INTRODUÇÃO A TESTES DE SOFTWARE

Teste de software

Os testes de software são uma parte essencial do processo de desenvolvimento de software. Eles ajudam a garantir que o software atenda às especificações e requisitos do usuário, funcionem conforme o esperado e sejam livres de erros e bugs.

Tipos de testes de software

Existem vários tipos de testes de software que podem ser realizados durante o processo de desenvolvimento de software. Cada tipo de teste se concentra em uma área específica do software e pode ser usado para verificar diferentes aspectos do seu funcionamento. Abaixo, estão alguns dos tipos de testes de software mais comuns:

1. **Testes de unidade:** Este tipo de teste verifica o funcionamento de cada componente individual do software. Os testes de unidade são geralmente escritos pelos desenvolvedores e ajudam a garantir que cada parte do software esteja funcionando corretamente.

2. **Testes de integração:** Este tipo de teste verifica como os componentes individuais do software interagem entre si. Os testes de integração são usados para verificar se as diferentes partes do software funcionam juntas de forma integrada e sem conflitos.

3. **Testes de sistema:** Este tipo de teste verifica o software como um todo, incluindo sua funcionalidade, desempenho e segurança. Os testes de sistema são usados para garantir que o software funcione conforme o esperado em diferentes cenários e ambientes.

4. **Testes de regressão:** Este tipo de teste é usado para verificar se as alterações feitas no software não afetaram sua funcionalidade existente. Os testes de

regressão ajudam a garantir que as correções de bugs e atualizações do software não causem novos problemas.

5. **Testes de aceitação:** Este tipo de teste é realizado para verificar se o software atende aos requisitos do usuário e está pronto para ser lançado. Os testes de aceitação são usados para garantir que o software seja testado completamente antes de ser entregue ao usuário final.

6. **Testes de desempenho:** Este tipo de teste verifica como o software se comporta em diferentes cargas de trabalho e volumes de dados. Os testes de desempenho são usados para garantir que o software funcione de forma eficiente e sem problemas de desempenho.

7. **Testes de segurança:** Este tipo de teste verifica se o software está protegido contra possíveis ameaças e ataques. Os testes de segurança são usados para garantir que o software seja seguro e protegido contra possíveis vulnerabilidades e ameaças.

Estes são apenas alguns dos tipos de testes de software mais comuns. A escolha do tipo de teste a ser utilizado dependerá do objetivo do teste e da área do software que precisa ser verificada.

Convenções e Estrutura

Para garantir que os testes de software sejam bem estruturados e fáceis de entender e manter, é importante seguir algumas convenções e estruturas. Abaixo estão algumas das principais convenções e estruturas que podem ser utilizadas nos testes de software:

1. **Nomeação dos testes:** Os nomes dos testes devem ser claros e descritivos, indicando o objetivo do teste e a funcionalidade sendo testada.

2. **Estrutura dos testes:** Os testes devem ser organizados em uma estrutura lógica e facilmente navegável, agrupando os testes relacionados em conjuntos ou suites de testes.

3. **Comentários:** É importante incluir comentários nos testes para ajudar a entender o que está sendo testado e por quê. Comentários também podem ser usados para explicar as condições de teste, entradas e saídas esperadas, e outras informações relevantes.

4. **Asserts e verificações:** É comum utilizar asserts e verificações nos testes para verificar se o resultado esperado é igual ao resultado real obtido durante a execução do teste. Essas verificações ajudam a identificar falhas ou erros no software.

5. **Cobertura de código:** É importante monitorar a cobertura de código dos testes para garantir que todos os casos de teste relevantes tenham sido executados. A cobertura de código pode ser medida por meio de ferramentas específicas que indicam o percentual de código testado.

6. **Reutilização de testes:** É recomendado reutilizar os testes sempre que possível para evitar redundância e melhorar a eficiência do processo de teste. Testes que verificam funcionalidades semelhantes ou relacionadas podem ser agrupados em conjuntos de testes.

Ao seguir essas convenções e estruturas, os testes de software se tornam mais organizados, fáceis de entender e manter, o que ajuda a aumentar a eficiência do processo de teste e a qualidade do software produzido.

