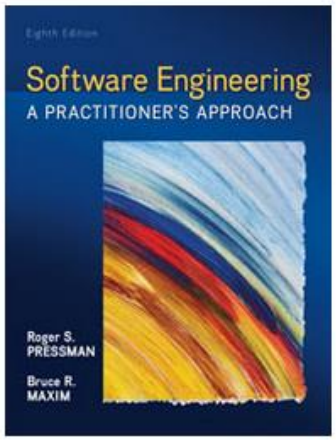
Unidade 13
Teste durante o Ciclo de Vida do Software

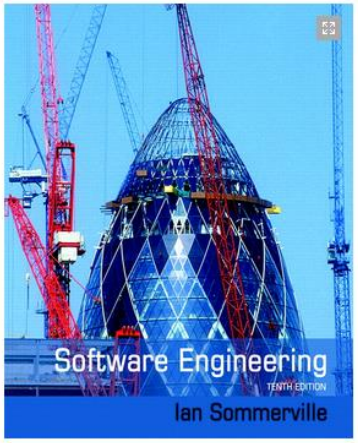
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP

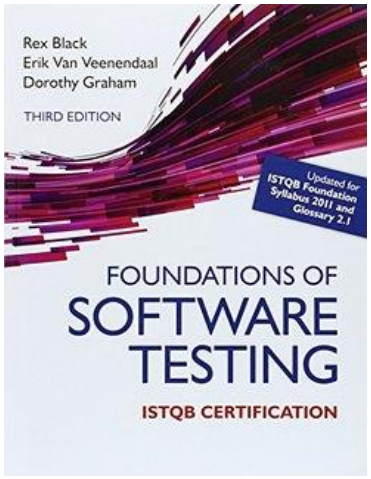
Bibliografia

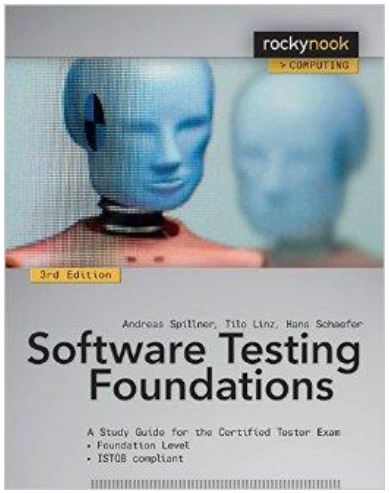
- **Software Engineering – A Practitioner's Approach – Roger S. Pressman – Eight Edition – 2014**
- **Software Engineering – Ian Sommerville – 10th edition - 2015**
- Engenharia de Software – Uma abordagem profissional – Roger Pressman - McGraw Hill, Sétima Edição - 2011
- Engenharia de Software – Ian Sommerville – Nona Edição – Addison Wesley, 2007
- **Software Testing Foundations – 4th edition – Andreas Spillner, Tito Linz, Hans Schaefer – 2014**
- **Foundations of Software Testing – Third Edition – Rex Black – ISTQB Certification**



Software Engineering: A Practitioner's Approach, 8/e







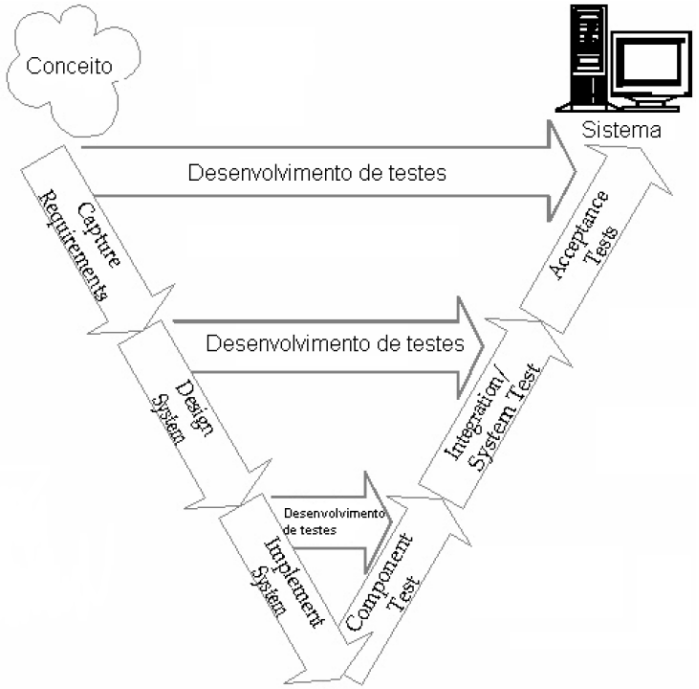
Modelos de Desenvolvimento de Software

- A atividade de teste está intimamente relacionada com as atividades de desenvolvimento de software;
- Diferentes modelos de processo de software necessitam de diferentes abordagens de teste;

Modelo V

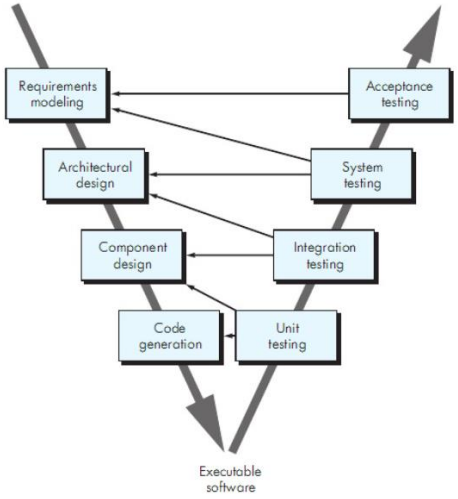
- Uma variação do modelo cascata, que descreve a relação entre as ações de **garantia de qualidade** e as ações do **processo de desenvolvimento de software**.

