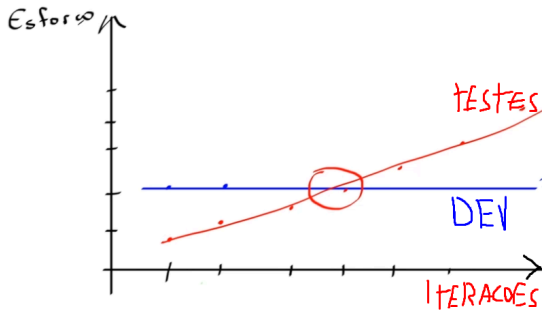
Testes Exploratórios

Módulo 1 – Introdução

Subtópico 1 – Testes Automatizados x Testes Manuais

Neste subtópico, aborda-se a importância da existência tanto dos testes automatizados quanto dos testes manuais, cada um sendo aplicável em contextos específicos.

Inicialmente, foi apresentada uma introdução com o auxílio de um gráfico que ilustrava a curva de desenvolvimento e testes de uma aplicação. Tal gráfico evidenciou a inviabilidade de utilizar unicamente testes manuais durante o processo de verificação de software.

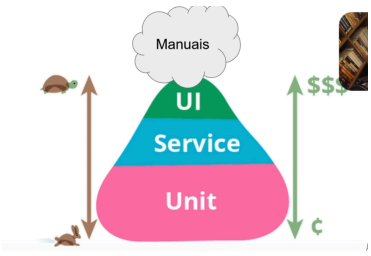


Como explicação complementar, destacou-se que a linha referente ao desenvolvimento de software tende a se manter constante ao longo do tempo da sprint (isto é, o tempo necessário para desenvolver funcionalidades permanece o mesmo). Em contrapartida, a linha correspondente aos testes tende a crescer, pois a cada nova funcionalidade desenvolvida, há a necessidade de aplicação de testes de regressão. Com isso, há um acúmulo progressivo de funcionalidades a serem testadas, impactando diretamente na entrega da aplicação.

Em sequência, foi exibido um segundo gráfico que comparava os testes manuais com os testes automatizados. O gráfico demonstrava que, à medida que mais testes manuais são executados, a curva de esforço e tempo de sprint cresce exponencialmente, dada a demanda de tempo envolvida na verificação manual de múltiplas funcionalidades.

Por outro lado, nos testes automatizados, observa-se que a curva de esforço cresce em menor proporção. Isso se dá pelo fato de que testes automatizados requerem menor esforço de adaptação (menor esforço na manutenção dos scripts) e são executados em tempo reduzido.

Posteriormente, foi discutida a Pirâmide de Testes como uma estratégia recomendada de abordagem. A imagem representativa da pirâmide organiza os tipos de testes em três camadas:



- **Testes Unitários (Unitários)**: Localizados na base, são os mais frequentes, rápidos e de menor custo. Valida-se aqui pequenas unidades de código individualmente.
- **Testes de Serviço (Service)**: Representam a camada intermediária, responsáveis pela verificação da integração entre componentes ou serviços. Têm custo e tempo de execução moderados.
- **Testes de Interface (UI)**: Situam-se no topo da pirâmide. Validam a aplicação por meio da interface gráfica, sendo os testes mais lentos e dispendiosos.

Adicionalmente, enfatizou-se a relevância dos **testes manuais**, indicados por uma nuvem na representação gráfica. Apesar de mais demorados e custosos, continuam sendo indispensáveis em muitos contextos. Tais testes viabilizam um planejamento mais qualitativo, sobretudo em cenários que exigem sensibilidade humana, como testes de usabilidade. Nestes casos, a intuição do testador é capaz de captar nuances que poderiam escapar à automação.

Em síntese:
Os testes automatizados são excelentes para garantir que funcionalidades técnicas operem de maneira eficiente, principalmente em processos repetitivos e estáveis. Por sua vez, os testes manuais, mesmo sendo mais onerosos em termos de tempo e esforço, são fundamentais em contextos complexos, permitindo validações mais personalizadas e refinadas, sobretudo na experiência do usuário.

