AGENDA 9

TESTES DE SOFTWARE

GEEaD - Grupo de Estudos de Educação a Distância

Centro de Educação Tecnológica Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO

EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO

CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

TESTES DE SOFTWARE

Expediente

Autor:

José Mendes da Silva Neto

Revisão Técnica:

Eliana Cristina Nogueira Barion

Revisão Gramatical:

Juçara Maria Montenegro Simonsen Santos

Editoração e Diagramação:

Flávio Biazim

São Paulo – SP, 2021

MERGULHANDO NO TEMA...

Na agenda anterior, falamos sobre qualidade de software, cujo objetivo é garantir que tanto o processo de desenvolvimento quanto o produto de software atinjam os níveis de qualidade especificados, para isso, é necessário levar em consideração o **VV&T** (Validação, Verificação e Teste):

- **Validação**: Estamos construindo o produto certo?

  A validação consiste em assegurar que o produto final corresponda aos requisitos do usuário;

- **Verificação**: Estamos construindo corretamente o produto?

  A verificação consiste em assegurar consistência, completude e corretude[1] do produto em cada fase e entre fases consecutivas do ciclo de vida do software.

- **Teste**: examina o comportamento do produto por meio da sua execução.

O teste de software é um controle de qualidade que pode envolver etapas desde a escolha das condições em que a aplicação será colocada à prova até a simulação de uso real dela e o desenvolvimento de relatórios sobre os resultados obtidos.

O propósito é verificar se o produto corresponde às funcionalidades esperadas no desenvolvimento e às necessidades dos usuários.

Qual a importância dos testes de software?

Ajuda que o seu time encontre pequenos erros que atrapalham a emissão de documentos fiscais e bugs que levem à perda de desenvolvimentos avançados.

(https://blog.tecnospeed.com.br/teste-de-software/)

O teste de software é um processo de checagem aplicado a programas de computador em diversas fases do desenvolvimento. Trata-se de uma etapa essencial para que o produto final seja entregue ao cliente funcionando dentro das expectativas. Contudo, há diversos tipos de testes que podem ser realizados para avaliar diferentes aspectos.

[1] Verificar se o software funciona como previsto.

Para elucidar melhor a sua importância, vamos tomar como exemplo uma empresa que presta serviços de missão crítica — ou seja, que não podem sofrer qualquer tipo de interrupção, como, por exemplo, controle de tráfego aéreo.

Nesse caso, estabilidade e resiliência estão entre os principais requisitos e, logicamente, não podem ser garantidos sem que testes rigorosos sejam feitos.

Mesmo o desenvolvimento de software voltado para entretenimento, como games e plataformas de conteúdo, requer testes para evitar que a aplicação feche repentinamente, apresente bugs ou retorne notificações de erro — o que geraria frustrações por parte dos usuários.

QTS – Qualidade e Teste de Software

Outros exemplos:

Um exemplo de teste de software que faltou planejamento: a instabilidade no acesso ao site para compra de ingressos no final do campeonato Paulista entre São Paulo e Corinthians em 2019.

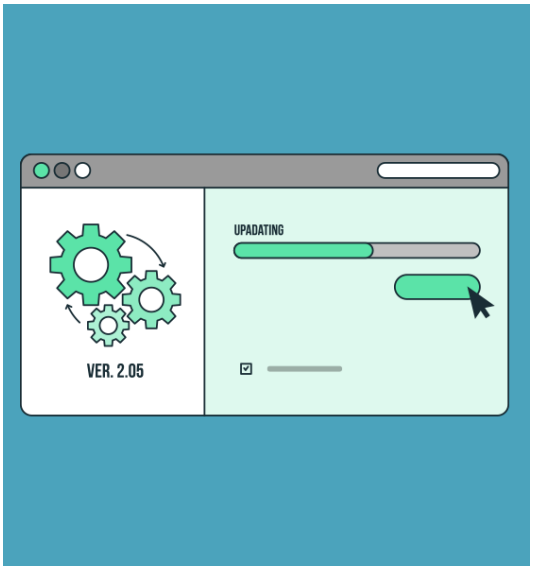
O mesmo aconteceu na busca por ingressos para a final da Copa do Brasil no mesmo ano, um jogo entre Inter e Athletico-PR.

