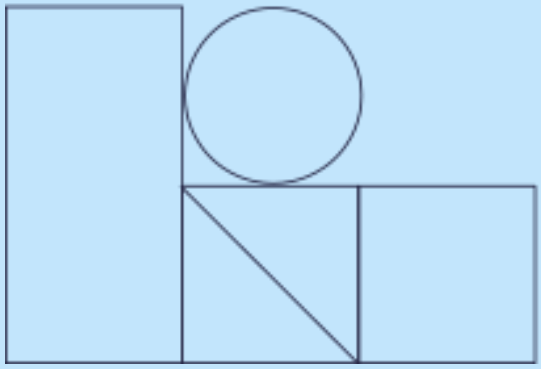
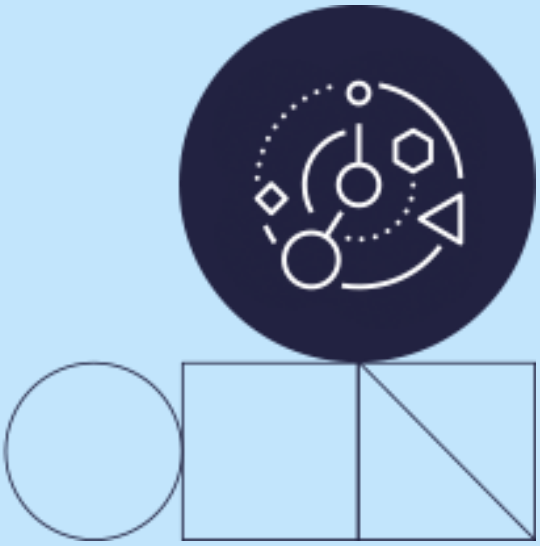
IBM SkillsBuild

IBM

Testes com Python

A importância do testing





Índice

Introdução

Como todos sabemos, o processo de criação do software é composto de várias fases. Desde sua concepção até sua implementação na produção, ele deve passar por vários momentos em que este software evolui, no entanto, há uma fase que não foi promovida tanto quanto deveria por causa de sua natureza. De fato, o testing, embora sua importância e necessidade sejam reconhecidas, vemos que em muitos casos é feito mal ou simplesmente não é feito de todo.

O que é testing de software

O testing de software ou GQ de software é uma disciplina em engenharia de software que permite ter processos de execução de um programa ou aplicação e uma metodologia de trabalho com o objetivo de localizar erros de software. Também pode ser descrito como o processo de validação e verificação de um programa ou aplicação de software.

É essencial ter em mente que o testing é paralelo ao processo de desenvolvimento de software. Como nosso produto está sendo construído, temos que realizar tarefas de testing de software para evitar problemas de funcionalidade e corrigir desvios do software antes de seu lançamento.

Por que o testing es importante

A um alto nível, o teste de software é necessário para detectar bugs no software e para testar se o software atende aos requisitos do cliente. Isto ajuda a equipe de desenvolvimento a corrigir erros e entregar um produto de boa qualidade.

Há vários pontos no processo de desenvolvimento de software onde o erro humano pode levar a um software que não atende às exigências do cliente. Algumas delas estão listadas abaixo.

- O cliente/pessoa que fornece os requisitos em nome da organização do cliente pode não saber exatamente o que é exigido ou pode esquecer de fornecer alguns detalhes, o que pode levar a características ausentes.
- A pessoa que está coletando as exigências pode interpretá-las mal ou não cumpri-las totalmente ao documentá-las.
- Durante a fase de projeto, se houver problemas no projeto, isto pode levar a erros no futuro.
- Os erros podem ser introduzidos durante a fase de desenvolvimento durante o erro humano, falta de experiência, etc.
- Os testadores podem perder erros durante a fase de teste devido a erro humano, falta de tempo, experiência insuficiente, etc.
- Os clientes podem não ter largura de banda para testar todas as características do produto e liberar o produto para seus usuários finais, o que pode fazer com que os usuários finais encontrem erros na aplicação.
- O negócio e a reputação de uma organização dependem da qualidade de seus produtos e, em alguns casos, até mesmo as receitas podem depender das vendas de produtos de software.

