

Testing I

Software Testing pertence a Engenharia de Software que nos ter processos, métodos de trabalho e ferramentas para identificar defeitos em nossos softwares paralelamente ao desenvolvimento dele, conforme a norma ISO25000 que regula todos os conceitos em nível de qualidade. Nos permitindo uma construção de acordo com as especificações do sistema conforme o usuário final precisa, corrigindo erros e desvios de softwares antes dele ser operacional.

Nesta disciplina veremos os conceitos envolvidos, as ferramentas relacionadas e usadas para execução de teste em cenários reais, exploraremos temas e tipos de teste, construção de casos de testes, execução de teste e como reportar defeitos encontrados.

Resumo do texto de introdução a disciplina de Testing I do curso CTD – Digital House.

[Certified Tech Developer | Digital House Brasil](#)

Fundamentação

O que é qualidade?

A definição formal é:

"Qualidade de software é o grau em que um sistema, componente ou processo atende aos requisitos especificados e às necessidades ou expectativas do cliente ou usuário" (IEEE Std 610, 1990).

Na indústria de tecnologia, embora toda equipe seja responsável pela qualidade do produto de software, o testador de software será a pessoa responsável por orientar e liderar as atividades relacionadas a qualidade para garantir que as necessidades do cliente sejam atendidas e a entregar um produto de qualidade.

Objetivos de aprendizagem:

- Conhecer e compreender a importância dos testes e da qualidade do software.
- Compreender e aprofundar-se sobre o ciclo de vida do processo de teste.
- Compreender e exercitar diferentes técnicas de teste.
- Criar e executar casos de teste apropriadamente.
- Compreender como relatar bugs e a importância deles.
- Conhecer os diferentes níveis e tipos de testes para determinar a conveniência de seu uso.
- Compreender e aprofundar-se nos testes de componentes (TDD e BDD).
- Conhecer e realizar testes de API.
- Aprender sobre diferentes frameworks de automação de teste e modelos de casos de teste automáticos.

Conteúdos:

Aula 1: Primeiros passos

- Um pouco de história
- Os 7 princípios do teste
- Aspecto psicológico do teste
- Os 3 pilares do desenvolvimento de software
- Qualidade
- Validação versus Verificação

Aula 2: Gestão de defeitos

- O que é um defeito?
- Erros, defeitos e falhas
- Ciclo de vida de um defeito
- Gestão de defeitos

Aula 3: Fundamentos e Gestão de Defeitos - Revisão

- Fechamento da semana (Aulas 1 e 2)

Aula 4: Design de testes

- Casos de teste
- Teste negativo e Teste positivo
- "Happy Path"
- Casos de uso
- Requisitos

Aula 5: Níveis e tipos de testes

- Modelos de desenvolvimento de software
- Níveis de teste
 - Teste de componentes
 - Teste de integração
 - Teste de sistema
 - Teste de aceitação
- Tipos de teste
 - Teste funcional
 - Teste não funcional

Aula 6: Design de testes, Níveis e tipos de testes - Revisão

- Fechamento da semana (Aulas 4 e 5)

Aula 7: Técnicas de testes

- Caixa preta e Caixa branca
- Análise de valor limite
- Partição de equivalência
- Teste de tabela de decisão
- Testes baseados na experiência

Aula 8: Testes estáticos e dinâmicos

- Testes estáticos
- Testes estáticos e Testes dinâmicos
- Processo de revisão

Aula 9: Técnicas de teste e Testes estáticos e dinâmicos - Revisão

- Fechamento da semana (Aulas 7 e 8)

Aula 10: Implementação e execução do teste

- Execução de casos de teste
- Criação de suíte
 - Smoke suites
 - Suítes de regressão
 -

Aula 11: Organização do teste

- Ambientes de teste (DEV, QA, UAT, STG, PROD)
- Métricas e relatórios

Aula 12: Implementação e execução do teste - Organização do teste - Revisão

- Fechamento da semana (Revisão da primeira parte da matéria)

Aula 13: Avaliação I

Aula 14: Introdução ao teste de componente

- Debugging
- Introdução ao teste de unidade

Aula 15: Teste Unitário

- Teste de unidade
- Primeiro teste unitário com JavaScript
- Teste de cobertura de sentença
- Teste de cobertura de decisão
- Ferramentas de cobertura de teste