(https://blog.tecnospeed.com.br/teste-de-software/)

Conhecendo a importância dos testes de software, confira agora quais são os principais tipos de teste de software, de acordo com a QTS – Qualidade e Teste de Software:

TESTE DE USABILIDADE

O teste de usabilidade é de suma importância para avaliar a qualidade do software no quesito experiência do usuário. A finalidade é compreender o quão intuitivo, compreensível e inteligível é a interface do programa para o usuário final.



Nesse caso, é fundamental a compreensão de como o software será utilizado. Por exemplo: nem sempre a aplicação é executada em primeiro plano, o que leva a equipe a construir uma interface que permaneça agradável quando redimensionada.

Além disso, como os perfis de usuário variam a cada projeto, o teste de usabilidade é realizado com esses grupos de pessoas, de modo que elas opinem acerca da facilidade de uso, das questões visuais (cores, fontes, poluição visual etc.), entre outros detalhes que prejudicam a experiência de uso.

TESTES FUNCIONAIS

O teste funcional consiste em uma série de testes (técnicas), cujo objetivo é atestar se a aplicação é capaz de desempenhar as funções que se propõe a fazer. As técnicas mais comuns englobadas pelo teste funcional são os testes denominados caixa-branca e caixa-preta.

O teste de caixa-branca tem como foco a análise do comportamento interno do software, ou seja, o seu código fonte. Já o teste de caixa-preta é feito em cima das funções que devem ser desempenhadas pelo programa.

TESTE DE INTEGRAÇÃO

Entre os principais tipos de teste de software está o teste de integração. A finalidade dele é analisar o comportamento do software quando interage com outras aplicações ou processos. Há diversas situações em que os testes de integração se mostram úteis — quando o software se comunica com um banco de dados ou servidor que estabelece a conexão dele com a internet, por exemplo.

A ideia é compreender se o processo em questão gera alguma falha/instabilidade. O mesmo vale para aplicações a serem utilizadas em cooperação pelo usuário final.

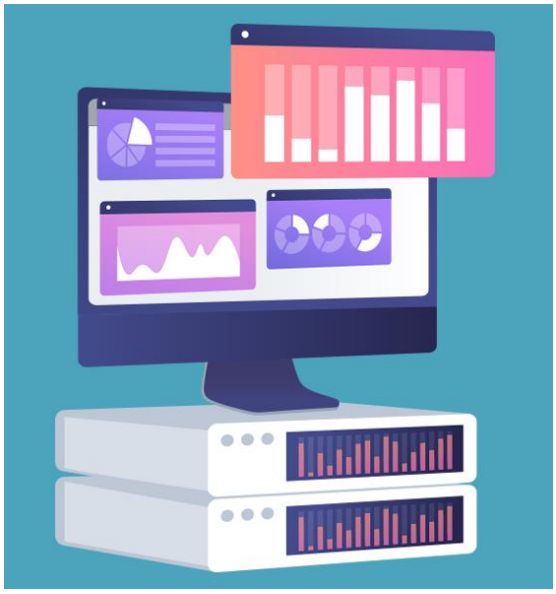
TESTES DE PERFORMANCE

Os testes de performance são uma série de análises voltadas para o desempenho do software mediante várias situações. A partir dos diagnósticos, a equipe é capaz de compreender os limites do programa sob diversas condições.

Para constatar a qualidade da aplicação, ela é submetida a avaliações que simulam eventos e situações previsíveis de acordo com a rotina do cliente, ou seja, testes de carga, estresse e estabilidade. Quaisquer falhas detectadas durante o teste são corrigidas pela equipe precisa e cirurgicamente.

TESTE DE CARGA E ESTRESSE

Você sabia que o software, assim como o hardware, pode apresentar problemas quando recebe cargas altas de processos e requisições? Isso acontece justamente para mensurar se as condições nas quais ele será submetido não vão comprometer o seu desempenho. Contudo, imprevisivelmente, a sobrecarga pode acontecer durante a utilização, gerando o que chamamos “estresse”. Como descobrir o nível de tensão capaz de levar o software a parar de funcionar? Nesse caso, é executado o teste de estresse, que consiste em simular eventos de carga excessiva, forçando o software ao extremo.



TESTE DE ESTABILIDADE

A partir do momento em que se planeja construir um software para o cliente, é necessário que o produto final apresente certa estabilidade, condizente com a carga de trabalho a ser suportada diariamente. O ideal, evidentemente, é que o software não sofra perda de performance depois de determinado tempo de uso.

