Checklist de planejamento de testes

@Raique Alfredo Pereira de Ramos

@Cassia Yumi

@Thais Nogueira

@Ádony Lagares

🖋️ Resumo Inicial

Antes de montar qualquer plano de testes, é essencial entender bem o que será entregue. Os requisitos funcionais (o que o sistema faz) e os não funcionais (como o sistema se comporta) precisam estar claros e bem documentados, porque eles são a base para definir os testes.

Para os requisitos funcionais é importante identificar as funcionalidade críticas que precisam ser testadas. Já para os requisitos não funcionais é importante identificar as métricas de desempenho que são necessárias avaliar.

As estórias dos usuários ajudam bastante a direcionar os cenários de teste. A estrutura "Como [tipo de usuário], eu quero [ação] para [benefício]" facilita a escrita e também a priorização do que é mais importante validar.

Uma ferramenta que ajuda muito no início do planejamento é o mapa mental, pois permite organizar as funcionalidades, visualizar dependências e levantar possíveis riscos de forma rápida e visual.

📄 Estrutura do Plano de Testes

O plano de testes precisa deixar claro:

- O que será testado (escopo)
- O que não será testado (fora de escopo)
- Tipos de testes (funcionais, não funcionais, regressão, etc.)
- Cronograma de testes
- Critérios de entrada e saída, incluindo o resultado esperado
- Ambientes de teste que serão utilizados
- Ferramentas e recursos que serão aplicadas e utilizados
- Responsáveis por cada etapa
- Possíveis riscos e como tratá-los

⚠️ Riscos e Priorização

É importante fazer uma análise de riscos logo no início. Mapear o que tem maior chance de dar problema ou causar impacto ajuda a definir o foco dos testes. A matriz de risco é útil para isso, cruzando impacto e probabilidade. Precisa também definir as estratégias de tratamento dos riscos, se ele será evitado, mitigado, aceito (ativa ou passivamente) ou transferido.

Para priorizar os testes, algumas técnicas que valem a pena considerar:

- Análise de Pareto: focar nos 20% que causam 80% dos problemas
- Priorização por Impulso: rápida e baseada na percepção sobre o que é mais crítico
- Priorização por Atração: baseia no que é mais utilizado ou desejado pelo usuário
- Análise da Cauda Longa: cuida de partes menos usadas, mas que ainda precisam funcionar bem

- Pairwise Testing: combinações de entradas e testes eficientes para cobrir vários cenários

Essas técnicas ajudam a usar melhor o tempo e os recursos disponíveis, especialmente quando nem tudo pode ser testado.

🚤 Integração com o Processo Ágil

O plano de testes precisa acompanhar o ritmo do projeto. É importante manter alinhamento com os artefatos ágeis, como:

- Product Backlog
- Sprint Backlog
- Épicos
- Features

O ideal é que os testes sejam pensados desde o início das sprints e adaptados conforme as prioridades mudam. A integração entre planejamento e execução precisa ser constante.

Também vale usar o Planning Poker para estimar o esforço necessário para cada teste. Mesmo que essa técnica seja mais comum para desenvolvimento, ela funciona bem para testes também.

Análise, Modelagem e Implementação

- **O que é Análise de Teste?**
  Entender o que deve ser testado, com base nos requisitos, riscos e impacto.
- **Modelagem:**
  Definir e organizar os cenários de teste a partir da análise feita. A modelagem transforma o entendimento em estrutura testável.
  - Lógica: também chamada de genérica ou tradicional. A modelagem lógica é rápida e barata e não é viciada (serve para vários testes). Contudo, faltam dados para execução, sendo ruim para fazer em caso de equipes dispersas.
  - Concreta: também chamado de específico ou físico. Ele detalha o que deve ser escrito para acessar o site, por exemplo, qual e-mail ou senha que devem ser digitados. Ele pode ter passos mais detalhados também. Prós: pode ser usado para equipes novas ou dispersas; mantém o conhecimento na empresa (se a equipe se desfizer, outra pode testar). Contras: ele é demorado de fazer, mais caro, é viciado (porque limita o que pode ser testado).
  - Data Driven: muda os dados, mas o teste é o mesmo.
  - Keyword Driven: usa palavras-chave como comandos, facilitando a reutilização.
- **Implementação:**
  Quando os testes são de fato escritos ou programados. Aqui entram os testes automatizados ou roteiros manuais.
- **Cenário: Ambiente:**
  Descreve onde o teste vai acontecer (sistema, navegador, versão, etc). Isso evita diferenças de comportamento por ambiente.
- **Cenário: Situações:**
  Define as condições e combinações em que os testes vão rodar (ex: usuário com perfil X, ação Y, resultado Z).
- **Suítes de Teste:**
  Agrupamento lógico dos casos de teste, por funcionalidade ou módulo. Isso ajuda na organização e execução.
- **Casos de Teste:**
  Cada caso descreve um caminho específico para testar uma funcionalidade. É o nível mais detalhado do plano.
- **TDD, BDD, ATDD:**
  Formas de guiar o desenvolvimento com base em testes:
  - **TDD**: escreve o teste antes do código.
  - **BDD**: foca no comportamento esperado, geralmente com linguagem acessível ao cliente.
  - **ATDD**: validação colaborativa entre cliente, dev e QA com base em testes desde o início.

- **Testes Unitários:**
  Validam partes específicas do código (funções, métodos). Garantem que cada componente funciona isoladamente.

👔 O que não pode faltar no plano de testes

- Requisitos (funcionais e não funcionais) bem definidos
- Estórias de usuário claras
- Escopo definido
- Orçamento/Recursos definidos
- Tipos de testes estabelecidos
- Matriz de risco montada
- Técnicas de priorização aplicadas
- Casos de teste organizados em suítes
- Modelagem e implementação previstas
- Estratégias como TDD, BDD ou ATDD se aplicável
- Testes automatizados (quando possível)
- Ferramentas e ambiente definidos
- Estimativas de esforço
- Integração com o processo ágil
- Documentação acessível

Seguindo esse processo, o processo de testes fica mais organizado, eficiente e conectado com os objetivos do projeto. Além de garantir mais qualidade na entrega, ajuda a reduzir retrabalho e surpresas de última hora.