**Material pra P1 de Teste de Software**

A **garantia** nada mais é do que uma declaração positiva sobre um produto ou serviço, que dá confiança. É a certeza de um produto ou serviço, que funcionará bem. Ele fornece uma garantia de que o produto funcionará sem problemas de acordo com as expectativas ou requisitos.

A **Garantia de Qualidade em Testes de Software** é definida como um procedimento para garantir a qualidade dos produtos ou serviços de software fornecidos aos clientes por uma organização. A garantia de qualidade se concentra em melhorar o processo de desenvolvimento de software e torná-lo eficiente e eficaz de acordo com os padrões de qualidade definidos para produtos de software.

A metodologia de Garantia da Qualidade tem um ciclo definido. As fases deste ciclo são:

Plano – A organização deve planejar e estabelecer os objetivos relacionados ao processo e determinar os processos necessários para entregar um produto final de alta qualidade.

Fazer – Desenvolvimento e teste de Processos e também "fazer" mudanças nos processos.

Verificar – Monitoramento de processos, modificação dos processos e verificação se ele atende aos objetivos predeterminados.

Agir – Um testador de Garantia da Qualidade deve implementar ações que sejam necessárias para alcançar melhorias nos processos.

**Controle de qualidade** popularmente abreviado como QC. É um processo de Engenharia de Software usado para garantir a qualidade em um produto ou serviço. Não lida com os processos usados para criar um produto; em vez disso, examina a qualidade dos "produtos finais" e o resultado final.

O principal objetivo do controle de qualidade é verificar se os produtos atendem às especificações e requisitos do cliente. Se um problema ou problema for identificado, ele precisa ser corrigido antes da entrega ao cliente.

Às vezes, o QC(Controle de Qualidade) é confundido com o QA(Garantia de Qualidade). O objetivo do controle de qualidade é examinar o produto ou serviço e verificar o resultado. Garantia de Qualidade em Engenharia de Software é examinar os processos e fazer alterações nos processos que levaram ao produto final.

**Teste de software** é testar um produto em busca de problemas antes que ele entre no ar. Foco no produto, Técnica corretiva, Medida reativa.

A **Garantia de Qualidade de Software** é sobre o processo de engenharia que garante a qualidade. Envolve atividades relacionadas à implementação de processos, procedimentos e normas. Foco no processo. Técnica preventiva. Medida proativa.

Para ser um testador eficaz, você precisa estar focado em maneiras de quebrar o software. Em certo sentido, sua intenção deve ser provar que "não funciona", mas essa abordagem por si só pode não ser tudo o que é necessário para o sucesso.

Ao mesmo tempo, você precisa de algumas habilidades especiais para poder comunicar os problemas encontrados. Apenas ser um "cara mau" que quebra tudo não é eficiente em uma perspectiva de longo prazo. Discuta um problema, não uma pessoa; Seja específico, não geral; Operar com fatos, não com julgamentos; Concentre-se no futuro, não no passado. Comunique-se a partir da posição do objetivo comum.

O trabalho de um testador de software significa não apenas encontrar bugs, mas preveni-los. Isso inclui análise de requisitos, otimização de processos e implementação de uma abordagem de teste contínuo. Nesse sentido, a mentalidade de um testador significa se preocupar com a qualidade durante todas as etapas do ciclo de vida do desenvolvimento de software. No desenvolvimento ágil, a qualidade é responsabilidade de toda a equipe, de modo que o foco principal dos testes ágeis é deslocado para a iniciativa e controle de atividades que previnam a ocorrência de defeitos.

Para ser um testador de sucesso, você precisa ser crítico para o software e gracioso para os desenvolvedores. Em outras palavras, um verdadeiro testador de software sabe não apenas como quebrar o software, mas como construir relacionamentos amigáveis e produtivos e elaborar o processo que evita defeitos.

