Análise e modelagem de testes - NoBugs

A seção 6 irá abordar o princípio da visão geral das Técnicas de Testes, dividindo-as em 3 categorias principais.

Visão Geral das Técnicas de Teste

Neste tópico podemos ter uma analogia como: a bússola do QA. Por aqui onde ele começará a pensar e modelar como, onde e quando aplicar tal técnica de teste. Sendo dividida em 3 grandes grupos, abaixo temos eles:

- **Técnicas Baseadas na Especificação (Caixa Preta):**
  - É baseado em uma análise mais voltada para o usuário, onde o QA não tem contato com o código que está por trás da aplicação. Mas pode ter acesso a um documento ou a nenhum, mas irá realizar interações como usuário comum.
  - Os seus casos de teste é independente de como está o software, o código em si. Por isso vem o termo “Caixa Preta”, você ainda não sabe o que há na parte de dentro.
  - Caso uma certa implementação mude, mas, mesmo assim continue aquele comportamento exigido pelo cliente, o seu caso de teste (da caixa preta), ainda permanecerá útil.
    - Abaixo alguns exemplos deste tipo, que serão explicados ao longo da apresentação:

Técnica	O que faz
Particionamento de Equivalência	Divide entradas em grupos.
Análise de Valor Limite	Testa os extremos desses grupos.
Tabela de Decisão	Testa combinações de regras.
Transição de Estado	Testa mudanças de estado do sistema.

- **Técnicas Baseadas na Estrutura (Caixa Branca):**

- Aqui é onde o QA irá ver o código, aquilo que não viu na Caixa Preta. Baseando-se por condições lógicas da estrutura.
- Já neste caso de teste, o teste depende de como está o software. Com isso esse modal apenas pode ser feito com certas implementações.
- O seu foco é garantir que todo o caminho do teste esteja ocorrendo bem.
- Alguns exemplos desta técnica:

Técnica	O que cobre
Cobertura de Instrução	Todas as linhas.
Cobertura de Decisão	Todos os ramos (if/else, etc).

- **Técnicas Baseadas na Experiência:**
  - E é aqui onde aquele famosa “bagagem” se torna importante para o QA, onde experiências em diferentes projetos e até nos mesmos, contam.
  - A eficácia dessa técnica leva em conta muito a habilidade que o QA irá ter e adquirir ao longo de sua carreira. Podendo detectar defeitos que podiam não aparecer na caixa preta e na caixa branca.
  - Mas esse não anda sozinho, ele é colocado como um complemento dos dois tipos já vistos.
  - Alguns exemplos de seu uso:

Técnica	Como funciona
Teste exploratório	Anda pelo sistema livremente.
Checklist	Usa uma lista padrão de testes que o mesmo cria.
Ataque	Tenta quebrar o sistema com o que sabe.

Particionamento de Equivalências

Ela é usada para meio que reduzir os seus testes sem perder a qualidade, lembrando que teste exaustivo fere um dos princípios do teste.

- Ele faz com que todos os dados envolvidos sejam agrupados e tratados da mesma forma, por equivalências lógica.
  - Primeiro identifica o campo, ou problema a ser colocado.
  - Verifica os valores dos dados em jogo.
  - Cria-se então as classes e verifica.
    - Ficará mais claro no exemplo a seguir:

*Campo de cadastro aceita **idade entre 18 e 60**.*

Partições:

- Classe válida: 18–60 → **testa o 30**, por exemplo.
- Classe inválida abaixo: <18 → **testa o 10**
- Classe inválida acima: >60 → **testa o 70**

Você NÃO precisa testar 18, 19, 20, 21...
Um valor de cada faixa já é suficiente, porque todos dentro do grupo são tratados igual.

- Ele pode ser usado quando a funcionalidade aceita 'n' valores múltiplos.
- E como falado, é usado quando se quer diminuir os testes. Mas com isso não justifica não testar, é uma ajuda no seu trabalho e não uma forma de "deixar para lá" as outras tarefas de teste.