- À medida em que a equipe de software cresce em direção ao lado esquerdo do V até a geração do código, a equipe se desloca do lado direito, realizando uma série de testes.



Níveis de Teste - Modelo V

- Teste de **componente** (unidade);
- Teste de **Integração**;
- Teste de **Sistema**;
- Teste de **Aceite**.



Modelos Evolucionários

- Desenvolvimento iterativo é o processo que estabelece os requisitos, modelagem, construção e teste de um software como uma série de desenvolvimentos menores;
- Dirigidos por risco e cronograma;
- O conjunto de funcionalidades cresce ao redor da funcionalidade central;
- Tem se tornado uma abordagem popular;
- **Exemplos**: Prototipação, RUP, Processos ágeis.
- O produto resultante de uma iteração pode ser testado em vários níveis como parte de seu desenvolvimento. **Teste de regressão** tem sua importância aumentada a cada iteração.
- **Verificação** e **Validação** podem ser efetuadas a cada incremento.

RUP® Rational Unified Process®

Verificação e Validação (V & V)

- Em Engenharia de Software (**IEEE 1012**):
  - **Validação**: **Estamos construindo o produto certo ?**
    O software está fazendo o que o usuário quer ?
    O software correto foi construído ?
  - **Verificação**: **Estamos construindo o produto corretamente?**
    O software está de acordo com a sua especificação ?
    O software foi construído corretamente ?

Independente do Modelo

- **Características gerais de um bom teste**
  - Atividade de teste para cada atividade de desenvolvimento. Isso corresponde a um **paralelismo** entre desenvolvimento e teste (Por exemplo, teste unitário);
  - Cada teste tem um objetivo específico dentro do seu **nível** (etapas ou fases);
  - Análise e modelagem de teste deve iniciar **cedo** – prevenção de bugs;
  - Testadores devem se envolver na revisão de documentos o mais cedo possível – prevenção de bugs.

Teste de Componente

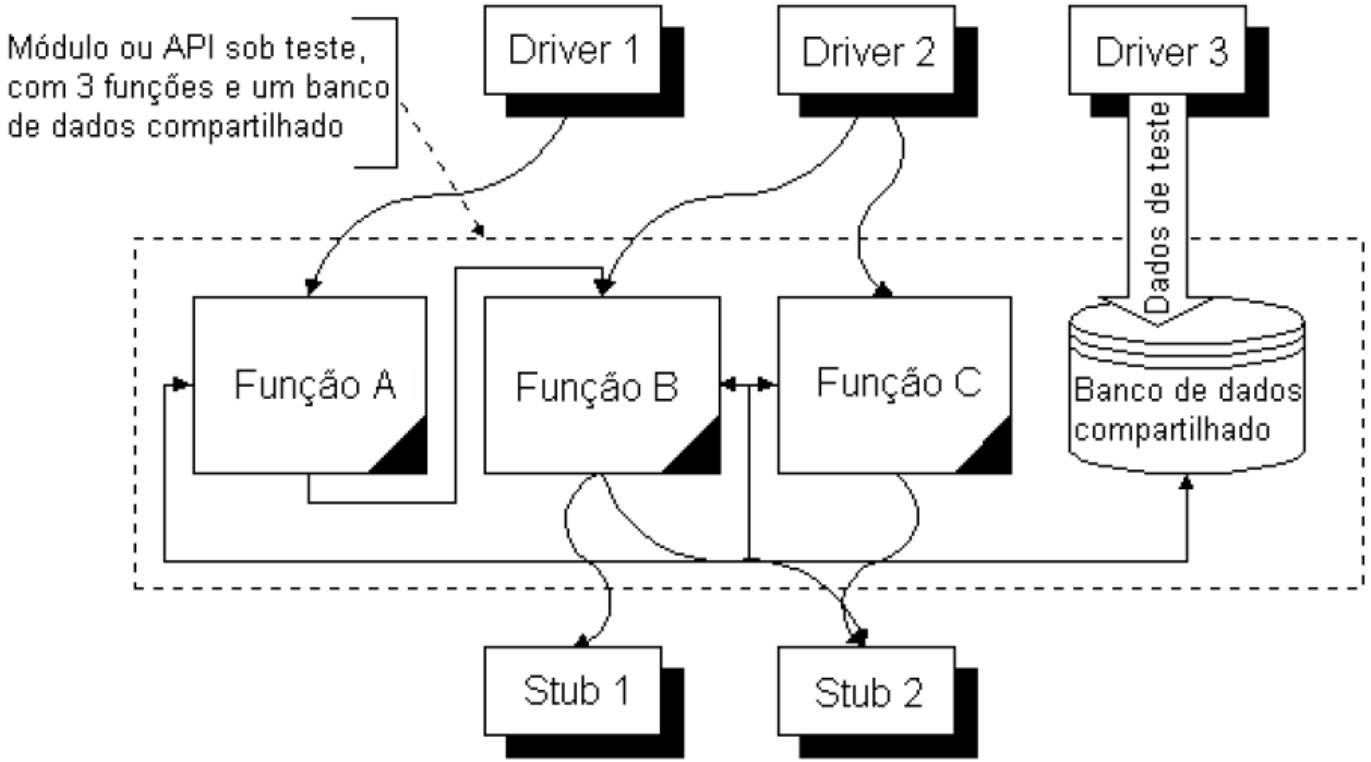
- Também chamado Teste **Unitário** ou Teste de **Unidade**;
- **Objetivo**: Encontrar **defeitos** (**bugs**) e verifica o funcionamento do software (módulos, programas, objetos, classes, etc) que são **testáveis separadamente**;
- **Base**: Código, Banco de Dados, Especificação de Componentes, Modelagem do software, etc;
- **Tipos de Teste**: Funcionalidade e Características específicas não funcionais tais como: uso de recursos (ex. falta de memória), desempenho, estrutural;
- **Item sob Teste**: Varia. O menor item testável de forma independente (função ou classe) ou componente separado fornecendo serviços para os outros;
- **Ambiente preparado e Ferramentas**: Framework de teste de unidade ou ferramenta de depuração. Controladores ("**drivers**") e simuladores ("**stubs**");
- **Responsabilidade**: Geralmente **programadores**.