Os usuários podem preferir comprar o produto de um concorrente em vez de um produto de baixa qualidade, o que pode resultar em uma perda de receita para a organização. No mundo de hoje, a qualidade é uma das principais prioridades de qualquer organização.

O que é um tester

Os testadores de software (também conhecidos como testers, seu nome em inglês) planejam e realizam testes de software de computadores para verificar se eles funcionam corretamente. Eles identificam o risco de bugs no software, detectam erros e os reportam. Eles avaliam o desempenho geral do software e sugerem maneiras de melhorá-lo.

Em muitos casos, a fase de testes foi relegada a uma fase final de pré-produção com tempo tão limitado que, em muitos casos, ela não pode garantir um testing efetivo.

Falando em testes funcionais, os testes podem ser mais valiosos, dados os resultados que oferecem (passados ou não passados), ao contrário, outros testes, tais como desempenho ou segurança, são relegados a um ponto menos quantificável, uma vez que não afeta sua funcionalidade direta e o negócio normalmente não vem com requisitos específicos.

Criação de perfil um tester

Em um projeto, o tester deve ser o segundo mais conhecedor do projeto (imediatamente após o gerente do projeto), sua arquitetura (após seus arquitetos), seu projeto (após seus projetistas) e seu desenvolvimento (após seus desenvolvedores). Há apenas um caso em que o testador deve ser aquele que mais sabe sobre algo, e esse é o teste. Além disso, em um perfil do testador de desempenho, o testador, além de todas as habilidades mencionadas acima, acrescentaria a de um matemático estatístico, já que, como os resultados de desempenho aparecem em gráficos e estatísticas, o testador deve saber expressar esses termos matemáticos de forma fácil e compreensível.

Como implementar um serviço de testing a partir do zero

Implementar um serviço de testes a partir do zero é uma tarefa complexa e demorada. Em projetos comprovados, vimos que pequenos mas eficazes passos foram dados em direção a um serviço de QA em contínua integração. Passos como a dedicação de pessoas especializadas neste campo, a implementação de ferramentas como Testlink para gerenciamento de testes, SonarQube para avaliar a qualidade do código, Jenkins para integração contínua ou Selenium para automação de testes. No futuro dos testes, horizontes como Big Data testing estão surgindo, portanto o futuro do testing está garantido.

Em nosso caso, usaremos unittest, uma estrutura de teste unitário que foi originalmente inspirada pela JUnit e tem um estilo similar às principais estruturas de teste unitário em outros idiomas. Suporta equipamentos(fixtures), conjuntos de teste e um test runner para permitir testes automatizados.

Os benefícios dos testes automatizados

A realização e execução de casos de teste é uma parte crítica dos serviços de desenvolvimento de software. Atingir altos níveis de qualidade não só beneficia o usuário final, mas também evita bugs no software que prejudicam o propósito do software. Dentro deste contexto, a equipe de GQ deve levar em conta o número de casos de teste a serem realizados e o tempo necessário para executá-los. Em certos casos, os testadores devem executar testes que levam muito tempo, tais como uma regressão completa do software; em outros casos, apenas completar os casos de teste de aceitação pode levar algum tempo. Seja como for, estes tipos de testes envolvem um investimento de recursos que poderiam ser dedicados a outros aspectos específicos do software que podem ser mais complexos.