Aula 16: Teste Unitário - Revisão

- Fechamento da semana (Aulas 14 e 15)

Aula 17: Introdução - Teste de API

- Teste de back end
- APIs
- Protocolo HTTP
- Postman - Parte 1

Aula 18: Teste de API

- Postman - Parte 2
- GET e POST
- Testes (js)
- Coleções em postman / variáveis de ambiente

Aula 19: Teste de API - Revisão

- Fechamento da semana (Aulas 17 e 18)

Aula 20: Fundamentos da automação de teste

- Introdução à automação
- Padrões de design
- Page Object Model
- Introdução ao Selenium
- Nosso primeiro teste com o Selenium
- Selenium WebDriver

Aula 21: Automação de teste

- Introdução ao Robot Framework
- Nosso primeiro teste com o Robot Framework

Aula 22: Automação de teste - Revisão

- Fechamento da semana (Revisão da segunda parte da matéria)

Aula 23: Avaliação II

Os sete princípios do Teste de Software:

1 - Teste demonstra a presença de defeitos. “Mas não prova que não há nenhum bug ali.”

2 - Teste exaustivo é impossível. “Em alguns casos, você poderia levar meses para testar todos os cenários.”

3 - Teste antecipando. “Prevenir é melhor que remediar.”

4 - Agrupamento de defeitos. “Um número pequeno de módulos (20%) contém a maioria dos defeitos descobertos (80%).”

5 - Paradoxo de Perspectiva. “Revise os seus testes para garantir que você não está sendo enganado pelos bugs. “

6 - Teste depende do contexto. “Na maioria das vezes, não é possível aplicar o mesmo teste de um sistema X em um sistema Y.”

7 - A ilusão da ausência de defeitos. “Um sistema sem bugs, porém que não atende às necessidades do usuário, não serve de muita coisa.”

Ciclo de vida de teste de software:

- Análise de requisitos:
 - “Identificar os tipos de teste que devem e podem ser executados”
- Frase de planejamento:
 - “Elaborar o plano de teste, recomendar ferramentas e estimar o tempo de trabalho e o custo aproximado para o projeto”
- Integração do caso de teste:
 - “Elaboração dos casos de teste e script”
- Configuração do Ambiente de teste:
 - “Configurar o ambiente e fazer uma lista de requisitos do sistema”
- Fase de implementação:
 - “Documentar resultados dos testes, registrar os erros e reportar os problemas”
- Encerramento:
 - “Discutir os resultados obtidos durante o ciclo de vida teste”

Exercício de Fixação / Micro desafio

Com base no conteúdo do Playground e no que foi visto nas aulas, debata na mesa de trabalho e responda às seguintes perguntas:

Disserte, com as suas palavras, sobre as principais subatividades realizadas dentro de cada atividade no ciclo de vida do teste de software.

Relacione cada princípio de teste com sua definição em uma planilha.

Relacione cada princípio de teste com sua definição.		
1	Não se pode provar que não há defeitos. Ele reduz a probabilidade de defeitos não encontrados no software, mas mesmo se eles não forem encontrados, o processo de teste não é uma demonstração de correção.	1- ausência de defeitos é uma falácia
2	Não é possível testar tudo, exceto em casos triviais. Em vez de tentar realizar testes exaustivos, devemos utilizar análises de riscos, técnicas de teste e de prioridades para centralizar os esforços do teste.	2- Testes exaustivos são impossíveis
3	Para detectar defeitos de forma prematura, as atividades de teste, tanto estáticas quanto dinâmicas, devem ser iniciadas o mais cedo possível no ciclo de vida do desenvolvimento de software para ajudar a reduzir ou eliminar alterações caras.	3- Teste prematuro economiza tempo e dinheiro
4	Em geral, um pequeno número de módulos contém a maioria dos defeitos encontrados durante o teste de pré-lançamento, ou é responsável pela maioria das falhas operativas.	4- Os defeitos são agrupados
5	Se os mesmos testes forem repetidos inúmeras vezes, eventualmente esses testes não encontrarão mais nenhum novo defeito. E para detectar um novo defeito, é importante revisar e atualizar regularmente os testes e os dados de teste existentes para adaptá-los e potencialmente encontrar mais falhas.	5- Paradoxo do pesticide
6	Dependendo do propósito do sistema ou da indústria, diferentes aplicações devem ser testadas diferentemente.	6- O teste depende do contexto
7	O sucesso de um sistema não depende apenas de encontrar erros e corrigi-los até que desapareçam, pois pode não haver erros mas surgir outros problemas. Existem outras variáveis que devem ser consideradas no momento de medir o sucesso.	7- Teste mostra a presença de defeitos e não a sua ausência

Erro, Defeito ou falha.