Teste de Componente tipicamente ...

- Envolve **acesso ao código**;
- É executado no **ambiente de desenvolvimento**;
- Requer **drivers**, **stubs**, e/ou ambiente preparado;
- É **feito pelo programador** que escreveu o código;
- Em geral, os bugs são corrigidos quando encontrados **sem qualquer relato**, o que reduz a transparência do processo de desenvolvimento com relação à qualidade.

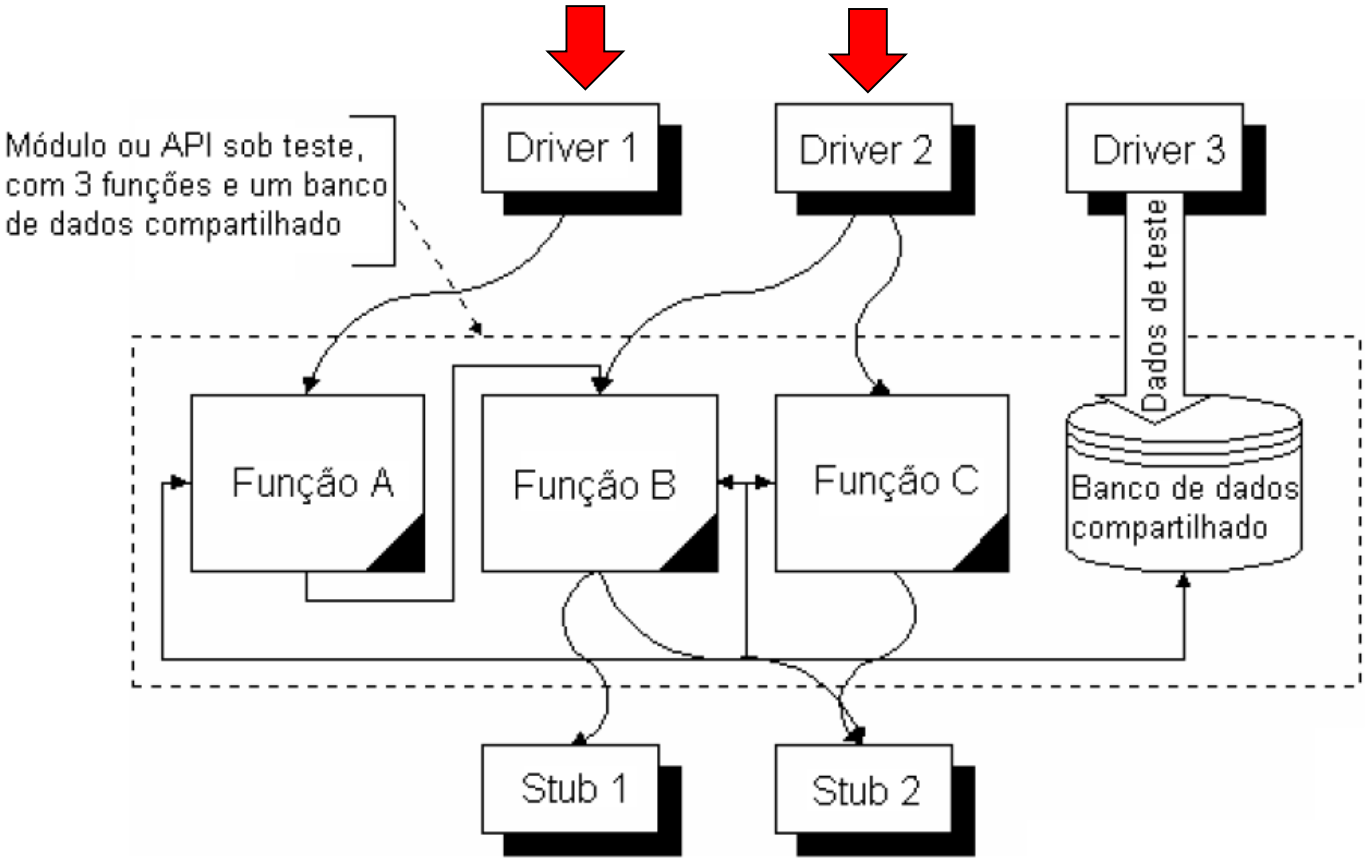
Drivers e Stubs

- Durante o teste de unidade, componente e integração – e para teste das API's – é frequentemente necessário **simular** **partes do fluxo de chamadas alcançáveis** a partir do **módulo sob teste**.



