Capítulo 2 - Teste ao Longo do Ciclo de Vida de Desenvolvimento de Software

2.1 Testes no contexto de um Ciclo de Vida de Desenvolvimento de Software

2.1.1 Impacto do ciclo de vida de desenvolvimento de software em testes

O teste deve ser adaptado ao **SDLC(Ciclo de Vida de Desenvolvimento de Software).** A escolha do SDLC tem impacto sobre:

→**Escopo e cronograma** das atividades de teste;

→O **nível de detalhamento** da documentação de teste;

→A **escolha das técnicas** de teste e da abordagem de teste

→**A extensão da automação** de testes;

→O **papel e responsabilidade** de um Testador;

2.1.2 Boas práticas de teste, independente do modelo SDLC

- Para **cada atividade de desenvolvimento, há uma ativdade de teste correspondente**.
- Diferentes **níveis de teste têm objetivos de teste específicos** e diferentes, evitando redundância.
- A **análise e a modelagem do teste começam durante a fase de desenvolvimento** correspondente do SDLC, princípio do teste antecipado.
- Os **testadores estão envolvidos na revisão dos produtos de trabalho** assim que os rascunhos dessa documentação estiverem disponíveis, de modo que esse teste antecipado e a detecção de defeitos possam apoiar a estratégia shift-left(teste antecipado).

2.1.3 Teste como um motivador para o desenvolvimento de software

Esse subtópico apresenta três práticas em que os testes são usados para **direcionar o desenvolvimento do software**. Todas elas seguem o princípio do **teste antecipado** (testar o mais cedo possível) e a abordagem **shift left**, trazendo os testes para as fases iniciais do desenvolvimento.

TDD (Test Driven Development)

No TDD, o desenvolvedor primeiro escreve um teste automatizado que falha, depois cria o código necessário para que o teste passe e, por fim, refatora o código para melhorar sua

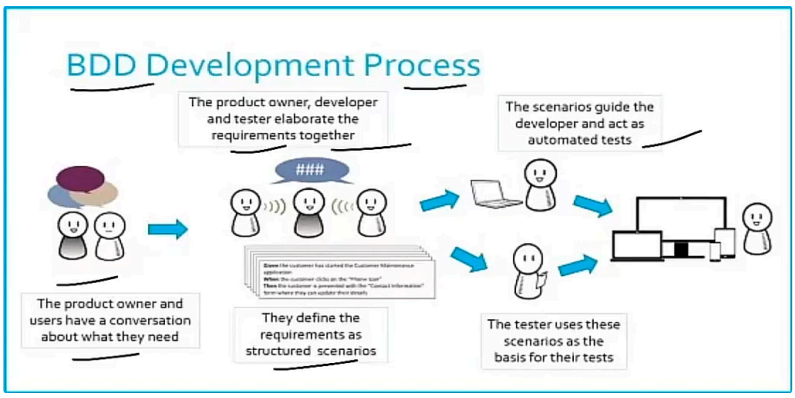
qualidade. Esse ciclo é repetido continuamente. É muito usado para testes de unidade, focando em pequenos trechos de código. Essa prática ajuda a evitar erros e melhora a manutenibilidade.

ATDD (Acceptance Test Driven Development)

No ATDD, os testes são baseados nos critérios de aceite das histórias de usuário e são escritos antes do desenvolvimento. Eles ajudam a garantir que o que está sendo desenvolvido atende exatamente ao que foi definido pelo cliente. O time todo participa da definição dos testes de aceite. Esses testes servem como referência durante o desenvolvimento e como validação ao final.

BDD (Behavior Driven Development)

O BDD é semelhante ao ATDD, mas se destaca por escrever os testes de forma simples e natural, geralmente usando a estrutura "Dado que / Quando / Então". Essa linguagem facilita a comunicação entre todos os envolvidos, inclusive pessoas não técnicas. Os cenários escritos podem ser convertidos em testes automatizados. O BDD deve envolver toda a equipe, e não apenas os testadores.



Essas três abordagens favorecem a qualidade desde o início, promovem melhor entendimento dos requisitos e reduzem retrabalho.

2.1.4 DevOps e Testes

DevOps é uma **abordagem organizacional** que visa criar a singergia, fazendo com que o desenvolvimento (incluindo os testes) e as operações **trabalhem juntos**.