Subtópico 2 – Testes por Roteiro x Testes Exploratórios

Neste subtópico, discutiu-se as diferenças entre os testes baseados em roteiros e os testes exploratórios, bem como suas aplicações práticas.

Os testes com roteiro seguem um caminho pré-definido, composto por etapas cuidadosamente planejadas antes da execução. Por exemplo, ao testar determinada funcionalidade de uma aplicação, o testador executa ações conforme um script estabelecido.

A principal vantagem dessa abordagem é a previsibilidade: os testes podem ser reproduzidos com exatidão e consistência. São especialmente eficazes em testes de regressão, pois previnem falhas e são passíveis de automação.

Por outro lado, os **testes exploratórios** caracterizam-se pela liberdade e criatividade do testador, que interage com o sistema sem roteiro fixo, buscando descobrir falhas que poderiam passar despercebidas em abordagens mais engessadas.

Essa liberdade investigativa permite que falhas inesperadas sejam detectadas, muitas vezes em áreas pouco exploradas em testes por roteiro. A limitação, no entanto, é que esse tipo de teste ainda não é automatizável.

Testes baseados em roteiros	Testes Exploratórios
Executado seguindo um roteiro	Executado de acordo com o propósito definido
Os passos são bem definidos	Apenas um direcionamento inicial
A fase de planejamento se dá na criação dos roteiros, não durante a execução do teste.	O planejamento se dá durante a execução do teste
Leva o tempo necessário para executar todos os passos	Possui um limite de tempo pré-determinado
Previne erros	Encontra erros
Pode ser automatizado	Não pode ser automatizado (ainda)
Como resultado final, temos "Sucesso" ou "Falha"	Como resultado final, temos: - "Sucesso" ou "Falha" - Dúvidas e observações realizadas - Mais entendimento e ideias para outras execuções

---

Subtópico 3 – A Automação como Aliada

Nesta seção, destaca-se que, embora os testes automatizados e os testes manuais sejam distintos, ambos podem (e devem) ser utilizados de forma complementar, compondo uma abordagem híbrida que maximize a eficiência dos processos de teste.

Exemplo 1 – Testes Automatizados
A equipe de QA pode implementar scripts para validar que:

- O usuário é capaz de inserir diferentes critérios de busca (como origem, destino e datas).
- A lista de resultados corresponde corretamente aos critérios informados.

- Os filtros (como preço e duração do voo) funcionam adequadamente.

Esses testes são executados periodicamente, especialmente após atualizações no sistema, garantindo que as funcionalidades principais estejam sempre operacionais.

Exemplo 2 – Testes Manuais
Simultaneamente, testadores manuais concentram-se em:

- Avaliar a clareza da interface de busca.
- Verificar a relevância dos resultados apresentados.
- Analisar a experiência de navegação em diferentes dispositivos (desktop e mobile).

A combinação dessas abordagens permite detectar tanto falhas técnicas quanto problemas de usabilidade. Por exemplo, testadores manuais podem identificar que usuários têm dificuldade em utilizar certos filtros, mesmo que tecnicamente estejam funcionando. Com base nesse feedback, ajustes podem ser feitos na interface, enquanto os testes automatizados continuam garantindo a estabilidade das funcionalidades.

Subtópico 4 – Gerenciamento de Massa de Dados

Este subtópico explora a importância do gerenciamento da massa de dados nos ambientes de teste.

Gerenciar dados de forma eficiente é essencial para simular cenários reais ou específicos durante os testes. Esses dados podem ser gerados manualmente, importados de ambientes de produção (com os devidos cuidados) ou criados por meio de scripts automatizados.

É fundamental que a massa de dados contemple uma diversidade de cenários: válidos, inválidos e casos-limite. Além disso, o ambiente de testes deve ser completamente segregado do ambiente de produção, a fim de evitar impactos não desejados.

Outro ponto crucial é a variação constante de entradas e ações no ambiente de testes, o que aumenta a cobertura e a robustez das validações. Destacou-se também a necessidade de restaurar o estado inicial do ambiente após cada execução, garantindo consistência nos resultados e evitando que testes subsequentes sejam comprometidos por execuções anteriores.

Subtópico 5 – Quando Parar de Testar?

