UniEVANGÉLICA
CENTRO UNIVERSITÁRIO

Análise e Projeto de Software

Universidade Evangélica de Goiás

Curso de Engenharia de Software

UniEVANGÉLICA
CENTRO UNIVERSITÁRIO

Análise e projetos de Software: Conceito de Pirâmides de Projetos de Software

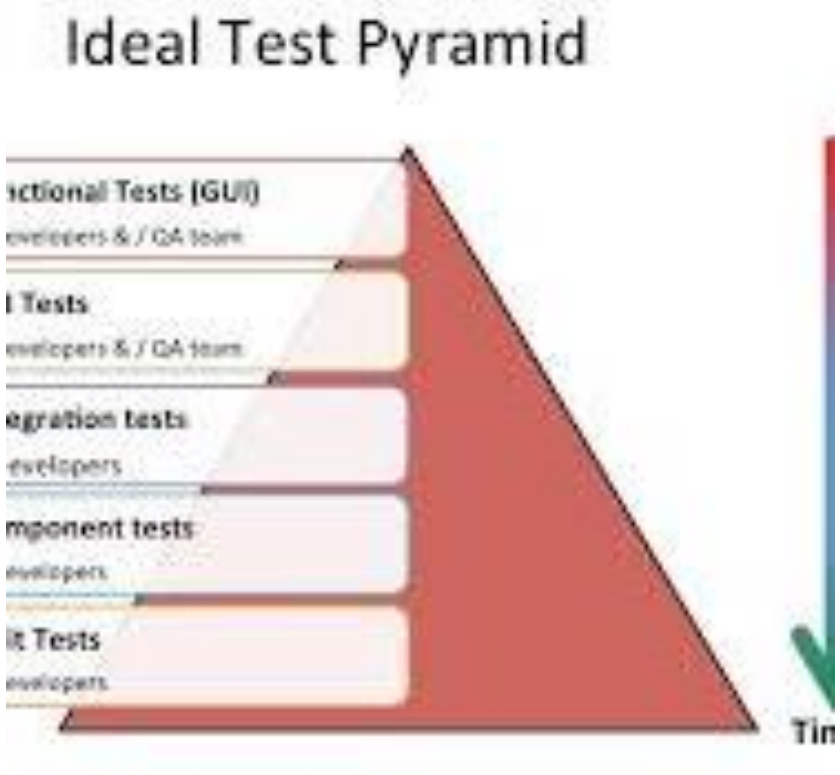
Definição e Origem

A **Pirâmide de Testes** orienta na construção de suítes de testes eficientes e sustentáveis para projetos de software.

“Cohn modelou a pirâmide baseando-se nos princípios de eficácia, manutenibilidade e velocidade da execução de testes.”

Desenvolvida inicialmente por Mike Cohn, a estratégia coloca ênfase em testes unitários, sendo complementada por testes de integração e, por último, teste de aceitação.

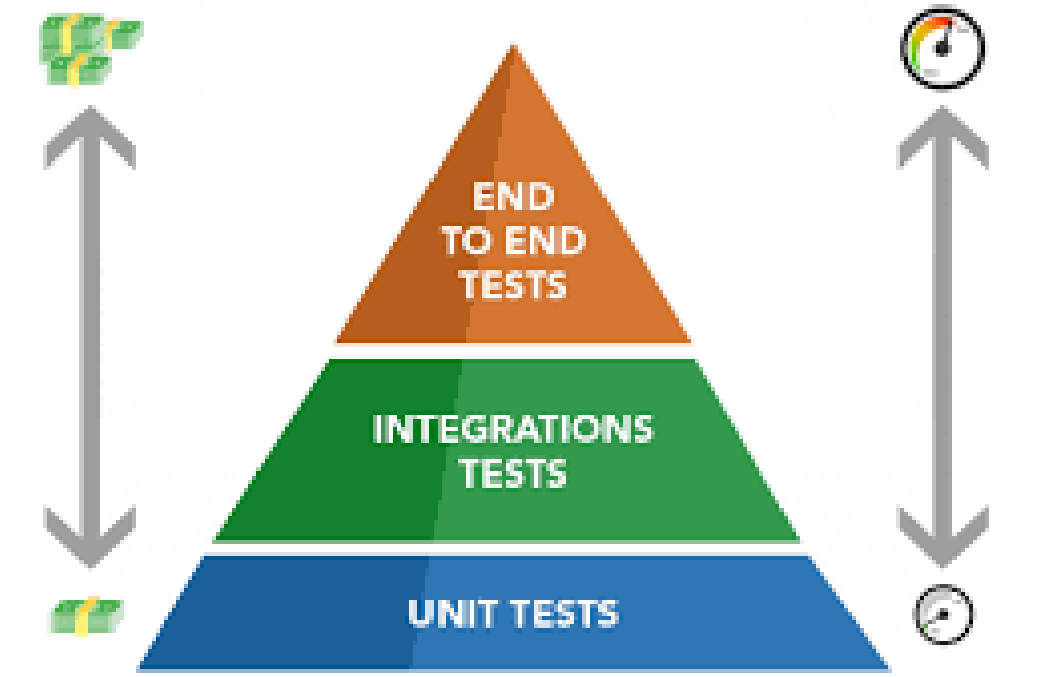
A abordagem promove um balanço custo-efetividade ao maximizar a confiabilidade do software com um investimento adequado em testes automatizados.



Definição e Origem

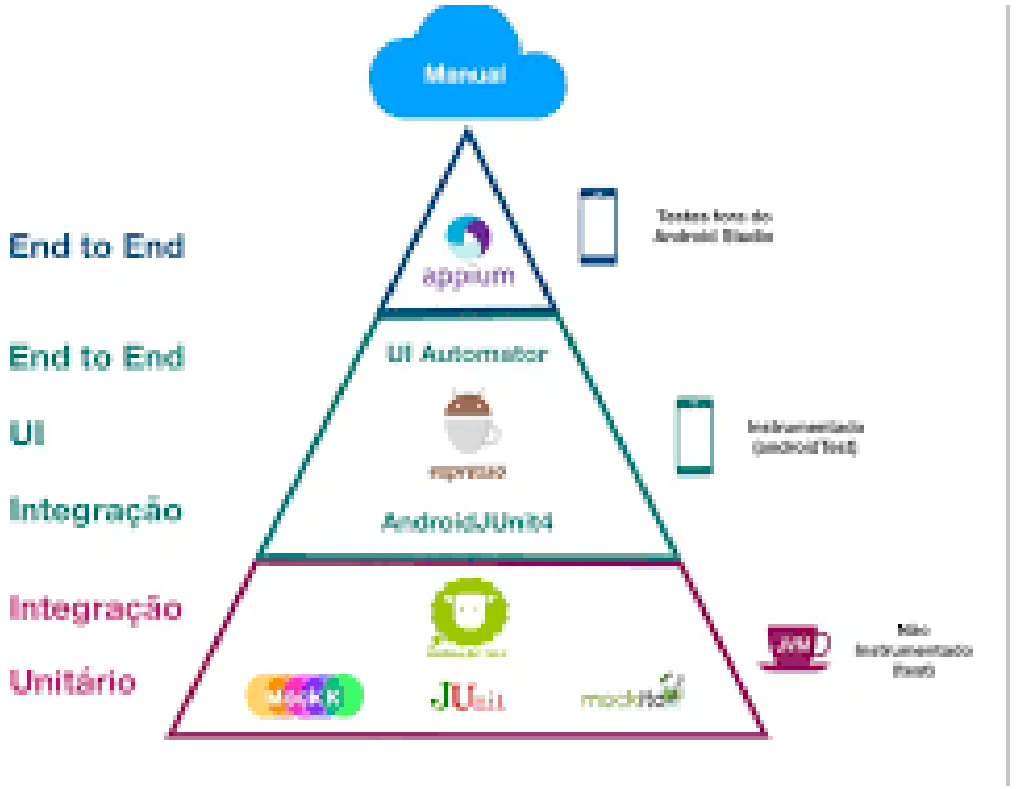
Desenvolvida inicialmente por Mike Cohn, a estratégia coloca ênfase em testes unitários, sendo complementada por testes de integração e, por último, teste de aceitação.

A abordagem promove um balanço custo-efetividade ao maximizar a confiabilidade do software com um investimento adequado em testes automatizados.