Introdução a Cobertura de Teste

A cobertura de teste é uma medida usada para avaliar a qualidade do teste de um software. Ela se refere à porcentagem do código-fonte de um software que foi

testado por meio de um conjunto de testes. Quanto maior a cobertura de teste, maior a probabilidade de o software ter menos erros e falhas quando usado em produção.

A cobertura de teste pode ser medida por meio de ferramentas específicas, que verificam quais partes do código foram executadas durante a execução dos testes. Essas ferramentas permitem que os desenvolvedores saibam quais áreas do código foram testadas e quais ainda precisam ser testadas para garantir que o software funcione corretamente.

Métricas de cobertura

Existem diferentes métricas de cobertura de teste, cada uma com seus próprios objetivos e benefícios. Algumas das principais métricas de cobertura de teste incluem:

1. **Cobertura de instruções:** mede o número de instruções executadas durante os testes em relação ao número total de instruções no código-fonte.

2. **Cobertura de ramificação:** mede o número de caminhos de execução de código que foram testados em relação ao número total de caminhos de execução possíveis.

3. **Cobertura de condição:** mede o número de expressões booleanas testadas em relação ao número total de expressões booleanas no código-fonte.

4. **Cobertura de decisão:** mede o número de decisões testadas em relação ao número total de decisões no código-fonte.

5. **Cobertura de linha:** mede o número de linhas executadas durante os testes em relação ao número total de linhas no código-fonte.

6. **Cobertura de classe:** Esta métrica mede a porcentagem de classes testadas em relação ao número total de classes no software. O objetivo é garantir que todas as classes do software tenham sido testadas adequadamente.

7. **Cobertura de método:** Essa métrica mede a porcentagem de métodos testados em relação ao número total de métodos no software. O objetivo é garantir que todos os métodos do software tenham sido testados adequadamente.

Ao utilizar ferramentas de cobertura de teste, os desenvolvedores podem identificar áreas do código que não foram testadas adequadamente e, assim, criar testes adicionais para aumentar a cobertura de teste. Isso ajuda a garantir que o software tenha um alto nível de qualidade e reduz o risco de erros e falhas durante o uso em produção.

Ferramentas de cobertura

Existem diversas ferramentas de cobertura de testes disponíveis no mercado, cada uma com suas próprias características e funcionalidades. Abaixo estão algumas das principais ferramentas de cobertura de testes:

JaCoCo: JaCoCo é uma ferramenta de cobertura de código-fonte de código aberto para Java. Ele suporta vários níveis de cobertura, incluindo cobertura de instruções, ramificações, linhas e condições. JaCoCo pode ser integrado em ferramentas de build como o Maven e o Gradle.

Cobertura: Cobertura é uma ferramenta de cobertura de código-fonte de código aberto para Java. Ele suporta cobertura de instruções, ramificações e linhas. Cobertura pode ser integrado em ferramentas de build como o Ant e o Maven.

Istanbul: Istanbul é uma ferramenta de cobertura de código-fonte de código aberto para JavaScript. Ele suporta cobertura de instruções, ramificações e linhas. Istanbul pode ser integrado com frameworks de teste como o Mocha e o Jasmine.

NUnit: NUnit é uma ferramenta de teste de unidade de código aberto para .NET. Ele inclui funcionalidades de cobertura de teste e suporta a cobertura de ramificações,

condições e métodos. NUnit pode ser integrado em ferramentas de build como o MSBuild.

PHPUnit: PHPUnit é uma ferramenta de teste de unidade de código aberto para PHP. Ele inclui funcionalidades de cobertura de teste e suporta a cobertura de instruções, ramificações e linhas. PHPUnit pode ser integrado em ferramentas de build como o Composer.

Essas são apenas algumas das ferramentas de cobertura de teste disponíveis no mercado. Ao escolher uma ferramenta de cobertura de teste, é importante avaliar suas funcionalidades e suporte para as linguagens de programação e frameworks utilizados no projeto. Além disso, é importante considerar se a ferramenta pode ser facilmente integrada com as ferramentas de build e testes existentes no projeto.