TESTE DE REGRESSÃO

Como o próprio nome diz, o teste de regressão é uma metodologia usada, entre outras coisas, para evitar a recorrência de um erro. Um exemplo comum de sua aplicação é quando o programador modifica o código — seja para eliminar um bug, seja para acrescentar funcionalidades — e procura identificar falhas até então inexistentes.

Esse tipo de comprometimento pode acontecer em função de problemas previamente corrigidos em uma versão anterior. Sabe quando um sistema passa por um update e, assim que as atualizações são instaladas, fica instável? Pois bem, o teste de regressão é essencial para impedir esse tipo de situação.

TESTE DE SEGURANÇA

Por sua vez, o teste de segurança é um dos mais importantes da lista, sobretudo quando falamos em software corporativo. Afinal, a proteção dos dados é imprescindível a toda e qualquer empresa que armazena informações no ambiente virtual — que é repleto de ameaças.

Esse teste é realizado por uma equipe especializada em Segurança da Informação, incumbida de avaliar se há brechas de segurança a partir de procedimentos, tais como análise de vulnerabilidade, coleta de informações e violação de senha.

Dessa maneira, o produto final só é entregue ao cliente quando os requisitos de segurança são devidamente preenchidos, garantindo à empresa que suas informações ficarão protegidas contra invasão cibernética.

TDD - TEST DRIVEN DEVELOPMENT / DESENVOLVIMENTO ORIENTADO A TESTE

O TDD é parte da metodologia XP (Extremme Programming) e também utilizado em diversas outras metodologias, além de poder ser utilizada livremente.

O que você tem a perder utilizando o TDD?

Bem, segundo Freeman et al, você não tem nada a perder, a não ser os seus bugs.

O TDD transforma o desenvolvimento, pois deve-se primeiro escrever os testes, antes de implementar o sistema. Os testes são utilizados para facilitar no entendimento do projeto, segundo Freeman os testes são usados para clarear a ideia em relação ao que se deseja em relação ao código. Segundo Kent Beck apud Freeman “Finalmente, consegui separar o projeto lógico do físico. Sempre me disseram para fazer isso, mas nunca ninguém tinha explicado como”, o TDD é a forma de se fazer isso. A criação de teste unitários ou de componentes é parte crucial para o TDD.

Segundo Presmann, “Os componentes individuais são testados para garantir que operem corretamente. Cada componente é testado independentemente, sem os outros componentes de sistema. Os componentes podem ser entidades simples, tais como funções ou classes de objetos, ou podem ser grupos coerentes dessas entidades”.

Mas não é só o teste unitário que vai trazer o sucesso à aplicação, é necessário testar o sistema como um todo, que segundo Sommerville (2010), “Os componentes são integrados para compor o sistema. Esse processo está relacionado com a busca de erros que resultam das interações não previstas entre os componentes”.

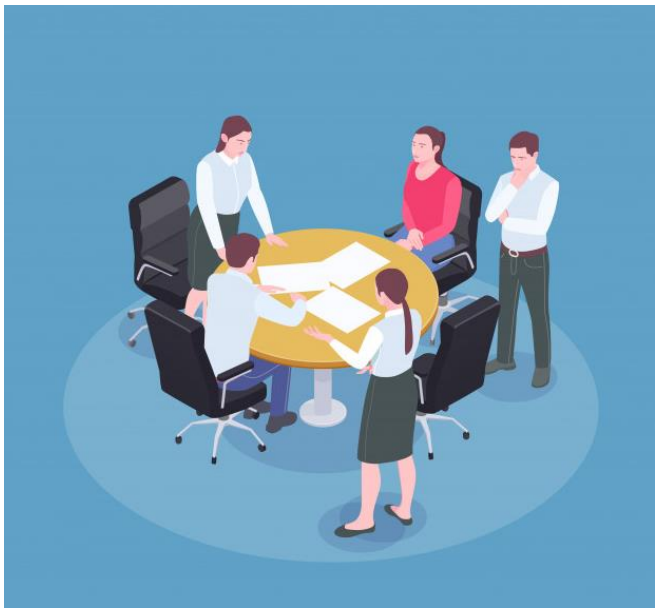
Um sistema é um conjunto de unidades integradas, por este motivo é importante os testes unitários para ver se no micromundo tudo funciona, mas também temos de testar a integração, ou seja, ao integrar dois ou mais componentes, devemos realizar testes para verificar se a integração funciona. Voltemos ao exemplo do carro, não adianta eu testar a roda, a porta, o volante, mas não testar o carro completo. Erros podem ocorrer, justamente no processo de montagem/integração de componentes.

