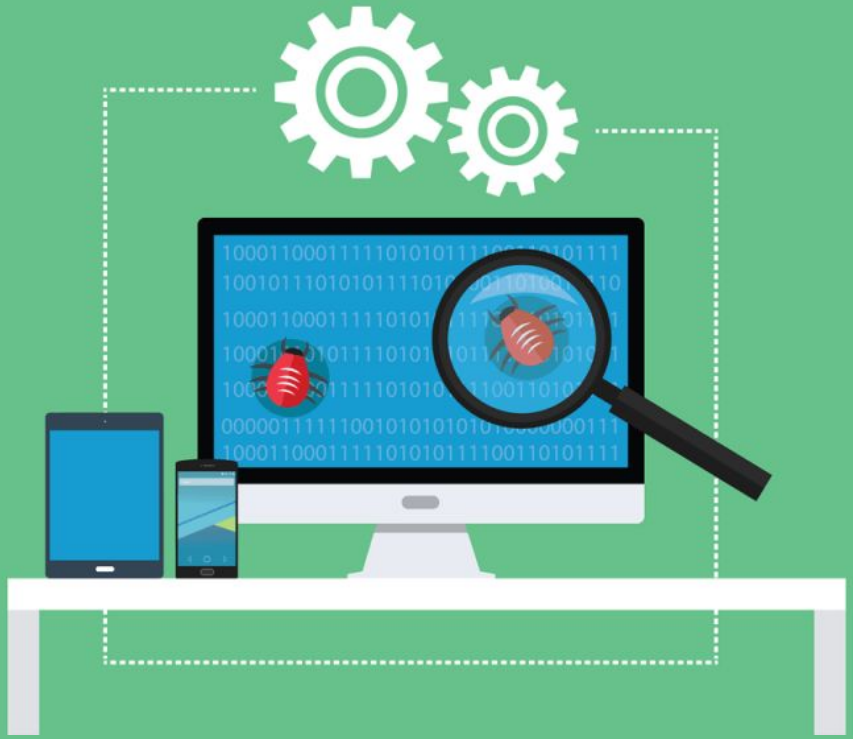


UniCesumar

Projeto, implementação e Teste de Software

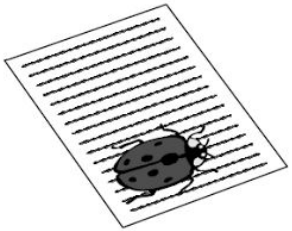
O Papel dos Testes Automatizados no Processo de Integração Contínua.

Prof. Esp. Dacio F. Machado

Aula de Hoje

UNIDADE V - Ferramentas de Teste de Software

- **O Papel dos Testes Automatizados no Processo de Integração Contínua.**
  - ❑ **Relembrando Testes Automatizados**
  - ❑ **Processo de Desenvolvimento e Integração Contínua (CI)**
  - ❑ **Integração Contínua com Testes Automatizados**
  - ❑ **Unindo os conceitos**
  - ❑ **Vantagens e Desafios**
  - ❑ **Técnicas e Ferramentas**



Relembrando Testes Automatizados

- **Conceito**

A automação de testes consiste em testar um software com outro software. Algo parecido com diversos robôs (vários scripts) construídos para usar o sistema no lugar dos usuários. Isso é válido pois podem ser mais rápidos na execução dos testes e detecção dos erros e trabalham em uma escala maior.

Os propósitos da automação de testes:

- Eleva a consistência e a cobertura dos testes.
- Diminui o tempo ou o esforço necessário para testar.
- Reduz os gastos relacionados aos testes.
- Melhora a eficiência no desenvolvimento de software de forma geral.
- Contribui para a elevação da qualidade do produto final.

