**C214A – Eng. de Software - LE1 2023/1**

**1) Conforme avançamos o desenvolvimento de um software, e junto com ele adicionamos testes de unidade, criamos uma suíte de testes. Por que é importante executarmos essa suíte de testes frequentemente? Que benefício elas nos trazem?**

R- Quando existe uma suíte de testes automatizados, o desenvolvedor pode executá-la após qualquer modificação. É chamado teste de regressão. Se algum erro foi introduzido no código, alguns testes irão falhar indicando o local do erro de forma precisa. Então, executar a suíte de testes frequentemente traz alguns benefícios, como por exemplo:

- Garantir que as alterações sejam integradas ao software sem problemas.

- Identificar rapidamente problemas que possam surgir durante o desenvolvimento do software, permitindo que a equipe de desenvolvimento os corrija em um estágio inicial antes que se tornem mais complexos e difíceis de corrigir.

- Reduzir os custos de desenvolvimento e manutenção do software. A correção de problemas mais tarde no ciclo de vida do software pode ser muito mais cara e demorada.

**2) O que você entende por teste automatizado? Qual a diferença para um teste manual?**

R- Teste automatizado é um tipo de teste de software que é executado por meio de ferramentas de software especializadas, em vez de ser executado manualmente por um testador humano.

Os testes automatizados são mais rápidos, precisos e escaláveis do que os testes manuais, permitindo que os desenvolvedores realizem mais testes em menos tempo e encontrem erros com mais facilidade. Além disso, os testes automatizados podem ser executados várias vezes, permitindo que os desenvolvedores verifiquem se as correções de erros funcionam corretamente, garantindo que não haja regressão.

**3) Quando utilizamos um framework para objetos mock, é esperado que ele nos auxilie na criação desses objetos. Escreva três características que você julga importante estar presente nesses frameworks, para que de fato os objetos mock possam contribuir para os testes.**

- Simular os comportamentos diversos das dependências que desejamos “mockar”.

- Retornar valores diferentes dependendo da situação ou do contexto em que estão sendo usados.

- Lançar exceptions.

- Modificar parâmetros.

Além de facilitar a legibilidade do código de teste e reduzir a complexibilidade quando o serviço externo é muito complexo.

**4) O que são objetos mock e em quais situações é interessante os utilizarmos? Quais os benefícios de utilizarmos objetos mock?**

R- Objetos mock são objetos simulados que “imitam” o comportamento de objetos reais de forma controlada. Eles são usados em situações onde é difícil ou impraticável criar e manipular objetos reais em um ambiente de teste. Os mocks são úteis em testes de unidade, onde o objetivo é isolar o código de um módulo específico e testá-lo de forma independente.

Alguns benefícios de usar objetos mock em testes:

Maior controle: Como os mocks são objetos simulados, é possível controlar com precisão o comportamento que eles exibem em um cenário de teste. Isso pode tornar os testes mais confiáveis e reproduzíveis.

Maior eficiência: Usar objetos mock pode ser muito mais eficiente do que criar e manipular objetos reais em um ambiente de teste. Os mocks podem ser criados e destruídos rapidamente e podem ser configurados para retornar resultados específicos sem precisar executar o código real que geraria esses resultados.

**5) Uma técnica muito utilizada por programadores é imprimir na tela, através de comandos, mensagens que os guiem durante a execução do software a fim de verificações. Por que essa técnica não pode ser considerada um teste automatizado?**

R- Porque mensagens na tela precisam ser validadas manualmente, ou seja, requer intervenção manual para verificar se o resultado do código está correto ou não.

**6) Um desenvolvedor está atuando em um arquivo Lanche.java, e implementou algumas funcionalidades. A fim de enviar o arquivo ao repositório central, executou o comando git add Lanche.java. Porém, antes de executar o git commit, observou que faltava um recurso na classe Lanche.java para lidar com lanches vegetarianos. Assim, fez a modificação necessária e em seguida executou o git commit. Entretanto, a modificação não chegou ao repositório central. O que aconteceu?**

R- Quando o desenvolvedor fez a modificação para lidar com lanches vegetarianos, ele não adicionou a versão modificada do arquivo ao índice do Git, usando novamente "git add" antes de fazer o commit. Toda vez que um arquivo é modificado, é necessário executar o “git add <arquivo>” novamente para que ele possa ser “commitado”. Portanto, a versão original do arquivo Lanche.java foi commitada, não incluindo as modificações para lidar com lanches vegetarianos.