(Fonte: https://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151. Acesso em 12/11/2020.)

O que se ganha com isso?

- Qualidade no código;
- Código mais simples e organizado;
- Documentação mais clara (componentização).

Vamos imaginar o seguinte cenário:



(Fonte: Freepik. Acesso em 12/11/2020).

Imagine que você esteja numa reunião de planejamento com o Scrum Master, Product Onwer e com o time de desenvolvedores. Você e a equipe alinham as funcionalidades do sistema com o programador e planejam as sprints, conforme foi combinado.

Então, o programador começa a pensar em desenvolver um teste (programa) para testar o programa que ele ainda irá desenvolver.

Achou estranho? Pois é, mas é assim mesmo!

Nesse programa de teste, ele fará a prova/avaliação para ter a certeza de que as rotinas que ele desenvolverá no sistema realmente irão funcionar. Então, ele faz um teste de comportamento para as rotinas e componentes que ele desenvolverá. Assim, o programador já está premeditando o que pode acontecer de problema com o programa que ele desenvolverá e o programa teste o ajudará a evitar possíveis problemas e ainda auxilia no planejamento das ações.

Vamos ao exemplo para entender melhor:

Imagine que o programador tenha saído da reunião de planejamento de sprints com o seguinte requisito:

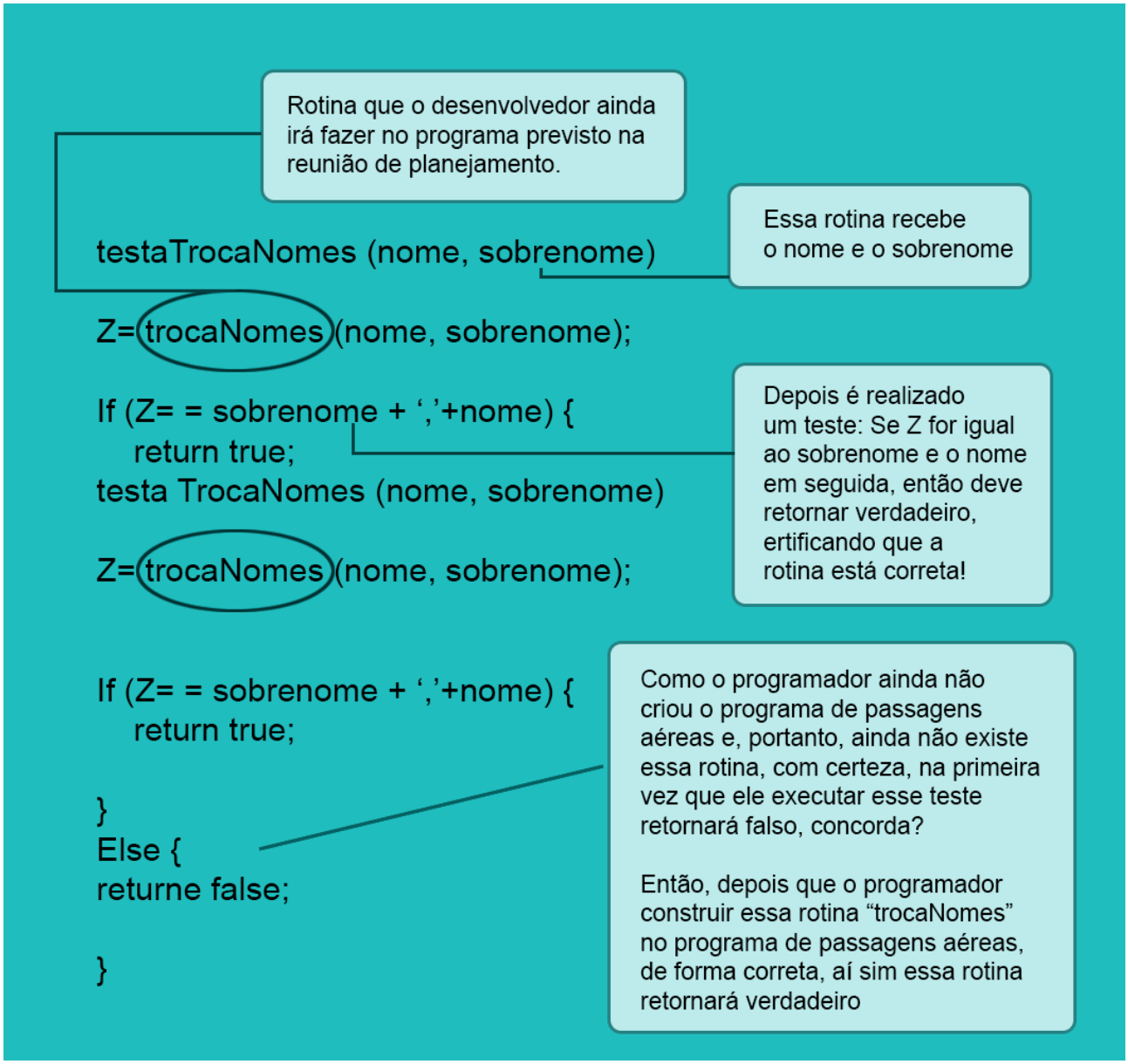
Dado um nome e sobrenome, o sistema deverá inverter a ordem, colocando o sobrenome do usuário e depois o nome, assim como acontece com os passaportes, citação de autores em trabalhos científicos e em demais situações.