Análise de Valor Limite (Boundary Value Analysis – BVA)

A **análise de valor limite** é uma extensão do particionamento de equivalência, utilizada quando a entrada pode ser representada como um **conjunto ordenado**, como números, datas ou sequências.

Princípios:

- Baseia-se na observação de que **erros ocorrem frequentemente nas extremidades dos intervalos válidos**.
- Cada partição válida ou inválida possui **valores mínimos e máximos**, chamados de **valores-limite**.
- Os testes devem ser construídos utilizando **valores nos próprios limites e valores ligeiramente fora deles** (logo acima ou abaixo).

Aplicabilidade:

- Pode ser utilizada em **qualquer nível de teste** (componente, integração, sistema).
- É relevante sempre que houver **intervalos numéricos definidos** como critérios de entrada.

Cobertura:

$$\text{Cobertura de Limites} = \frac{\text{Limites testados}}{\text{Total de limites identificados}}$$

Exemplo:

Imagine um campo que aceita **ano de nascimento entre 1900 e 2010**. Os valores de teste seriam:

- **Valores válidos nos limites**: 1900 e 2010
- **Valores fora dos limites**: 1899 e 2011

Assim, os valores testados seriam: 1899, 1900, 2010, 2011 — verificando se o sistema aceita os valores válidos e rejeita os inválidos.

Vantagens:

- Tem maior eficácia na detecção de erros do que o particionamento de equivalência isoladamente.
- Foca nos **pontos mais críticos** para falhas no sistema.

Teste com Tabela de Decisão

O **teste baseado em tabela de decisão** é indicado quando o comportamento do sistema depende de **múltiplas condições combinadas**, especialmente quando essas condições resultam em diferentes ações.

CONDIÇÕES	Regras 1	Regras 2	Regras 3	Regras 4	Regras 5	Regras 6	Regra 7	Regra 8
Ocupa cargo de chefia ?	S	S	S	S	N	N	N	N
Idade maior que 40 anos ?	S	S	N	N	S	S	N	N
Mais de 2 anos no cargo ?	S	N	S	N	S	N	S	N
AÇÕES								
Exame especial	X	X	X		X	X		
Exame normal				X			X	X

onde: S=sim; N=não; X=ação a ser executada.

Princípios:

- Representa de forma estruturada as **regras de negócio complexas**.
- É composta por:
  - **Condições (entradas)** nas linhas superiores.
  - **Ações (saídas)** nas linhas inferiores.
  - **Regras (colunas)** que representam cenários distintos a serem testados.

Aplicabilidade:

- Ideal para sistemas com **lógica de negócio condicional** complexa, como validação de cadastros, cálculo de tarifas ou concessão de benefícios.
- Facilita a **visibilidade e rastreabilidade** das decisões do sistema.

Exemplo simplificado:

Cada coluna representa uma **regra** distinta que deve ser transformada em um **caso de teste**.

Cobertura:

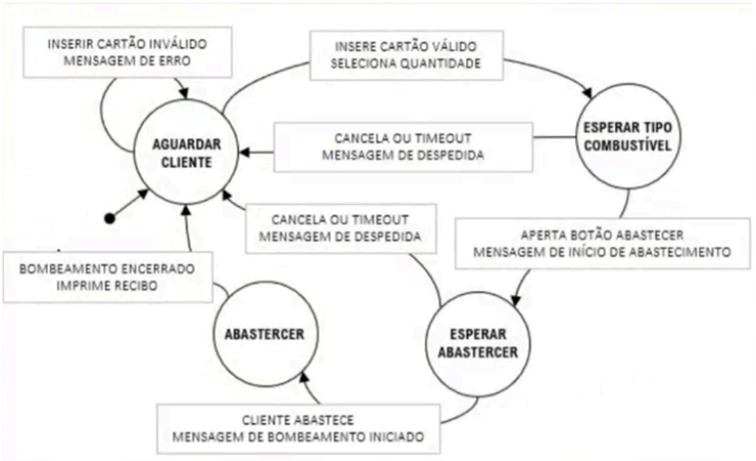
- O número de colunas determina a **quantidade mínima de casos de teste** para cobertura total da tabela.