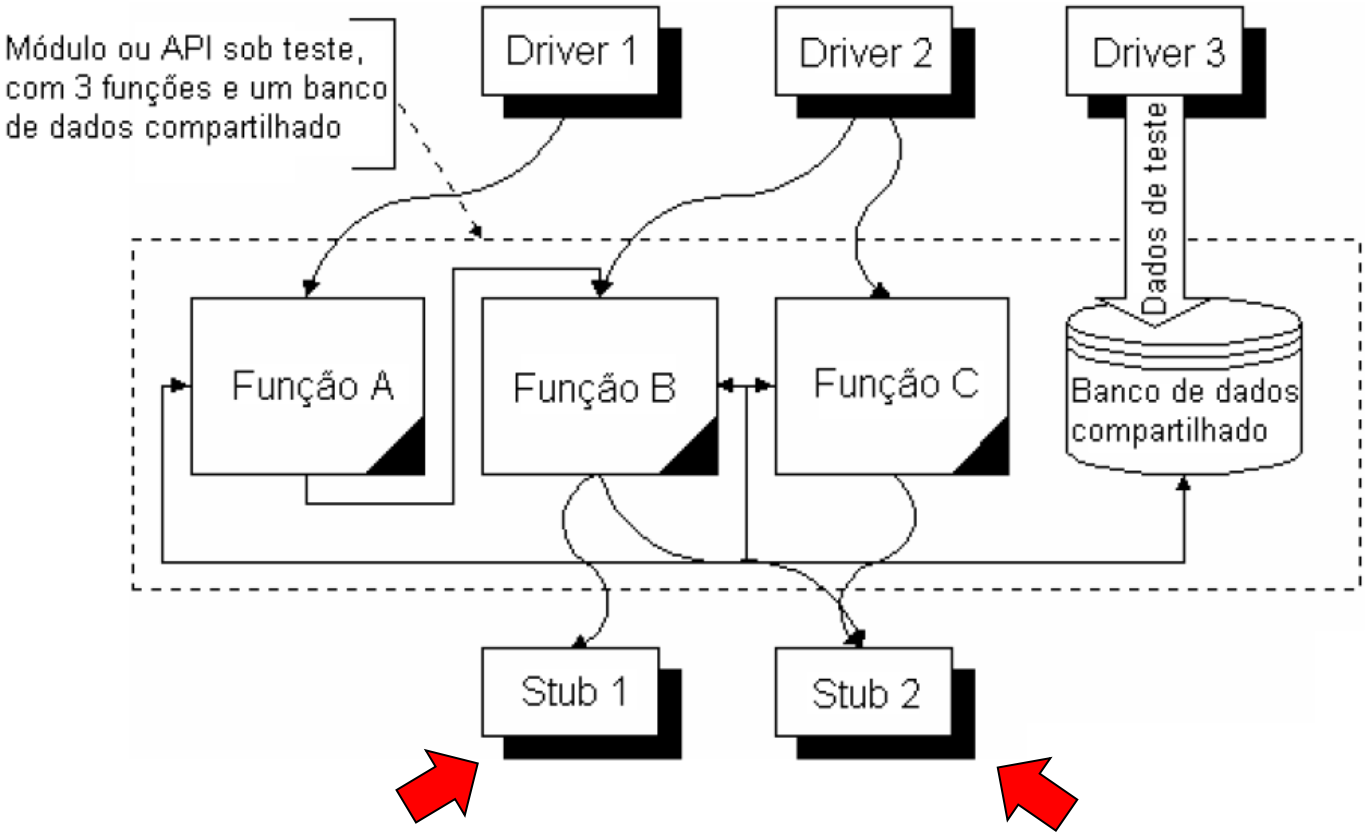
Drivers

- Função que **chama** o módulo sob teste.



Stubs

- Função **chamada**, diretamente ou indiretamente, pelo módulo em teste.



Teste de Integração

- **Objetivo**: Encontrar bugs (defeitos), adquirir confiança e reduzir os riscos nos relacionamentos e **interfaces** entres pares e grupos de componentes no software em teste, assim que as partes forem unidas (comunicação).
- **Base**: Projeto, arquitetura, esquemas, fluxo de dados, riscos de qualidade.
- **Tipos de Teste**: Funcionalidade, uso de recursos, desempenho.
- **Item sob Teste**: Builds (Compilação do sistema inteiro) ou backbones (seleção de partes do sistema);
- **Responsabilidade**: Idealmente ambos – testadores e programadores.
- Foco na integração (**comunicação**) entre os módulos e não em suas funcionalidades.

Técnicas de Integração

- **Bottom up**:
  - Iniciar com os módulos da camada mais **baixa**;
  - Usar **drivers** apropriados;
  - Testar;
  - Repetir o processo, **trocando** os drivers por módulos, até terminar;
  - Inicia com os módulos operacionais e termina com os módulos de controle;
  - Técnica muito usada na prática;
  - Não utiliza **stubs**, pois os módulos são testados de baixo para cima;
  - **Boa estratégia para isolar problemas nas camadas operacionais**;

Técnicas de Integração

- **Top down**:
  - Iniciar com os módulos da camada mais alta;
  - Usar **stubs** apropriados;
  - Testar;
  - Repetir o processo, trocando os stubs por módulos, até terminar;
  - **Boa estratégia para isolar problemas nas camadas de controle (comunicação)**;

Técnicas de Integração

- **Backbone**:
  - Iniciar com os módulos mais críticos;
  - Construir o backbone inicial;
  - Usar drivers e stubs;
  - Testar;
  - Repetir o processo, substituindo os stubs e drivers por módulos, na ordem de risco;
  - Bom isolamento do bug e encontra bugs de integração na ordem de risco;
  - **Interessante para teste de transações específicas do sistema**;

Técnicas de Integração

- **Big bang**:
  - Reunir todos os módulos testados;
  - Juntá-los;
  - Testar;
  - Rápido, mas onde está o bug ?
  - Método tardio (**sem prevenção**, pois o sistema já está pronto);
  - Geralmente feito após a execução de alguma das estratégias anteriores.