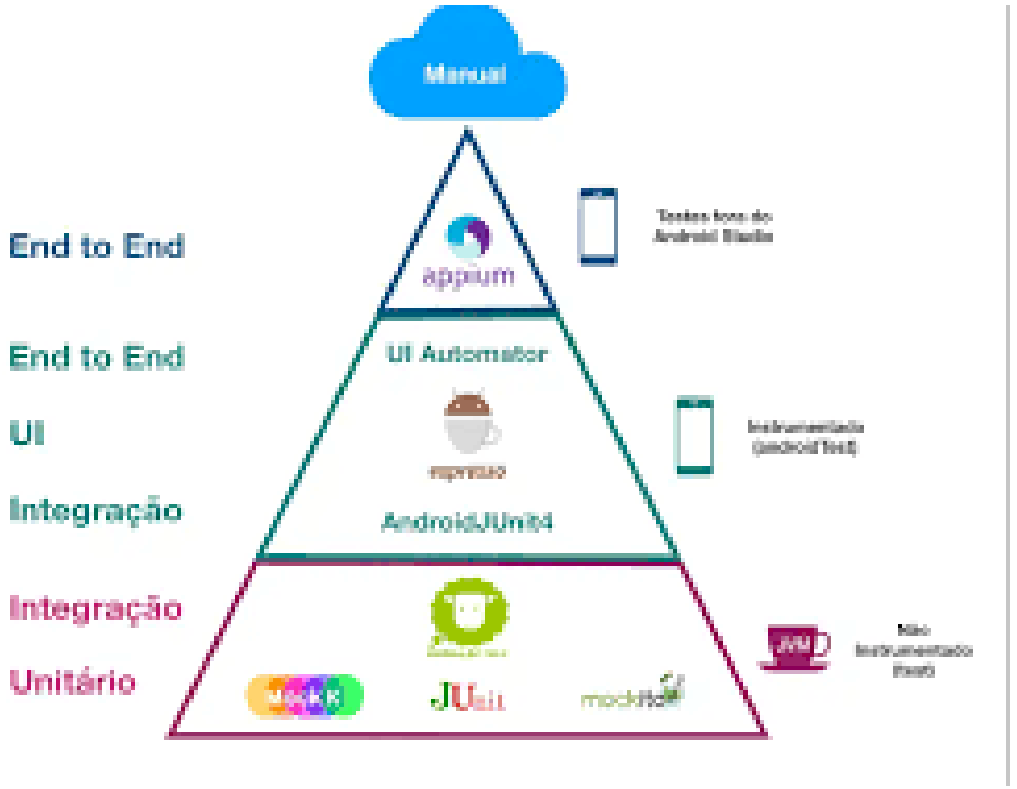
Camadas da Pirâmide

- A **base da Pirâmide**, representando a maior quantidade de testes, é composta por testes unitários. Esses são pequenos e rápidos, focando em verificar a corretude de uma função ou classe isoladamente.
- Acima deles situam-se os **testes de integração**, que avaliam como diferentes módulos ou serviços interagem entre si.



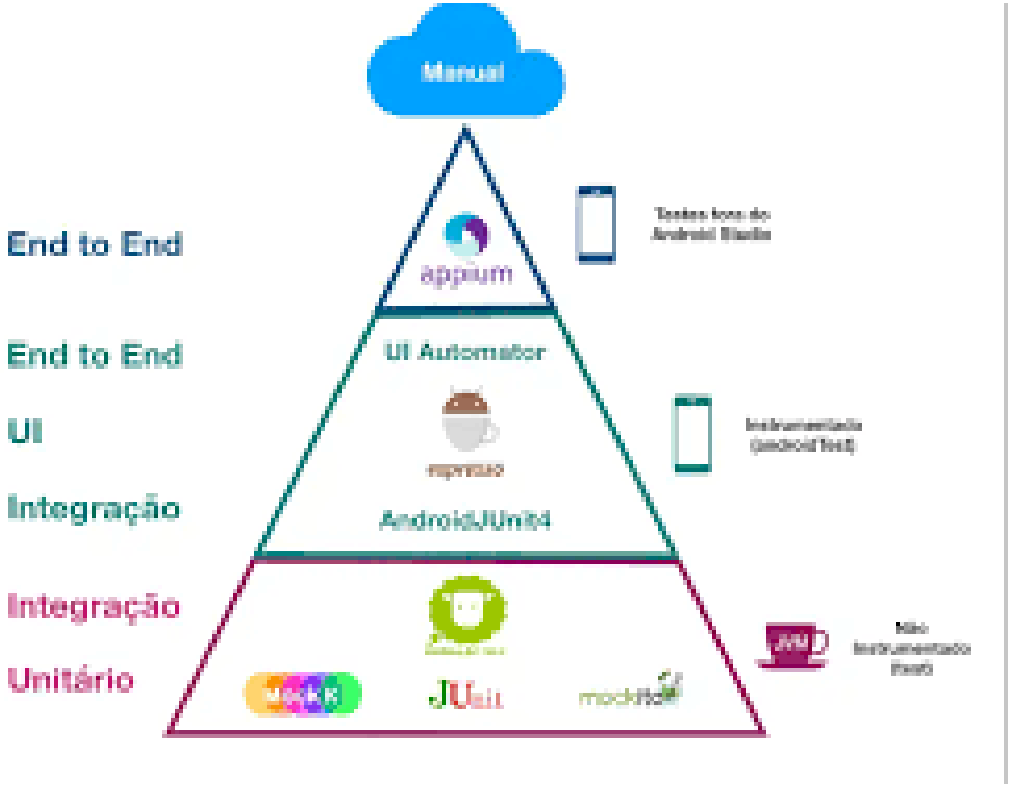
Camadas da Pirâmide

- No **intermédio**, encontramos os **testes de serviço** (também conhecidos como testes de API), garantindo que a comunicação entre diferentes partes do sistema ocorra corretamente.
- Subindo a pirâmide, estão os **testes de interface do usuário (UI)**, que simulam a interação do usuário com o aplicativo e verificam a camada visual e de usabilidade.



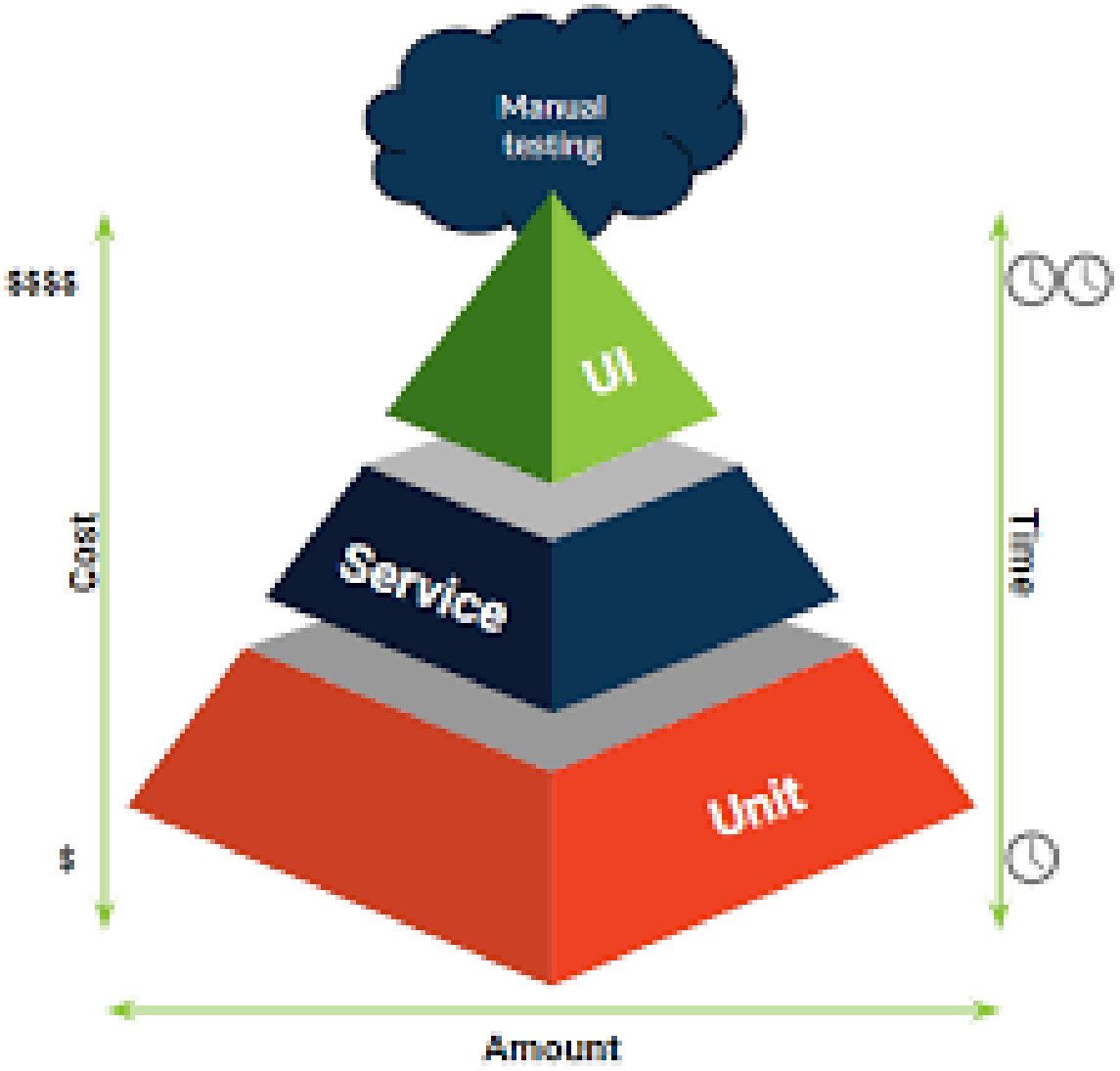
Camadas da Pirâmide

- Por fim, na **ponta da pirâmide**, localizam-se os **testes manuais** e **testes exploratórios**, essenciais para a identificação de defeitos não cobertos pelos testes automatizados.
- Além destes, existe uma camada transversal: os **testes de contrato**, fundamentais para assegurar que as APIs mantenham a consistência e não quebrem contratos estabelecidos.



