dokumen Spesifikasi Sistem:** Penguji memulai dengan menyusun dokumen spesifikasi yang berisi semua fungsionalitas dan kebutuhan dari keseluruhan sistem, sebagai panduan pengujian.
2. **Membuat Skenario Pengujian Berdasarkan Fungsionalitas Sistem:** Berdasarkan spesifikasi, penguji menyusun skenario pengujian yang mencakup semua fitur dan alur sistem untuk memastikan bahwa sistem berfungsi sesuai dengan spesifikasi.
3. **Melakukan Pengujian pada Keseluruhan Sistem:** Pengujian dilakukan pada keseluruhan sistem yang terintegrasi, menjalankan semua skenario yang telah disusun untuk memverifikasi bahwa semua fungsionalitas bekerja dengan baik.

4. **Menganalisis Hasil dan Memperbaiki Masalah yang Ditemukan:** Setelah pengujian selesai, hasilnya dianalisis untuk menemukan bug atau masalah, dan pengembang memperbaikinya sebelum pengujian ulang dilakukan.

Studi Kasus: Penguji menjalankan seluruh skenario pemesanan tiket bioskop dari awal hingga akhir, termasuk memilih film, kursi, jadwal, dan menyelesaikan pembayaran. Penguji memastikan bahwa semua langkah dalam proses pemesanan berfungsi dengan baik dan tidak ada kesalahan yang terjadi di antara langkah-langkah tersebut.

F. Regression Testing

Regression testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa setelah modifikasi pada kode atau penambahan fitur, fungsionalitas yang sudah ada sebelumnya tidak terpengaruh. Ini penting untuk menjaga kestabilan sistem setelah adanya perubahan dan memastikan bahwa tidak ada bug baru yang muncul akibat perubahan tersebut.

Langkah Proses:

1. **Mengidentifikasi Perubahan yang Dilakukan:** Penguji mengidentifikasi perubahan kode atau fitur baru yang telah dimodifikasi atau ditambahkan pada perangkat lunak untuk mengetahui area yang memerlukan pengujian ulang.
2. **Menentukan Area yang Mungkin Terpengaruh:** Penguji mengidentifikasi bagian lain dari sistem yang mungkin terpengaruh oleh perubahan tersebut, termasuk fitur-fitur lama yang mungkin terganggu oleh pembaruan.
3. **Melakukan Pengujian Ulang pada Area Terkait:** Penguji menjalankan ulang test case lama yang relevan serta test case baru jika diperlukan untuk memastikan bahwa semua fitur lama masih berfungsi dengan benar setelah perubahan.
4. **Menganalisis Hasil dan Memperbaiki Masalah yang Ditemukan:** Setelah pengujian selesai, penguji meninjau hasil untuk memastikan bahwa semua fungsionalitas lama tidak terganggu, dan jika ada masalah, perbaikan segera dilakukan.

Studi Kasus: Setelah memperbarui sistem pembayaran untuk mendukung e-wallet baru, regression testing dilakukan untuk memastikan bahwa metode pembayaran lama seperti kartu kredit masih berfungsi dengan benar. Penguji melakukan pengujian pada semua metode pembayaran yang ada untuk memastikan tidak ada yang terganggu setelah pembaruan.

2. Lakukan Evaluasi Hasil Testing dan Identifikasi Penyebab Masalah dari suatu kasus!

Langkah-Langkah Evaluasi Hasil Testing:

1. **Menganalisis Data Hasil Pengujian:** Setelah pengujian perangkat lunak dilakukan, langkah pertama adalah mengumpulkan semua hasil dari berbagai pengujian yang telah dijalankan, seperti White-box testing, Black-box testing, Unit testing, dan lainnya. Hasil ini berupa laporan dari setiap skenario pengujian, termasuk output yang dihasilkan, log error, dan perilaku yang tidak sesuai dengan ekspektasi.
2. **Mencatat Bug atau Kesalahan yang Ditemukan:** Setelah semua hasil pengujian dianalisis, catat setiap bug atau kesalahan yang ditemukan. Bug ini bisa berupa kesalahan logika, kegagalan fungsional, atau performa yang tidak sesuai. Setiap bug harus didokumentasikan dengan deskripsi lengkap, termasuk langkah-langkah untuk mereproduksi masalah, kapan masalah terjadi, serta kondisi lingkungan yang menyebabkan bug muncul.
3. **Memprioritaskan Bug Berdasarkan Dampak:** Bug yang ditemukan kemudian diprioritaskan berdasarkan dampaknya terhadap sistem. Misalnya, bug yang menyebabkan kerusakan sistem secara keseluruhan atau masalah keamanan akan mendapat prioritas tinggi, sementara bug kecil yang hanya mempengaruhi tampilan visual dapat diprioritaskan lebih rendah.
4. **Membandingkan Hasil dengan Ekspektasi:** Selanjutnya, hasil pengujian dibandingkan dengan ekspektasi yang ditetapkan dalam spesifikasi awal. Ini membantu menentukan apakah aplikasi berfungsi sesuai kebutuhan pengguna dan apakah semua fitur telah bekerja sebagaimana mestinya.
5. **Meninjau Pengaruh Perubahan Kode:** Jika pengujian dilakukan setelah ada perubahan atau penambahan fitur (misalnya pada regression testing), hasil pengujian harus dibandingkan dengan kondisi sebelum perubahan dilakukan. Ini membantu memastikan bahwa perubahan tidak merusak fungsionalitas yang ada sebelumnya.
6. **Mengidentifikasi Area yang Memerlukan Pengujian Ulang:** Setelah bug diperbaiki, pengujian ulang harus dilakukan pada area yang terkena dampak. Pengujian ulang memastikan bahwa perbaikan yang dilakukan telah menyelesaikan masalah dan tidak menimbulkan bug baru (yang disebut regression bugs).
7. **Menyusun Laporan Akhir:** Setelah semua pengujian dan perbaikan selesai, hasil akhir dari pengujian dan perbaikan bug didokumentasikan dalam sebuah laporan. Laporan ini mencakup ringkasan bug yang ditemukan, langkah perbaikan, dan hasil dari pengujian ulang.

