O que estudar para a certificação: CTFL, BSTQB, ISTQB

Material extraído do Canal Pessonizando de Vinicius Pessoni, 2020
https://www.youtube.com/playlist?list=PLEqTHftpM91NDCvWp-oF_1CPl-yKZAxzc

https://viniciuspessoni.com/downloads/

Carmen Cabral
11/2023

Índice

Teste de software: por onde começar

✓Aprender os **conceitos básicos** de teste: o que é teste; por que precisa testar; o que é um processo de teste; quais as **técnicas de teste** (caixa preta, branca e cinza; baseado em defeito; mutação; exploratório; BDD; TDD; e outras); objetivo do teste.
✓Escolher qual **área de testes** quer trabalhar: testes para **frontend**; testes para **backend/API**; testes para **mobile**; testes para **Agile**; testes para games; testes para **IA**; testes para **security**; testes para **usability**; testes para sistema bancário; testes para BD; testes para sistemas embarcados/avião; testes para automóveis; testes para equipamentos médicos.
✓Escolher com qual **linguagem de programação** quer trabalhar: java; javascript/frontend; python; kotlin.
✓Procurar **tutoriais** das tecnologias: cucumber; selenium; cypress.
✓Decida se quer fazer uma **certificação**.
✓Fazer **mentoria** para decidir quais pontos serão trabalhados.

Entenda os 7 princípios do teste de software que todo engenheiro de software deve saber

1º Teste demonstra a presença de defeitos. Mas não garante a ausência total de defeitos.
2º Teste exaustivo é impossível. É necessário priorizar e verificar os riscos para focar os esforços de testes.
3º Teste antecipado. Trazer o teste para as fases iniciais do desenvolvimento de software.
4º Agrupamento de defeitos. Baseia-se no princípio de Pareto, um número pequeno de módulos (20%) contém a maioria dos defeitos descobertos (80%).
5º Paradoxo do pesticida. Executar o mesmo conjunto de teste e não encontrar mais defeitos depois de um tempo. Nesse caso, são necessárias revisões e atualizações para testar outros pontos do sistema.
6º Teste depende do contexto. Não é possível aplicar o mesmo teste de um sistema em outro.
7º A ilusão da ausência de erros. Encontrar e consertar defeitos não ajuda se o sistema não atender às expectativas e necessidades dos usuários.

Paradoxo do pesticida: como saber se seus testes são bons?

Se forem executados os mesmos testes, não serão encontrados defeitos novos.
Teste automatizado: para o caminho A, B, C.
Ocorreu um novo erro, mudando o caminho para A, B, C, D
Cobertura dos testes: abrangência do sistema que será testado. O teste automatizado existente não cobre esse novo erro. Ajuda a definir os riscos dos testes.
Relatórios de cobertura de teste: necessários para Compliance (ligados à lei) e para Auditoria.
Testes de regressão: nova funcionalidade foi incluída, então precisa fazer novo teste e ampliar a cobertura de testes.
Revisão dos pacotes de teste: para atualizar os testes.
Fazer medições: para saber quanto gastar e quais recursos alocar em quais funcionalidades.

Programador/Developer x Testador/Tester/QA: qual a diferença?

Conhecimentos necessários: requisitos, arquitetura, processos, projetos, testes, ferramentas.
Coach de qualidade: nas equipes com várias pessoas para ajudar a desenvolver a qualidade. Dar suporte.
Foco nas funcionalidades/**construir** o software x Foco na qualidade/**entregar** o software com qualidade.

Testa tudo! Caso de teste: por que e como fazer

Formado por 3 partes: entrada; ação; saída
Como criar casos de teste? (**técnicas para criar casos de teste**) Partição de equivalência, análise de valor limite, teste baseado em defeito, teste exploratório.
Como fazer os casos de teste? BDD (*Behavior Driven Development*/Desenvolvimento Baseado no Comportamento).

Por que automatizar testes de software?