Exemplo:

Nome: João
Sobrenome: Silva
Retorno: Silva, João.

Para isso, o programador irá criar um programa de teste com o nome “trocaNomes”.

Veja como ficaria a rotina de teste para testar a função que o programador ainda irá construir para trocar a ordem do nome e sobrenome:



Exemplo de retorno: Silva, João

Entendeu? Pronto, acabamos de conhecer como funciona um teste unitário do TDD – fazer uma rotina ou classe do método que o programador ainda irá construir.

Então, observamos que:

- A rotina falhou na primeira execução porque ainda não existia a codificação;
- A rotina de teste passou a funcionar certinho porque o código foi programado corretamente;
- Depois o programador codificou o futuro programa.Dessa forma, o programador consegue testar o funcionamento da rotina do seu sistema, por meio do programa de teste.

Além de tudo, os programas testes auxiliam muito na documentação do sistema, pois nele o programador está expressando exatamente aquilo que deverá ser implementado e os resultados almejados para o sistema.

Então, vamos listar as vantagens em adotar o TDD:

- **Design e Manutenibilidade**: quando o TDD guia a implementação de um software, naturalmente esse software terá um desenho menos complexo e com características altamente modularizáveis porque será forçado a escrever códigos limpos e desacoplados, evitando possíveis erros antes que chegue ao usuário final.