- DevOps promove
  - **Autonamia** da equipe
  - **Feedack rápido**
  - **Ferramentas** integradas
  - Práticas técnicas como

        - Integração Contínua (CI - Continous Integration)
        - Entrega Contínua (CD - Continous Delivery)
    - Isso permite que as equipes criem, testes e liberem códigos de alta qualidade mais rapidamente por meio de um pipeline de entrega DevOps
- Benefícios do DevOps para Testes
    - Feedback rápido sobre a **qualidade do código** e se as alterações afetam negativamente o código existente;
    - O CI promove uma abordagem shift-left(no inicio do ciclo) nos testes, incentivando os desenvolvedores a enviar códigos de alta qualidade acompanhados de testes de componentes e análise estática;
    - Promove **processos automatizados**, como CI/CD, que facilitam o estabelecimento de ambientes de teste estáveis;
    - Aumenta a visão das **características de qualidade não funcionais** (ex: performance, confiabilidade);
    - A automação por meio de um pipeline de entrega **reduz a necessidade de testes manuais repetitivos;**
    - O risco na **regressão é minimizado** devido à escala e ao alcance dos testes de regressão automatizados;
- Riscos e Desafios do DevOps
    - O **pipeline de entrega** de DevOps deve ser definido e estabelecido;
    - As **ferramentas** CI/CD devem ser **introduzidas e mantidas**;
    - A automação de testes requer recursos adicionais e **pode ser difícil de estabelecer e manter.**
        - **DevOps necessita de um alto nível de testes automatizados,** porém testes manuais ainda são necessários (principalmente da perspectiva do usuário).

2.1.5 Abordagem Shift-Left

É um sinônimo do princípio do teste antecipado.

Left(Esquerda), pois adiciona o teste nas etapas iniciais (normalmente ilustrada à esquerda).

O Shift-Left sugere que **os testes devem ser feitos mais cedo**(ex: não esperar que o código seja implementado ou que os componentes sejam integrados).

Mas isso **não significa que os testes posteriores no SDLC(Ciclo de Desenvolvimento) devam ser negligenciados.**

Testes executados mais cedo vão **otimizar os testes das outras etapas**, pois tendem a encontrar defeitos com antecedência.

- **Boas práticas:**
  - **Revisão da especificação sob a perspectiva de testes:** Essas atividades de revisão das especificações geralmente encontram possíveis defeitos, como ambiguidades, incompletude e inconsistências;
  - **Escrever casos de teste antes de o cóigo ser escrito**: Fazer com que o código seja executado em conjunto de testes durante a sua implementação;
  - **Usar a CI e CD,** pois ela vem com **feedback rápido** e testes de componente automatizados para acompanhar o código-fonte quando ele é enviado ao repositório de código;
  - **Concluir a análise estática do código-fonte antes do teste dinâmico** ou como parte de um processo automatizado;
  - **Realizar testes não funcionais começando no nível de teste do componente, sempre que possível:** Essa é uma forma de shift-left, pois esses tipos de **testes não funcionais tendem a ser realizados mais tarde** no SDLC, quando um **sistema completo** e um ambiente de teste representativo estão disponíveis

> *Pode ser necessário **treinamento, esforço e/ou custos adicionais** no inicio do processo, mas espera-se que economize esforços/ou custos no final do processo.*

2.1.6 Retrospectivas e melhorias no processo

2.2 Níveis de Teste e Tipos de Teste

NÍVEIS DE TESTE

Grupos de atividades de teste que são organizados e gerenciados juntos. Estão **relacionados a outras atividades** dentro do SDLC. Níveis de teste **podem se sobrepor** no tempo.

TIPOS DE TESTE

São grupos de atividades de teste **relacionadas a características de qualidade específicas** e a maioria dessas atividades de teste **pode ser realizada em todos os níveis de teste**.

2.2.1 Níveis de Teste

Os nívies de teste usados no syllabus são:

- **Testes de componente (unidade)**
- **Teste de integração entre componentes**
- **Teste de sistema**
- **Teste de integração entre sistemas**
- **Teste de aceite**

→Os níveis de teste são caracterizados pelos seguintes **atirbutos**:

- Objeto de teste (ou seja, o que está sendo testado);
- Objetivos do teste;
- Base de teste, referenciada para derivar casos de teste;
- Defeitos e falhas;
- Abordagens e responsabilidades específicas;

*Para cada nível de teste, é necessário um **ambiente de teste adequado.***

TESTE DE COMPONENTE (UNIDADE)

- É o teste na **menor parte testável** de um sistema (Função / Componente / Classe)
- Requer **acesso ao código-fonte**
- Normalmente **realizado pelo próprio desenvolvedor**
- Deve ser **isolado**, **sem integração nenhuma** com outro componente ou parte do sistema
- É um **teste muito rápido** de ser executado
- No ágil, esse teste pode ser feitos antes de desnvolver o código (TDD)
- Deve ser um **teste automatizado**
- Teste de regressão **automatizados** trazem **confiabilidade** nesse nível

Objetivos:

- Reduzir o risco;
- Verificar os comportamentos **funcional** e **não funcional** do componente;
- Construir a confiança na qualidade do compoente;
- Encontrar defeitos no componente;
- **Evitar que os defeitos espalhem pra níveis mais altos de teste.**

TESTE DE INTEGRAÇÃO

- **Foco na integração** entre componentes ou sistemas
- Requer **acesso ao código-fonte** (para integração entre componentes)

- Pode ser **realizado pelo desenvolvedor** (componentes) **ou testador** (sistemas)
- Quanto **maior o escopo** da integração **mais difícil é isolar os defeitos**, por isso a **integração contínua** é comum hoje em dia.