Neste subtópico, discutiu-se os critérios que podem indicar o momento adequado para encerrar as atividades de teste de uma aplicação.

Inicialmente, apresentaram-se os seguintes critérios:

- **Prazos (Deadlines)**: Os testes são interrompidos quando o tempo previsto no cronograma se esgota. É uma prática comum em projetos com prazos rigorosos.
- **Esgotamento de Recursos**: O término dos testes também pode ocorrer quando se esgotam os recursos disponíveis, como equipe, tempo ou orçamento.
- **Meta de Cobertura de Testes Atingida**: A interrupção ocorre quando se alcança uma cobertura pré-estabelecida (de código, requisitos ou funcionalidades). Contudo, esse critério não é totalmente confiável, visto que as metas podem variar ao longo do projeto.
- **Redução da Taxa de Erros**: Os testes podem cessar quando a taxa de erros atinge um patamar considerado aceitável, especialmente se as falhas restantes forem de baixa criticidade.
- **Etapas de Lançamento (Alpha/Beta/RC)**: Os testes podem ser encerrados ao fim das fases de desenvolvimento, como Alpha (testes internos), Beta (testes com usuários selecionados) ou Release Candidate (versão pronta para produção).

Adicionalmente, foi apresentada uma pirâmide conceitual com três pilares essenciais para qualquer projeto de software: **Tempo**, **Dinheiro** e **Qualidade**. Tais pilares estão inter-relacionados, sendo necessária, muitas vezes, a escolha de dois em detrimento de um:

- **Tempo**: Relaciona-se ao prazo de entrega.

- **Dinheiro**: Refere-se ao orçamento disponível.
- **Qualidade**: Reflete a robustez e confiabilidade do software.

O papel do QA é, portanto, garantir a qualidade da aplicação, mesmo diante das limitações impostas pelos demais pilares.

Por exemplo:

- **Tempo e Dinheiro**: Se o prazo for apertado e o orçamento limitado, será difícil garantir alta qualidade. Isso pode resultar em menos testes, comprometendo a cobertura e dificultando a identificação de bugs.
- **Qualidade e Dinheiro**: Se o foco for garantir alta qualidade, com um orçamento adequado para ferramentas e equipe, o tempo de entrega pode ser mais longo para permitir que todas as funcionalidades sejam testadas com profundidade.
- **Qualidade e Tempo**: Se o objetivo for garantir qualidade dentro de um prazo curto, será necessário investir mais dinheiro — por exemplo, contratando mais pessoas ou automatizando processos para acelerar o trabalho.

Analogia do Muro x Grade (no contexto de um QA):

- **Muro (Proteção Alta, Custo Alto)**: Representa um sistema de QA robusto e abrangente. Envolve testes detalhados, documentação rigorosa e processos estruturados. É ideal para produtos críticos, onde a qualidade é essencial, mas exige mais tempo, recursos e investimento financeiro.
- **Grade (Proteção Moderada, Custo Baixo)**: Representa uma abordagem mais leve e ágil. Com testes básicos e verificações de qualidade, oferece proteção moderada, sendo ideal para projetos menores ou fases iniciais. Apesar de menos robusta, se bem estruturada, pode atingir um nível aceitável de proteção.

Resumo: A escolha entre “muro” e “grade” depende das prioridades da empresa. O muro é indicado para produtos críticos, enquanto a grade permite flexibilidade e velocidade. O desafio é encontrar um equilíbrio entre qualidade e recursos disponíveis.

Subtópico 6 - Planejamento de testes

Nesse subtópico, foi introduzido o conceito de planejamento de testes, destacando a importância da preparação adequada e o uso da linguagem Gherkin.

- **Foco e Organização**: Ter objetivos claros e um ambiente organizado é essencial. Dividir tarefas com ferramentas de gerenciamento e quebrar processos em etapas menores facilita a execução e evita o estresse de última hora.
- **Uso da Linguagem Gherkin**: A linguagem Gherkin permite descrever os requisitos de forma simples e compreensível. Ela estrutura os cenários de teste com clareza, garantindo que todos entendam o que deve ser testado e quais são os critérios de aceitação.