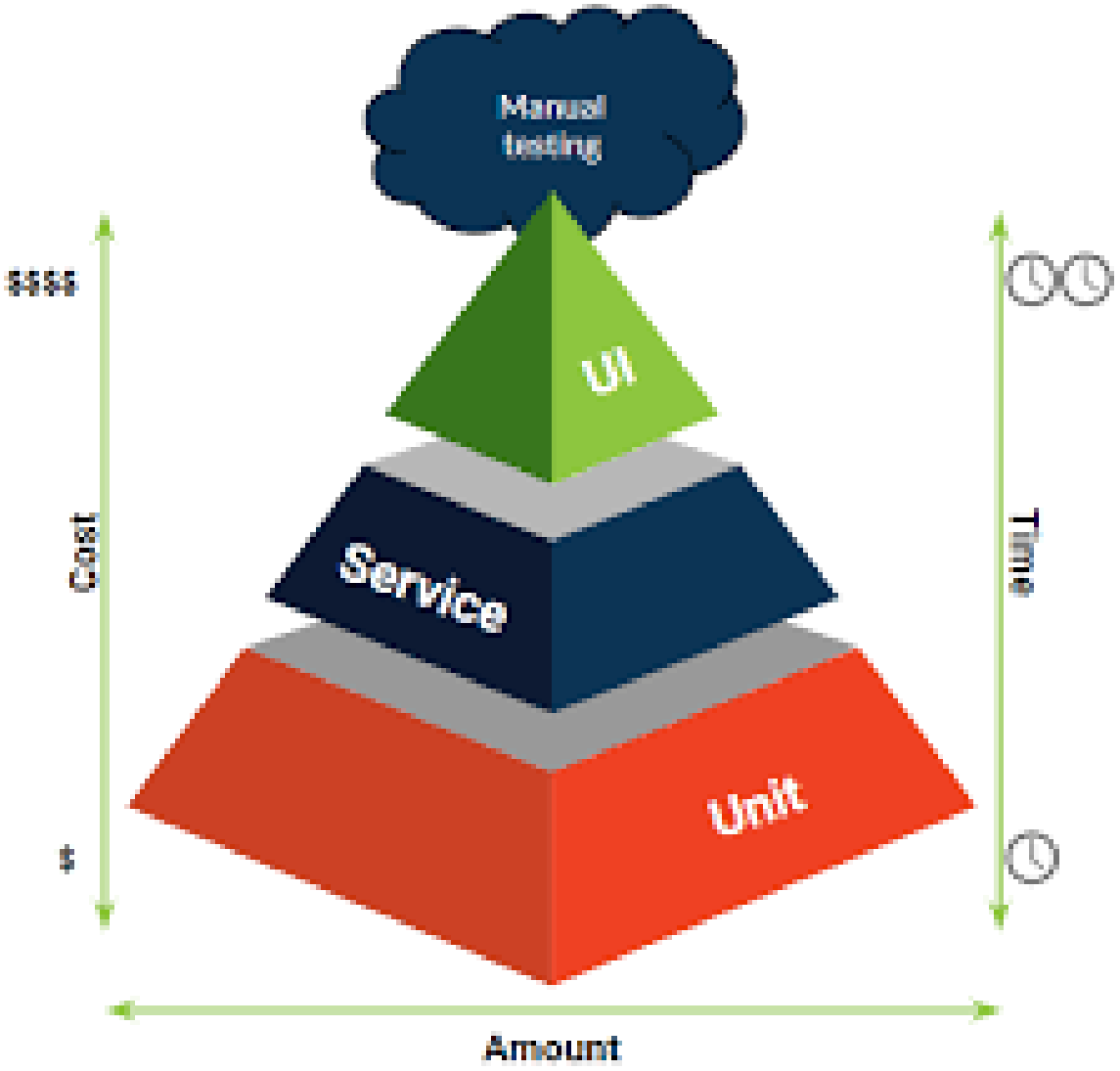
Importância da estratégica

- Investir em uma estrutura de testes robusta é condição sine qua non para startups de tecnologia que almejam crescimento sustentável e qualidade de produto. Os testes garantem que funcionalidades se mantenham íntegras ao longo de atualizações e incrementações.
- **A adoção de uma pirâmide de testes bem definida reduz custos operacionais a longo prazo.** Ela otimiza a detecção precoce de falhas, um fator crítico para o sucesso e escalabilidade da aplicação.



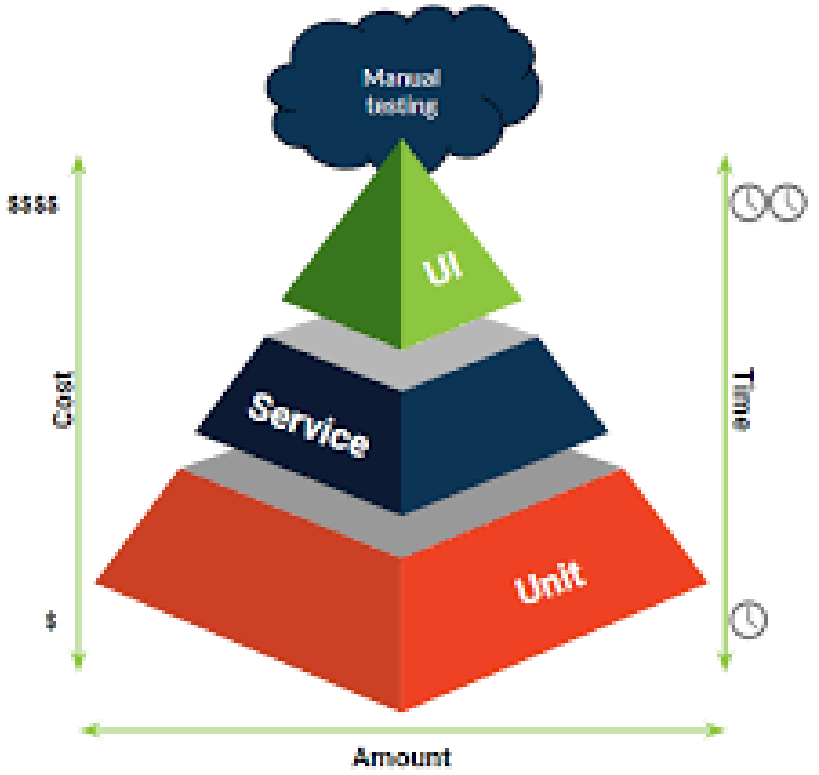
Importância da estratégia

- Desenvolver uma suíte de testes escalonável é um dos maiores desafios e diferencial estratégico. O equilíbrio certo entre diferentes níveis de testes resulta em um ciclo de vida de desenvolvimento mais ágil e confiável, promovendo o contínuo aprimoramento do produto sem sacrificar a confiança do usuário ou a integridade do sistema.



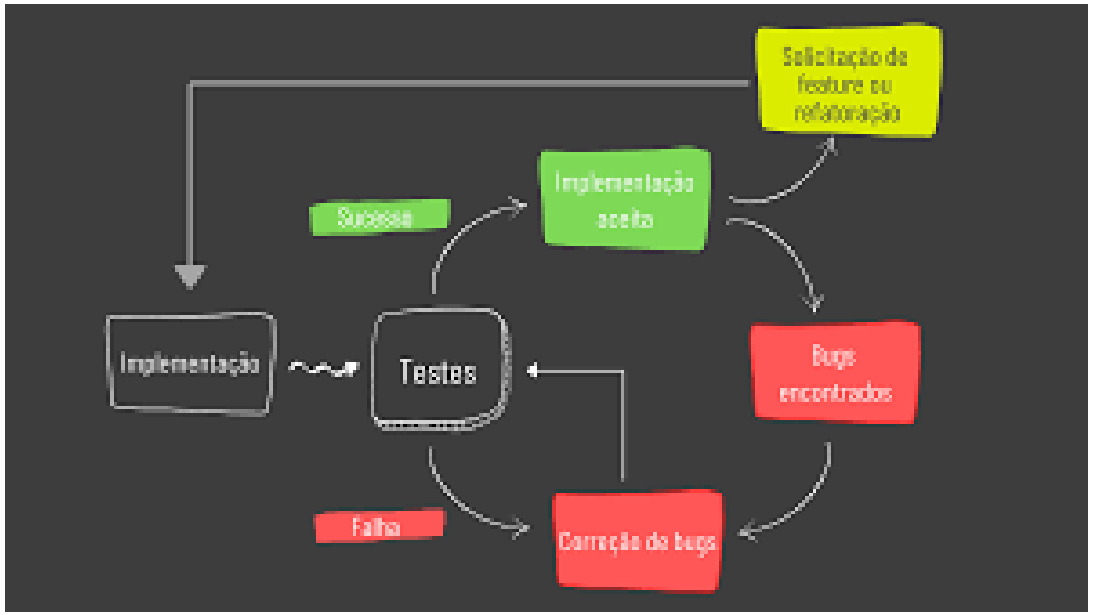
Importância da estratégia

- A prática eficiente de testes influencia diretamente na reputação e na competitividade da startup no mercado. Startups que dominam as técnicas de testagem e integram esses processos nas rotinas de desenvolvimento se destacam, oferecendo produtos mais estáveis e com menor taxa de falhas. A pirâmide de testes, portanto, não é apenas uma ferramenta de qualidade, mas também um **alicerce para o sucesso comercial e a satisfação do cliente.**