Ao testar o software, você deve começar assumindo que o software não funciona. Na verdade, não poderia funcionar. É inconcebível que funcione. Não é apenas bug, é total e completamente errado. Alguém está realmente tentando enganá-lo, passando este software como "funcionando".

Com esse entendimento firmemente em sua mente, force o software a provar que você está errado. Faça-o executar todas as ações e demonstrar todos os comportamentos até que ele consiga incontestavelmente convencê-lo do contrário, que não é uma farsa ou uma fachada e sim, um software funcional.

Grandes testadores são capazes de se imaginar no lugar do usuário, para prever o que eles podem fazer, como eles podem estar confusos, ou por que eles podem ficar frustrados. Entender profundamente os usuários e emular seu comportamento permite que os testadores encontrem bugs antes que eles escapem e causem problemas reais para usuários reais.

Identificar e alavancar suposições é fundamental para ser um testador eficaz. Quando você se senta para testar algum novo recurso em um aplicativo não trivial, há um número quase ilimitado de possíveis ações ou caminhos que você poderia testar que poderiam levar a um bug. As suposições são exatamente o que você usa para restringir esse escopo infinito de possíveis testes ao conjunto razoável com maior probabilidade de encontrar problemas. A habilidade crítica é ser capaz de avaliar com precisão as suposições para determinar se elas estão guiando corretamente seu comportamento ou se estão levando você ao erro.

Se o teste pudesse ser destilado em uma lista simples de ações, não precisaríamos de testadores. Mesmo que pudesse ser reduzido a um conjunto previsível e determinístico de etapas com base em um conjunto finito de entradas (critérios de aceitação e SUT, por exemplo), ainda não precisaríamos de testadores. O teste é uma atividade não linear e imprevisível. Requer pensamento crítico, criatividade e depende tanto do instinto e da intuição quanto na execução algorítmica de ações.

Como testador, você testa o software, mas mantém em mente que o software é apenas o produto final de um longo processo. É o resultado final de um grupo de humanos colaborando ao longo do tempo para criar algo maior e mais complexo do que qualquer um deles poderia ter criado individualmente. Defeitos no software não apenas saltam para a existência, eles são sintomas de problemas nesse processo. Assim, como testador, você precisa se preocupar com o processo de desenvolvimento de software e o comportamento dos seres humanos dentro dele, tanto quanto você precisa se preocupar com o produto de software final. Se você estudar a natureza humana, logo perceberá que os seres humanos estão longe de robôs deterministas com memória, motivação e atenção perfeitas. Na verdade, somos exatamente o contrário.

O **White Box Testing** é uma técnica de teste na qual a estrutura interna, o design e a codificação do software são testados para verificar o fluxo de entrada-saída e melhorar o design, a usabilidade e a segurança. No teste de caixa branca, o código é visível para os testadores, por isso também é chamado de teste de caixa clara, teste de caixa aberta, teste de caixa transparente, teste baseado em código e teste de caixa de vidro. O teste de caixa branca envolve o teste do código de software para o seguinte: Lacunas de segurança interna; Caminhos quebrados ou mal estruturados nos processos de codificação; O fluxo de entradas específicas através do código; Saída esperada; Teste de cada instrução, objeto e função em uma base individual; A funcionalidade de loops condicionais;

Os testes podem ser feitos nos níveis de sistema, integração e unidade de desenvolvimento de software. Um dos objetivos básicos do teste de caixa branca é verificar um fluxo de trabalho para um aplicativo. Envolve testar uma série de entradas predefinidas em relação às saídas esperadas ou desejadas para que, quando uma entrada específica não resultar na saída esperada, você tenha encontrado um bug.

Dois passos para fazer o teste de caixa branca são entender o código-fonte e também criar casos de teste e executá-los. A seguir estão importantes técnicas de teste de caixa branca: Cobertura de Demonstrativos; Cobertura de Decisão; Cobertura de Filiais; Cobertura de Condições; Cobertura de Caminhos; Teste de fluxo de dados; Teste de fluxo de controle.