Características Dos Testes Automatizados

1. **Conciso**: deve ser simples e direto, sempre que possível.
2. **Explícito**: deve relatar claramente quaisquer falhas ou desvios.
3. **Replicável**: pode ser executado várias vezes com os mesmos resultados.
4. **Robusto**: resiste a interferências externas e produz resultados consistentes.
5. **Necessário**: detecta divergências entre o esperado e o implementado.
6. **Clareza/Manutenção**: possui código compreensível e fácil de manter.
7. **Eficiente**: tem bom desempenho durante a execução.
8. **Independente**: não depende de outros testes e pode ser executado isoladamente.
9. **Rastreável**: está ligado diretamente aos requisitos e suas origens.

Processo De Automação De Testes

1. **Decisão pela automação**: avaliar o retorno sobre o investimento, compreender os tipos e abordagens de teste para evitar erros.
2. **Escolha da ferramenta**: estabelecer critérios para selecionar a ferramenta de automação, destacando seus benefícios, limitações e restrições.
3. **Implantação na organização**: aplicar procedimentos e boas práticas que orientarão a automação dentro da empresa.
4. **Planejamento e desenvolvimento dos testes automatizados**: estruturar e criar os testes automáticos para o projeto.
5. **Execução e monitoramento**: rodar os testes criados e coletar métricas como cobertura, progresso e eficiência.
6. **Avaliação e aperfeiçoamento**: realizar análises para identificar melhorias no processo de automação.

Relembrando o Processo de Desenvolvimento

O ciclo de vida do desenvolvimento de software (SDLC) é um processo estruturado que as equipes de desenvolvimento seguem para criar software com qualidade, de forma econômica e segura. Suas etapas são:

- **Análise**
- **Planejamento**
- **Design (Projeto)**
- **Implementação (Codificação)**
- **Testes**
- **Implantação (Entrega)**
- **Manutenção**

Essas fases estão geralmente conectadas e podem ser realizadas em sequência ou simultaneamente, conforme o modelo adotado.

Modelos de desenvolvimento de software

- **Cascata:** Segue etapas sequenciais rígidas, sendo adequado a projetos simples e bem definidos.
- **Modelo em V:** Associa testes a cada fase do desenvolvimento em uma estrutura linear semelhante à cascata.
- **Iterativo:** Repete ciclos de desenvolvimento para refinar funcionalidades com base em versões anteriores.
- **Ágil** Divide o projeto em sprints com entregas rápidas e feedback contínuo.
- **DevOps** Integra desenvolvimento e operações com foco em automação, colaboração e entregas contínuas. O DevOps adiciona novos processos e ferramentas à metodologia ágil, principalmente a automação de grande parte do pipeline de CI/CD.

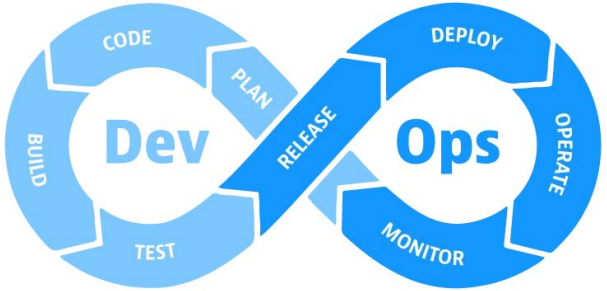
Integração Contínua (CI)

A integração contínua (CI) é um processo de desenvolvimento de software em que os desenvolvedores integram novo código à base de código durante todo o ciclo de desenvolvimento.