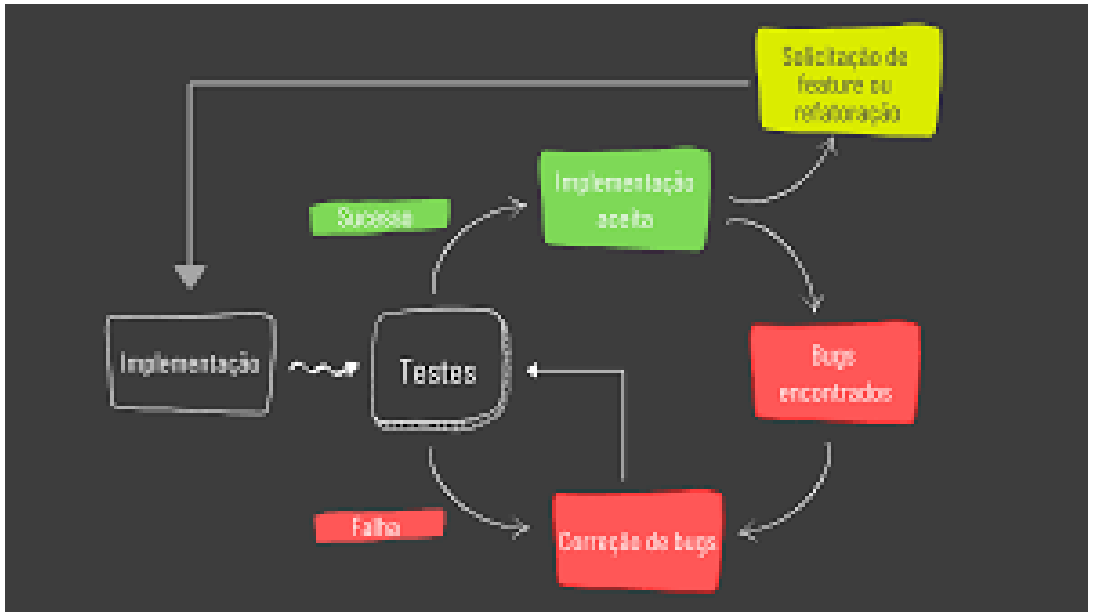
Elaborando Testes Unitários

- Os testes unitários são essenciais na base da pirâmide de testes, priorizados pela sua agilidade e precisão em isolar problemas. Com foco em pequenas porções do código, eles validam a funcionalidade das menores unidades logicamente isoláveis, os componentes ou métodos individuais, garantindo que cada parte funcione corretamente por si só.



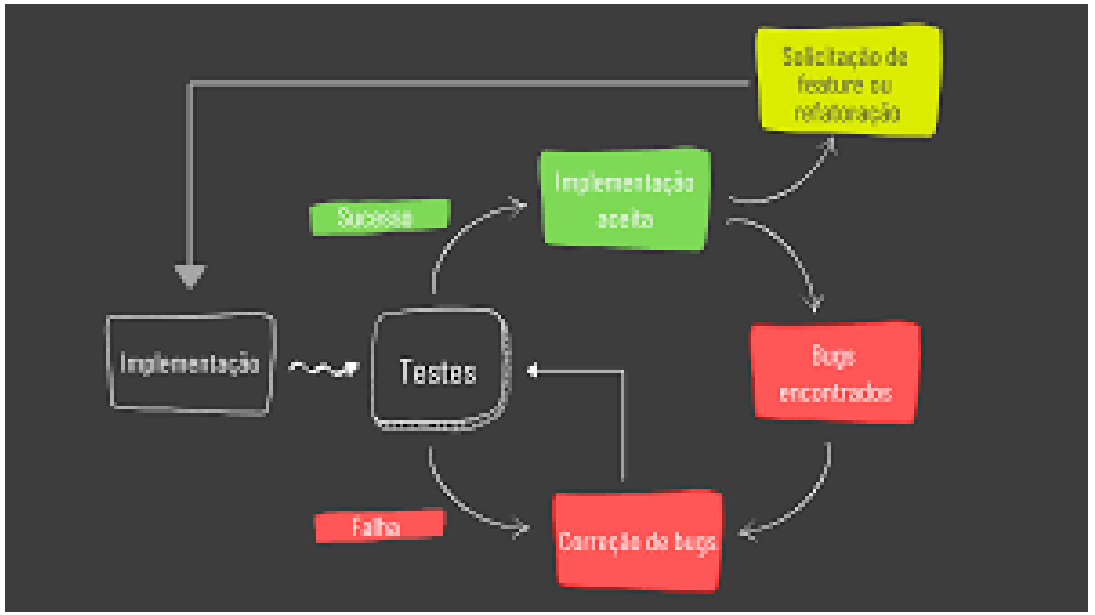
Elaborando Testes Unitários

- A minuciosidade nestes testes é crucial, e a adoção de estratégias como o **TDD (Test-Driven Development)** pode ser benéfica na construção de um software robusto. Ao escrever o teste antes mesmo da funcionalidade, afirmamos que o desenvolvimento estará alinhado com os requisitos e que apenas o código necessário será produzido, reduzindo a complexidade e aumentando a manutenabilidade.



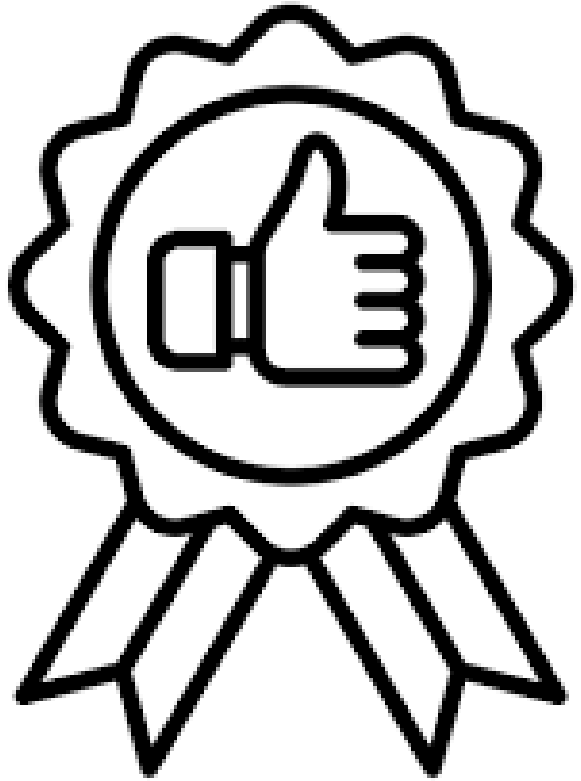
Elaborando Testes Unitários

- Na prática, a automação dos testes unitários **é um investimento** que se traduz em eficiência operacional. Ferramentas como JUnit, para Java, ou PyTest, para Python, viabilizam a execução de testes automatizados frequentes, o que é fundamental para a integração contínua e para uma construção iterativa e de alta qualidade do software.



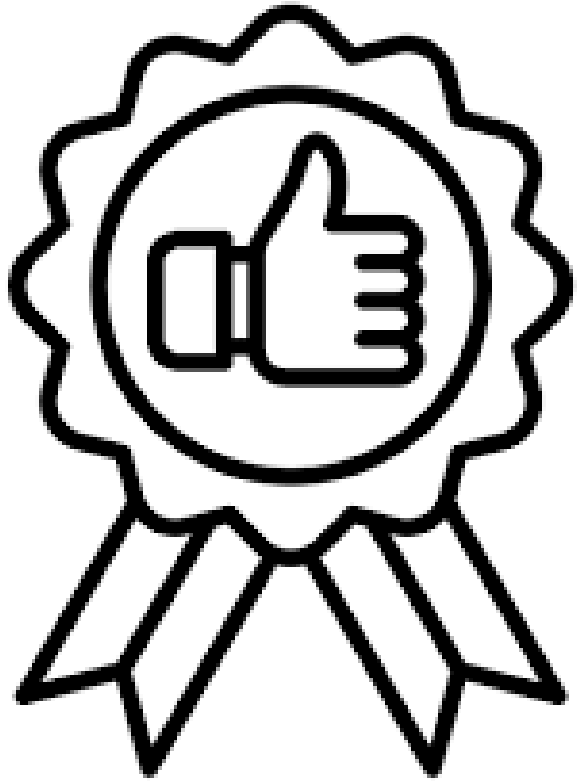
Práticas Recomendadas

- Uma abordagem estratificada ao planejar e executar testes **é fundamental para o sucesso da pirâmide de testes.** Esta estratégia garante que diferentes níveis de testes sejam devidamente abordados, promovendo uma cobertura eficaz e abrangente do código.
- Priorize a automação nos diferentes estágios de testes. Embora o investimento inicial em testes automatizados possa parecer alto, eles proporcionam um retorno significativo ao aumentar a confiabilidade do software e acelerar o ciclo de desenvolvimento ao longo do tempo.