A seguir estão algumas vantagens que deixam clara a importância da automação de testes de software:

- **Economia de tempo e esforço.** Por exemplo, um sprint pode incluir várias características novas que precisam ser testadas, mas além disso, deve-se verificar que estas não afetaram o sistema como um todo. Nessas ocasiões, fazer uma regressão é sempre muito útil. Se os casos de teste forem automatizados, eles são executados e os testers só têm que rever os resultados, permitindo que eles se concentrem no teste manual das novas características.
- **Verificar se os erros do passado não estão sendo repetidos.** Caso um bug tenha sido corrigido, a automatização do caso de teste que o verifica pode garantir que este bug não seja reintroduzido por uma mudança em outras funcionalidades.
- **Verificar se o produto é funcional para o usuário.** Ao concentrar os casos de teste na funcionalidade mínima do software, a equipe de QA pode garantir que a nova versão não afetará a utilidade mínima para o usuário. A automação

destes casos de teste é uma grande ajuda porque, estando próximos do lançamento, os testadores podem se concentrar na nova funcionalidade enquanto os testes de aceitação são executados de forma automatizada.

- **Verificar se não há nenhuma funcionalidade quebrada na 'build'.** Ao implantar uma versão do produto, seja ela uma nova característica ou uma correção, a 'build' deve ser verificada. Neste processo de Integração Contínua/Desdobramento Contínuo (CI/CD por sua sigla em inglês), a automação de casos de teste focados na verificação da 'build' é de grande ajuda, pois garante uma funcionalidade mínima.

Testes automatizados de software: diretrizes essenciais

A fim de colher todos os benefícios de um testing automatizado de software, é crucial seguir estas diretrizes:

- **Seleção dos casos de teste a serem automatizados.** Esta tarefa requer tempo e esforço, portanto, ter selecionado os casos de teste com antecedência é muito útil quando é hora de executar os testes (seja em regressão ou teste de aceitação). É fundamental selecionar os casos de teste que cobrem a maioria dos módulos do produto ou escolher aqueles que cobrem a funcionalidade mais crítica do software.
- **Seleção da ferramenta correta para a criação de casos de teste.** É vital considerar as ferramentas que serão utilizadas para criar um framework para implementá-las. Desta forma, é mais fácil aumentar a cobertura dos testes em relação aos casos de testes automatizados e o tempo necessário para executá-los. Além disso, todos os aspectos a serem cobertos como GUI, API, validações de bancos de dados, etc., devem ser considerados.
- **Implementando casos de teste que podem ser executados em múltiplos ambientes de teste.** É crucial que os casos de teste sejam úteis para todos os ambientes porque, no caso de testes em execução em ambientes semelhantes aos de produção, só seria necessário alterar os dados utilizados para cada caso. Isto torna muito mais fácil atualizá-los.
- **Levando em conta a atualização dos casos de teste.** Em caso de mudanças na lógica do software ou no próprio produto, os casos de testes automatizados precisam ser fáceis de atualizar. Além disso, devem ser fáceis de depurar para bugs, pois o software também pode exigir correções a serem adicionadas.

É vital que a equipe de QA leve em conta as diretrizes acima para otimizar a automação de testes de software e garantir que seja uma ferramenta que facilite seu trabalho. Caso contrário, a automação corre o risco de se tornar uma tarefa adicional que requer tempo e esforço. Quando implementados corretamente, os testes automatizados trazem muitos benefícios tanto para o desenvolvimento quanto para a verificação do software.

O objetivo

Às vezes, orçamentos apertados não permitem um recurso dedicado. Em todos os casos, é aconselhável reservar parte do orçamento para fazer el testing.

É muito difícil propor uma melhoria quantificável graças à qualidade, pois estamos falando de melhorias qualitativas e não quantitativas, no entanto, é possível ver a necessidade de testing graças a erros que ocorreram ao longo da história. É bem conhecido que em dias específicos os serviços podem cair, como por exemplo o Black Friday em grandes lojas ou o início da campanha do imposto de renda. Também os fãs de videogames estarão cientes do "patch do dia 1", um patch de correção de bugs lançado no mesmo dia do lançamento do produto. Estes erros poderiam ser previstos graças à execução correta da fase de testing.

Tipos de testes

Dentro do mundo do desenvolvimento de software, um nível crescente de importância está sendo colocado em testes e automação de testes.