Teste de Sistema

- **Objetivo**: Encontrar bugs (defeitos), adquirir confiança e reduzir os riscos no **comportamento global** e particular, funções e **respostas do sistema sob teste como um todo**;
- **Base**: Requisitos, projeto de alto nível, casos de uso, riscos de qualidade, experiência, listas de checagem , ambientes;
- **Tipos de Teste**: Funcionalidade, segurança, desempenho, confiabilidade, usabilidade, portabilidade, etc.
- **Item sob Teste**: **Sistema completo**, no ambiente de teste o mais realista possível;
- **Responsabilidade**: Tipicamente **testadores** independentes.

Teste de Aceite

- **Objetivo**: Demonstrar que o produto está pronto para entrega/lançamento. Procurar defeitos não é o principal foco do Teste de Aceite;
- **Base**: Requisitos, contratos, experiência;
- **Tipos de Teste**: Funcional, Portabilidade, Desempenho;
- **Item sob Teste**: **Sistema completo**, algumas vezes no ambiente de produção ou do cliente;
- **Ambiente preparado e Ferramentas**: Usualmente GUI;
- **Responsabilidade**: Geralmente usuários ou clientes, mas também testadores independentes.

Formas de Teste de Aceite

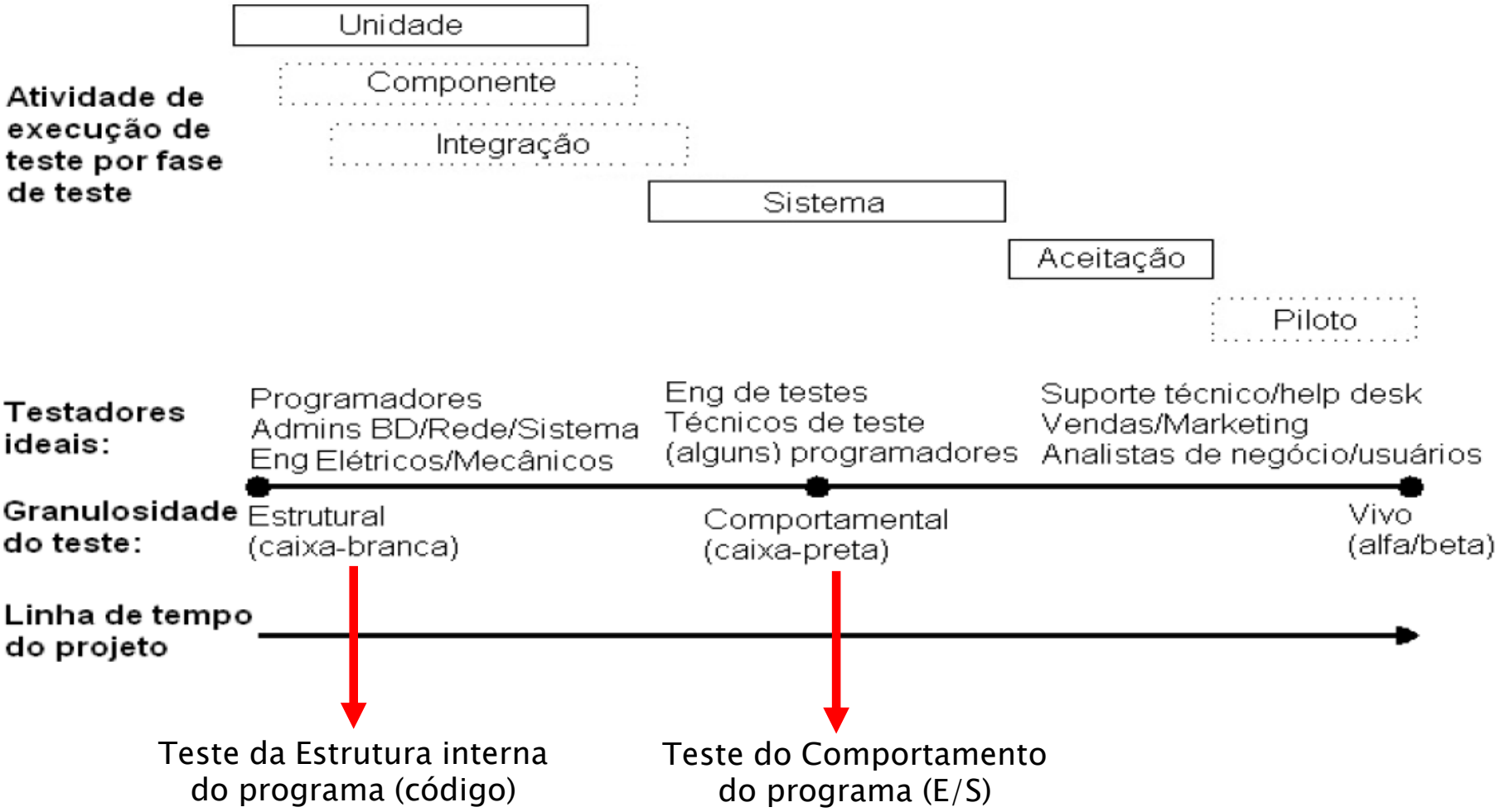
- **Teste de Aceite do Usuário ("UAT")** : Normalmente verifica se o sistema está apropriado para o uso por um usuário com perfil de negócio;

- **Teste Operacional**: Aceite pelos administradores do sistema (por exemplo, backup/restore, recuperação de desastre, gerenciamento de usuário, manutenção, segurança);

- **Teste de Contrato e Regulamento**: Verificação da Conformidade aos requisitos, regulamentos, ou normas contratualmente acordados ou de exigência legal;

- **Teste Alfa, Beta e de Campo**: Desenvolvedores de software, muitas vezes, precisam obter feedback de clientes em potencial no mercado. **Alfa teste** é feito no "site" da organização em que o produto foi desenvolvido (**infra do desenvolvedor**). **Beta teste**, ou **teste de campo**, é feito pelas pessoas em suas próprias localidades (**infra do cliente**). Ambos os testes são feitos por usuários em potencial e não por desenvolvedores.

