APRESENTAÇÃO
DE APOIO

QUALIDADE E TESTE DE SOFTWARE

Ricardo Beck – Aula 01

PUCRS online uol edtech

Professores

RICARDO BECK
Professor Convidado

Possui um histórico profissional de mais de 20 anos de experiência em Qualidade de Software com certificação internacional em Teste de Software (CSTE), tendo trabalhado com um extenso portfólio de produtos multinacionais aplicado a estratégias e processos no estado da arte. Há 10 anos se dedica à gerência de projetos, atuando como Scrum Master/Product Owner e, mais recentemente, sendo gerente sênior de P&D na HP Brasil. Possui certificações em Gestão de Pessoas pela FGV-Rio, SCM, PO e Agile Coaching pela Scrum Alliance, além de Fotografia pela ESPM.

DANIEL CALLEGARI
Professor PUCRS

Doutor em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul. Possui Especialização em Gestão Empresarial (Sebrae / ANFE / Itália) e Certificações Microsoft e IBM. Associado da Sociedade Brasileira de Computação (SBC). Foi sócio-diretor de duas empresas de tecnologia e sempre equilibrou a atuação acadêmica com o mundo empresarial. Atualmente é professor Adjunto da PUCRS e Coordenador do Curso de Ciência de Dados e Inteligência Artificial. Tem ampla experiência na área da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: bancos de dados, engenharia de software, gerenciamento de projetos. Durante o doutorado, desenvolveu um algoritmo para reconfiguração dinâmica de projetos de software e realocação de recursos, que ganhou o prêmio de primeiro lugar em um congresso da área. Atualmente ministra disciplinas principalmente de engenharia de software, banco de dados e programação, além de coordenar equipes em cooperação com a Dell Computadores do Brasil.

Ementa da disciplina

Introdução aos conceitos de teste unitário, teste de integração, teste de UI. Introdução aos conceitos de garantia de qualidade de software. Estudo de métricas voltados ao controle de qualidade no desenvolvimento de software.

Bem-vindos

Nas próximas horas vamos conversar sobre diversos aspectos da qualidade de software e entraremos em conceitos de testes.

É um grande prazer poder compartilhar com vocês mais de 20 anos de experiência nessa área.

O teste de software é um exercício de disciplina.

ÍNDICE

Aula 01

- O que é qualidade?
- Projetos complexos
- Introdução aos conceitos de teste unitários
- Testes de integração
- Testes de interface com o usuário
- Testes de acessibilidade
- Testes de performance

ÍNDICE

- Testes de localização
- Introdução aos conceitos de garantia de qualidade
- Confiabilidade de software
- Estudo de métricas voltados ao controle de qualidade no processo de desenvolvimento
- Definição de Retorno de investimento

O que é qualidade?

- Qualidade é uma sensação, segundo o QAI, que um produto ou serviço atende a necessidade do cliente.
- Segundo o IEEE, é o grau de conformidade de um sistema, componente ou processo com seus respectivos requisitos.

O que é um projeto complexo?

- São projetos que demandam uma efetiva gestão de riscos no processo tendo como dependências internas e externas. Dependências como diferentes áreas de conhecimento e múltiplas organizações.

Ciclo de vida de projetos de software

Waterfall - Cascata
Projetos regulares com etapas definidas para:
Investigação, prototipação, desenvolvimento, testes e liberação em produção

Ciclo de vida ágil
Projetos interativos onde a cada etapa de tempo (sprint) ocorre uma entrega de valor.

Tipos de testes e suas características

Para que exista qualidade é necessário que exista repetibilidade no processo e consistência nos fluxos.
A segmentação de domínios específicos de teste tenta endereçar camadas direcionadas de qualidade.
Qualidade é caracterizada como uma “sensação” pelo Quality Assurance Institute, o usuário ou cliente sabe se um produto tem “qualidade” se esse produto, serviço ou artefato serve ao seu propósito.

Conceito de caixa transparente

O conceito de “caixa transparente” advém da possibilidade de conhecermos as partes “internas” do artefato a ser testado.

As estratégias que orbitam o domínio da “caixa transparente” aproveitam o conhecimento de como o código foi criado, sua tecnologia e seu acoplamento.

