White Box Testing

Pengertian

* White Box Testing adalah pengujian terhadap struktur internal, rancangan serta kode program dari suatu Perangkat Lunak.
* Kode program harus dapat dilihat oleh tester.
* Biasanya dilakukan oleh pengembang atau programmer perangkat lunak.
* Disebut juga Clear Box Testing, Open Box Testing, Structural Testing, Transparent Box Testing, Glass Box Testing, Code-based Testing.

Tujuan

Tujuan utama dari white box testing adalah untuk memastikan bahwa setiap bagian dari kode program atau struktur perangkat lunak berfungsi sesuai dengan spesifikasi dan dirancang dengan benar. Testing jenis ini juga dapat membantu mengidentifikasi potensi masalah sebelum perangkat lunak dirilis, meminimalkan risiko kesalahan, dan mendukung pengembangan perangkat lunak yang lebih handal dan efisien.

* Verifikasi Logika dan Struktur Kode

Memastikan bahwa logika dan struktur kode program sesuai dengan desain dan spesifikasi yang telah ditentukan.

* Menguji Setiap Jalur Eksekusi

Melibatkan pengujian setiap jalur eksekusi dalam kode untuk memastikan bahwa semua kemungkinan percabangan dan kondisi telah diuji.

* Identifikasi Kesalahan Logika

Mengidentifikasi kesalahan atau kelemahan dalam logika atau alur pemrograman yang mungkin tidak terlihat dalam pengujian black box.

* Optimasi Kode dan Kinerja

Memberikan wawasan tentang kinerja dan efisiensi kode program, memungkinkan pengembang untuk mengoptimalkan kode agar berjalan lebih efisien.

* Assurance terhadap Keamanan

Memverifikasi bahwa implementasi keamanan telah diintegrasikan dengan benar dan tidak ada celah keamanan yang signifikan dalam kode.

* Penilaian Kualitas Kode

Menilai kualitas kode program dengan menilai sejauh mana kode tersebut mematuhi standar pemrograman, pedoman, dan praktik terbaik.

Hal yang Diuji

Dalam White Box Testing, pengujian difokuskan pada struktur internal dan logika program. Beberapa hal yang diuji meliputi:

* Aliran Program (Path Testing)

Menguji setiap jalur eksekusi dalam program untuk memastikan bahwa semua path telah diuji.

* Kondisi dan Keputusan (Condition and Decision Testing)

Memastikan bahwa semua kondisi logika dan keputusan dalam program telah diuji untuk semua kemungkinan nilai.

* Unit Testing

Menguji setiap unit atau komponen secara terpisah untuk memastikan bahwa masing-masing berfungsi sebagaimana mestinya.

* Integration Testing

Menguji interaksi antara unit atau komponen yang berbeda untuk memastikan bahwa integrasinya berjalan dengan baik.

* Control Flow Testing

Memastikan bahwa kontrol aliran program berjalan sesuai dengan yang diharapkan.

* Boundary Testing

Menguji nilai-nilai batas atau kondisi ekstrem untuk memastikan bahwa program menangani mereka dengan benar.

* Statement Coverage

Memastikan bahwa setiap pernyataan (statement) dalam kode telah dijalankan setidaknya sekali selama pengujian.

Tahapan Pengujian

* Pemahaman kode program

Penguji harus memahami kode program yang digunakan serta keamanan dalam program. Penguji juga harus dapat menemukan isu keamanan dan mencegah dari serangan.

* Pembuatan dan eksekusi test case

Penguji membuat beberapa test case untuk setiap maupun serangkaian proses dalam aplikasi. Selanjutnya penguji melakukan pengujian secara manual atau otomatis sesuai test case yang dibuat.

Teknik Pengujian

* Basis Path Testing

Basis Path Testing adalah teknik pengujian yang fokus pada jalur eksekusi yang melibatkan setiap pernyataan minimal satu kali selama pengujian.

Cara Kerja:

1. Identifikasi jalur-jalur yang mungkin dalam program.
2. Pilih setiap jalur dan pastikan bahwa setiap pernyataan dijalankan minimal satu kali.
3. Tujuannya adalah untuk mencakup setiap jalur secara eksplisit.

* Data Flow Testing

Data Flow Testing menguji aliran data melalui program selama eksekusi dan mengidentifikasi tempat-tempat di mana data dapat dimodifikasi atau digunakan secara tidak benar.

Cara Kerja:

1. Analisis aliran data melalui program.
2. Mengidentifikasi variabel-variabel yang diubah atau digunakan dalam setiap bagian program.
3. Membuat skenario uji yang fokus pada aliran data dan mengidentifikasi masalah seperti penggunaan variabel tanpa inisialisasi atau kondisi balikan yang tidak diinginkan.