Teste Pervasivo (diferentes)



Por que Teste Pervasivo ?

- Diferentes participantes podem testar diferentes especifidades (diferentes habilidades para diferentes estágios);

- Diferentes especifidades enfatizadas em cada nível (ou fase)
  - Teste **Unitário**: primariamente **estrutural**;
  - Teste de **Sistema**: primariamente **comportamental**;
  - Teste de **Aceite**: primariamente real (**sistema vivo**);

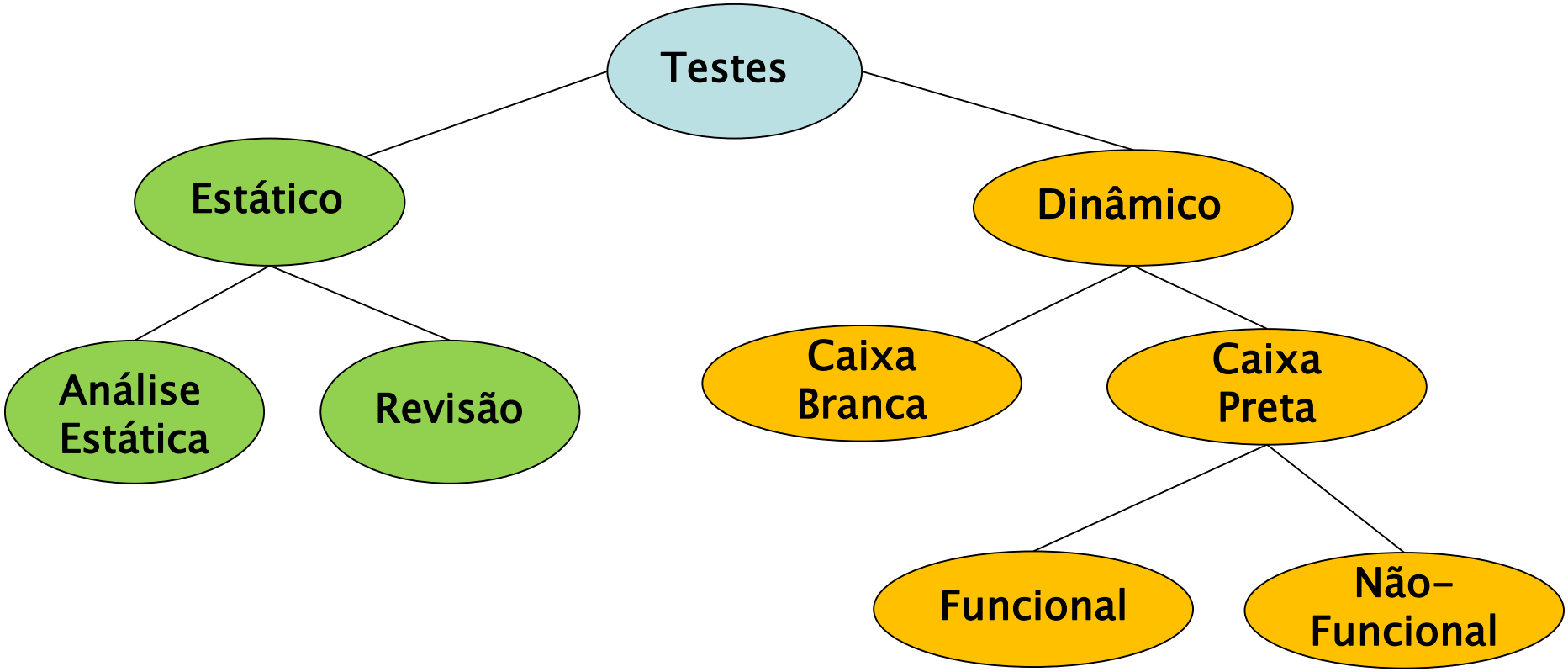
Teste deve contaminar o Desenvolvimento

- **Tarefas de Teste devem ocorrer durante todo o esforço de desenvolvimento;**
- A execução do teste é planejada com múltiplos ciclos para permitir tempo de correção.

ID	Tarefa	Abr	Mai	Jun	Jul	Ago	Set
1	Desenvolvimento de requisitos de produto e escrita de plano de projeto						
2	Análise de riscos de qualidade e escrita do plano de testes						
3	Escrita de especificações de produto						
4	Escrita de especificações do sistema de teste						
5	Desenvolvimento do produto						
6	Desenvolvimento do sistema de teste / casos de teste						
7	Depuração das unidades do produto						
8	Depuração do sistema de teste / casos de teste						
9	Execução de teste de unidade / componente						
10	Execução do teste de integração						
11	Execução do teste de sistema						
12	Execução do teste de aceite						