**7) Dentre os sistemas de controle de versões, temos os centralizados e os distribuídos. Comente sobre cada um deles, mostrando suas diferenças.**

R- Sistemas de controle de versão centralizados: possuem um servidor central que armazena todas as versões do código fonte. Os desenvolvedores fazem check-out de uma cópia do código do servidor central para trabalhar em suas modificações localmente. Quando o trabalho é concluído, as alterações são enviadas de volta para o servidor central e são incorporadas na versão principal do código.

Sistemas de controle de versão distribuídos: não possuem um servidor central. Cada desenvolvedor mantém uma cópia completa do repositório, incluindo todo o histórico de alterações. Isso permite que os desenvolvedores trabalhem offline e de forma independente, sem depender de uma conexão com um servidor central. Quando as alterações são concluídas, elas são mescladas com as versões de outros desenvolvedores.

Sistemas de Controle de Versões Centralizados e baseado em arquitetura cliente/servidor.

- Único servidor com repositório e o sistema de controle de versão.

- Clientes acessam o servidor para obterem a versão mais recente.

- Clientes atualizam o servidor com o commit.

Sistemas de Controle de Versões Distribuídos (DVCS)

- Cada cliente possui um repositório local completo.

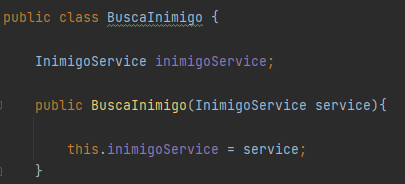
- Na teoria os clientes (peers) são funcionalmente equivalentes.

- Na prática existe uma máquina principal com uma versão de referência do código fonte.

- Clientes podem trabalhar de forma independente e off-line realizando commits no repositório local.

Resumidamente, uma das principais diferenças entre esses dois tipos de sistemas é a sua arquitetura. Enquanto os sistemas centralizados dependem de um servidor central, os sistemas distribuídos não têm um ponto centralizado de falha e são mais descentralizados. Isso faz com que os sistemas distribuídos sejam mais flexíveis e adaptáveis a diferentes cenários de colaboração.

**8) Analisando a Figura que mostra a classe BuscaInimigo, o que está acontecendo no construtor? Qual a importância desse código para testes com objetos mocks?**



R- O construtor está injetando uma dependência externa na classe BuscaInimigo. Ao injetar uma dependência externa na classe, torna-se mais fácil criar objetos mocks (objetos simulados) para testar a classe em questão. No caso da classe BuscaInimigo, é possível criar um objeto mock para a classe InimigoService e injetá-lo no construtor da classe BuscaInimigo, permitindo que sejam criados testes isolados e independentes para a classe, sem depender de recursos externos.

**9) Qual o objetivo de utilizar Interface na implementação de objetos mock?**

R- O objetivo de utilizar uma interface na implementação de objetos mock é permitir a criação de objetos que se comportam como objetos reais, mas com comportamentos e funcionalidades específicas para testes. Ao criar uma interface para uma classe, é possível definir um conjunto de comportamentos que a classe deve ter. Isso pode ser útil na criação de objetos mock, pois permite que o mock implemente apenas o conjunto de comportamentos necessários para o teste. Então, é possível criar um objeto mock que se comporte como um objeto real, mas que tenha comportamentos específicos para testes, como por exemplo retornar valores pré-definidos ou simular comportamentos de erro.

**10) Utilizamos um sistema de controle de versões para resolvermos, basicamente, dois problemas. Comente, brevemente, sobre esses problemas.**

R- Manter a versão/controlar mudanças: Permite que várias pessoas trabalhem em um mesmo arquivo ou projeto ao mesmo tempo, sem perder as alterações feitas por outros colaboradores. O sistema de controle de versões rastreia quem fez o que, quando e por que, tornando possível visualizar o histórico de alterações e revertê-las caso necessário.

Compartilhar documentos: Facilita a colaboração entre os membros da equipe, pois cada um pode trabalhar em sua própria cópia do arquivo ou projeto e, posteriormente, mesclar todas as alterações em uma versão final. Isso minimiza conflitos de edição e garante que todos tenham acesso às mesmas informações atualizadas.