Vantagens:

- Organiza cenários de forma clara e sistemática.
- Evita omissão de combinações importantes.
- Auxilia na automação e validação junto a stakeholders.

Teste de Transição de Estado

O **teste de transição de estado** é utilizado para verificar o comportamento de sistemas que operam de acordo com **estados distintos** e que mudam de estado em resposta a eventos.



Princípios:

- Um diagrama de transição de estado mostra os possíveis estados do software, bem como a forma como o software entre, sai e transita entre os estados.
- Uma transição é iniciada por um evento (p. ex., entrada do usuário de um valor em um campo)
- A mudança de estado pode fazer com que o software execute uma ação (p. ex., gerar uma mensagem de cálculo ou de erro).

- Uma tabela de transição de estado mostra todas as transições válidas e potencialmente inválidas entre estados, bem como os eventos, condições de proteção e ações resultantes para transições válidas.
- Os diagramas de transição de estado normalmente mostram apenas as transições válidas e excluem as transições inválidas
- Os testes podem ser projetados para cobrir uma sequência típica de estados, para exercitar todos os estados, para exercitar cada transição, para executar sequências específicas de transições ou para testar transições inválidas (cobertura)

Ferramentas:

- **Diagrama de Transição de Estado**: ilustra os estados válidos e as transições permitidas entre eles.
- **Tabela de Transição de Estado**: detalha todas as transições possíveis (válidas e inválidas), eventos, condições e ações esperadas.

Tipos de cobertura:

- **Cobertura de estados**: garante que todos os estados sejam atingidos.
- **Cobertura de transições**: garante que todas as transições possíveis sejam testadas.
- **Cobertura de sequência de transições**: garante que sequências específicas de eventos sejam testadas.
- **Teste de transições inválidas**: garante que o sistema lide corretamente com entradas inesperadas.

Exemplo prático:

No caso de uma bomba de gasolina autônoma com pagamento por cartão:

- O objetivo é testar todas as **transições** do sistema, iniciando e finalizando sempre no estado inicial.
- **Resposta correta**: são necessários **4 testes**:
  - 1 caminho ideal (fluxo de sucesso completo).
  - 3 caminhos cobrindo diferentes situações de erro ou intervenção.

Vantagens:

- Muito eficaz para sistemas com **fluxos de navegação, estados e ciclos**.
- Permite detectar falhas em **sequências incorretas de ações** ou ausência de transições esperadas.

Testes de Caixa-Branca

Testes que se baseiam na estrutura interna do objeto, ou seja, na parte literalmente operacional de um corpo de projeto (o código do projeto), utilizada em todos os níveis de teste, mesmo sendo usualmente comuns em testes de componentes (unitários) e na integração entre os componentes.

Existem 2 tipos de **técnicas de teste de caixa branca:**

Teste de Instrução

Técnica que verifica a execução de **linhas de código individuais** (comandos/sentenças) sem avaliar condições ou ramificações. Vejamos algumas etapas desse teste

- **Análise Estática do Código:**
  - O sistema de testes identifica todas as instruções executáveis no código (por exemplo, atribuições, chamadas de funções, loops, etc.).
- **Instrumentação do Código:**
  - O código é "instrumentado", isto é, são inseridas **marcas ou contadores** que registram quais instruções foram executadas durante o teste.
  - Essa prática é chamada de **instrumentação de código**.
- **Execução dos Casos de Teste:**
  - Os casos de teste são executados e a ferramenta coleta dados sobre **quais instruções foram atingidas**.
- **Geração de Relatório de Cobertura:**
  - Um relatório é emitido indicando a **porcentagem de instruções cobertas** e, em muitos casos, **destacando visualmente** as que não foram alcançadas.