Cobertura vs Mutação

A cobertura de testes é uma métrica quantitativa que mede a porcentagem de código-fonte que é exercido pelos testes. A cobertura pode ser medida em diferentes níveis, como cobertura de instruções, cobertura de ramificações, cobertura de condições, etc. A cobertura de testes é útil para avaliar a abrangência dos testes, identificar áreas do código que não estão sendo testadas adequadamente e definir metas de cobertura para orientar o desenvolvimento de testes adicionais.

A mutação, por outro lado, é uma técnica de avaliação qualitativa dos testes que visa medir a capacidade dos testes em detectar erros no código-fonte. A mutação envolve a introdução de pequenas mudanças no código-fonte, como a inversão de operadores e a modificação de valores de variáveis, para criar novas versões (mutantes) do código-fonte. Em seguida, os testes são executados novamente em cada uma das versões mutantes para avaliar se eles são capazes de detectar as mutações. A ideia é que, se os testes são capazes de detectar a maioria das mutações, então eles são considerados eficazes.

Em resumo, enquanto a cobertura de testes mede a abrangência dos testes, a mutação mede a eficácia dos testes. Ambas as abordagens são importantes para garantir a qualidade dos testes em um software. A cobertura pode ajudar a identificar áreas do código que precisam de testes adicionais, enquanto a mutação pode ajudar a identificar testes que não são capazes de detectar erros no código-fonte.

Princípios FIRST

FIRST é um acrônimo que representa cinco princípios para o desenvolvimento de testes efetivos e confiáveis. Abaixo estão os princípios FIRST:

Fast (Rápidos): Os testes devem ser rápidos para serem executados. Isso significa que os testes devem ser projetados para rodar rapidamente e sem demora, para que os desenvolvedores possam obter um feedback rápido sobre o código que estão escrevendo.

Isolated (Isolados): Cada teste deve ser independente e isolado dos outros testes. Isso significa que os testes devem ser projetados para serem executados em qualquer ordem, sem interferir nos resultados dos outros testes. Além disso, os testes devem ser projetados para não dependerem de dados externos ou do estado do sistema para serem executados.

Repeatable (Repetíveis): Os testes devem ser repetíveis em qualquer ambiente. Isso significa que os testes devem ser projetados para serem executados da mesma forma, independentemente do ambiente em que estão sendo executados. Isso inclui garantir que os testes não dependam de configurações específicas do sistema ou de recursos externos que possam variar de ambiente para ambiente.

Self-validating (Autovalidáveis): Os testes devem ser autovalidáveis, ou seja, os resultados dos testes devem ser facilmente verificáveis. Isso significa que os testes devem ser projetados para gerar resultados claros e objetivos, que possam ser facilmente avaliados pelos desenvolvedores.

Timely (Oportunos): Os testes devem ser escritos em tempo hábil. Isso significa que os testes devem ser escritos o mais cedo possível no ciclo de desenvolvimento, para que os desenvolvedores possam obter feedback rápido sobre o código que estão escrevendo. Os testes também devem ser atualizados regularmente para garantir que estejam em sincronia com as mudanças no código-fonte.

Seguir os princípios FIRST pode ajudar a garantir que os testes sejam efetivos, confiáveis e fáceis de manter, o que pode melhorar a qualidade do software e acelerar o processo de desenvolvimento.

Teste Smels

Test Smells são sinais de problemas nos testes de um software, incluindo testes mal escritos, redundantes, lentos, difíceis de manter e com pouca cobertura. A identificação de Test Smells pode ajudar a melhorar a qualidade dos testes e torná-los mais efetivos, levando a um software mais confiável e de alta qualidade.

Abaixo estão alguns exemplos de Test Smells:

Testes longos e complexos: Testes que são muito longos e complexos podem ser difíceis de entender, manter e executar. Eles também podem ser vulneráveis a erros e difíceis de depurar.

Testes redundantes: Testes que cobrem o mesmo comportamento ou funcionalidade em diferentes testes podem ser redundantes e desnecessários. Isso pode aumentar o tempo de execução dos testes e dificultar a manutenção dos testes.

Testes sem sentido: Testes que não têm nenhum propósito claro ou que não testam nada de útil podem ser uma perda de tempo e recursos.

Testes com dependências externas: Testes que dependem de serviços ou recursos externos podem ser vulneráveis a falhas de conectividade ou mudanças nas

dependências. Eles também podem ser lentos e difíceis de executar em diferentes ambientes.