Os seguintes são os principais tipos de testes utilizados durante todo o ciclo de criação do programa.

Teste unitário

O teste de unidade é um método de teste que se concentra em examinar "unidades" individuais ou fragmentos de código. O principal objetivo dos testes unitários é determinar a integridade lógica: que um pedaço de código faça o que é suposto fazer.

Geralmente, neste tipo de teste, métodos ou funções individuais serão testados como unidades e, dependendo do tamanho e complexidade do código, também como classes.

Eles são testados isoladamente e depois as dependências típicas são removidas ou ridicularizadas.

Um exemplo disso poderia ser uma função que envia mensagens aos usuários de um banco de dados. Entretanto, como o que queremos fazer é um teste de unidade, não usaríamos um banco de dados real: faríamos uma chamada para um ponto final com stub, que retorna os dados que normalmente esperaríamos de um banco de dados. Dessa forma, a única funcionalidade a ser testada é este fragmento de código ou unidade.

A maioria dos idiomas tem pelo menos uma estrutura de teste unitário recomendada para si mesmos (por exemplo, Java → JUnit , Python → PyUnit o PyTest , JavaScript → Mocha , Jest , Karma , etc.)

Teste de integração

O teste de integração é um método de teste que se concentra no exame conjunto de vários componentes.

O principal objetivo dos testes de integração é garantir a integridade da relação e do fluxo de dados entre componentes ou unidades. Ou seja, se eles funcionam corretamente uma vez que "montamos" todo o código e o colocamos em funcionamento.

Tipicamente, primeiro faremos testes unitários para testar a integridade lógica das unidades individuais. Em seguida, faremos testes de integração para garantir que a interação entre essas unidades se comporte como esperado. Continuando com o exemplo anterior, um teste de integração neste caso seria executar o mesmo teste contra um banco de dados real. Com bancos de dados reais, existem cenários e comportamentos adicionais a serem considerados.

"Teste de integração" é um termo amplo e engloba qualquer teste onde múltiplos componentes estão envolvidos. Posteriormente, uma grande variedade de tecnologias e estruturas pode ser usada, incluindo as mesmas usadas anteriormente em testes unitários, ou estruturas separadas baseadas no comportamento.

Teste de ponta a ponta (E2E, sistema)

Os testes de sistema, ou testes de ponta a ponta (E2E), concentram-se no exame do comportamento de um sistema de ponta a ponta.

O objetivo principal dos testes de ponta a ponta é garantir que toda a aplicação ou sistema como uma unidade se comporte como esperamos, independentemente do funcionamento interno.

Em essência, os testes de unidade e integração são tipicamente testes de "caixa branca", enquanto os testes E2E são tipicamente testes de "caixa preta". Um exemplo de um teste E2E pode ser "Obter dados

de um usuário". A entrada poderia ser um simples pedido de GET para um caminho específico, e então verificamos se a saída retornada é o que esperamos. Como o sistema obteve esses dados é irrelevante.

Como podemos ver, os testes E2E só podem verificar o comportamento geral, e é por isso que são necessários testes unitários e de integração. Pode ser que, embora o resultado esteja correto, a forma como o resultado é obtido internamente esteja incorreta, e um teste E2E não detectaria isto.

Para os testes E2E, normalmente usamos estruturas baseadas no comportamento.

Podemos utilizar estruturas como Cucumber, Postman, SoapUI, Karate, Cypress, Katalon, etc. Muitas estruturas de teste API são usadas para testes E2E porque uma API é normalmente a forma como as aplicações interagem entre si.

Teste de aceitação

Os testes de aceitação são geralmente uma fase do ciclo de desenvolvimento.

O principal objetivo dos testes de aceitação é verificar se um determinado produto ou característica foi desenvolvido de acordo com as especificações estabelecidas por um cliente ou um interessado interno, como um gerente de produto.