* Loop Testing

Loop Testing berfokus pada pengujian loop atau perulangan dalam program untuk memastikan bahwa loop berfungsi dengan benar dan tidak menyebabkan masalah seperti iterasi yang tak terbatas atau keluar dari loop terlalu dini.

Cara Kerja:

1. Mengidentifikasi loop dalam program.
2. Menguji berbagai skenario yang melibatkan loop, termasuk kondisi awal, kondisi akhir, dan perubahan nilai variabel pengontrol loop.
3. Memastikan bahwa loop berhenti ketika seharusnya dan menjalankan jumlah iterasi yang diharapkan.

Keuntungan

* Dapat mencakup semua jalur eksekusi dan struktur internal program, mengidentifikasi kesalahan logika dan kegagalan algoritma.
* Efektif untuk menguji setiap unit atau komponen secara terpisah.
* Dapat mengidentifikasi cacat dan masalah implementasi internal.
* Dapat memberikan hasil yang sangat baik dengan keterampilan pengujian dan pemrograman yang tinggi.

Kekurangan

* Memerlukan pemahaman mendalam tentang kode program.
* Kurang efisien untuk pengujian fungsionalitas sistem secara keseluruhan.
* Fokus pada implementasi internal mungkin kurang relevan untuk perspektif pengguna.
* Pengujian tidak efektif jika kode program tidak tersedia atau terlalu kompleks.

Black Box Testing

Pengertian

* Black Box Testing adalah Pengujian terhadap fungsionalitas suatu aplikasi tanpa melihat struktur internal, detail implementasi atau jalur dari suatu Perangkat Lunak.
* Berbasis spesifikasi dan kebutuhan perangkat lunak.
* Fokus pada input output suatu aplikasi.

Tujuan

Tujuan utama dari black box testing adalah untuk menilai fungsi dan perilaku perangkat lunak tanpa memerhatikan implementasi internalnya. Dalam black box testing, fokus utama adalah pada input dan output perangkat lunak, serta bagaimana perangkat lunak merespons input yang diberikan.

* Verifikasi Fungsionalitas

Memastikan bahwa perangkat lunak berperilaku sesuai dengan spesifikasi fungsional yang telah ditetapkan.

* Menguji Integrasi Sistem

Menguji bagaimana komponen atau modul perangkat lunak berinteraksi satu sama lain untuk memastikan integrasinya berjalan dengan baik.

* Menguji Kinerja dan Responsivitas

Menilai sejauh mana perangkat lunak dapat menanggapi input dengan cepat dan merespons dengan hasil yang sesuai.

* Identifikasi Kegagalan dan Bug

Mengidentifikasi bug atau kesalahan dalam fungsionalitas yang dapat mempengaruhi pengalaman pengguna atau kinerja perangkat lunak.

* Evaluasi User Experience

Mengevaluasi pengalaman pengguna dengan menguji antarmuka pengguna dan interaksi.

* Verifikasi Sesuai dengan Kebutuhan Pengguna

Memastikan bahwa perangkat lunak memenuhi kebutuhan pengguna dan kebutuhan bisnis yang telah ditentukan.

Hal yang Diuji

Dalam Black Box Testing, fokusnya adalah pada fungsionalitas dan respons sistem tanpa memerhatikan implementasi internal. Hal-hal yang diuji meliputi:

* Input Validation

Memastikan bahwa sistem dapat menangani berbagai jenis input dengan benar dan memberikan respons yang tepat.

* Output Validation

Memeriksa apakah output sistem sesuai dengan harapan berdasarkan input tertentu.

* Usability Testing

Menguji sejauh mana sistem mudah digunakan oleh pengguna akhir.

* Functional Testing

Memastikan bahwa setiap fungsi atau fitur bekerja sesuai dengan spesifikasi.

* Integration Testing

Menguji interaksi antara berbagai bagian atau modul sistem.

* System Testing

Menguji sistem secara keseluruhan untuk memastikan bahwa semua komponen bekerja bersama dengan baik.

* Performance Testing

Memastikan bahwa sistem dapat menangani beban kerja yang diharapkan dan memberikan kinerja yang memadai.

* Security Testing

Menguji kerentanan keamanan sistem dan memastikan bahwa data dijaga dengan baik.

* Compatibility Testing

Menguji sejauh mana sistem kompatibel dengan lingkungan dan platform yang berbeda.

Tahapan Pengujian

* Pemeriksaan Spesifikasi dan Kebutuhan Sistem

Tahap ini melibatkan pemeriksaan dokumen spesifikasi dan kebutuhan sistem untuk memahami secara menyeluruh tentang apa yang diharapkan dari perangkat lunak.

* Pemilihan Input (Positif dan Negatif)

Tester memilih input yang valid (skenario uji positif) dan input yang tidak valid (skenario uji negatif) berdasarkan spesifikasi dan kebutuhan sistem.