Vantagens do teste de caixa branca: Otimização de código encontrando erros ocultos; Os casos de testes de caixa branca podem ser facilmente automatizados; O teste é mais completo, pois todos os caminhos de código geralmente são cobertos;

Desvantagens do teste de caixa branca: O teste de caixa branca pode ser bastante complexo e caro; Os desenvolvedores que geralmente executam casos de teste de caixa branca o detestam. O teste de caixa branca feito pelos desenvolvedores não é detalhado e pode levar a erros de produção; O teste de caixa branca é demorado, aplicativos de programação maiores levam tempo para testar completamente.

O **teste de caixa cinza** é uma técnica de teste de software para testar um produto ou aplicativo de software com conhecimento parcial da estrutura interna do aplicativo. O objetivo do teste de caixa cinza é pesquisar e identificar os defeitos devido à estrutura de código inadequada ou ao uso inadequado de aplicativos. Nesse processo, erros específicos de contexto relacionados a sistemas da Web são comumente identificados. Aumenta a cobertura de testes, concentrando-se em todas as camadas de qualquer complexo.

Em Engenharia de Software, o Gray Box Testing oferece a capacidade de testar os dois lados de um aplicativo, a camada de apresentação, bem como a parte de código. É útil principalmente em Testes de Integração e Testes de Penetração.

O teste de caixa cinza é realizado pelos seguintes motivos: Ele fornece benefícios combinados de ambos os testes de caixa preta e teste de caixa branca; Ele combina a entrada de desenvolvedores, bem como testadores e melhora a qualidade geral do produto; Reduz a sobrecarga de longo processo de teste de tipos funcionais e não funcionais; Ele dá tempo livre suficiente para um desenvolvedor corrigir defeitos. As técnicas usadas para o teste de caixa cinza são: Teste de matriz, Teste de regressão(Este teste é realizado com base nos dados históricos dos defeitos do sistema anterior), Teste de padrão.

O **teste de caixa preta** é um método de teste de software no qual as funcionalidades de aplicativos de software são testadas sem ter conhecimento da estrutura interna do código, detalhes de implementação e caminhos internos. O Black Box Testing concentra-se principalmente na entrada e saída de aplicativos de software e é inteiramente baseado em requisitos e especificações de software. Também é conhecido como Teste Comportamental.

A seguir estão as estratégias de teste proeminentes entre as muitas usadas no teste de caixa preta: Teste de Classe de Equivalência, Teste de Valor de Limite, Teste de Tabela de Decisão.

Existem muitos tipos de Teste de Caixa Preta, mas os seguintes são os mais proeminentes: Teste funcional (Este tipo de teste de caixa preta está relacionado aos requisitos funcionais de um sistema; é feito por testadores de software.); Teste não funcional(Esse tipo de teste de caixa preta não está relacionado ao teste de funcionalidade específica, mas a requisitos não funcionais, como desempenho, escalabilidade e usabilidade.); Teste de regressão(O teste de regressão é feito após correções de código, atualizações ou qualquer outra manutenção do sistema para verificar se o novo código não afetou o código existente.)

O teste de caixa preta fornece abstração do código e se concentra no esforço de teste no comportamento do sistema de software. O foco principal do teste de caixa preta é a validação de seus requisitos funcionais. O teste de caixa preta facilita o teste de comunicação entre os módulos.

Um **oráculo de teste** é um mecanismo; diferente do próprio programa que pode ser usado para verificar a exatidão da saída do programa para os casos de teste. Conceitualmente, podemos considerar o teste de um processo no qual os casos de teste são dados ao oráculo de teste e ao programa em teste.