Testes com baixa cobertura: Testes que não cobrem adequadamente o código-fonte podem deixar áreas críticas do software sem testes adequados. Isso pode levar a erros não detectados e aumentar o risco de falhas no software.

Identificar e corrigir Test Smells pode ajudar a melhorar a qualidade dos testes e torná-los mais efetivos. Isso pode levar a um software mais robusto, confiável e de alta qualidade.

Testabilidade

Testabilidade é a capacidade de um software ser testado de maneira eficiente e efetiva. Em outras palavras, um software é considerado testável quando é fácil de testar e produz resultados confiáveis durante os testes.

A testabilidade é um atributo importante de um software, pois impacta diretamente a qualidade do produto final. Quanto mais testável um software é, mais fácil é encontrar e corrigir defeitos durante o processo de desenvolvimento e garantir que o software atenda às necessidades do usuário.

Desenvolvimento dirigido por testes

Desenvolvimento dirigido por testes (TDD) é uma abordagem de desenvolvimento de software em que os testes automatizados são criados antes do código ser escrito. Com o TDD, o desenvolvedor escreve um teste que falha antes de escrever o código que faz o teste passar. Em seguida, o código é escrito e refatorado para torná-lo mais legível e manutenível, e o teste é executado novamente para garantir que ele passe.

O TDD é uma prática que visa garantir que o software desenvolvido atenda aos requisitos e tenha uma boa qualidade. Ele ajuda a reduzir o tempo e o custo de desenvolvimento, além de minimizar a ocorrência de defeitos e bugs no software.

O TDD é uma prática comum em metodologias ágeis de desenvolvimento de software e é amplamente utilizado em projetos de software em todo o mundo.

Ciclo TDD(Red-Green-Refactor)

O ciclo TDD, também conhecido como ciclo "Red-Green-Refactor", é um processo iterativo utilizado no desenvolvimento de software guiado por testes (TDD). O ciclo consiste em três etapas:

1. **Red:** O desenvolvedor escreve um teste automatizado que irá falhar inicialmente. Essa fase é chamada de "Red" porque a barra de progresso do teste fica vermelha indicando que o teste falhou.

2. **Green:** O desenvolvedor escreve a quantidade mínima de código necessário para fazer o teste passar. Essa fase é chamada de "Green" porque a barra de progresso do teste fica verde, indicando que o teste foi bem-sucedido.

3. **Refactor:** O desenvolvedor melhora o código para torná-lo mais legível, manutenível e eficiente. Essa fase é chamada de "Refactor" porque o desenvolvedor refatora o código existente para torná-lo mais limpo e organizado, sem alterar a funcionalidade do software.

O ciclo TDD é repetido diversas vezes, com o desenvolvedor escrevendo novos testes para cobrir novas funcionalidades ou casos de uso, e repetindo o ciclo Red-Green-Refactor para cada novo teste. Com o tempo, o software é desenvolvido iterativamente, com cada nova iteração adicionando mais funcionalidades e melhorias.

Mocks e Test Doubles

Mocks e Test Doubles são técnicas utilizadas em testes de software para isolar e testar partes específicas do código, sem depender de outras partes externas que podem não estar disponíveis ou podem afetar o resultado dos testes.

Um Mock é um objeto simulado que imita o comportamento de um objeto real, mas é usado para testar outros objetos em vez de ser usado em produção. Eles são usados para testar a interação entre objetos e para verificar se um objeto está se comunicando corretamente com outro. Um exemplo comum é o uso de um Mock para simular a interação com um banco de dados, permitindo que os testes ocorram sem precisar acessar o banco de dados real.

Test Doubles é um termo mais genérico que se refere a objetos que substituem outros objetos durante os testes. Isso inclui Mocks, mas também pode incluir outros tipos de objetos como Stubs, Spies e Fakes. Stubs são objetos que fornecem respostas pré-definidas para chamadas de métodos, enquanto Spies são objetos que registram informações sobre chamadas de métodos. Fakes são objetos que imitam o comportamento de objetos reais, mas geralmente têm uma implementação mais simples e menos completa.