Esse teste utiliza uma métrica chamada ***cobertura de Instrução, utilizada para*** avaliar se o **corpo do código** foi testado de forma abrangente. Ela é representada pela quantidade de instruções executadas sobre todas as instruções.

É ótima para identificar trechos de códigos, porém pode ocorrer do fluxo do código está correto, porém apresentar defeitos lógicos.

Teste de Decisão

É uma técnica de teste estrutural (caixa branca) que foca em testar todas as decisões existentes no código e o código executado com base nessas decisões. veja s seguintes etapas do processo:

1. **Identificação das Decisões Lógica**
   - O testador analisa o código-fonte (ou fluxogramas, pseudocódigo, diagramas) e **mapeia os pontos de decisão**.

- Exemplo: estruturas como `if (condição)`, `while (condição)`, `case`, etc.

2. **Extração dos Caminhos**
   - Para cada decisão, são considerados **dois caminhos**:
     - Quando a condição for **verdadeira**
     - Quando for **falsa**
3. **Criação de Casos de Teste**
   - São formulados **casos específicos** para **forçar cada resultado da decisão**.
   - Caso não haja testes cobrindo ambos os lados de uma decisão, o teste é **incompleto**.
4. **Execução e Avaliação**
   - Os testes são executados e analisados para verificar se **ambas as alternativas** de cada decisão foram, de fato, exercitadas.

Esse teste utiliza uma métrica chamada **Cobertura de Decisão**, que serve para verificar se **todas as condições lógicas do código** foram testadas em **ambas as direções possíveis**: quando são **verdadeiras (true)** e quando são **falsas (false)**.

Cada ponto de decisão, como um `if` ou um `while`, pode levar o código por caminhos diferentes dependendo do resultado da condição. A métrica é calculada dividindo o número de resultados de decisão que foram efetivamente testados (por exemplo, testar tanto o “sim” quanto o “não” de uma condição) pelo número total de possibilidades existentes. Assim, se um `if` só for testado com a condição verdadeira, a cobertura será de 50%, mas se for testado com as duas possibilidades, a cobertura será de 100%.

Experiencia

conjunto de abordagens no qual o testador utiliza seu **conhecimento prévio**, **intuição** e **experiência prática acumulada** para projetar e executar casos de teste.

A técnica de experiência é dividida em 3 tipos:

Características Distintivas

- **Improvisação orientada pela experiência**: decisões são tomadas com base na *intuição* e *vivência prévia* do testador.
- **Iteratividade contínua**: o testador aprende sobre o sistema enquanto o testa e ajusta os testes conforme descobre novos comportamentos.

- **Documentação adaptativa**: os resultados podem ser registrados conforme necessário, geralmente em forma de anotações informais ou relatórios dinâmicos.
- **Abordagem centrada no ser humano**: confia-se no julgamento do testador, ao invés de depender apenas de scripts automatizados.

Quando Utilizar?

- Em **fases iniciais** do projeto, quando há pouca documentação.
- Para **descobrir falhas inesperadas**, que não seriam facilmente identificadas com testes tradicionais.
- Quando se deseja **avaliar a usabilidade**, comportamento e estabilidade do sistema sob diversas interações humanas.

CheckList

Os testes são organizados por uma **lista de verificação estruturada** contendo itens, perguntas ou critérios que o testador deve seguir ou validar durante a execução dos testes. Ele é utilizado para guiar o processo de verificação, garantindo que pontos importantes não sejam negligenciados.