Erro: é caracterizado como passo, processo ou definição de dados incorretos, como por exemplo, uma instrução ou comando incorreto. O erro é a causa de um defeito, mas não necessariamente sempre causa um defeito, pois a linha que contém um erro pode nunca ser executada.

Defeito: ocorre durante a execução de um programa e caracteriza-se pela diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa, fruto da execução de um defeito.

Falha: é o desvio da especificação, isto é, produção de uma saída que difere da saída esperada, do que foi especificado. A falha ocorre em decorrência ao defeito.

Erro faz parte do universo físico, causado por pessoas e pode ocasionar a manifestação de um defeito em um produto, ou seja, a construção de um software de forma diferente ao que foi especificado. O defeito gera uma falha, um comportamento inesperado em um software e afeta diretamente o usuário final da aplicação, inviabilizando a utilização de um software.

Os três conceitos estão relacionados pois um depende do outro:

Erro é produzido pelo equívoco de uma pessoa causando um defeito no software provocando então uma falha no sistema. Exemplo:

O desenvolvedor implementa um software incorretamente cometendo um erro, isso torna em algo errado no código (um defeito) e quando executado causa um mal funcionamento (uma falha).

O objetivo como testadores é fornecer qualidade dentro de um sistema de software.



Gestão de defeitos

Processo Geral:

- Detectar
- Registrar
- Investigação e monitoramento
- Classificação/Resolução
- Detectar

No processo geral os objetivos são fornecer quaisquer defeitos, eventos adversos, problemas isolados para garantir a correção do produto, dar ideias de melhorias em seu processo de desenvolvimento e teste.

Escrever relatório de defeitos

Relatar os defeitos de forma eficiente pode garantir uma correção mais rápida. Na hora de criar um relatório devemos nos atentar aos seguintes tópicos:

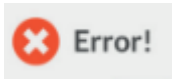
Os bugs devem ter identificadores exclusivos.

Uma falha deve ser reproduzível para relatá-la, se o defeito não for reproduzível então não é um defeito.

Ser claro e específico, escrevendo sobre o que está acontecendo e sobre o que é relevante.

Relatar cada passo realizado para chegar ao erro em questão.

Atributos	Descrição	Exemplo
ID	Abreviatura do identificador, um código único e irrepitível que podem ser números ou letras, ou uma combinação dos dois.	001 - Teste
Título	O título deve ser curto e específico, que se entenda a partir dele o que queremos reportar.	Login – Entrar com campos em brancos.
Descrição	Descrever um pouco mais sobre o erro, ou seja, o que deixamos de fora no título devemos explicar aqui.	Na tela de login, se eu deixar os campos nome e senha vazios e pressionar Enter, me leva para a página principal.
Data do relatório de defeito	A data em que o tester encontrou o defeito para saber posteriormente o tempo em que foi resolvido.	23/04/21
Autor	O nome do tester que encontrou o defeito, caso o desenvolvedor tenha alguma dúvida ele saberá a quem consultar.	Úrsula Rosa
Versão	É um número que nos diz em qual versão a aplicação está.	1.0.0
Ambiente	O ambiente em que testamos (desenvolvimento, QA, produção).	Desenvolvimento.
Passos para reproduzir	Os passos a seguir para chegar ao defeito encontrado.	1) Entrar na aplicação. 2) Deixar em branco o campo nome. 3) Deixar em branco o campo senha. 4) Clicar no botão “Entrar”.