✓Ajudar a fazer os **testes de regressão** para serem executados quando ocorrem modificações no software para verificar se nada foi 'quebrado'.
✓Agilizar a execução de **fluxos longos** (ex. compra num site).
✓Tornar os **testes repetíveis** de forma sistemática.
✓Liberar o Tester para fazer tipos de **testes diferentes**.
✓Sem automação não se consegue CI/CD (*Continuos Integration/Continuos Delivery/Deployment*)

Certificação de TI vale a pena?

CTFL: *Certified Tester Foundation Level*.
ISTQB: *International Software Testing Qualifications Board*

Por que testar programas?

***Objetivo Primário**: Fazer o produto falhar para revelar defeitos.
***Objetivo Secundário/Funcional**: Para mostrar que funciona. Verificar se o que foi pedido, foi entregue.
***Objetivo Secundário/Não Funcional**: apoiar as funções, mas que não foram pedidas pelos usuários (segurança, usabilidade, desempenho, acessibilidade).
***Danos**: Podem causar danos materiais (sistemas bancários) e até mesmo a morte (equipamentos médicos, máquinas de raio-x, automóveis, aeronaves, foguetes).

Algoritmo, Programa, Software e Sistema

Algoritmo: sequência finita (instruções) de passos para solucionar um problema.
Programa: conjunto de instruções codificadas em alguma linguagem de programação para solucionar algum problema.
Software: programa com configurações e documentações para instalação.
Sistema: conjunto de partes interrelacionadas/software que colcaboram para realizar uma ação.
Em resumo:
*Algoritmo: instruções não codificadas.
*Programa: instruções codificadas.
*Software: programa documentado para instalar e configurar.
*Sistema: vários softwares interligados.

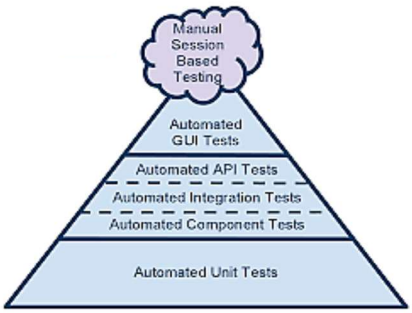
Teste de software vai acabar?

Precisa aprender sobre testes **automatizados**/***Automation Engineer***. ***Full Stack Tester***.
O teste **manual** está acabando, mas ainda é utilizado em:
***cenários** onde a validação e exploração manual continuam sendo indispensáveis.
***situações** onde o **tempo** para validar uma nova funcionalidade/requisito é **curto**.
***cenários** onde a viabilidade da automação não é favorável, por questões financeiras..
*atividades relacionadas a **design de testes**.

Pirâmide de teste automatizado

https://medium.com/@gianegf/pir%C3%A2mide-de-testes-uma-boa-estrat%C3%A9gia-para-automa%C3%A7%C3%A3o-de-testes-na-pr%C3%A1tica-1d87e64c3a44

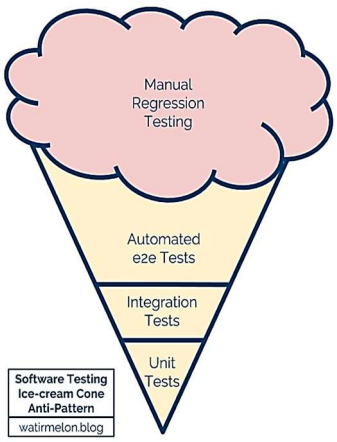
Padrão



Padrão
Da base ao topo (do mais simples e rápido aos mais complexos e lentos):
1º Unidade
2º Integração
3º Sistema
4º ***API***/Serviço
5º ***GUI***/Interface
6º *End to end (e2e)*: precisam de muitas partes do software.

***Testes determinísticos**: os que sempre que executados com as mesmas **entradas**, retornam as mesmas **saídas** / resultados esperados.
Testes não determinísticos/ ***flaky tests*** (esquisitos): mesma entrada e saídas diferentes. Às vezes passa e às vezes não.