Também são conhecidas expressões como “caixa branca, caixa preta e caixa cinza”

Conceito de unidade para teste

- Uma “unidade” pode ser considerada a menor parte indivisível do código.
- Em geral temos:
- DD path - Caminho decisão à decisão
- DU path - Declaração e Uso das variáveis

```
...
if (a == b)
{
    b = b+1
}
if (b > a) {
   b = b -1
}
```

```
a = 0
b = 0
...
if (a ==
b) {
    b = b+1
}
if (b > a)
{
   b = b -1
}
```

Necessidade de isolamento entre componentes

As unidades devem ser isoladas entre si, de forma a garantir o mínimo de interferência externa quando os estímulos forem recebidos.

Para que sejam criados os níveis de isolamento é necessário se criar Stubs, chamados de esqueletos que sempre respondem a um valor conhecido quando a classe ou método depende de algum agente externo.

Os estímulos gerados devem ser rastreáveis e repetitivos.

Exemplos de níveis

Nível de método:

```
Método soma ( inteiro a, inteiro
b)
    Retorna a+b
```

Nível de método (testes):

```
Esperado= [2, 1. 0, -1, erro]
Esperado[0] = soma(1,1)
Esperado[1] = soma(0,1)
Esperado[2] = soma(0,0)
Esperado[3] = soma(1,-2)
Esperado[4] = soma(A,1)
```

Nível de classe:

```
Classe Instalacao{

Metodo publico instalar(caminho)
   validaRegistroApp()
   copiaArquivos(caminho)
   verificaInstalacao()

Metodo privado
validaRegistroApp()
    ....

Metodo privado
copiaArquivos(caminho)
    ...

Metodo privado
verificaInstalacao()
...

}
```

Nível de classe (testes):

```
Esperado=[Pass, Pass, Fail]
Esperado[0]=Instalacao.instalar("C:\temp")
Esperado[1]=Instalacao.instalar(".\temp")
Esperado[2]=Instalacao.instalar("@#$%")
```

Critérios ambientais

- HD cheio
- Usuário sem direitos de instalação
- Diretório protegido
- Diretório inexistente
- Arquivos somente para leitura

Estratégias de teste unitário

TDD = Test Driven Development

Cria-se o teste unitário antes de começar a codificação.

Nessa estratégia os testes devem sempre falhar inicialmente, após a falha se escreve o código para que o teste passe.

Estratégias de teste unitário

BDD = Behavior Driven Development
Entende-se a demanda da unidade para que sejam criados os testes. Podem ser baseados em “promessas” ou requisitos funcionais / não-funcionais.

Como um (pessoa ou perfil)
Eu gostaria de (funcionalidade)
Para que (objetivo ou benefício)

Critérios de aceitação:
Dado que (critério inicial)
Quando (evento que inicia a ação)
Então (resultados esperados)

Diferenças entre teoria e prática

No dia-a-dia cria-se o código e depois testa-se.

Poucos times conseguem ter a disciplina de criar os testes primeiro para verificarem a "falha" para depois implementar o código.

Ferramentas de verificação de código são boas referências para a saúde do código, assim como o entendimento do esforço faltante para complementação de testes.

A regra de ouro é 80% ou mais de cobertura de código.

Integração dos testes unitários no dia-a-dia

A integração dos testes unitários deve iniciar tão logo o projeto comece, pois muitas vezes o código feito para investigação e pesquisa é mantido para o produto final.

Os sistemas de build devem concentrar a execução dos testes unitários de forma a prover uma execução automática em diferentes níveis.

A cobertura de código deve ser analisada para fins de aprovação ou rejeição do PR (pull-request), no caso de uso de GitHub.

Ferramental para criação/execução e relatórios

Java - IDE Eclipse - JUnit - https://github.com/junit-team/junit5

.NET - Visual Studio - NUnit - https://github.com/nunit/nunit

JavaScript - Mocha - https://github.com/mochajs/mocha

Jest - https://github.com/facebook/jest

Criação e apresentação de relatórios:

Allure - https://github.com/allure-framework

Processos de Build

Os processos de build devem integrar a capacidade de validar branches progressivamente bem como executar os testes unitários de acordo com o regime requisitado.

Estratégia 1 - executar todos os testes unitários após cada commit. Essa forma provê uma grande confiabilidade no código a ser integrado, porém causa filas no aguardo para validação.

Estratégia 2 - executar os testes relativos ao código a ser inserido na branch principal. Essa estratégia prevê uma maior velocidade e pontualidade dos processos. Todavia, cria-se uma demanda de executar os testes completos ao final de um período de tempo (dia, semana ou sprint).

Processos de review e merge

Os processos de revisão para o time de desenvolvimento voltado à qualidade devem se ater aos padrões de código definidos pelo time e também a cobertura de testes unitários provida.