Um Sistema de Controle de Versões oferece uma solução para os problemas anteriores.

- Repositório (compartilhado) para armazenar a versão mais recente do código.

- Permite recuperar versões mais antigas (versionamento).

**11) Dado o tamanho e a complexidade de produtos de software, com todas as suas dependências, não é suficiente apenas compilar e disponibilizar o executável. Para isso é necessário automatizar o processo de build. Descreva com suas palavras o que você entende por automatização do build. O que está acontecendo? Quais etapas, normalmente, estão presentes e o que elas fazem?**

R- Automação de build é um processo essencial no desenvolvimento de software que envolve a utilização de ferramentas e scripts para compilar, testar, empacotar e distribuir o software de forma mais eficiente e confiável. Pode-se dizer, resumidamente, que é o processo de geração do software como produto.

O processo de automação de build envolve, geralmente, as seguintes etapas:

Gerenciamento de dependências: Envolve a utilização de gerenciadores de pacotes para baixar as dependências necessárias para o projeto. As dependências são gerenciadas por meio de arquivos de configuração, como o arquivo "pom.xml" no caso do Maven.

Compilação: É a etapa em que o código-fonte é compilado em um executável.

Testes: Após a compilação, é importante testar o software para garantir que ele esteja funcionando corretamente. Isso envolve a execução de testes automatizados que foram criados como parte do processo de desenvolvimento.

Empacotamento: Depois que o software foi compilado e testado, é hora de empacotá-lo em um formato que possa ser distribuído. Isso geralmente envolve a criação de um arquivo de instalação ou um arquivo executável.

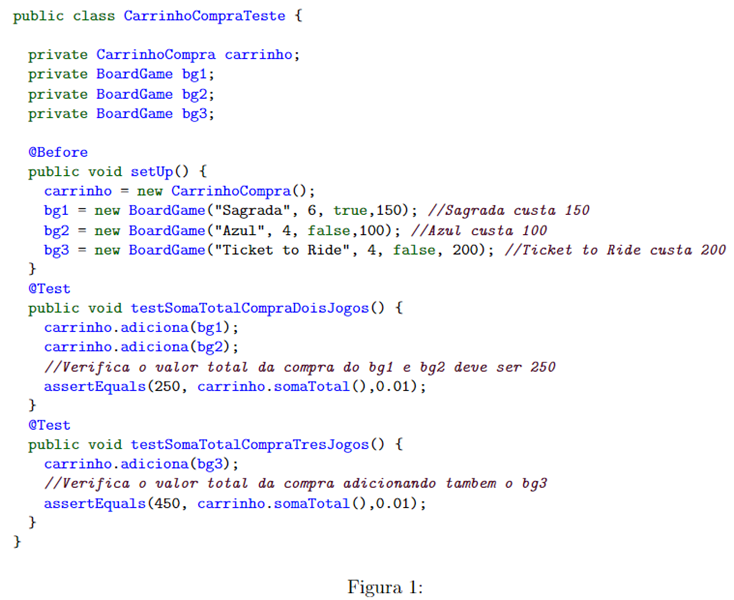
Distribuição: Por fim, o pacote é distribuído aos usuários ou aos servidores que o executarão.

**12) Existe uma característica fundamental para implementação de objetos mock manualmente, qual é essa característica?**

R- A característica fundamental para escrever um código de objetos mock manualmente é ter um entendimento profundo das interfaces ou contratos que esses objetos mock devem seguir. É necessário saber quais métodos e propriedades o objeto real deve fornecer, a fim de que o objeto mock possa imitá-lo de forma precisa e confiável.

Também é necessário, para implementação de objetos mock manualmente, fazer a injeção de dependências e um desacoplamento na classe, ou seja, adequar a classe para ser testável.

**13) Considere o código na Figura 1. É uma classe de teste para a classe CarrinhoCompra. Existem dois testes. Um para validar a compra de dois jogos, e um para validar a adição de um terceiro jogo. Existe algum erro? Se sim, explique o erro e o que precisa ser feito para que o teste fique adequado.**



R- Sim, existe um erro.

Testes unitários são independentes, um teste não pode depender de outros testes. Então, no testSomaTotalCompraTresJogos foi adicionado ao carrinho somente o “bg3”. Logo, o valor total da compra não será 450, pois os valores de “bg1” e “bg2” não fazem parte da somaTotal.