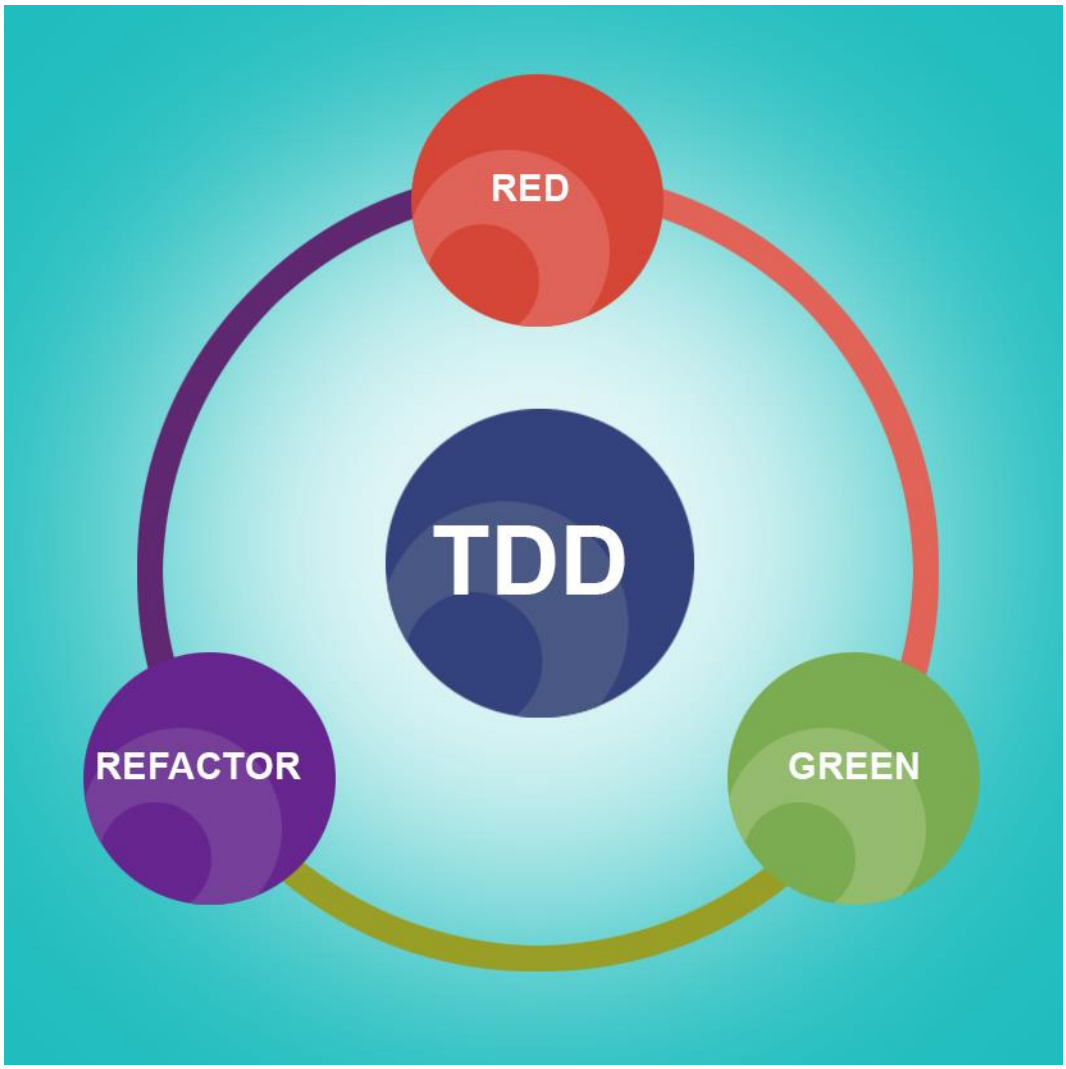
- **Confiança**: rotinas já testadas. Confiança para fazer alterações no código sem temer que possa acontecer algum problema.

-**Documentação**: as descrições que são escritas nos testes documentam os requisitos que foram implementados.

- **Redução de bugs em produção**: estudos comprovam que a prática do TDD, além dos benefícios mencionados, impacta de 40 a 80% de redução dos bugs em produção.

Entendendo que o tdd é uma prática, é preciso assimilar todos os conceitos e regras

Assim, fique atento ao cliclo de um TDD:



O processo de desenvolvimento com TDD é sempre: **Red**, **Green** e **Refactor**.

Red: Como vimos no exemplo do programa de passagens aéreas, o primeiro passo é escrever um teste que falha. Sim! É para falhar mesmo porque o código da sua funcionalidade ainda nem existe! Escrevemos o programa teste primeiro para depois escrever realmente as rotinas do sistema que se pretende desenvolver.

Depois execute o teste e acompanhe a falha. Pode ser um código falso mesmo que retorne o erro.

Green: desenvolva primeiro a solução mais simples, depois teste novamente e se o teste não passar – der errado, retorne à etapa anterior.

Refatore a sua funcionalidade e agora escreva-a por completo. Nessa etapa, elimine redundância e acoplamentos deixando o design do código mais limpo e legível.

E pronto! Passe para a próxima funcionalidade iniciando um novo teste!

TESTES DE SOFTWARE COM O METODOLOGIA SCRUM NA LINHA DE FRENTE

Renata é analista de testes de software e está iniciando um projeto de desenvolvimento de sistemas em uma nova empresa que adota a metodologia Scrum. Na semana passada ela participou de uma reunião de planejamento de Sprint, onde os membros do time de Scrum se reuniram com o Product Owner Agnaldo, gerente de pós venda da empresa cliente que solicitou o software. Nesse planejamento, os membros da equipe discutiram sobre os requisitos desejados e em conjunto definiram quais características da aplicação serão desenvolvidas naquele Sprint, bem como a duração do Sprint.

A partir dessas definições o projeto começou a ser executado e todos os membros iniciaram sua atuação e Renata percebeu que dentro de um time scrum, todos são responsáveis por tudo, ou seja, os papéis são compartilhados, portanto a responsabilidade não é apenas dos analistas de testes, mas de todo o time. No Scrum, todos são *testers* e todos são desenvolvedores.