* Penentuan Output yang Diharapkan

Tester menentukan output yang diharapkan untuk setiap input yang telah dipilih. Ini melibatkan pemahaman terhadap bagaimana sistem seharusnya merespons berdasarkan input tertentu.

* Pembuatan Kasus Uji

Berdasarkan input yang dipilih dan output yang diharapkan, tester membuat kasus uji yang mencakup langkah-langkah yang perlu diambil dan kondisi yang harus dipenuhi selama pengujian.

* Eksekusi Kasus Uji

Kasus uji dieksekusi menggunakan input yang telah ditentukan. Selama eksekusi, data pengujian dikumpulkan, dan perilaku sistem diamati.

* Pembandingan Output dengan Output yang Diharapkan

Setelah eksekusi, output yang dihasilkan oleh sistem dibandingkan dengan output yang diharapkan. Ini membantu dalam mengidentifikasi apakah sistem berperilaku sesuai dengan harapan atau tidak.

* Perbaikan dan Uji Ulang

Jika ada kesalahan atau ketidaksesuaian antara output yang dihasilkan dan yang diharapkan, tester mencatat dan melaporkan bug atau masalah tersebut. Pengembang kemudian melakukan perbaikan, dan pengujian ulang dilakukan untuk memastikan bahwa perbaikan tersebut efektif.

Teknik Pengujian

* Equivalence Partitioning

Membagi input data ke dalam kelompok atau partisi yang setara, sehingga satu tes dari setiap partisi dianggap cukup.

Contoh:

Jika input memerlukan angka antara 1 dan 100, maka partisi setara bisa berupa nilai kurang dari 1, nilai di antara 1 dan 100, dan nilai lebih dari 100.

* Boundary Value Analysis/Limit Testing

Menguji nilai di batas rentang valid dan di luar batas tersebut.

Contoh:

Jika sebuah aplikasi menerima input angka antara 1 dan 100, maka uji batas akan mencakup nilai 1, 100, dan nilai di luar rentang tersebut.

* Comparison Testing

Membandingkan hasil dari versi perangkat lunak yang berbeda atau sistem yang berbeda untuk memastikan konsistensi.

Contoh:

Membandingkan output dari versi perangkat lunak yang lama dan baru untuk memeriksa apakah ada perbedaan yang tidak diinginkan.

* Sample Testing

Mengambil sejumlah sampel dari fungsionalitas atau fitur tertentu untuk diuji, tanpa mencakup keseluruhan sistem.

Contoh:

Menguji beberapa skenario penggunaan utama atau fitur kunci pada aplikasi.

* Robustness Testing

Menguji kemampuan sistem untuk menangani situasi yang tidak terduga atau input yang tidak valid.

Contoh:

Memberikan input yang tidak valid atau memasukkan data yang besar untuk melihat bagaimana sistem menanggapi situasi tersebut.

* Behavior Testing

Menguji perilaku atau respons sistem terhadap situasi tertentu.

Contoh:

Memeriksa bagaimana sistem merespons terhadap tindakan pengguna atau kondisi tertentu, seperti input yang salah atau transaksi ganda.

* Requirement Testing

Menguji apakah perangkat lunak memenuhi semua persyaratan fungsional dan non-fungsional yang telah ditentukan.

Contoh:

Menguji setiap fitur dan fungsi untuk memastikan sesuai dengan spesifikasi yang ada.

* Performance Testing

Menguji kinerja aplikasi di bawah kondisi tertentu, seperti beban tinggi atau waktu respons yang cepat.

Contoh:

Mengukur waktu respons sistem saat menghadapi beban pengguna yang tinggi.

* Endurance Testing

Menguji seberapa baik sistem dapat menangani beban atau aktivitas tinggi dalam jangka waktu yang lama.

Contoh:

Menjalankan aplikasi dalam waktu yang lama untuk mengevaluasi apakah ada penurunan kinerja seiring waktu.

* Cause-Effect Relationship Testing

Menguji hubungan sebab-akibat antara input dan output.

Contoh:

Menentukan bagaimana perubahan pada input dapat mempengaruhi output dan menguji skenario untuk mengevaluasi hubungan tersebut.

Keuntungan

* Tidak memerlukan pengetahuan detail tentang kode program, lebih fokus pada fungsionalitas dan antarmuka eksternal.
* Memastikan bahwa sistem berperilaku sesuai dengan harapan pengguna karena melibatkan pengujian dari perspektif pengguna.
* Efektif untuk menguji fungsionalitas sistem secara keseluruhan.
* Tidak memerlukan akses atau pemahaman terhadap kode program.

Kekurangan

* Tidak efektif dalam mengidentifikasi kesalahan logika atau masalah implementasi internal.
* Tidak memberikan kejelasan mendalam tentang kualitas implementasi internal.
* Jika persyaratan kurang jelas, dapat mengakibatkan pengujian yang tidak memadai.