Muitas vezes quando postergados os testes unitários acabam por promover novas atividades futuras de retrabalho, conhecidas como dívidas técnicas.

A integração do código gerado com o branch principal só deve ser feita (aprovada) após verificações de padronagem de código, testes unitários e avaliações estáticas de código (como exemplo, Lint - https://en.wikipedia.org/wiki/Lint_(software))

Testes de integração

Os testes de integração avaliam o ecossistema no qual os diferentes métodos ou classes se integram compondo a solução.

As estratégias de teste de integração tendem a exercitar incrementalmente a união parte a parte das unidades.

Exemplo:
Em um teste de integração de uma calculadora, seriam feitos testes unitários para cada método (soma, subtração, divisão e multiplicação). A+B, A-B, A/B e A*B

No caso de integração, os testes devem exercitar o conjunto incremental dos métodos soma seguida de subtração e divisão. A+(B-A)*B+B/A

Estratégias de integração de componentes

Top-down - cria-se testes para a camada mais superior da solução. O grande problema dessa abordagem é a grande quantidade de variáveis para diagnóstico da causa raiz. Mas a complexidade dos testes é mais baixa.

Bottom-up - onde os métodos e classes vão sendo testados aos poucos mantendo partes isoladas. A desvantagem dessa abordagem é o consumo excessivo de tempo para execução e manutenção dos testes a cada interação de software.

Sanduíche - um misto das duas abordagens anteriores, onde partes mais críticas do sistema são testadas individualmente e partes mais estáveis são testadas em alto nível pela integração.

Ferramental para criação/execução e relatórios

Muitos sistemas de build já incluem opções de integração para auxiliar o processo.

Jenkins - Test Harness

Codeway - Test Pipeline - Usando Jest, JUnit e outros frameworks

Front-end / Back-end

Enquanto o Back-end é testado através de um conjunto de Rest APIs, e outras estruturas mantidas sem interface com o usuário, o Front-end é exercitado manipulando-se os componentes e controles visuais disponibilizados pela aplicação/solução ou serviço.

Testes de Interface com o usuário

São os testes realizados manipulando-se os controles disponíveis visualmente.

Principais pontos:
*** Verificação do alinhamento dos componentes visuais.**
*** Verificação dos padrões de ancoragem, padronização das cores, fontes e espaçamentos**
*** Verificação do vocabulário usado.**

Clock

Focus sessions

Timer

Alarm

Stopwatch

World clock

Sign in

Settings

Welcome to focus sessions

Achieve your goals and get more done with focus sessions. Tell us how much time you have, and we'll set up the rest. After each focus period, you'll get a short break. This will help you recharge and get ready for your next task.

Get started

Diferenças entre testes de front-end e testes de interface gráfica

- Validação de campo
- Tipos de arquivos
- Instalação
- Propriedades

- Número de passos para completar a ação
- Padronização do vocabulário
- Padronização do espaçamento
- Padronização da paleta de cores

Ferramental para criação/execução e relatórios

Mental maps

Word

Jira

Azure DevOps

Mapas mentais e identificação de problemas

Mapeamento de atividades e número de passos

Caso de Uso - Instalação de software
Mapeamento de atividades (Cenários de teste)

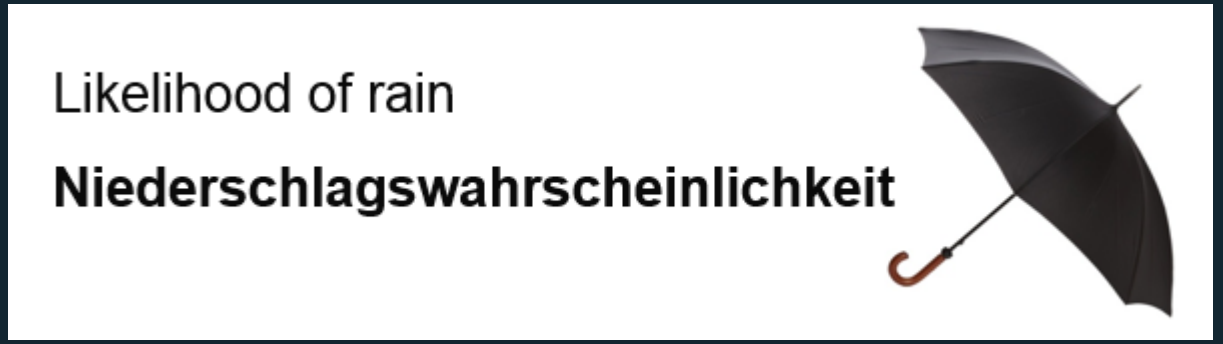
HD limpo

USB drive

Diretório compartilhado

Número de passos (Cenários de teste)
Instalação simples 10 passos
Instalação avançada 30 passos

Localização



Deve ser pensado no tempo de concepção da solução, pois as linguagens suportadas influenciarão diretamente no processo de criação da experiência do usuário.