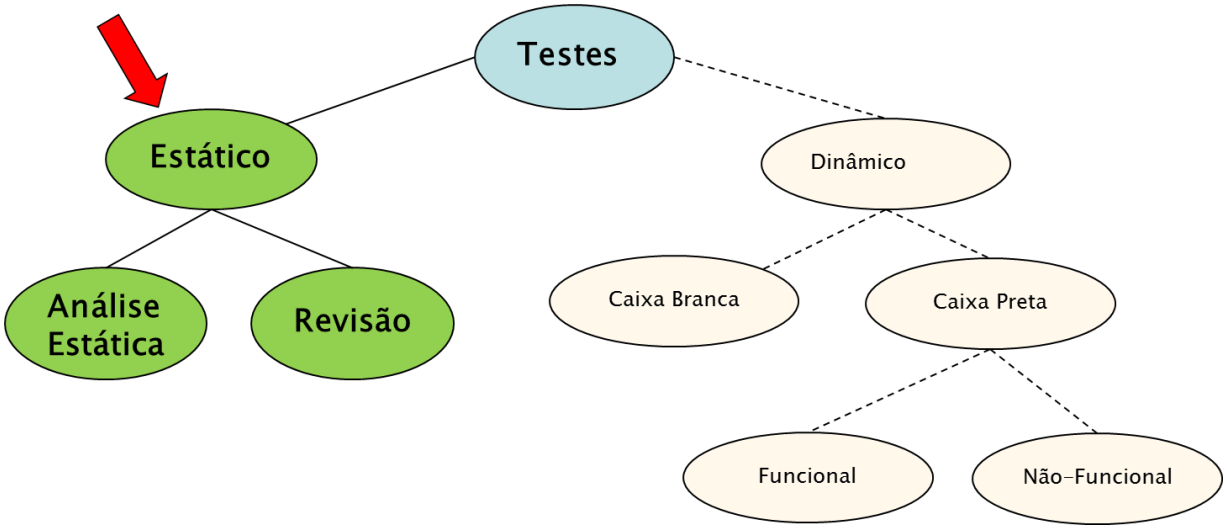
Comparando os Tipos de Teste

- Testes **Estáticos** **NÃO** **executam** programas;
- Testes **Dinâmicos** **executam** programas.



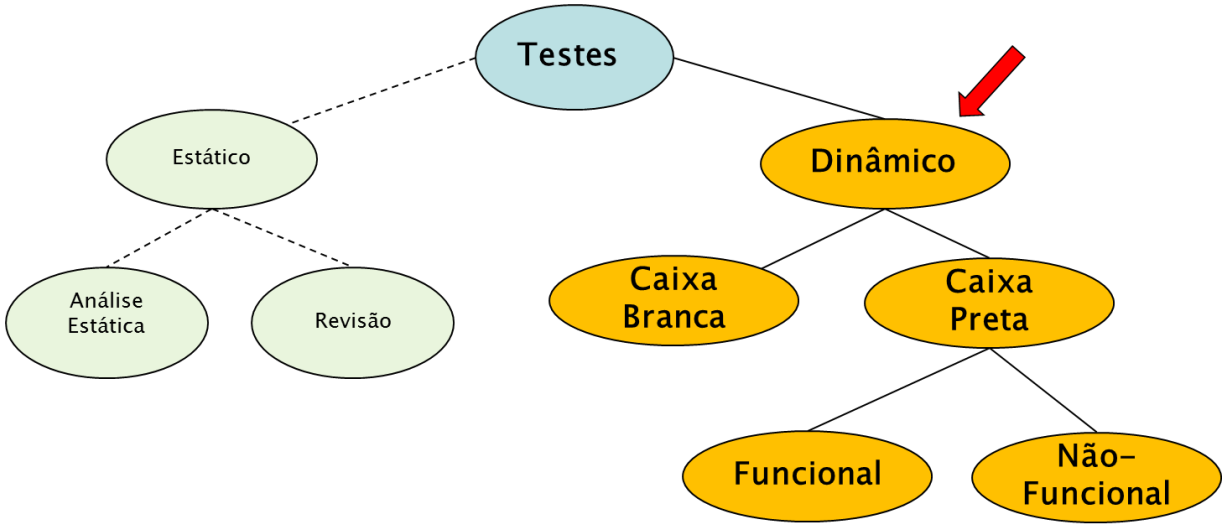
Teste Estático

- Quando se consulta um documento ou mesmo o código fonte SEM executar o programas, estamos desenvolvendo um **TESTE ESTÁTICO**;
- Portanto, com TESTE ESTÁTICO **NÃO** se **executa** o programa;
- O teste de revisão corresponde à examinar documentos, diagramas, textos, etc.
- A Análise Estática consiste em se utilizar ferramentas para investigar o código ou partes do código de programas, em busca de melhorias. Exemplo: **FxCop** - Microsoft



Teste Dinâmico

- Feito com o programa em execução, por exemplo exibindo telas...
- Testes relacionados a Mudanças (Regressão e Confirmação);
- Podem ser de **Caixa-Branca** ou de **Caixa-Preta**;
- Teste **Caixa-Branca** verifica a estrutura lógica do programa;
- Teste **Caixa-Preta** é baseado na especificação (Comportamento) do programa;
- O Teste Caixa-Preta subdivide-se em **Teste Funcional** (Negócio) e **Não-Funcional** (Tecnologia);



Testes Funcionais

- São baseados em **funções** (descritas na documentação do projeto ou compreendidos pelos testadores);
- As funções representam "**o que**" o software faz;
- Podem ser realizados em todos os níveis de teste;
- Considera o **comportamento externo** do software (**Teste Caixa-Preta**);
- Pode haver funções **não** documentadas e, nesse caso, os testadores precisam entender o significado de "**comportamento razoável**" (**bom senso**) .