Objetivos do Teste de integração:

- Reduzir risco;
- Verificar se os comportamentos **funcionais e não funcionais** das interfaces estão projetados e especificados;
- **Construir confiança na qualidade das interfaces**;
- Encontrar defeitos (que podem estar nas próprias interfaces ou nos componentes ou sistemas);
- Evitar que os defeitos espalhem para níveis mais altos de teste.

> O teste de **INTEGRAÇÃO DE COMPONENTES** foca nas interações e interfaces entre componentes integrados. É executado após o teste de componente e **geralmente é automatizado**. No desenvolvimento iterativo e incremental, os testes de integração de componentes **geralmente fazem parte do processeo de integração contínua;**
>
> O teste de **INTEGRAÇÃO DE SISTEMA** concentra-se nas interações e **interfaces entre sistemas,** pacotes e microserviços. O teste de integração do sistema também pode abranger interações e interfaces fornecidas por **entidades externas** (ex: serviços da web).

TESTE DE SISTEMA

Teste do Sistema em si sendo executando, próximo ao cenário do usuário final.

- Fluxos de ponta a ponta(end-to-end)
- Produz informações que são usadas para **tomar decisões** de liberação
- Tambem pode verificar **requisitos legais**/regulatórios
- Testes de regressão **automatizados** trazem confiabilidade nesse nível

Objetivos:

- Reduzir o risco;
- Verificar se os componentes **funcionais** e **não funcionais do sistema** estão como projetados e especificados;
- Valdiar se o sistema está completo e funcionará como esperado;
- **Criar confiança na qualidade do sistema como um todo;**
- Encontrar defeitos

- Evitar que os defeitos espelhem para níveis mais altos de teste ou produção.

TESTE DE ACEITE

Concentra-se na validação e na demonstração dos sistema. Preferencialmente, o teste de aceite deve ser realizado pelos usuários previstos.

Não tem como objetivo encontrar defeitos mas sim **garantir que o sistema esteja funcionando da maneira como especificado** para o usuário final. Defeitos encontrados durante esse teste são considerados **grandes riscos** para o projeto.

Responsáveis: clientes, usuários de negócios, proprietários de produtos, operadores de um sistema e stakeholders.

Tipos de Teste de Aceite:

→ **Teste de Aceite de Usuário (UAT)**

- Certificar que o sistema atenda as necessidades do usuário.
- Validar o uso do sistema por usuários em um ambiente de produção simulado

→ **Teste de Aceite Operacional (OAT)**

Garantir que os operadores ou administradores do sistema possam manter o sistema funcionando adequadamente para os usuários no ambiente de produção.

- São realizados testes como:
  - Backups e Restauração
  - Performance
  - Vulnerabilidades de Segurança

→ **Teste de Aceite Contratual e Regulatório**

Garantir que a conformidade contratual ou regulatória foi alcançada.

- Critérios devem ser definidos quando as partes assinam o contrato para o desenvolvimento do Software.
- Resultados podem ser acompanhados por testemunhas ou agências reguladores.

Alfa

- Realizado no site(local) da organização.
- Pode ser feitos por clientes, operadores ou testadores independentes.

Beta

- Testes realizados fora da organização, ou seja em local próprio.
- Pode ser feito por clientes em potencial ou operadores;

O que é um Teste Funcional

“O que” o sistema deve fazer

Avalia as **funções** em que o sistema deve executar

Devem ser realizados em **todos os níveis de teste**

Eficácia medida atráves da **cobertura funcional**

Pode exigir conhecimentos especiais na **regra de negócio**

O que é um Teste Não Funcional

“Quão bem” o sistema se comporta

Avalia as características de um software (ex: usabilidade, desempenho, segurança)

Pode exigir conhecimentos especiais na tecnologia(para vulnerabilidades, desempenho, etc.)

Teste de Caixa Preta

Baseado nas **especificações do sistema**

Deriva testes da documentação externa ao objeto de teste

O principal é **verificar o comportamento do sistema em relação às suas especificações**

Teste de Caixa Branca

Baseado na **estrutura interna** do sistema (código, arquitetura, etc)

Eficácia medida através da **cobertura estrutural** (código, interfaces entre componentes)

Pode exigir conhecimento especiais em como código é construido