




# Aprofundando em Análise de Testes e Report de Issues

## Pirâmide de Testes [↗](#)

A **pirâmide de testes** é uma metáfora visual criada por Mike Cohn para representar a estratégia ideal de automação de testes em diferentes camadas de um sistema. O objetivo é alcançar qualidade com eficiência, otimizando tempo e recursos.

### Estrutura da Pirâmide: [↗](#)

1 vbnet			
 Testes End-to-End (UI) completos	- Mais lentos	- Menor quantidade	- Testam fluxos
 Testes de Integração entre componentes	- Médios em número e performance	- Verificam comunicação	
 Testes Unitários	- Maioria dos testes	- Mais rápidos e baratos	
- Validam partes isoladas do código			

### Aplicação em Front-End: [↗](#)

- **Testes unitários:** validam componentes isolados (ex: botão, input).
- **Testes de integração:** testam componentes em conjunto (ex: formulário com validação).
- **Testes E2E (End-to-End):** simulam interações reais do usuário no navegador (ex: Cypress, Playwright).

**Boa prática:** mantenha muitos testes unitários, um número moderado de testes de integração e poucos E2E focados nos fluxos críticos.

## 2. Gestão de Issues [↗](#)

A **gestão de issues** é o processo de registrar, acompanhar, priorizar e resolver problemas identificados durante os testes. Ferramentas como Jira, GitHub Issues, Azure DevOps e Linear são amplamente usadas para isso.

### Ciclo de Vida de uma Issue: [↗](#)

1. **Aberta (Open):** o bug é reportado.
2. **Em análise (Triage):** a equipe avalia severidade/prioridade.
3. **Em progresso (In Progress):** o time de desenvolvimento começa a correção.
4. **Para testar (To Test):** o QA verifica a correção.
5. **Fechada (Closed):** issue resolvida e validada.
6. **Reaberta (Reopened):** se o bug persistir.

### Boas Práticas: [↗](#)

- Crie um **template de issue** com:
  - Título claro
  - Passos para reproduzir
  - Resultado atual x esperado
  - Prints ou vídeos

- Ambiente/teste usado
  - Utilize **labels** como `bug`, `enhancement`, `critical`, `frontend`, etc.
  - Acompanhe através de **quadros Kanban ou Scrum**.
- 

### 3. Como Aprender a Reportar Bugs de Forma Eficiente [↗](#)

#### 1. Clareza é tudo: [↗](#)

- Evite termos vagos como “não funciona”.
- Use verbos diretos e específicos: "Erro ao clicar no botão 'Salvar'".

#### 2. Descreva os passos para reproduzir: [↗](#)

- Exemplo:

```
1  css
```

```
1. Acesse a página de login 2. Insira e-mail válido 3. Deixe o campo senha em branco 4. Clique em "Entrar"
Resultado: aplicação retorna erro 500 Esperado: exibir validação de campo obrigatório
```

#### 3. Utilize evidências: [↗](#)

- **Prints, GIFs e vídeos** ajudam desenvolvedores a entender o problema.
- Ferramentas úteis: Loom, OBS, Lightshot, DevTools (Network e Console).

#### 4. Seja colaborativo: [↗](#)

- Evite linguagem acusatória.
- Inclua sugestões, se possível.

**Dica:** faça triagem junto com o time de produto para garantir que o que é bug ou melhoria esteja bem definido.

---

### 4. Gestão do Ciclo de Vida dos Testes [↗](#)

O **ciclo de vida de testes de software** é o processo completo de planejamento, criação, execução e encerramento dos testes.

#### Etapas principais: [↗](#)

##### 1. Planejamento de Testes:

- Definição de escopo, critérios de entrada/saída e responsabilidades.

##### 2. Análise de Testes:

- Estudo dos requisitos para identificar o que deve ser testado.

##### 3. Design de Casos de Teste:

- Criação de cenários e scripts baseados nos requisitos.

##### 4. Implementação e Execução:

- Execução manual ou automática dos testes.

##### 5. Registro e Análise de Resultados:

- Verificação de falhas, logs, prints.

##### 6. Encerramento de Testes:

- Relatório final com cobertura, métricas e lições aprendidas.
-

## 5. Análise de Riscos com PRISMA [↗](#)

**PRISMA (PRocessus Instrumenté de Suivi des Modes de défaillance par Arbre)** é uma técnica usada para análise qualitativa de riscos.

### Etapas adaptadas para testes de software: [↗](#)

#### 1. Identificação dos riscos:

- Ex: falhas em login, perda de dados, não conformidade com LGPD.

#### 2. Categorização dos riscos:

- Técnicos, de negócio, de segurança, de acessibilidade etc.

#### 3. Análise de impacto e probabilidade:

- Alta, Média ou Baixa.

#### 4. Priorização de testes com base nesses riscos.

### Exemplo em Front-End: [↗](#)

Risco	Probabilidade	Impacto	Ação
Login inválido não mostra erro	Alta	Média	Teste de validação
Quebra de layout no Safari	Média	Alta	Teste cross-browser

---

## 6. Testes Baseados em Riscos (Risk-Based Testing) [↗](#)

### Conceito: [↗](#)

Testar **primeiro o que pode quebrar mais facilmente** ou ter **maior impacto para o negócio**.

### Benefícios: [↗](#)

- Maximiza cobertura com menos esforço.
- Ajuda a priorizar testes e recursos.

### Ferramentas: [↗](#)







#### Matriz de Rastreabilidade: [↗](#)

Rastreia requisitos → riscos → casos de teste → bugs

Requisito	Risco Associado	Caso de Teste	Bug Encontrado
Login	Falha de autenticação	CT01 - Login válido	BUG-001

#### Matriz de Impacto: [↗](#)

Avalia a gravidade dos riscos e a probabilidade de ocorrência.

Impacto \ Probabilidade	Alta	Média	Baixa
Alta			
Média			

Baixa			
-------	---	---	---



Testes front-end para áreas vermelhas  devem ser priorizados.

## Aprofundamento em Testes Front-End [↗](#)

### Características específicas: [↗](#)

- Alto risco de regressão visual
- Dependência de comportamento do usuário
- Sensível a ambientes (navegador, tela, resolução)

### Tipos de testes aplicáveis: [↗](#)

1. **Testes Visuais:** (ex: Percy, Applitools)
  - Detectam diferenças de pixels entre builds.
2. **Testes de Responsividade:**
  - Validam a adaptação do layout em diferentes dispositivos.
3. **Testes de Acessibilidade:** (ex: Axe, Lighthouse)
  - Verificam uso de ARIA, contraste, navegação por teclado.
4. **Testes Funcionais Automatizados:** (ex: Cypress, Playwright)
  - Validam fluxos: login, filtros, carrinhos etc.
5. **Testes de Usabilidade:** com usuários reais ou ferramentas como Hotjar.

### Boas práticas: [↗](#)

- Automatize os fluxos principais.
- Faça testes exploratórios com foco em acessibilidade e comportamento inesperado.
- Valide feedbacks visuais (tooltips, loaders, alerts).

## Conclusão [↗](#)

A garantia de qualidade no front-end exige organização, estratégia e empatia com o usuário final. Ao dominar a pirâmide de testes, reportar bugs com clareza, gerenciar riscos com foco em impacto e rastreabilidade, e aprofundar os testes na interface, o QA consegue gerar um valor real para o produto.