Dentro dos testes de aceitação, também pode haver várias fases, tais como α-testing ou β-testing. À medida que grande parte do mundo de desenvolvimento de software caminha para processos ágeis, os testes de aceitação do usuário se tornaram muito menos rígidos e mais colaborativos.

É importante observar que embora os testes de aceitação possam verificar que a aplicação se comporta como o usuário pretendia, ela não verifica a integridade do sistema. Outra advertência de teste de aceitação do usuário é que há um limite para os casos e cenários que uma pessoa pode conceber; é por isso que os métodos de teste automatizados acima são importantes, pois cada caso de uso e cenário é codificado de forma rígida.

Teste da caixa branca (estrutural, caixa transparente)

Os testes da caixa branca (também chamada de caixa estrutural ou transparente) descrevem testes ou métodos onde os detalhes e o funcionamento interno do software em teste são conhecidos.

Uma vez que conhecemos as funções, métodos, classes, como funcionam e como são montados, estes testes são geralmente ótimos para examinar a integridade lógica do código.

Por exemplo, podemos saber que existe uma peculiaridade na forma como um determinado idioma lida com determinadas operações. Poderíamos escrever testes específicos para isso, que de outra forma não saberíamos escrever em um cenário de caixa preta.

Os testes unitários e de integração são geralmente testes de "caixa branca”.

Teste da caixa preta (funcional, comportamental, caixa fechada)

Em contraste, o teste de caixa preta (também chamado de teste funcional, comportamental ou de caixa fechada) descreve qualquer teste ou método onde os detalhes e o funcionamento interno do software a ser testado são desconhecidos.

Como você não conhece nenhum dos detalhes, não podemos realmente criar casos de teste que abordem cenários específicos ou enfatizem a lógica específica do sistema.

Tudo o que sabemos é que para um determinado pedido ou entrada, espera-se um certo comportamento ou saída. Portanto, o teste da caixa preta testa principalmente o comportamento de um sistema. Os testes de ponta a ponta são frequentemente testes de caixa preta.

Teste da caixa cinza

O teste da caixa cinza é apenas uma combinação híbrida de caixa preta e caixa branca.

O teste da caixa cinzenta leva a facilidade e simplicidade do teste da caixa preta (por exemplo, entrada → saída) e visa sistemas específicos relacionados com o código de teste da caixa branca.

A razão pela qual existe o teste da caixa cinza é porque o teste da caixa preta e o teste da caixa branca sozinhos podem perder funcionalidades importantes.

- Os testes da caixa preta são apenas testes de que você obtém uma determinada saída para uma determinada entrada. Ele não testa a integridade dos componentes internos; podemos obter o resultado correto por puro acaso.
- Os testes da caixa branca se concentram na integridade das unidades individuais e como elas trabalham em conjunto, mas às vezes são insuficientes para encontrar defeitos em todo o sistema ou em vários componentes.

Ao combinar os dois tipos juntos, o teste da caixa cinza pode abranger cenários mais complicados para validar verdadeiramente que uma aplicação é sólida em estrutura e lógica.

Teste manual

Como o nome sugere, testes manuais são testes nos quais um usuário especifica manualmente a entrada ou interage com um sistema. Eles também podem avaliar manualmente os resultados.

Este método de teste geralmente pode ser lento e propenso a erros. Grande parte da indústria de software avançou em direção a testes automatizados juntamente com a adoção de princípios ágeis.

Hoje, os usuários podem testar manualmente um produto beta para aceitação, casos de borda e cenários de nicho.

Teste estático

Os testes estáticos descrevem qualquer método ou método de teste onde nenhum código real é executado.

Na realidade, isto inclui a revisão do código junto com outros testers, verificação manual da lógica e integridade de funções, classes, etc.

Assim como os testes manuais, os testes estáticos podem ser lentos e propensos a erros, e os testes estáticos são geralmente realizados como uma primeira linha de defesa para detectar problemas muito óbvios.