Ele os checklist usam uma estrutura de Sessão que definem o objetivo de cada teste:

Estrutura de Sessão

1. **Charter** (Objetivo)

```
"Explorar comportamento do checkout com cupons inválidos"
```

2. **Timebox**
   - 90 minutos
   - ⏸️ Pausas a cada 25 min
3. **Resultados**:
   - Bugs encontrados
   - Áreas de risco identificadas
   - Novos cenários para teste

Ataque

uma abordagem onde testamos os oponentes do produto atacando eles. Uma ótima analogia seria ver se um carro realmente está a prova de balas atirando no próprio carro, com o detalhe que o carro está em fase desenvolvimento.

Objetivo

- Encontrar falhas rapidamente.
- Validar a robustez do sistema contra entradas inesperadas ou uso incorreto.
- Expor **cenários não cobertos** por testes funcionais formais.

Exemplos de Ataques

- Inserir dados absurdos, como strings com milhares de caracteres em campos curtos.
- Forçar uso incorreto da interface: cliques múltiplos, sequências rápidas de comandos, ações fora da ordem lógica.
- Testar limites de carga, como usuários simultâneos além do previsto.
- Inserir comandos maliciosos em campos de entrada (ex: scripts ou SQL injection, em testes de segurança).

Técnica de Ataque

Um exemplo de checklist que pode orientar ataques de teste:

Item	Descrição
Entrada inválida	O sistema trata entradas fora do padrão?
Fluxo inesperado	O que acontece se o usuário quebrar a sequência normal de uso?
Ações repetidas	O sistema responde bem a comandos duplicados rapidamente?
Segurança	Campos de entrada aceitam scripts, comandos ou SQL?
Limites	E se o campo de upload receber um arquivo de 5GB?
📉Estresse	O sistema aguenta 1000 usuários simultâneos?

Escrita colaborativa de história de usuários

Abordagens de teste baseadas na colaboração

As histórias de usuários tem três aspectos principais – Chamado de Conjunto 3c

Cartão: meio onde se escreve a história de usuário – mais comum ser escrito no Jira, Trello, Notion

Conversação: explicação de como o software será usado – Pode ser documentado ou verbal

Confirmação: são os critérios de aceite, onde é confirmado o que o software deverá fazer

Formato de como se escreve a história de usuário:

Como (ator ou persona) médico

Quero(meta a ser cumprida) acessar o histórico de um paciente

Para que eu possa(valor de negócio resultante para a função) ver como atender melhor um paciente

Pode-se usar brainstorming ou mapas mentais para criar as histórias de usuários;

Boas histórias de usuários devem ser: independente, negociável, valiosa, estimável, pequena e testável – Técnica Invest

Se um stakeholder não souber como testar a história de usuário deve se avaliar se ela realmente tem algo valioso ou se ele está por dentro do software.

Critérios de aceite

São as condições que a implementação da história de usuários deve atender para ser aceita.

Os critérios de aceite são resultados da conversação entre a equipe.

São usados para:

Definir o escopo da história de usuário;

Chegar a um consenso entre os stakeholders;

Descrever os cenários positivos e negativos;

Servir como base para o teste de aceite da história do usuário;

Forma de escrever critérios de aceite:

Orientado a cenários – no formato BDD

Orientado por regras - Lista de pontos de verificação

ATDD – Desenvolvimento Orientado por Teste de Aceite

É uma abordagem que prioriza o teste.

Os casos de teste são criados antes da implementação da história do usuário.

São criados por clientes, desenvolvedores e testadores, podendo ser executados de forma manual ou informatizada.

Linha do tempo de como Funciona o processo no ATDD

Primeiro: Criação da história de usuário (definições e critérios de aceite);

Segundo: Criação dos casos de teste de aceite – são baseados no passo anterior, pode ser feito pela equipe ou só pelo testador, ajudam na implementação correta da história;

Terceiro: Desenvolvimento (é feito baseado nos critérios de aceite, mas a cada pedaço que o desenvolvedor vai desenvolvendo ele usa o TDD, como boa prática de qualidade);

Quarto: Testes – automação dos testes, execução dos testes e testes exploratórios.