Poderosa técnica para salvar seu emprego. Vamos falar sobre Teste Unitário

Salve padawan, inspirado na Aceleração da GFT de QA e seguindo meu cronograma de roteiro em mainframe, neste finzinho de domingo no início de primavera, vou contar a respeito de projetos que participei e que as atribuições era gerar código, documentação e evidências de testes.

Era uma época em que não existiam metodologias ágeis, então seguíamos rotinas de trabalho de acordo com o cliente e o instinto adquirido pela experiência, afinal após queimarmos o bolo algumas vezes, sabemos qual a temperatura certa do forno, evitando cair na maldição do Dr. Ivon Safe e sempre olhando para onde a agulha da seringa esta apontada, evitando que seja o nosso bumbum na mira.

Venho de uma época em que a metodologia Ágil era apenas discussão acadêmica, papers e palestras em nichos restritos, no mundo mainframe ainda estávamos na época copy-paste dos testes unitários e validações de programas.

O que é Teste Unitário.

Em nossa atividade profissional, quando fazemos modificações em software em projetos evolutivos, precisamos garantir que nossas alterações cumprem aquilo que foi requisitado, que nosso trabalho não afete o desempenho do aplicativo.

Pense nisso o teste unitário é nosso capacete ou cinto de segurança, pois serve para validar nosso trabalhando, antes de entregarmos para o Analista de Negócio, garantindo que não ocorrera abends ou falhas durante a execução em Produção. Por isso ao planejarmos as alterações e antes de codificar, criamos uma lista de validação, para testarmos após quando concluirmos nossas alterações.

Então respondendo a dúvida inicial: Teste Unitário é o menor teste efetuado em um software, sua execução e sucesso garante que o software está pronto para as próximas etapas de validação.

Como fazer um plano de testes unitários.

Primeiro deve-se avaliar o impacto da alteração, quanto maior, mais necessário são os testes, em mainframe Cobol, os testes unitários são divididos em pré e pós alteração, primeiro temos que garantir que o programa funciona e esta não tem nenhum erro oculto.

Normalmente avaliamos os IF, Performs, Call e etc, fazendo um mapa do caminho critico, sabendo qual o resultado esperado se tudo correr bem, depois criar um caminho alternativo com as bifurcações que podem ocorrer.

Com este esboço podemos criar o Plano, utilizando uma planilha do MS Excel e com ela numerando cada caso a ser testado, indicando uma breve explicação e qual o resultado esperado, ao concluir o teste, colocamos evidencias no documento, mas antes de avançarmos, vamos fazer algumas definições.

Defeito - ocorre quando usamos comandos sem domina-lo perfeitamente e ocorrem falhas inesperadas.

Erro - é a manifestação do defeito, ocorre quando se espera uma reação e ocorre outra, um valor anormal etc

Falha - é o comportamento diferente do esperado, uma falha pode ocorrer por diversos erros e curiosamente alguns erros podem nunca gerar uma falha.

Anomalia – é um ato que acontece na execução do programa, fora dos atos normais e não estava planejado ou previsto, podendo tanto ser um erro/falha, como um acontecimento casuístico que não impacta ou prejudica.

Com essas definições em mente, temos que planejar dois tipos de bateria de testes:

o caminho feliz: o programa deve ir do começo ao fim sem erro;

o caminho tenebroso: o programa deve falhar na sua execução:

o caminho incerto: o programa deve estar preparado para movimentos aleatórios, únicos que o usuário final poderá fazer, acredite em mim, isso é mais comum do que imagina nossa vã filosofia.

ABEND ou falha catastrófica

Os mais antigos no mundo da computação usam a expressão ABEND, um neologismo que surgiu da concatenação de duas palavras. ABnormal End, final anormal, ocorre quando o programa termina inesperada devido há uma falhar ou erro na execução.

Ao codificar tenha em mente que devemos ter estruturas internas para capturar o máximo de erros possíveis, planejando uma saída programada e sem surpresas, mas ao mesmo tempo temos que planejar erros inesperados e ter um estrutura para capturar este tipo de erro.

Num artigo futuro irei comentar sobre qualidade, existe uma norma técnica que define a qualidade do software: **ISO/IEC 9126.(NBR 13596), que serve de instrumento para planejar testes e garantir a qualidade do software.**

**Evite a maldição do DR. Ivon Safe.**

**Antes de sair codificando, leia a Documentação de Requisitos de Sistema (DRS), estude o funcionamento atual do software, executando, avaliando os inputs e outputs, sabendo qual é sua dependência.**

**E tendo duvidas, pergunte, não tenha vergonha em perguntar, pior e assumir algo e sua assunção for errada e gerar danos, neste caso a agulha da seringa será fatal.**

Como ter um bom plano de teste?