Untuk mengidentifikasi akar penyebab masalah dalam pengujian perangkat lunak, metode **"5 Why's"** dan **Diagram Fishbone (Ishikawa)** sering digunakan.

Teknik "5 Why's"

"5 Why's" adalah metode sederhana namun efektif untuk menggali akar penyebab masalah dengan cara bertanya "Mengapa?" sebanyak lima kali atau lebih hingga penyebab dasar dari masalah terungkap.

Masalah: transaksi pembayaran sering gagal.

Ajukan Pertanyaan "Mengapa?" untuk Setiap Penyebab:

1. **Mengapa transaksi pembayaran gagal?** Karena sistem menolak transaksi beberapa metode pembayaran.
2. **Mengapa sistem menolak beberapa metode pembayaran?** Karena API dari layanan pembayaran pihak ketiga mengalami timeout.
3. **Mengapa API layanan pembayaran mengalami timeout?** Karena ada terlalu banyak permintaan yang diterima pada waktu bersamaan.
4. **Mengapa ada terlalu banyak permintaan pada waktu bersamaan?** Karena aplikasi tidak menerapkan throttling atau pembatasan jumlah permintaan ke API.
5. **Mengapa throttling tidak diterapkan?** Karena pengembang tidak mempertimbangkan skenario lalu lintas yang tinggi saat merancang sistem.

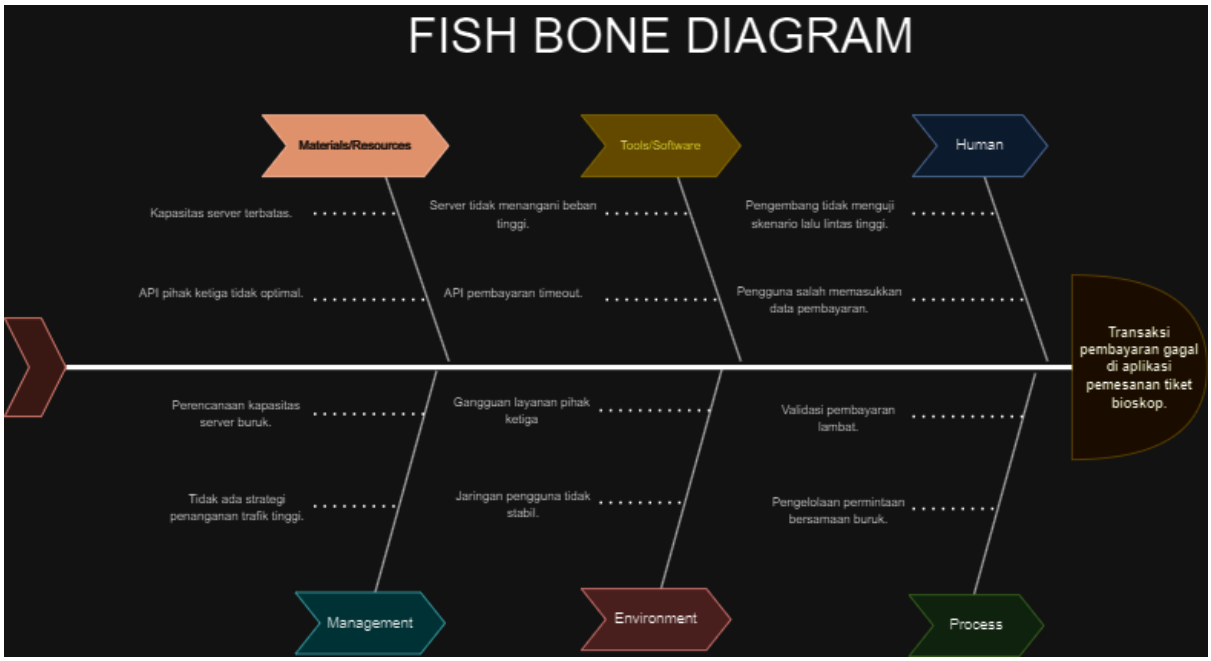
Akar Masalah: Dengan menanyakan "Mengapa?" sebanyak lima kali, kita dapat menemukan akar penyebab masalah, yaitu kurangnya implementasi throttling di aplikasi.