Em tempo de design de produto é necessário o entendimento dos possíveis países que usarão solução. Em termos gerais, aplicações em nuvem devemos priorizar o Inglês para desenvolvimento pois não se sabe

As palavras, frases e textos usados devem ser adaptados ao contexto e de acordo com a localização diferentes expressões podem ser usadas.

Isso reflete no processo de qualificação pois em regra será feita uma avaliação por proximidade, ou seja se a solução deve suportar Inglês e Mandarim, provavelmente o executor de testes pode não conhecer Mandarim suficiente para identificar problemas de estrutura, mas pode comprar contra um Oráculo de teste.

Acessibilidade

Garante acesso universal à solução, usando atalhos de teclado, voz, alto-contraste e fluxos simples.

O processo de testes deve (sempre que necessário) avaliar diferentes perspectivas, como:
uso de teclado para completar ações,
*** Diferentes modos de tela (retrato, paisagem)**
*** Diferentes resoluções de tela**
*** Alto contraste**

Light e Dark temas

Muito usado nos smartphones, os temas claros e escuros

O processo de teste deve cobrir os dois modos, duplicando os fluxos

Ferramental para criação/execução e relatórios

Entre as alternativas correntes, o site http://acessibilidadebrasil.org.br/joomla/softwares/ provê uma gama interessante de soluções para validação

Os relatórios devem apresentar o percentual de cobertura para os diferentes parâmetros. Bem como cobrir os critérios definidos no W3C (https://www.w3.org/WAI/)

Quais estratégias aplicar?

O “custo” do defeito tende a aumentar conforme o processo de desenvolvimento de Software.

A estratégia mais eficiente é aquela que permite cobrir as características de maior prioridade primeiro. Em um site de compra de passagens, por exemplo, temos que priorizar a facilidade do acesso para diferentes pesquisas e filtros de resultados.

O processo de qualidade deve estar sempre atrelado às expectativas de uso da solução.

Testes de performance

São avaliações/inspeções periódicas que devem ser feitas visando um cenário de uso específico.

As seguintes perguntas devem ser feitas:
*** Quantos usuários devem acessar (em média) a solução por minuto?**
*** Quais são os limites de escalonamento (máquinas, memória, HD)?**
*** Quais são os SLAs definidos (prazos e limites)?**

Definição de testes de performance

Baseado nos parâmetros anteriores, devemos:

Estabelecer cenários incrementais de carga (100 usuários por minuto)

- **100 usuários por minuto = 100 / 60 = 1.7 usuários por segundo (2 usuários por segundo).**

1 - Carga inicial de 10 usuários por minuto por 5 minutos, adicionando-se 10 ao final do tempo.

2 - Carga inicial de 100 usuarios por minuto por 20 min, adicionando-se 10 ao final do tempo.

3 - Carga de inicial de 200 usuários por minuto por 1 hora.

Definição de Baseline

A baseline é definida como o padrão atual ou o padrão definido.

Através dos testes anteriores pode ser definida uma baseline atual.

No caso, por exemplo, digamos que com a carga de 100 usuários por minuto o sistema se manteve em 20% de alocação (geral) com tempos de resposta inferior a 1 segundo.

Caso na próxima liberação o tempo de resposta dobre e a alocação de infra siga igual, podemos identificar uma quebra de baseline causada por código, que deve ser avaliada (a página pode ter aumentado de tamanho por exemplo)

Quanto tempo demora para carregar a página?

Uma das unidades mais básicas em testes de performance é a identificação de tempos de fila e tempo de resposta.

Onde Tempo de resposta é o tempo total (tempo de fila + tempo de processamento).

O tempo para carregar uma página vai depender da disponibilidade do servidor, conexão do usuário e tamanho (em bytes) dos componentes visuais.

Ferramental para criação/execução e relatórios

LoadRunner - https://www.microfocus.com/en-us/products/loadrunner-professional/overview
JMeter - https://jmeter.apache.org/
Ambas as ferramentas geram relatórios para análise.