1. Leia o documento de requisitos do sistema
2. Elabore cenários de testes de acordo com o DRS
3. De acordo com os cenários de testes crie casos de teste.
4. Execute, solucione os erros encontrados, reveja o DRS e recomece o ciclo.

No decorrer dos testes colete evidencias, chamada normalmente de corte e costura, pois normalmente usamos o print-screen, control+c e control+v no documento das evidencias, recordando.se de identificar o caso de teste e como chegou ao resultado apresentado.

Tipos de testes possíveis

Teste de Caixa Preta ou Funcional;

Teste de Caixa Branca ou Estrutural;

Teste de Caixa Cinza;

Análise baseada em erros;

Em um próximo artigo iremos explorar cada um dos tópicos abordados em testes unitário, lembre-se que cada alteração afetara seu programa, tente prever possíveis pontos de colapso, faça analisando cada parada que o programa faz, por isso o teste de mesa é fundamental.

O que é teste de mesa?

Nos primórdios da computação era uma técnica utilizada para debugar programas, funciona da seguinte maneira, imprimimos o código fonte expandindo, marcamos todos os pontos que geram paradas no processamento procedural. Isto é IF, Whiles, Performs, Calls,Evaluate (no bom e velho Cobol), converta para a sua linguagem favorita.

Com a massa de teste iniciamos o processamento mental, seguindo o fluxo e verificando o que acontece a seguir, atualmente esse processo foi aprimorado e coloca-se Display, indicando a passagem ou não em determinado ponto.

E um processo perigoso, mas graças ao Github, Changeman e Endevour, essa tarefa ficou bem mais fácil, pois esses softwares mapeiam as mudança, indicando o que mudou entre as guardas, permitindo saber que não ficou nenhum rabicho de código, situação que possa gerar danos e confusões.

Esta é uma das melhores funções de programas de controle de versão, indicam as últimas mudanças, e auxiliam na manutenção / debugação de software.

Ambientes de Teste.

Na maioria das instalações existem 3 tipos de ambientes, que são usados para o teste:

Ambiente de Desenvolvimento: neste ambiente os dados não muito fiáveis, devido a quantidade de programadores manipulando as informações e fazendo teste atrás de testes, funciona para pegar os erros primários, um pouco enviesada e sem respeitar as regras de negócio, sendo usado predominante em Teste Unitários e com muito sufoco nos Testes de Integração do Sistema.

Ambiente de Aceite neste ambiente os dados são mais fiáveis, respeitando as regras do negócio na medida do possível, sendo que executa processamentos diários para manter a integridade dos dados e de tempos em tempos, recebe informações copiadas do ambiente de produção, sendo utilizados em Testes de Aceite e em alguns casos nos Testes de Integração do Sistema.

Ambiente de Produção neste ambiente é onde a magia ocorre, são os dados reais, completamente fiáveis e razão de ser de nossa atividade como programadores, nada de ruim deve acontecer aqui, as consequências são imprevisíveis, cabeças rolaram se algo falhar aqui.

Em algumas situações críticas usamos o Ambiente de Produção para executar testes pilotos, onde o software a ser implantado é executado com grupos restritos de usuários, para efetuar testes antes de entrar em execução massiva.

Use Comentários

Quando estiver codificando, lembre-se sempre que outras pessoas irão trabalhar com seu código, coisas obvias para nós, que lemos a documentação inicial e nos dedicamos a codifica-lo, outros não terão tanta sorte assim, por isso deixar comentários informativos ajudam muito na hora da mão na massa.

Documentação de Evidencia dos Testes

Tenha em mente que dois terços do tempo de seu trabalho será em execução de testes e documentação. Por isso procure criar uma lista abrangente de testes, uma dica para uma massa de teste segura:

- Explore os limites do negócio;

- Explore valores nulos;

- Explore situações anômalas;

- Explore falhas operacionais: parâmetros incorretos, falha de comunicação, problemas incomuns já documentados;

- Faça um passeio aleatório, misturando situações;

Lei Geral de Proteção de Dados Pessoais

Quando estiver fazendo testes e estiver usando dados copiados de produção, tenha muito cuidado com dados sensíveis, a LGPD está em vigor e as punições são severas, expor dados ou manipular estas informações sem cuidado adequados podem gerar graves problemas a instituição.

Conclusão

Como de costume, fizemos um passeio na área de desenvolvimento, falando sobre programação na vertente dos Testes Unitários, onde partilhei um pouco do conhecimento em testes.

O principal cuidado é planejar os testes, leia com atenção a documentação existente, conheça a Área de Negócio, conversando com o usuário e sentindo as necessidades e as dores existentes.

Tenha atenção aos dados copiados e com a criação dos testes, pois um plano enviesado não encontrara erros e apenas gerara trabalho sem fim, espero sucesso em sua bateria de testes e que a força esteja com você.