A saída dos dois é então comparada para determinar se o programa se comportou corretamente para casos de teste. Para ajudar o oráculo a determinar o comportamento correto, é importante que o comportamento do sistema ou componente seja especificado de forma inequívoca e que a especificação em si esteja livre de erros. Existem alguns sistemas onde os oráculos são gerados automaticamente a partir de especificações de programas ou módulos. Com esses oráculos, temos a certeza de que a saída do oráculo é consistente com as especificações.

**Priorização de caso de teste** ou TCP, como o nome sugere, é o processo de priorizar casos de teste em um conjunto de testes com base em diferentes fatores que podem ser qualquer coisa, desde a cobertura de código e funcionalidade até módulos de risco / críticos, recursos, etc. Ele fornece uma abordagem para executar casos de teste altamente significativos primeiro de acordo com algumas medidas e, em seguida, produzir o resultado desejado, como revelar as falhas mais cedo e fornecer aos testadores o feedback.

Por que a priorização de casos de teste é importante? Porque o teste é uma das fases mais criticamente importantes do ciclo de vida do desenvolvimento do software, que consome recursos significativos em termos de custo, esforço e tempo. A parte difícil do teste é o gerenciamento de riscos, o planejamento do teste, o valor do custo e a análise sobre qual teste executar para seu projeto específico. Executar todos os casos de teste em um conjunto de testes pode exigir uma grande quantidade de esforço e, assim, aumentar o custo do teste de regressão. À medida que o tamanho do software cresce, o conjunto de testes também cresce e, portanto, requer mais esforço para manter o conjunto de testes. Para aplicações complexas, é impossível e impraticável testar exaustivamente cada cenário. Priorizar casos de teste com base nos riscos percebidos e nas necessidades do cliente pode reduzir com eficiência o número de casos de teste necessários para testar um aplicativo. Priorizar casos de teste também ajuda a atender aos marcos do projeto, além de atender aos requisitos e expectativas do cliente. A detecção precoce de bugs também pode ser alcançada.

Tipos de priorização de caso de teste: Priorização geral de casos de teste(Nessa priorização, os casos de teste são priorizados com base em como eles serão úteis para versões modificadas subsequentes do produto. Não requer qualquer conhecimento das versões modificadas.) e Priorização de caso de teste específico da versão(Nessa priorização, priorizamos os casos de teste de modo que eles sejam úteis em uma versão específica do produto. Isso requer conhecimento de todas as mudanças que foram feitas no produto. Isso é realizado antes do teste de regressão da versão modificada.)

TCP baseado em risco: Esse tipo de priorização usa a análise de risco para identificar possíveis áreas problemáticas que, se falharem, podem levar a más consequências. Portanto, os casos de teste são priorizados tendo em mente as possíveis áreas problemáticas. As seguintes etapas são executadas na análise de risco: Listar possíveis problemas; Atribuir a probabilidade de ocorrência para cada problema; Calcular a gravidade do impacto para cada problema.

TCP baseado em histórico: Nesta técnica, os casos de teste são priorizados principalmente com base no histórico dos casos de teste, ou seja, os resultados de execução anteriores são verificados para os casos de teste. Ele é usado para determinar as possíveis chances de falha no teste e aqueles que são mais propensos a falhar são priorizados sobre aqueles que têm menores chances de falha.

TCP baseado em reconhecimento de custo: Nesta técnica, os casos de teste são priorizados com base no fator de custo. Os casos de teste, que são menos dispendiosos, serão priorizados em relação aos mais custosos. Inclui: Custo do processo de teste de regressão; Custo de levantamento de requisitos; Custo de analisar se um caso de teste deve ser selecionado ; Custo de priorização de casos de teste.

Sem dúvida, a priorização de casos de teste é uma das maneiras mais eficazes de produzir produtos de alta qualidade de acordo com o mercado e os padrões dos usuários finais. É uma maneira de priorizar e agendar casos de teste de prioridade de alto nível para prioridade de baixo nível. Esse método é melhor para minimizar o custo, o esforço e o tempo durante a fase de teste de software.