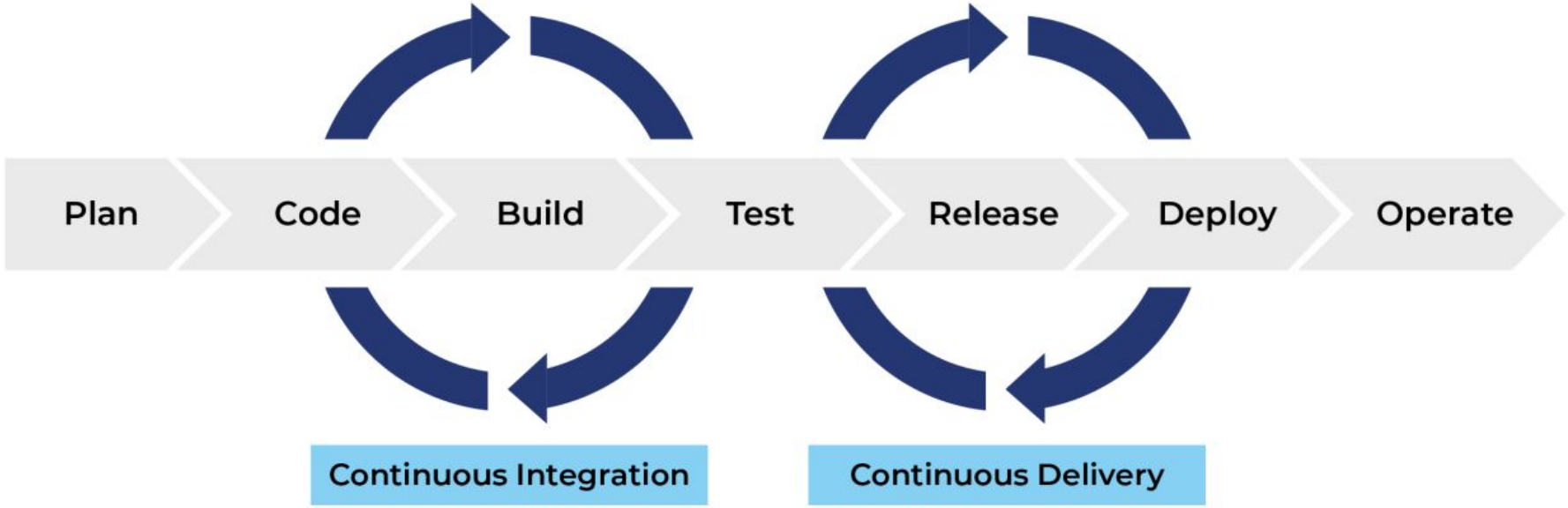
A **CI** foi uma resposta aos desafios dos processos associados à integração e implementação e são fundamentais para as práticas modernas de DevOps.

Ajudam a simplificar o processo de criação, fornecendo feedback rápido sobre o desempenho da integração.

Faz com que as equipes detectem e corrijam erros antes que eles afetem o software



CI/CD


Plan
Code
Build
Test
Release
Deploy
Operate
Continuous Integration
Continuous Delivery

Principais componentes e processos de CI

- Repositórios centrais de código fonte
  - sistemas de controle de versão (VCSs), como o Git e o Bitbucket.
- Servidores de integração contínua
  - configuráveis para compilar os projetos em diferentes plataformas, centralizam as operações e oferecem uma base estável e confiável para o desenvolvimento de software.
- Integração de código
  - alterações de código várias vezes ao dia, priorizando pequenas alterações focadas em tarefas ou funcionalidades específicas.

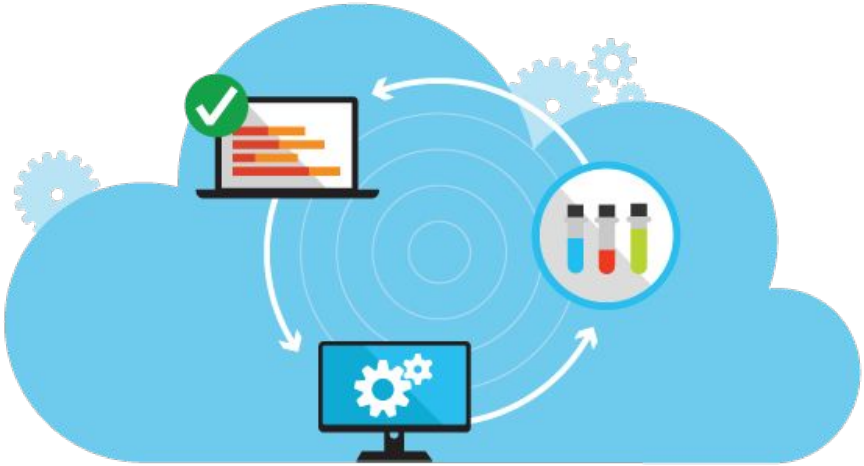
Principais componentes e processos de CI