Em oposição à metodologia tradicional, onde as especificações e requisitos dirigem os testes, no Scrum os testes são aplicados com maior ênfase nas partes que mais geram valor para o produto.

Pensando nisso, o time precisou ser muito mais crítico, visto que os testes não são desenvolvidos de maneira desordenada, mas focando no que é importante.

Desse modo, tivemos que difundir o conhecimento sobre o produto entre todos, para que o time tivesse condições de avaliar os principais pontos de impacto no software.

Portanto, ao nos comunicarmos, tornamos todas as etapas, inclusive os testes, transparentes.

No fim das contas, o ambiente que era, dependendo da situação, desconfortável devido

Às intrigas entre os times (*Testadores vs. Desenvolvedores*), com Scrum, tudo passou a ser colaborativo, ou seja, todos sentiam-se confortáveis para sugerir uns aos outros o que testar e de que forma testar.

A equipe, segmentou todo o projeto em pequenas *features/tasks*, aumentando a compreensão sobre a utilidade e prioridade dos itens, visto que a divisão em pequenos pedaços instigou a criticar sobre a contribuição de cada um para o todo.

“Não pense em todo o produto! ”Vá por partes!”

Nesse sentido, o desenvolvimento e os testes, foram direcionados para aquilo que realmente agrega valor ao produto.

Quando o time de desenvolvimento da Renata dividiu o que precisa ser testado em pequenas parcelas, conseguiram enxergar o quanto aquilo estava somando ao produto.

Assim, a decisão do time sobre o que deveria ser priorizado tornou-se mais assertiva, já que tiveram a ciência de que aquilo que foi e que seria testado contribuiria para o projeto.

(Adaptado de https://blog.geekhunter.com.br/como-aplicar-scrum-em-um-projeto-de-teste-de-software/. Acesso em 13/11/2020.)

E QUAIS FERRAMENTAS UTILIZAR PARA IMPLEMENTAR O PROCESSO DE TDD AO DESENVOLVIMENTO?

Não existe uma perspectiva das melhores ou piores ferramentas e/ou linguagens. Você pode usar qualquer linguagem de programação que você conhece para fazer os programas testes, escolhendo a opção mais próxima da sua realidade de conhecimento e habilidades técnicas dentro da sua equipe. No entanto, existem diversas ferramentas que nos fazem ganhar mais tempo.

Veja alguns exemplos para diferentes linguagens são:

- JUnit: O JUnit é um framework de teste para Java, que permite a criação de testes unitários. Além disso, está disponível como plug-in para os mais diversos IDE'S como Eclipse, Netbeans etc.
- TesteNG: Outra ferramenta de teste unitária, disponível para Java;
- PHPUnit: Framework XUnit para teste unitário em PHP, também é possível integrar aos IDE's assim como o JUnit;

- SimpleTest: Outra ferramenta para realização de teste para PHP. Além de possibilitar os testes unitários, é possível realizar MOCKS e outros testes;
- NUnit: Framework de teste no molde XUnit para a plataforma .NET;
- Jasmine: Framework para teste unitário de JavaScript;
- CUnit: Ferramenta para os testes unitários disponível para Linguagem C;
- PyUnit: Framework Xunit para testes na linguagem Python.

(Fonte: https://www.devmedia.com.br/ferramentas-de-suporte-ao-teste-de-software/28642. Acesso em 12/11/2020.)

As ferramentas de gerenciamento podem ser utilizadas em todas as etapas do ciclo de vida do teste e têm como grande vantagem a centralização de todas as informações relacionadas à evolução dos testes realizados no software, permitindo que informações sejam coletadas e acompanhadas, possibilitando uma visão mais ampla para a tomada de decisão, fornecendo artefatos de acordo com os interesses e expectativas.

Para aprofundar o seu conhecimento em testes de software e implementar o processo de TDD, escolha uma ferramenta que utilize linguagens com as quais você já está familiarizado. A ferramenta JUnit pode ser interessante para você que já conhece a linguagem Java e que utiliza IDE'S Eclipse e Netbeans, por exemplo.

A ferramenta **PHPUnit** também pode ser interessante para você que também tem familiaridade com o PHP.

Para se aprofundar seus conhecimentos nessas ferramentas, utilize os links acima e bons estudos!

Fontes Imagéticas:

Imagens: freepik.com, Pixabay , Arquivo do GEEaD