Anti padrão/*Anti-Pattern Ice-cream Cone*



Testes exploratórios/SBET (*Session Based Exploratory Testing*): verificar os ***edge cases***/casos que acontecem de vez em quando. (ficam no topo da pirâmide)

Níveis e técnicas de teste

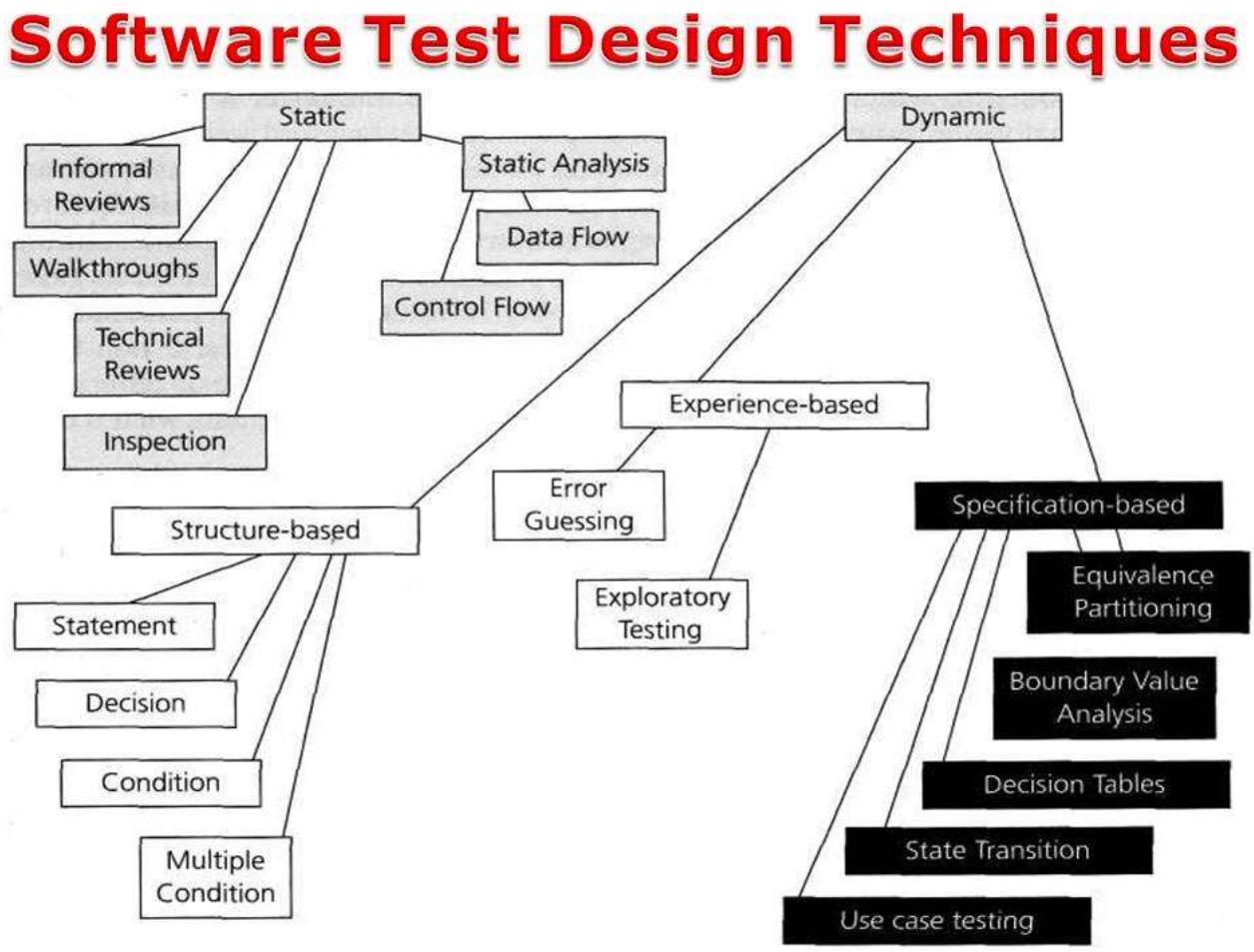
Engenharia de Software:
***Técnicas Estáticas** (não executam o software; inspeção; revisão; análise estática) para a **qualidade**.
***Técnicas Dinâmicas** para os testes de **software**.