- Crie automação
  - Servidores e ferramentas de integração contínua (Jenkins,GitHub, AWS CodePipeline e GitLab CI) monitoram o repositório central em busca de alterações no código. Quando encontradas rotinas de automação de processos são disparadas.
- **Testes automatizados**
  - executar uma série de testes para validar o código antes que ele seja mesclado com a base de código.

Os testes são um componente vital dos processos de integração contínua.

A importância de testar o código na CI

Testes automatizados são um dos pilares da integração e implantação contínuas. Ao incorporar testes contínuos aos seus pipelines de DevOps, você pode melhorar dramaticamente a qualidade do seu software.



Integração Contínua com Testes Automatizados

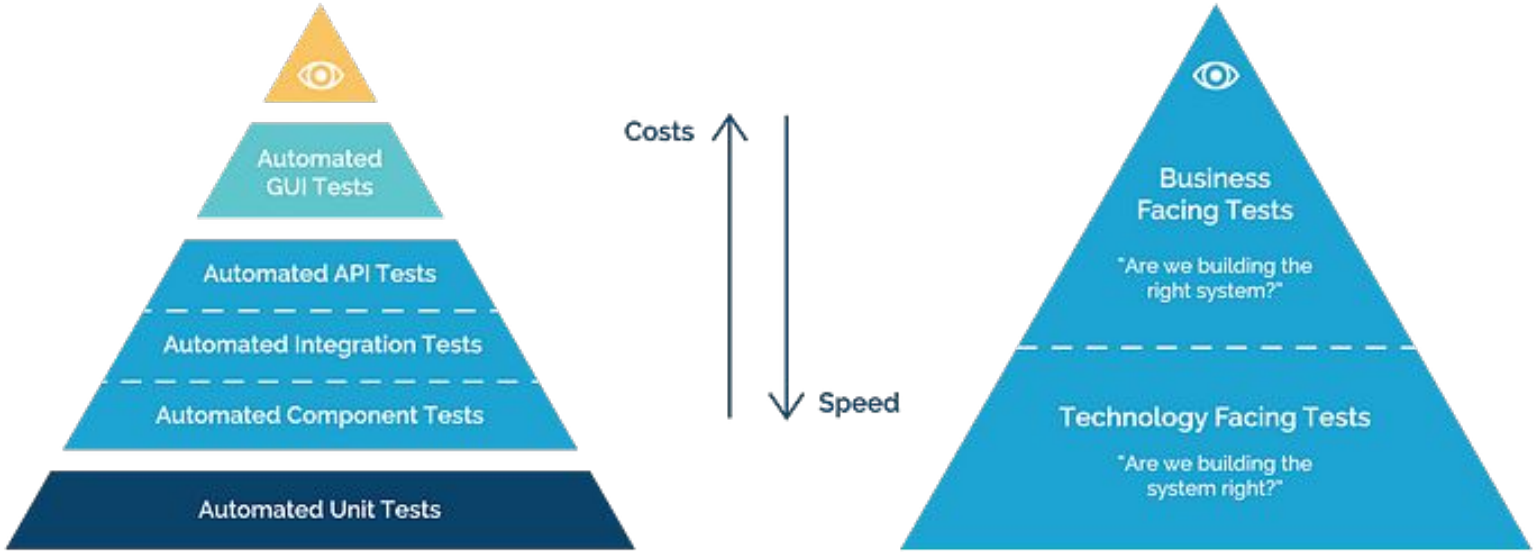
- Para que DevOps e a CI funcione bem, é essencial realizar commits, builds e testes frequentes, porém não é realista executar um conjunto completo de testes manuais uma ou mais vezes ao dia.