SLA e performance

SLA (Service Level Agreement) é um contrato firmado entre partes que estabelece limites de serviço. Muitas vezes define disponibilidade de serviço (99.999% do tempo funcionando) ou tempos de resposta.

Através das avaliações de performance podemos identificar possíveis falhas ou problemas com o SLA, assim como prover redundância como forma de mitigar possíveis riscos.

Introdução aos conceitos de garantia de qualidade

Estratégia de teste: Documento contendo a abordagem a ser executada nos diferentes níveis de qualidade.

Plano de testes: Documento contendo o conjunto de pré-requisitos e casos de teste.

Casos de teste: Conjunto de passos e resultados esperados visando testar uma única habilidade.

Pré-requisitos: Estado necessário para início do teste. Ex.: aplicação instalada.

Passos de teste: Conjunto de ações detalhadas para o teste.

Resultado esperado: De acordo com a especificação de Software qual é a ação esperada após o passo?

Relatório de testes: Documento contendo os resultados em forma Passou/Falhou para métricas.

Caso de teste: Instalação

Passo 1: copie o instalador para C:

Passo 2: clique duas vezes no instalador

Passo 3: verifique o resultado esperado 1

Passo 4: cancele o processo

Passo 5: verifique o resultado esperado 2

Resultado esperado 1: o instalador abre e mostra as informações para instalação

Resultado esperado 2: o processo deve ser cancelado e os arquivos criados devem ser apagados

Processos de qualidade

Divisão em camadas, com cobertura de funcionalidades e critérios não funcionais.

O processo de qualidade deve iniciar no momento de concepção de projeto. Quando a elucubração da solução é feita baseada em um problema específico.

A qualidade deve se ater ao que é necessário para resolução efetiva do problema.

No momento de design devem ser avaliados os critérios de acessibilidade e consistência.

No momento de desenvolvimento devem ser vistos todos os critérios funcionais e não funcionais.

Repetibilidade do processo

O processo de qualidade deve ser auditável a qualquer momento, para tanto é necessário que os processos que o compõem sejam bem documentados e se mantenham atualizados.

Bem como o planejamento dos testes, passos e resultados esperados.

O versionamento dos documentos é imprescindível para que a rastreabilidade da evolução seja auditável.

Teste de software é um exercício de disciplina.

Condições controladas de execução

Para que o processo seja repetível, são necessárias condições controladas.

São elas:
*** Mesmo padrão de HW: CPU, memória, hd etc...**
*** Mesmo padrão de Sistema Operacional e patches aplicados: Windows 11 build XYZ**
*** Mesma versão de browser: FireFox versão abc**
*** Mesmas condições iniciais: Imagem de HD feita para resetar o ambiente de testes.**

Relatórios de teste

Os relatórios de teste devem demonstrar o andamento da qualidade no produto.

Devem conter:

- **% de requisitos cobertos, com número de testes passando e falhando**
- **Número de defeitos por prioridade**

Priorização dos itens identificados

Os defeitos achados devem sempre ser priorizados de acordo com:

*** Impacto ao usuário: o quanto o defeito impede o usuário de completar a sua ação. Existe forma de completar a ação mesmo com esse defeito?**
*** Frequência de ocorrência: Sempre ocorre? Ocorre às vezes? Somente algumas vezes? Somente em cenários específicos?**

Métricas voltadas ao controle de qualidade

O controle das métricas é necessário para apoiar as decisões gerenciais.

Métricas mais usadas:
*** Número de defeitos por prioridade (por versão)**
*** Número de defeitos por Sistema/Browser/requisito**

Definição de ROI

O retorno de investimento (Return On Investment) é um cálculo que auxilia o entendimento dos gastos comprometidos com a qualidade e o que poderia ter sido o prejuízo por não aplicar.

De maneira geral:
*** Quantidade de pessoas do desenvolvimento,**
*** Quantidade de pessoas na qualificação,**
*** Número de defeitos achados,**
*** Etapa em que foram achados os defeitos.**

Estratégia de Qualidade de Software

Para compor a estratégia é necessário entender o propósito da solução.

Estabelecer o que será feito em cada etapa do desenvolvimento de software (concepção, design, implementação e validação).

Sendo que em processos ágeis o cenário muda a cada Sprint.

A estratégia deve conter:
Quais processos serão executados durante a concepção (pode ser uma reunião de entendimento)
Em tempo de design (características de acessibilidade, padronização do texto, vocabulário, espaçamento)
Em tempo de implementação (execução dos testes fim a fim, limitações etc)

Fechamento e conclusões


PUCRS online
uol edtech