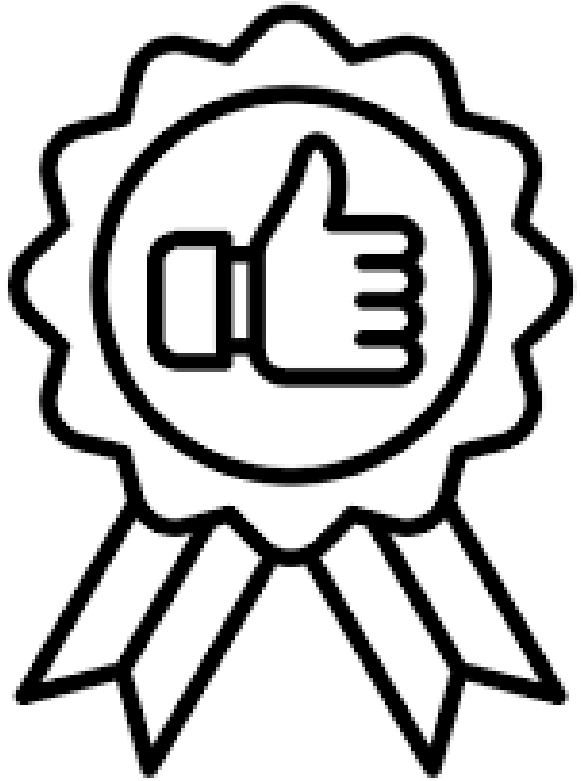
Práticas Recomendadas

- Busque uma integração entre as equipes de desenvolvimento e operações. A prática de DevOps favorece um ambiente colaborativo que agiliza o feedback e a entrega contínua, assegurando que mudanças no software possam ser rapidamente verificadas e integradas.
- Implemente monitoramento e logging robustos como parte do processo de qualidade. Estas ferramentas são fundamentais para identificar problemas em produção rapidamente e fornecer insights valiosos para possíveis melhorias no ciclo de testes e no próprio código.



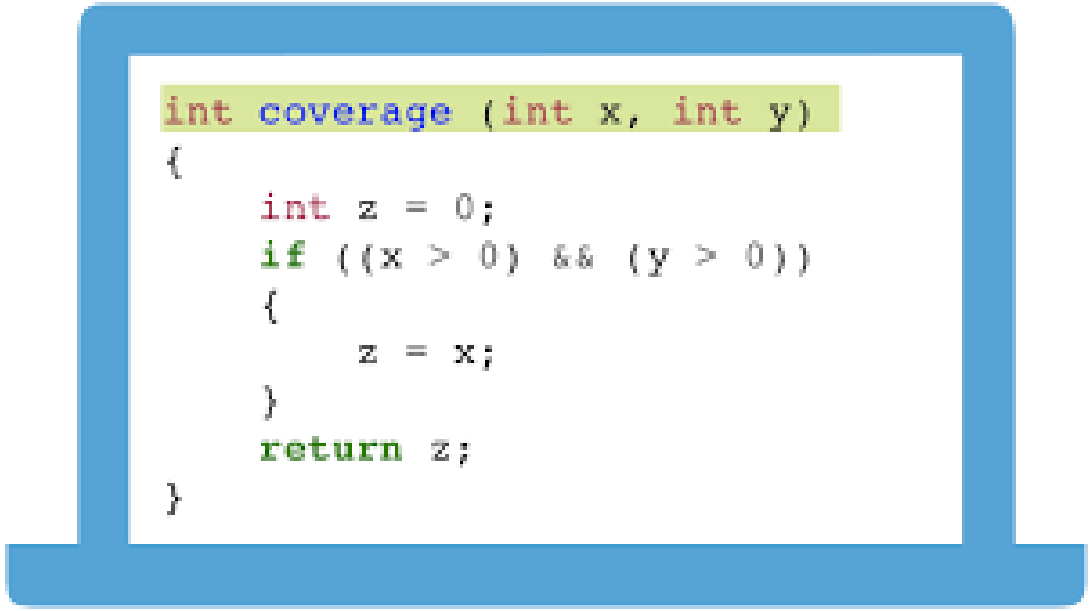
Práticas Recomendadas

- Ademais, reserve um tempo para revisões de código e refatorações constantes. Este hábito **promove** não apenas a **qualidade do código**, mas também o **aperfeiçoamento** contínuo da base de testes, mantendo-a alinhada com o desenvolvimento e evolução do **software.**



Automação e Cobertura de Código

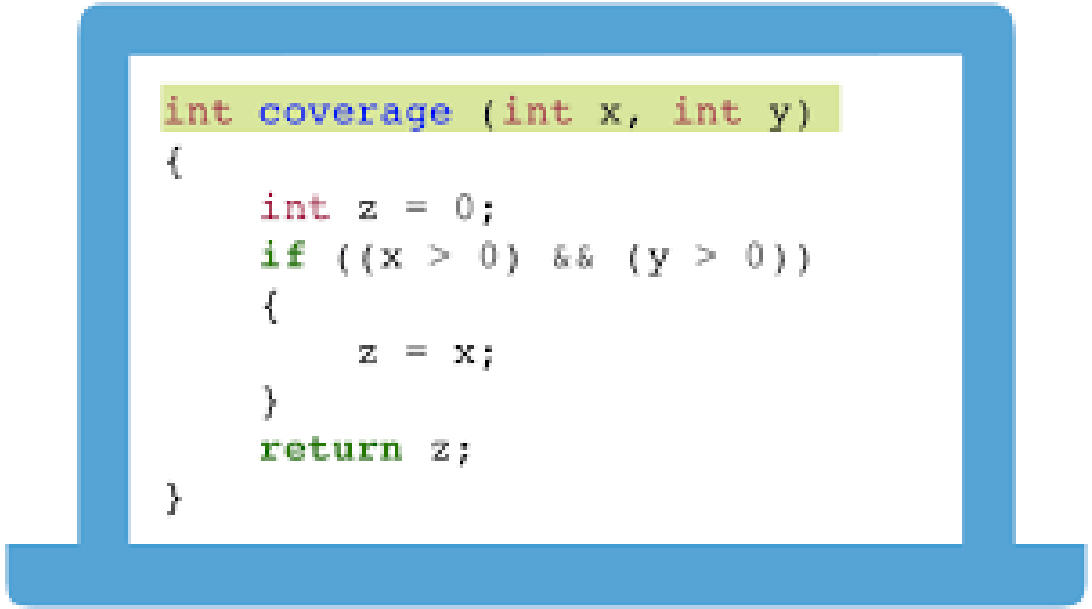
- A automação de testes é um pilar central na garantia da qualidade do software. Ela viabiliza a execução frequente e consistente de testes, fundamentais para detectar regressões e validar novas funcionalidades rapidamente.
- Nesse contexto, a cobertura de código emerge como um indicador crítico de qualidade. Trata-se da proporção do código-fonte que é efetivamente verificada por testes automatizados. **Quanto maior a cobertura, maior é a confiança** de que as alterações no código não introduzirão defeitos inadvertidamente. No entanto, uma cobertura de 100% não é, por si só, uma panaceia; é crucial que os testes sejam bem projetados e focados na lógica de negócios crítica.



```
int coverage (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Automação e Cobertura de Código

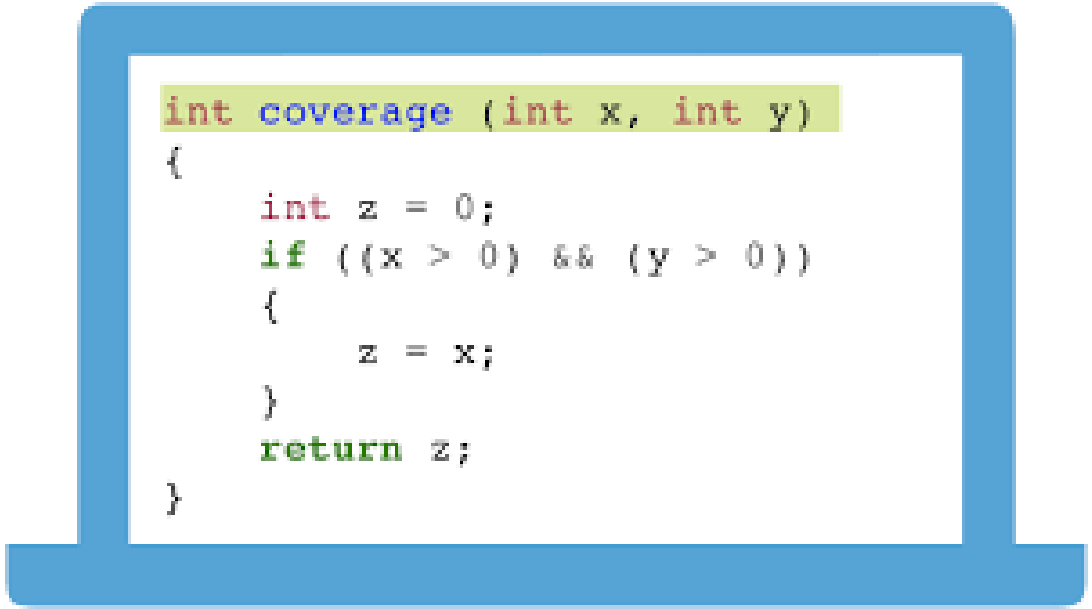
- Para aumentar a eficácia da automação, é essencial adotar práticas de integração contínua (CI). Isso inclui a execução de testes automatizados a cada commit, garantindo que todos os incrementos no código sejam imediatamente validados e que as falhas sejam rapidamente detectadas e corrigidas.