- As equipes de DevOps utilizam testes automatizados para testar continuamente o novo código ao longo do processo de desenvolvimento. É por isso que os testes automatizados são uma parte essencial de qualquer pipeline de CI/CD.

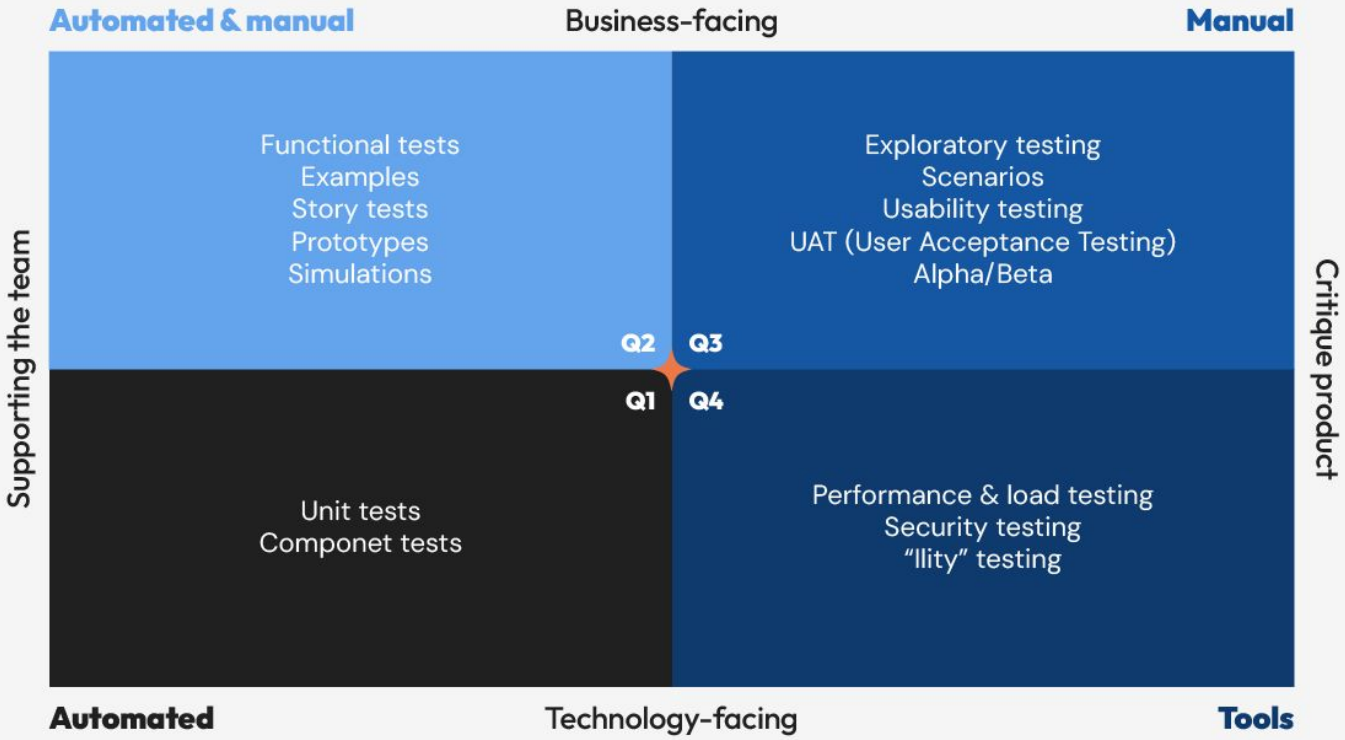
Integração Contínua com Testes Automatizados

- Para que os testes automatizados sejam confiáveis, é essencial que sejam executados sempre da mesma forma, com ambientes de teste consistentes e atualizados conforme as mudanças na produção.