Ao usar Mocks e Test Doubles, os testes podem ser executados de forma mais rápida e precisa, pois os objetos simulados são controlados e configurados especificamente para cada caso de teste. Isso permite que os desenvolvedores identifiquem e resolvam problemas com mais rapidez e facilidade, resultando em um software mais confiável e de alta qualidade.

Caixa Preta e Caixa Branca

Caixa Preta e Caixa Branca são técnicas de teste usadas em testes de software para verificar o comportamento do software.

O teste de Caixa Preta é uma técnica de teste em que o testador avalia o software sem se preocupar com o código-fonte ou a lógica interna do software. O foco

principal do teste de Caixa Preta é avaliar a funcionalidade do software de acordo com as especificações e requisitos do usuário. Essa técnica envolve a entrada de dados no software e a avaliação dos resultados de saída esperados, sem se preocupar com a forma como o software realiza o processamento. O teste de Caixa Preta é uma técnica valiosa para testar a interface do usuário, a segurança, a usabilidade e a conformidade regulatória.

O teste de Caixa Branca é uma técnica de teste em que o testador avalia o software com base na lógica interna e no código-fonte do software. O objetivo do teste de Caixa Branca é verificar se o software está funcionando conforme o esperado e se o código-fonte do software está sendo executado corretamente. Essa técnica envolve a análise do código-fonte do software e a execução de testes em diferentes caminhos lógicos, fluxos de controle, fluxos de dados e decisões do programa. O teste de Caixa Branca é uma técnica valiosa para testar a integridade do código-fonte do software, a cobertura de código e a eficiência do software.

Ambas as técnicas de teste são importantes e complementares, e devem ser usadas em conjunto para garantir que o software seja testado de forma completa e precisa.

Seleção de Dados de Teste

A seleção de dados de teste é uma etapa crucial no processo de teste de software. É o processo de selecionar um conjunto de dados de entrada que será usado para testar o software em questão. Esses dados de teste devem ser cuidadosamente escolhidos para garantir que todas as funcionalidades do software sejam testadas e que as falhas possíveis sejam identificadas.

Existem várias técnicas para a seleção de dados de teste, incluindo:

1. Teste de casos limite: esta técnica envolve a seleção de valores extremos ou limites para os dados de entrada. Isso ajuda a identificar comportamentos inesperados do software.

2. Teste baseado em equivalência: esta técnica envolve a divisão do conjunto de dados de entrada em classes de equivalência. Cada classe de equivalência contém um conjunto de valores de entrada que devem produzir o mesmo resultado de saída. O objetivo é testar apenas um valor de cada classe de equivalência para economizar tempo e esforço.

3. Teste de mutação: esta técnica envolve a modificação do código-fonte do software de forma controlada para criar mutantes. Os mutantes são variações do código-fonte original que contêm erros conhecidos. O objetivo é testar se os casos de teste identificam os erros nos mutantes.

4. Teste de regressão: esta técnica envolve a seleção de um conjunto de dados de teste que já foram usados para testar o software em versões anteriores. O objetivo é verificar se o software continua a funcionar corretamente após as mudanças no código-fonte.

5. Teste de estresse: esta técnica envolve a seleção de dados de entrada que testam os limites do software. O objetivo é verificar se o software continua a funcionar corretamente mesmo sob cargas de trabalho extremas.

Cada uma dessas técnicas tem suas próprias vantagens e desvantagens. O processo de seleção de dados de teste deve ser cuidadosamente planejado e executado para garantir que o software seja testado de forma completa e precisa.

QUALITY ASSURANCE (QA)

Quality Assurance (QA) ou garantia da qualidade, é uma disciplina que visa garantir a qualidade do software por meio da implementação de processos e procedimentos padronizados. A equipe de QA é responsável por garantir que o software

atenda aos requisitos do usuário e às especificações de qualidade, além de identificar e corrigir qualquer erro ou defeito antes que o software seja lançado ao público.

Os testes de software são uma parte essencial da garantia de qualidade. A equipe de QA deve criar e executar planos de teste para garantir que o software seja testado completamente antes do lançamento. Eles também devem fornecer relatórios de bugs e trabalhar com a equipe de desenvolvimento para corrigir quaisquer problemas identificados.

Em resumo, a garantia da qualidade e os testes de software são essenciais para garantir que o software seja confiável, seguro e atenda às expectativas do usuário.