Módulo 2 - Estratégias Iniciais

Subtópico 1 - Estratégia em Freestyle Testing

O **freestyle testing** é uma abordagem flexível onde os testadores exploram a aplicação sem seguir roteiros rígidos, usando intuição e experiência para encontrar bugs.

- Permite maior liberdade e criatividade.
- Ajuda a encontrar problemas de usabilidade e falhas não previstas.
- Benefícios: descoberta de bugs inesperados, maior engajamento e, às vezes, economia de tempo.

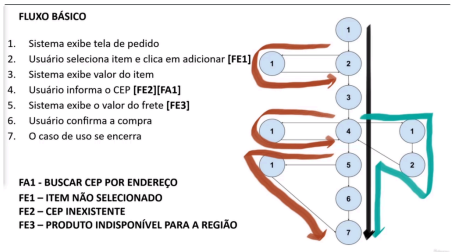
Subtópico 2 - Estratégia Baseada em Cenários

Essa estratégia simula o uso prático do sistema, avaliando como ele se comporta em situações reais. A seguir, tópicos que ajudam na construção de cenários:

- **Adicionar Passos**: Verifica como o sistema lida com novas ações ou funcionalidades.
- **Remover Passos**: Testa se o sistema continua funcional ao excluir etapas do fluxo.
- **Alterar Passos**: Avalia a adaptação do sistema a mudanças no processo.
- **Repetir Passos**: Verifica se loops e iterações funcionam sem erros.
- **Substituir Passos**: Observa como o sistema reage à troca de componentes.
- **Executar Igual (mas Diferente)**: Testa variações sutis com o mesmo fluxo aparente.

Exemplo de Fluxo de Caso de Teste (Sistema de pedidos):

1. Sistema exibe tela de pedido.
2. Usuário seleciona item e clica em adicionar.
   - [FE1] Item não selecionado → erro.
3. Sistema exibe valor do item.
4. Usuário informa o CEP.
   - [FE2] CEP inexistente.
   - [FA1] Buscar CEP por endereço.
5. Sistema exibe valor do frete.
   - [FE3] Produto indisponível para a região.
6. Usuário confirma a compra.
7. Fluxo se encerra.



Subtópico 3 - Estratégia Baseada em Feedbacks

Foca na análise de informações vindas da própria aplicação e dos usuários para guiar os testes.

- **Densidade de Erros Reportados**: Foco nas áreas com maior incidência de falhas.
- **Priorização do Especialista**: Uso da experiência técnica para identificar pontos críticos.
- **Criticidade de Cenários**: Cenários que podem causar grandes impactos têm prioridade.
- **Cobertura**: Ajuste da cobertura conforme os feedbacks, priorizando áreas sensíveis.

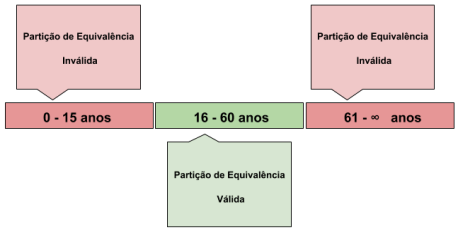
Subtópico 4 - Estratégia Baseada em Técnicas

Visa aplicar métodos estruturados para otimizar o esforço de testes.

- Permite testar menos, mas com mais inteligência.
- Foco em entradas críticas, limites e funcionalidades complexas.
- **Limitação**: Mesmo com testes bem feitos, é impossível garantir 100% de ausência de falhas.
- **Conclusão**: O ideal é combinar cobertura eficiente, priorização de risco e aceitação do risco residual.

Subtópico 5 - Técnica de Classe de Equivalência

Trata-se de uma técnica essencial para otimizar os testes.



- **Definição**: Agrupa dados de entrada em classes tratadas da mesma forma pelo sistema.
- **Objetivo**: Reduz o número de testes, testando apenas valores representativos de cada classe.
- **Exemplo**:
  Campo de idade aceita de 0 a 100.
  Classes:
  - Válidos: 0 a 100.
  - Inválidos: negativos, acima de 100, letras.

Essa técnica cobre bem os principais casos com menos esforço, mas pode não detectar erros em sistemas mais complexos se usada isoladamente.