- A pirâmide de testes é uma ferramenta para priorizar testes automáticos em um pipeline de CI/CD, em termos tanto do número relativo de testes quanto da ordem em que eles são executados.


Automated
GUI Tests
Automated API Tests
Automated Integration Tests
Automated Component Tests
Automated Unit Tests
Costs
Speed
Business
Facing Tests
"Are we building the
right system?"
Technology Facing Tests
"Are we building the
system right?"

Agile testing quadrants



Vantagens e Desafios

Validação contínua: cada mudança no código é verificada para evitar falhas e regressões.

Feedback rápido: os testes retornam resultados mais rapidamente que os testes manuais.

Correções mais eficientes: bugs são resolvidos com mais facilidade logo após serem introduzidos.

Maior confiabilidade: elimina erros humanos ao repetir tarefas de teste.

Execução em paralelo: permite escalar os testes para ganhar tempo, conforme a infraestrutura disponível.

Vantagens e Desafios

Embora a automação de testes elimine muitas tarefas monótonas e repetitivas, ela não torna desnecessária a equipe de QA. Em vez de gastarem tempo em tarefas repetitivas, podem se concentrar em definir casos de teste, escrever testes automatizados e aplicar sua criatividade e engenhosidade em testes exploratórios.

Falsos positivos/negativos: testes instáveis ou mal definidos geram falhas injustificadas ou deixam passar erros reais.

Manutenção constante: mudanças no software exigem atualizações frequentes nos testes, aumentando o custo de manutenção.

Tempo de execução elevado: muitos testes ou testes lentos tornam o pipeline demorado e dificultam a agilidade.

Ambiente instável: dependências externas tornam os testes inconsistentes; é necessário isolar e estabilizar o ambiente.

Técnicas e Ferramentas

- Diversas ferramentas e frameworks estão disponíveis para facilitar a automação de testes.
- Entre as mais populares estão o JUnit, amplamente usado em projetos Java, e o NUnit, voltado para a plataforma .NET. Outras opções incluem Selenium, Cypress, TestComplete e Robot Framework, cada uma com suas particularidades e vantagens, dependendo do contexto e das necessidades do projeto.
- Na unidade anterior, vimos várias técnicas de teste, e os conceitos de ferramentas e técnicas são importante no processo de teste. É entender as técnicas de teste primeiro, para que consiga entender suas ferramentas
- Pois, a ferramenta é um recurso para o testador, mas sem a técnica é insuficiente para conduzir os testes.