```
int coverage (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Automação e Cobertura de Código

- Além da implementação de testes automatizados robustos, empresas de tecnologia devem investir em ferramentas de análise de cobertura de código. Estas ferramentas ajudam a identificar áreas não testadas e a dar visibilidade sobre a qualidade dos testes existentes. Com insights precisos, é possível direcionar esforços onde realmente importa, otimizando testes e melhorando de forma contínua o padrão de qualidade do código entregue.



```
int coverage (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Ferramentas e Frameworks

Selecionar a ferramenta certa é um desafio e tanto.

- Cada linguagem e ecossistema possuem seus *frameworks* e ferramentas preferenciais de teste. Programas como **JUnit** para Java, **pytest** para **Python** e RSpec para **Ruby**, representam apenas a ponta do iceberg. Para testes de interface de usuário, ferramentas como **Selenium** e Cypress são frequentemente escolhidas devido à sua robustez e facilidade de integração com pipelines de CI/CD.

Ferramentas e Frameworks

A integração com sistemas de CI/CD impulsiona a automação.

- Os testes unitários podem ser implementados via xUnit - um padrão para frameworks de teste de unidades que suporta várias linguagens de programação. Ao mesmo tempo, frameworks como TestNG disponibilizam funcionalidades avançadas para testes mais complexos.

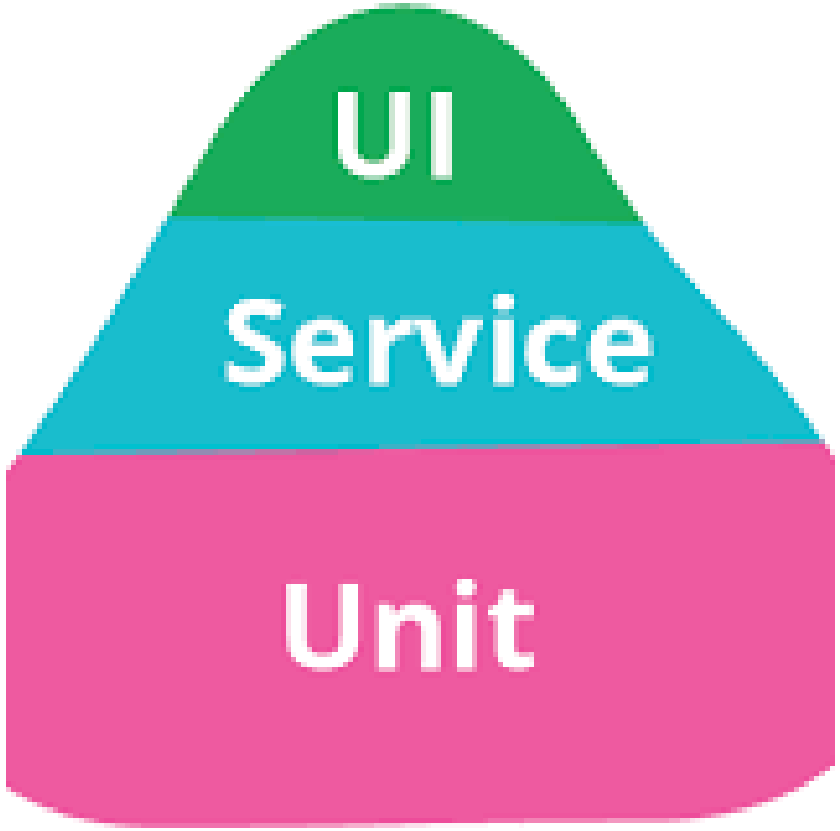
Ferramentas e Frameworks

Frameworks de BDD como Cucumber potencializam a colaboração.

- Para garantir uma visão holística do processo de testagem, as empresas precisam adotar ferramentas de monitoramento e visualização de progresso dos testes, como SonarQube ou **Code Climate**. Estas plataformas oferecem uma análise detalhada da qualidade e da cobertura do código, ajudando a manter um alto padrão de excelência do produto final.

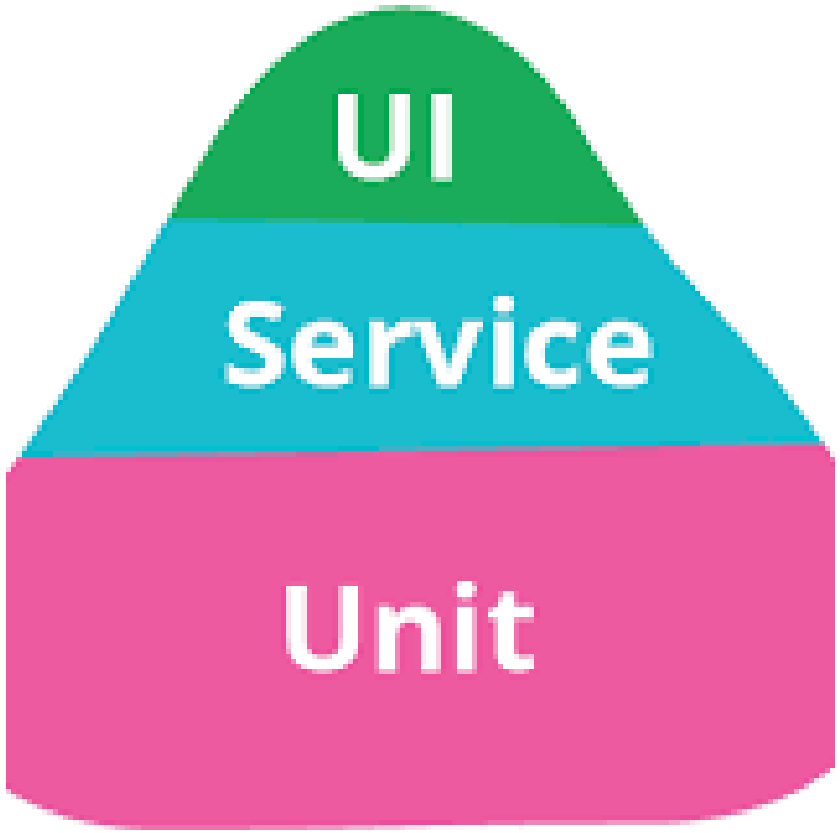
Integrando com Testes de Serviço

- Os testes de serviço, também conhecidos como testes de integração, são essenciais para validar a comunicação e o funcionamento correto entre diferentes componentes de um sistema. Em startups de tecnologia, a implementação desses testes deve garantir que todos os serviços e APIs trabalhem conjuntamente de forma eficiente, respeitando os contratos estabelecidos e garantindo a integridade dos dados. O uso de **ferramentas** de mock, como WireMock e Mountebank, pode ser altamente benéfico para simular serviços externos nas fases de teste.



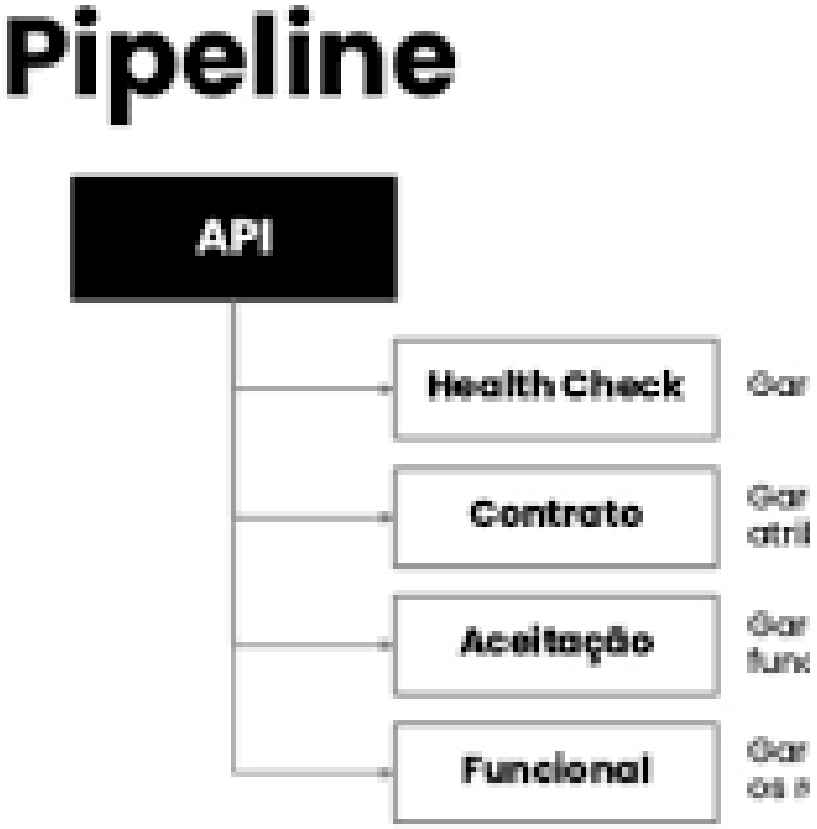