Muitas empresas se envolvem em revisões de código antes que o trabalho de um engenheiro seja fundido no ramo principal. Estas revisões de código são muito úteis para economizar tempo.

Testes dinâmicos

Os testes dinâmicos descrevem qualquer método de teste ou método de teste no qual o código está sendo executado de fato.

Geralmente, todos os métodos de teste acima são dinâmicos, exceto os testes manuais e às vezes de aceitação. Tipicamente, você executa scripts automatizados ou usa estruturas para executar entradas em seu sistema.

Testes visuais / de interface com o usuário (testes de navegador)

Os testes de interface de usuário ou de navegador descrevem testes que examinam especificamente a integridade e o comportamento dos componentes da interface de usuário.

Muitas vezes, ao utilizar um website, espera-se que certas ações resultem em certos estados.

Os testes da IU verificam se isso acontece corretamente. Por exemplo, a forma como implementamos certo código CSS pode falhar no Firefox, mas não no Chrome. Os testes do navegador podem verificar isto.

Há muitas estruturas populares de teste de navegador, como Selenium, Cypress, TestCafe, SauceLabs, KatalonStudio, BrowserSync, Robot, etc.

Teste de fumaça

Os testes de fumaça referem-se apenas a um subconjunto menor de verificações para verificar razoavelmente se um sistema está funcionando.

Eles consistem na escolha e execução de um conjunto não exaustivo de testes que examinam a funcionalidade central.

Um exemplo disso pode ser testar apenas alguns fluxos de usuários, como "Obter dados de um usuário" acima. Não é exaustivo, mas dado que a maior parte de nossa aplicação envolve o login de um usuário, fazendo um pedido e obtendo dados de algum lugar, este teste, ou alguns testes similares, pode nos dar confiança razoável de que nosso sistema é funcional e funcional.

Normalmente, os testes de fumaça são executados quando os usuários esperam que as mudanças não tenham tido nenhum impacto significativo sobre a lógica e a função geral. Pode ser caro e demorado executar o conjunto completo de todos os testes a cada vez, de modo que os testes de fumaça são usados como uma medida de segurança econômica que pode ser executada com mais freqüência.

Teste de regressão

O teste de regressão é um método de teste para verificar se alguma característica funcional anterior se rompeu (ou regrediu) repentinamente.

Isto freqüentemente inclui a execução de toda a unidade, integração e testes do sistema para garantir que nenhuma funcionalidade tenha mudado de forma inesperada.

Os testes de regressão são muitas vezes demorados e podem ser muito caros, portanto, às vezes as pessoas fazem testes de fumaça, especialmente se não se espera que mudanças recentes afetem logicamente todo o sistema.

Muitas vezes, quando as pessoas montam CI/CD, elas executam testes de fumaça em quase todos os compromissos, enquanto as suítes de regressão podem ser executadas em intervalos definidos ou em grandes funções para garantir uma integração contínua e perfeita.

Teste de carga

Os testes de carga testam a resposta de uma aplicação ao aumento da demanda.

Isto inclui testes para influxos repentinos de solicitações ou usuários que poderiam colocar uma pressão inesperada sobre o sistema. Os testes de carga são freqüentemente realizados como parte dos testes de segurança para garantir que uma aplicação e seu sistema não possam ser DDOS.

Também são realizados testes de carga para verificar a quantidade máxima de dados que um sistema pode tratar a qualquer momento. É fundamental para ajudar as equipes a determinar fórmulas de escalonamento eficazes e implementação de alta disponibilidade (HA).

Testes de inserção

O teste de inserção (ou penetração) é uma forma de teste de segurança que envolve a verificação da robustez e segurança de uma aplicação.

Todas as formas pelas quais uma aplicação pode ser comprometida (roteiro cruzado, entrada não sanitizada, ataques de buffer overflow, etc.) são exploradas para verificar como o sistema lida com ela. O teste de penetração é uma parte importante para garantir que uma empresa não seja vítima de violações graves.