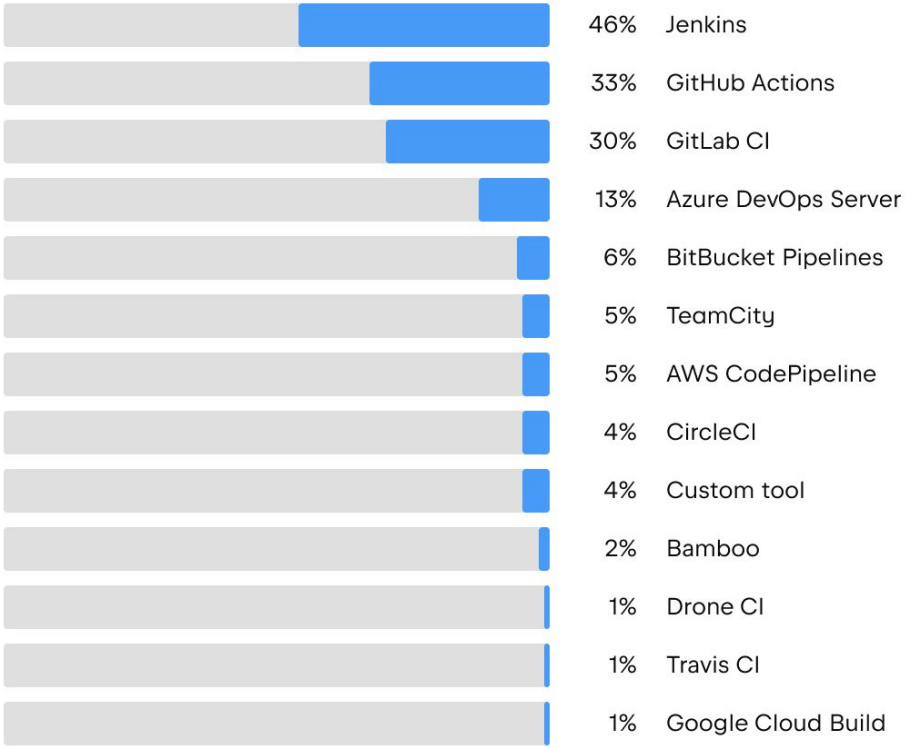
Técnicas e Ferramentas

Ferramentas Populares e Frameworks de Teste

- **Ferramentas de CI/CD:**
  - *Jenkins:* servidor open-source altamente customizável.
  - *GitHub Actions:* integração nativa do GitHub para pipelines como código.
  - *GitLab CI/CD:* integra pipelines configuráveis diretamente no GitLab.
  - *CircleCI:* serviço na nuvem para CI/CD com foco em paralelismo.
  - Outros: *Travis CI*, *Azure DevOps*, *Bamboo* etc.

Essas ferramentas monitoram o repositório e disparam *jobs* de build e teste automaticamente

Which continuous integration (CI) tool do you regularly use in your company?



Técnicas e Ferramentas

- **Frameworks de Teste:**
  - *JUnit* (Java) – padrão para testes de unidade em Java.
  - *PyTest* ou *unittest* (Python) – frameworks populares no ecossistema Python.
  - *NUnit* (C#/.NET) – similar ao JUnit para .NET.
  - *Selenium*, *Cypress* – para testes de interface web (E2E/funcionais) com automação de navegador.
  - *TestNG*, *JUnit5* – extensões de JUnit para Java.
  - *Cucumber* – para testes de aceitação baseados em comportamento (BDD).
  - *Robot Framework* – ferramenta genérica para automação em diversas linguagens.

Desenvolvimento orientado por teste

- Em um pipeline de IC com teste automático, é sempre interessante melhorar a cobertura do teste.
- Cada nova função que segue pelo pipeline de IC deve ser acompanhado de um conjunto de testes para assegurar que o novo código se comporte como o esperado.
- O Test Driven Development (Desenvolvimento orientado por teste, TDD) é uma prática de escrever o código de teste e os casos de teste antes de fazer qualquer código de função real.
- Como parte do processo de testes automatizados em CI, o desenvolvimento orientado por testes cria o código de forma iterativa e testa um caso de uso por vez; e notifica as equipes sobre o desempenho do código em todas as áreas funcionais da aplicação.

Em suma

- **Automação de Testes:** Utiliza scripts para executar testes de software automaticamente, aumentando eficiência, cobertura e qualidade. Reduzir esforço manual, custos, e tempo; melhorar consistência e detectar falhas mais cedo.
- **Integração Contínua (CI):** Prática de integrar código frequentemente para detectar erros rapidamente e garantir estabilidade contínua. Incluem repositórios de código, servidores CI, automações e testes a cada mudança no código.
- **Importância dos Testes na CI:** Testes automatizados garantem validações frequentes, feedback rápido e entregas mais confiáveis. **Vantagens:** Menor retrabalho, mais agilidade e maior confiabilidade nas entregas.
- **Desafios:** Testes frágeis, alto custo de manutenção, lentidão e dependência de ambiente podem comprometer a eficácia.
- **Ferramentas e Técnicas:** CI/CD (Jenkins, GitHub Actions, GitLab CI etc.) e frameworks de teste (JUnit, PyTest, Selenium, etc.) viabilizam a automação.

OBRIGADO

daciofmf@gmail.com