Diagram Fishbone (Ishikawa)

Diagram Fishbone, juga dikenal sebagai diagram sebab-akibat atau Ishikawa, adalah alat visual yang digunakan untuk mengidentifikasi penyebab potensial dari suatu masalah. Diagram ini membantu memetakan masalah utama dan kategori penyebabnya, biasanya meliputi: **Manusia, Proses, Alat, Lingkungan, Bahan, dan Manajemen**.

Masalah: "Transaksi pembayaran gagal pada aplikasi pemesanan tiket bioskop."

Diagram Fishbone:



Penejelasan Diagram Fishbone:

1. Manusia (Human)

Kategori ini merujuk pada kesalahan yang mungkin disebabkan oleh manusia, baik dari sisi pengembang maupun pengguna aplikasi.

- **Pengembang tidak menguji skenario lalu lintas tinggi**: Pengembang mungkin tidak mengantisipasi atau tidak melakukan pengujian pada kondisi di mana banyak pengguna menggunakan sistem secara bersamaan, yang menyebabkan aplikasi gagal menangani volume tinggi.
- **Pengguna salah memasukkan data pembayaran**: Pengguna aplikasi mungkin salah dalam memasukkan informasi kartu kredit, seperti nomor kartu atau tanggal kedaluwarsa, yang menyebabkan transaksi gagal diproses.

2. Proses (Process)

Kategori ini melibatkan alur atau proses dalam sistem yang mungkin menyebabkan kegagalan.

- **Validasi pembayaran lambat**: Sistem mungkin memproses validasi pembayaran dengan lambat, baik karena koneksi yang buruk dengan penyedia layanan pembayaran atau algoritma yang tidak efisien, menyebabkan timeout atau kegagalan.

- **Pengelolaan permintaan bersamaan buruk**: Ketika banyak pengguna mencoba memesan tiket secara bersamaan, sistem mungkin tidak dapat menangani permintaan ini dengan baik, yang mengakibatkan terjadinya kegagalan transaksi.

3. Alat/Perangkat Lunak (Tools/Software)

Kategori ini mengacu pada kegagalan perangkat lunak atau teknologi yang digunakan dalam aplikasi.

- **API pembayaran timeout**: Kegagalan dalam integrasi dengan API layanan pembayaran pihak ketiga (seperti gateway pembayaran) dapat terjadi karena API tersebut sering kali mengalami timeout, terutama saat volume transaksi tinggi.
- **Server tidak menangani beban tinggi**: Server aplikasi mungkin tidak memiliki kapasitas yang cukup untuk menangani permintaan dari banyak pengguna secara bersamaan, yang menyebabkan server overload dan kegagalan transaksi.

4. Lingkungan (Environment)

Kategori ini mengacu pada faktor eksternal yang dapat mempengaruhi sistem, seperti jaringan dan koneksi.

- **Jaringan pengguna tidak stabil**: Masalah dengan jaringan internet pengguna, seperti sinyal yang lemah atau koneksi yang terputus, dapat menyebabkan transaksi gagal karena data pembayaran tidak diteruskan ke server dengan baik.
- **Gangguan layanan pihak ketiga**: Jika layanan pihak ketiga seperti penyedia pembayaran mengalami gangguan, maka proses transaksi di aplikasi pemesanan tiket akan terganggu dan gagal.

5. Bahan/Resource (Materials/Resources)

Kategori ini mencakup sumber daya teknis dan infrastruktur yang mendukung aplikasi.

- **Kapasitas server terbatas**: Infrastruktur server yang terbatas mungkin tidak mampu menangani banyak permintaan secara bersamaan, yang dapat menyebabkan lambatnya respons atau bahkan kegagalan pemrosesan.
- **API pihak ketiga tidak optimal**: Layanan API dari pihak ketiga mungkin tidak dioptimalkan untuk skala besar atau volume transaksi tinggi, yang dapat menyebabkan transaksi tertunda atau gagal.

6. Manajemen (Management)

Kategori ini melibatkan keputusan dan perencanaan manajemen yang dapat berdampak pada performa sistem.

- **Perencanaan kapasitas server buruk**: Tim manajemen mungkin tidak melakukan perencanaan kapasitas server yang memadai, sehingga server tidak siap untuk menangani peningkatan lalu lintas pengguna, yang berujung pada kegagalan transaksi.

- **Tidak ada strategi penanganan trafik tinggi**: Manajemen mungkin tidak merencanakan strategi khusus untuk menangani lonjakan permintaan pengguna, seperti dalam waktu-waktu penjualan tiket yang ramai, yang menyebabkan server tidak dapat menangani lalu lintas dengan baik.