Integrando com Testes de Serviço

- A automação dos testes de serviço é **fundamental na agilidade** do processo de desenvolvimento e na detecção precoce de defeitos. Dessa forma, é recomendável a integração com sistemas de Integração Contínua e Entrega Contínua **(CI/CD)**, utilizando ferramentas como Jenkins e Travis CI. Essa estratégia ajuda a acelerar o lançamento de novas features e a assegurar a estabilidade do sistema antes de chegar ao ambiente de produção.



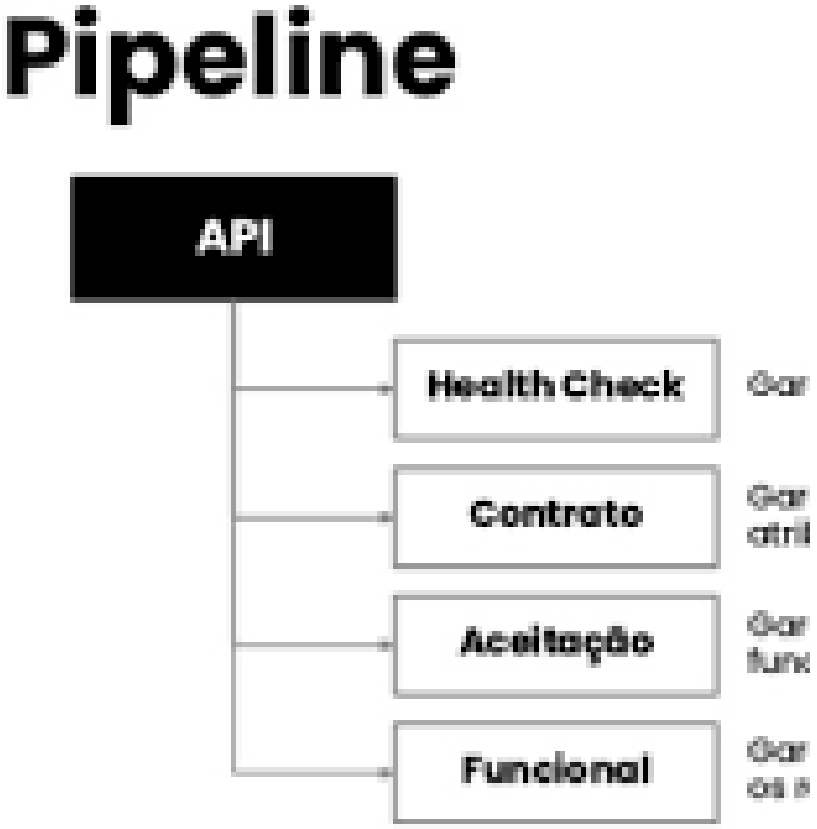
Testes de Integração e API

- Testes de integração são essenciais para assegurar que os componentes do sistema interajam corretamente. Estes testes verificam se os módulos individuais, quando combinados, funcionam como previsto, permitindo identificar falhas nas interfaces entre eles.
- No contexto de APIs, o teste de integração ganha contornos adicionais. É fundamental que as APIs, que atuam como pontos de contato entre diferentes sistemas ou partes de um mesmo sistema, operem de acordo com as especificações e expectativas dos consumidores. Esses testes verificam se os endpoints respondem adequadamente às requisições de entrada e se os dados são corretamente manipulados e devolvidos pelo sistema.



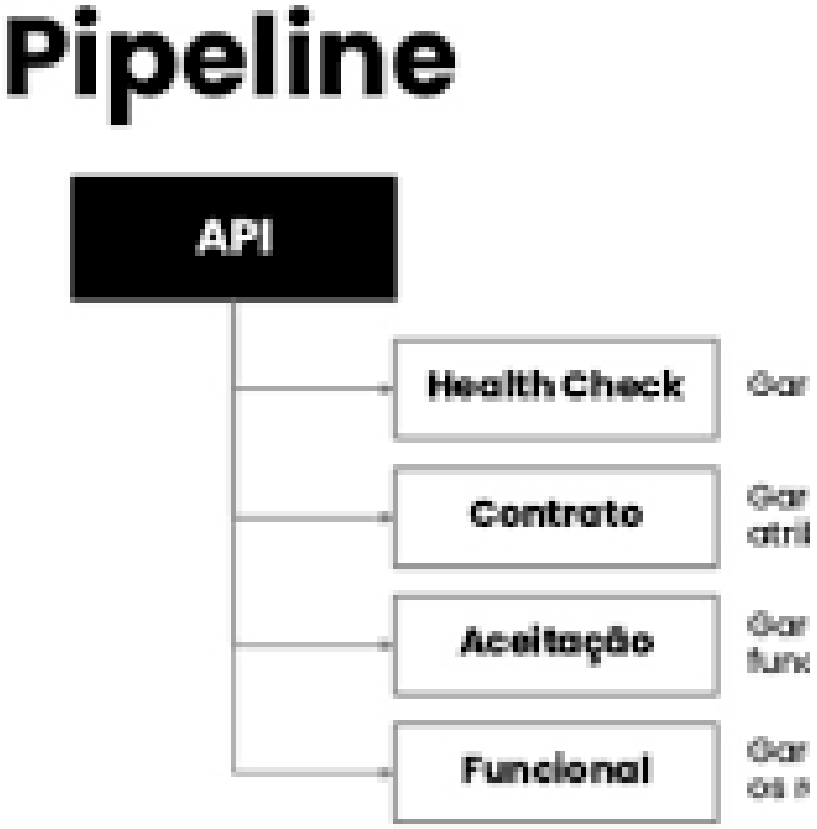
Testes de Integração e API

- Empregar técnicas de teste de contrato, tais como o **Pact** ou o **Swagger**, fornece uma camada de garantia que as APIs continuarão a atender aos contratos definidos mesmo após mudanças no código. Isso assegura a compatibilidade entre os serviços e previne falhas de integração que poderiam passar despercebidas até etapas mais avançadas do desenvolvimento.



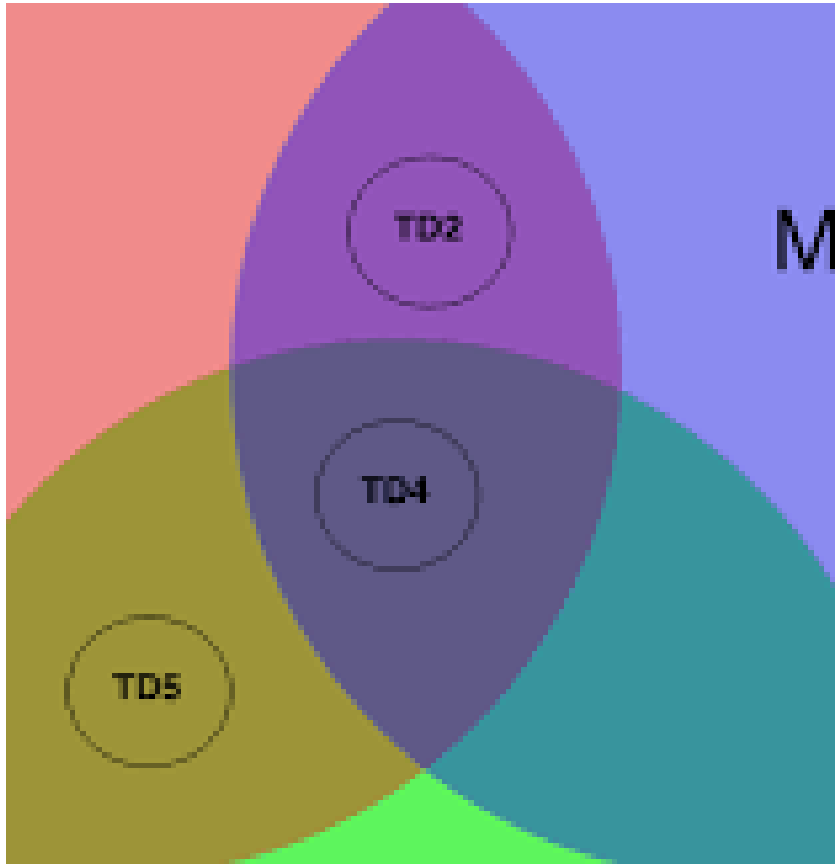
Testes de Integração e API

- Ademais, a simulação de chamadas de API utilizando ferramentas como Postman e SoapUI, ou mesmo frameworks de teste automatizado, permite uma abordagem de teste mais granular. Tais ferramentas possibilitam testar aspectos específicos das APIs, desde autenticação e autorização até a validação de parâmetros e o tratamento de erros.
- A cobertura completa dos cenários de teste de integração e API é crítica para startups em crescimento. Garantir essa cobertura ajuda a prevenir defeitos que poderiam comprometer a satisfação do usuário final, e consequentemente, o sucesso do produto no mercado.



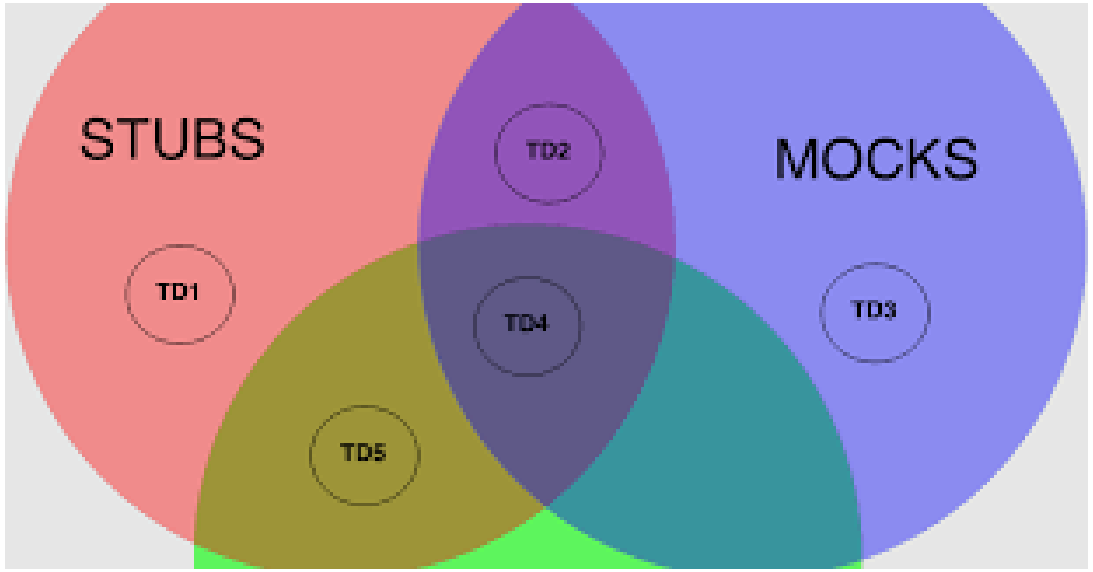
Mocks e Stubs

- **Mocks** e **stubs** são elementos fundamentais na automação de testes. Atuam isolando partes do sistema para verificar comportamentos específicos.