NÍVEIS DE TESTE
***Unidade / Componente**: verificam algo específico do software, como método, função, classe.
Componente: nível de serviço.
***Integração**: unidades testadas em conjunto. Entre componentes ou entre sistemas.
Utilizam ***mocker/faker/stub***: objetos que fingem outros que estão fora do sistema testado, como BD.
***Sistema**: está funcionando como deveria, como o esperado.
***Regressão**: verificar se nada foi quebrado ao introduzir uma nova funcionalidade ou alterar uma que já exista. Testes de **Sanidade**/*Sanity*.
***Aceitação/Validação**: verificar se o sistema cumpre os requisitos solicitados.
Testes **Alfa** (interno: algumas pessoas da empresa testam);
Testes **Beta** (externo: algumas pessoas/clientes/*friendly client* testam);
***Canary** *testing*: testar uma nova versão ou uma nova funcionalidade com usuários.
****Smoke***/Fumaça

TÉCNICAS DE TESTE
***Baseadas em Estruturas/*Structure-based*/Caixa Branca**:
Teste de Declaração/*Statement Testing*;
Teste de Decisão/*Decision Testing*;
Teste de Condição/*Condition Testing*;
Teste de Condição Múltipla/*Multiple Condition Testing*.

Baseadas em Experiência/*Experience-based:
Teste Exploratório/*Exploratory Testing*;
Erro de advinhação/*Error guessing*.

***Baseadas em Comportamento/Especificação/Requisitos/Specification-based**:
Partição de Equivalência/*Equivalence Partitioning (EP)*;
Análise de Valor Limite/*Boundary Value Analysis (BVA)*;
Tabelas de Decisão/*Decision Tables Testing*;
Transição de Estado/*State Transition Testing*;
Teste de Caso de Uso/*Use Case Testing*.


Software Test Design Techniques
Static
Informal Reviews
Walkthroughs
Technical Reviews
Inspection
Static Analysis
Data Flow
Control Flow
Dynamic
Experience-based
Error Guessing
Exploratory Testing
Structure-based
Statement
Decision
Condition
Multiple Condition
Specification-based
Equivalence Partitioning
Boundary Value Analysis
Decision Tables
State Transition
Use case testing

Requisito funcional e não funcional

Requisito funcional: tudo o que o cliente pede para o sistema fazer. (aceitação, sistema, regressão)
Requisito não funcional: tudo o que o sistema precisa ter, como segurança, desempenho (quantidade de usuários), usabilidade, acessibilidade.

Requisitos, regras de negócio, user stories, critérios de aceitação, cenários (condições) e casos de teste

***Requisitos**: o que os usuários pedem (funcionais) e o que o sistema precisa (não funcionais).

***Regras de negócio**: regras específicas do negócio, da atividade principal da empresa.

****User stories***: conceito dos métodos ágeis, como o Scrum. Menor pedaço de trabalho que conseguimos entregar. Tem um formato definido (como/papel; eu quero; para que)
Ex. **como** vendedor da farmácia; **eu quero** faturar uma venda; **para** ter mais agilidade e organização na farmácia.
Ex. **como** vendedor da farmácia; **eu quero** faturar uma venda de forma rápida (3 seg); **para** que meu cliente não desista da compra.
A partir de uma *user stories*, podemos escrever um requisito funcional e não funcional.

***Critérios de aceitação**: conceito dos métodos **ágeis** que nos diz o que está bom e o que não está.
Critério de aceitação da *user stories*. Elementos usados para saber se a *user stories* está pronta ou não.
Ex em Gherkin.
Dado: que eu preencho os dados de uma venda com informações válidas.
Quando: faturar minha venda.
Então: venda deve ser realizada com sucesso.
Cucumber é uma ferramenta de **BDD**, utilizada para descrever critérios.
https://cucumber.io/

***Cenários de testes/condições**: possíveis cenários de uso para o software. O que deverá ser testado.

***Casos de testes**: agrupa os cenários. Contém um ou mais cenários. Como fazer os testes.

Requisitos e regras de negócio >>
User stories e critérios de aceitação >>
Aplicar as Técnicas de teste >>
Cenários de testes/condições e Casos de teste