Atributos	Descrição	Exemplo
Resultado esperado	É o que esperamos que aconteça ou seja exibido na aplicação de acordo com seus requisitos.	Não deve logar na aplicação sem um nome de usuário e/ou senha válidos.
Resultado obtido ou atual	É o que realmente aconteceu ou o que a aplicação nos mostrou. Pode ou não corresponder ao resultado esperado, se não corresponder, é detectado um erro ou bug.	Entre na aplicação sem nome de usuário e sem senha.
Gravidade	Quão grave é o defeito que encontramos, podendo ser uma dessas categorias: bloqueado, crítico, alto, médio, baixo ou trivial.	Crítico
Prioridade	Com isso dizemos o quão rápido se deve resolver o defeito, podendo ser alta, média ou baixa.	Alta
Status do defeito	Os status podem ser: novo, adiado, duplicado, rejeitado, atribuído, em andamento, corrigido, aguardando verificação, em verificação, verificado, reaberto e fechado.	Novo
Referências	Link para o caso de teste com o qual encontramos o erro.	com.ar/TC-001-User-Login
Imagem	Pode-se anexar uma captura da tela do erro, o que nos permite demonstrar que o erro ocorreu e ajuda o desenvolvedor a localizar o erro.	

Exercícios de Fixação

Objetivo é familiarizar-se com a criação de relatórios de defeitos (bug report).

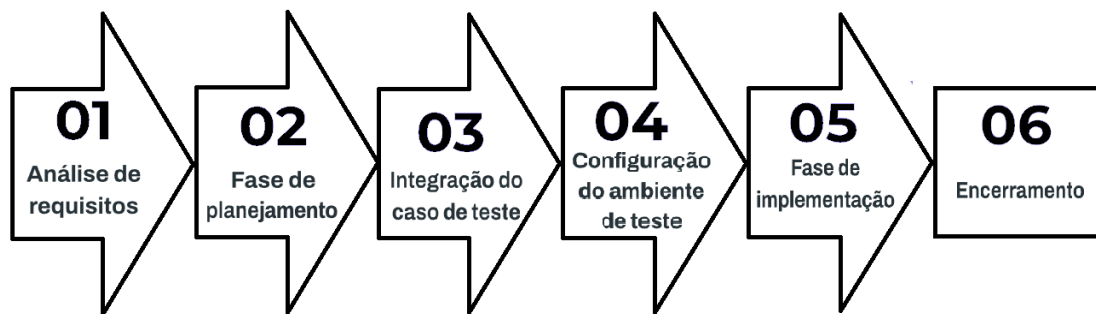
Micro desafio - Etapa I

Convidamos você a revisar o conteúdo do Playground relacionado aos relatórios de defeitos e criar um documento no Google Sheets que servirá como um modelo para criar todos os bugs reports que forem necessários. Esta atividade é uma atividade em grupo a ser realizada com os seus colegas na mesa de trabalho.

Micro desafio - Etapa II

Uma vez que tenha criado o seu modelo para os bugs reports, você deverá acessar a plataforma Demo QA (<https://demoqa.com>) e escolher uma ou mais aplicações para testar. O objetivo aqui é encontrar pelo menos dois defeitos e adicioná-los ao template que você criou na etapa anterior.

Ciclo de vida do teste de software



O defeito deve ter campos mínimos para ser tratado, podendo ser eles:

- ID
- Título
- Descrição
- Resultado final
- Resultado esperado
- Status
- Passos
- Reportado por

Exercícios de Fixação

Objetivo

Reforçar os conceitos aprendidos na semana relacionados a:

O que é um defeito?

Erros, defeitos e falhas.

Como relatar um defeito.

Desafio

O objetivo desta prática será trabalhar com a aplicação do BugBank!. Portanto, vamos procurar defeitos dentro dela. As características a considerar são: (DICA: Você pode se basear nos próprios requisitos do Bugbank <https://bugbank.netlify.app/requirements>)

Login:

O e-mail deve ter um formato válido, ou seja, deve incluir um “@” e um domínio, caso contrário, exibir uma mensagem: “O formato do e-mail está incorreto”.

O número de caracteres permitidos no campo de senha deve ser maior ou igual a 1 e menor ou igual a 10. Você não deve permitir a entrada de mais caracteres.

Caso o nome de usuário e a senha não coincidam com os dados cadastrados, exiba a mensagem: “Os dados estão incorretos. Verifique-os e tente novamente.” Deve ser permitido fechar a mensagem.

Crie uma conta:

Espera-se que preencha os campos de nome, sobrenome, e-mail e senha.

Possibilidade de redefinir a senha por esquecimento ou perda.