- Ao desenvolver testes unitários, **mocks** permitem simular objetos que interagem com o código-teste. Isso facilita o controle das condições de teste, criando cenários previsíveis e consistentes.



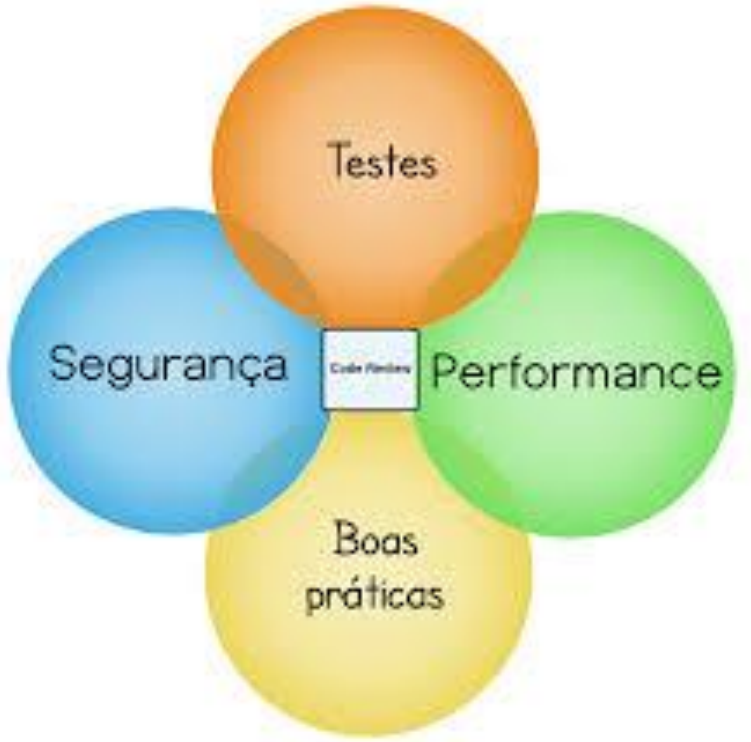
Mocks e Stubs

- **Stubs**, por outro lado, são implementações simplificadas que imitam as funcionalidades de componentes do software. Eles oferecem respostas predefinidas a chamadas de métodos ou funções.
- A utilização de ambos em conjunto possibilita a simulação fidedigna de sistemas externos e componentes complexos. Isso melhora significativamente a eficácia dos testes unitários e de integração.
- É importante escolher a ferramenta adequada para mocks e stubs com base na linguagem e necessidades do projeto. Essa escolha será crucial para a robustez dos testes.



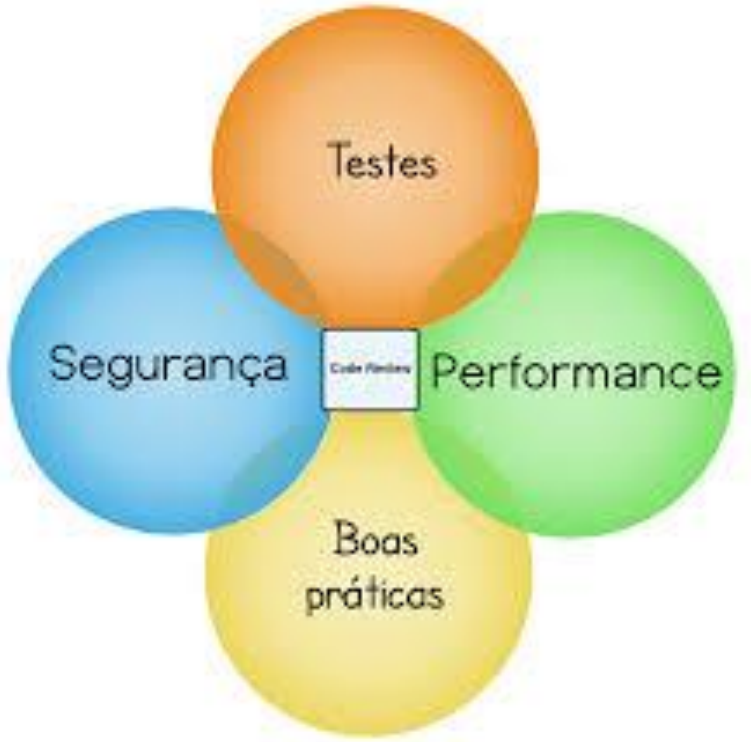
Segurança e Performance

- É imperativo que nas fases iniciais de desenvolvimento, segurança e performance sejam tratadas como prioridades. Essas áreas devem ser continuamente aperfeiçoadas e testadas para garantir a entrega de um software robusto e confiável.



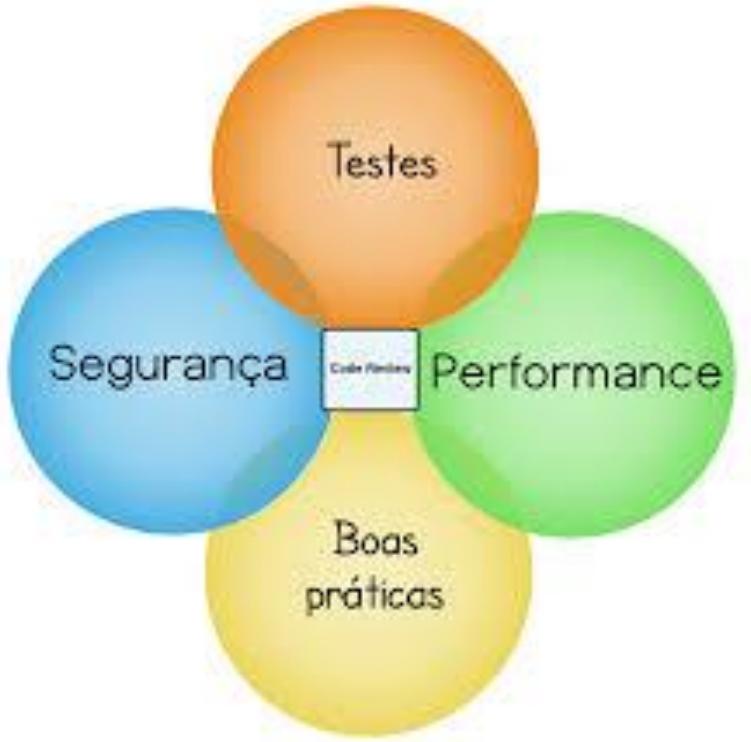
Segurança e Performance

- No contexto da **pirâmide de testes**, enfatiza-se que testes de segurança e performance não são simples acréscimos ao processo de desenvolvimento, mas componentes críticos que demandam atenção especial. Devemos considerar, por exemplo, a implantação de testes de penetração e testes de carga, que buscam respectivamente identificar vulnerabilidades de segurança e avaliar a capacidade do software sob condições de uso extremo. Ambos devem ser planejados estrategicamente, desde as fases mais tempranas e se intensificar progressivamente, evoluindo com a complexidade do produto.



Segurança e Performance

- Ressalta-se também a importância de usar **ferramentas especializadas** e realizar **auditorias de código** regularmente, identificando e mitigando riscos a tempo. A prática da **revisão de código** entre pares é igualmente valiosa, promovendo não só a segurança, mas também otimizando a performance ao identificar gargalos de desempenho prematuramente.



Segurança e Performance

- Finalmente, é essencial que a estratégia de testes para segurança e performance seja parte integrante do ciclo de desenvolvimento (CI/CD) e esteja alinhada aos objetivos do negócio. A implementação de práticas como **DevSecOps** fortalece a postura de segurança de aplicativos no ambiente ágil e contribui para a alta disponibilidade e resiliência do sistema, dois pilares fundamentais para a aceitação do software pelo mercado e a satisfação continuada dos usuários.