Exemplo - Funcionalidade

- Ação requerida **NÃO** está presente, está inacessível ou seriamente danificada;
  - Calculadora **sem** a função de adição;
  - Função de adição implementada, a tecla "+" **não** funciona;
  - Apenas soma números inteiros, **não** números reais.
- Ação correta, resultado errado;
  - Função Soma: 2 + 2 = **3**
- Ação correta, resultado correto, efeito colateral errado;
  - Função Dividir: 5/5= **I** (formato de numeral romano)

Testes Não-Funcionais

- Relacionados à características do software;
- Incluem, mas não se limitam a: teste de **performance**, teste de **carga**, teste de **estresse**, teste de **usabilidade**, teste de **interoperabilidade**, teste de **confiabilidade** e teste de **portabilidade**;
- É o teste de "**como**" o sistema trabalha;
- Podem ser realizados em todos os níveis de teste;
- Podem ser referenciados a um modelo de qualidade como definido na norma "Engenharia de Software – Qualidade do Produto de Software – **ISO 9126**.

Teste de Estrutura (Arquitetura) do Software

- Teste **Caixa-Branca**;
- Pode ser feito em todos os níveis de teste;
- Deve ser baseado na arquitetura do software, por meio de uma **hierarquia de chamadas**;
- **Cobertura** é a **extensão que uma estrutura foi exercitada por um conjunto de testes**, expresso como uma porcentagem de itens cobertos;

Teste de Manutenção

- Uma vez desenvolvido, um sistema pode ficar **ativo** por anos ou até décadas;
- Durante esse tempo o sistema e seu ambiente podem ser **corrigidos**, **modificados** ou **completados**;
- Exemplos de modificações incluem melhorias ("**releases**"), mudanças corretivas e emergenciais, além de mudanças do ambiente, como atualização em SO ou BD e correções ("**patches**");
- Além de testar o que foi alterado, o teste de manutenção **inclui teste de regressão massivo** para as partes do sistema que não foram testadas;
- A determinação de como um sistema pode ser afetado por mudanças é chamado **Análise de Impacto**, e pode ser usada para ajudar a decidir quantos testes de regressão serão realizados;
- Pode se tornar uma tarefa complicada se as especificações estiverem **desatualizadas** ou **incompletas**.

Gatilhos para Manutenção

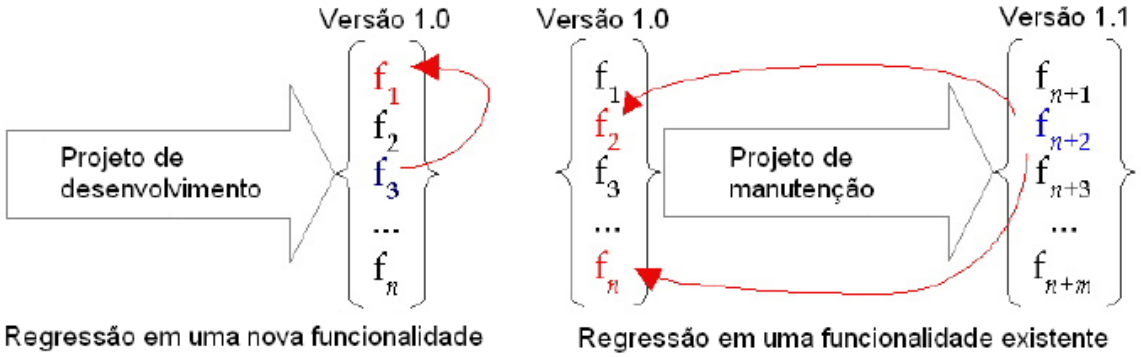
- **Modificação**: melhorias, correções de bugs, mudanças no ambiente operacional, correções;
- **Migração**: Um novo ambiente suportado;
- **Retirada**: Fim da vida de um subsistema.

Regressão

- Teste de manutenção tem por objetivo avaliar a própria mudança, mas – principalmente – avaliar se o restante do sistema está operacional (**side effect**);

- A regressão corresponde ao comportamento errado de uma função, atributo, ou funcionalidade previamente correta devido a mudança. Pode afetar tanto funcionalidades existentes quanto novas.

- Pode ser **local** (correção cria novo bug), **exposta** (correção revela bug existente) ou **remota** (correção em uma área quebra alguma coisa em outra área).



Estratégias de Regressão

- **Estratégia 1**: Repetir todos os testes

- **Estratégia 2**: Repetir alguns testes (estratégia seletiva). A seleção dos testes pode levar em conta rastreabilidade, análise de impacto, análise de risco. Pode-se usar cobertura de código para a avaliação do risco.

- **Estratégia 3**: Quando possível, a liberação do sistema deve ser feita de forma gradativa.

Padrão de Qualidade ISO 9126 (*)

- ✓ **Funcionalidade**: Conveniência, precisão, interoperabilidade, segurança, conformidade (atendimento a normas).
- ✓ **Confiabilidade**: Maturidade (robustez), tolerância a falhas, capacidade de recuperação, conformidade.
- ✓ **Usabilidade**: Facilidade de entendimento, capacidade de aprendizagem, operação, atratividade, conformidade.
- ✓ **Eficiência**: Comportamento temporal, utilização de recursos, conformidade.
- ✓ **Manutenibilidade**: Facilidade de entender código, possibilidade de efetuar mudanças, estabilidade, testabilidade, conformidade.
- ✓ **Portabilidade**: Adaptabilidade, instalabilidade, coexistência, substituição, conformidade.

Alvo de Teste Funcional

Alvo de Teste Não-Funcional

Características

Sub-características

* Substituída pela norma ISO 25010

Técnicas Estáticas

- Ao contrário das técnicas dinâmicas, as **técnicas estáticas** **não** pressupõem a execução do software que está sendo testado;
- Elas podem ser **manuais** (revisão) ou **automatizadas** (Análise Estática);
- Revisões variam desde **informais** até muito **formais**.
- Ferramentas podem executar alguns tipos de testes estáticos;
- Técnicas estáticas podem ser usadas para requisitos e projetos, além de código, esquema de bancos de dados, documentação, etc.

Revisões – Custos e Benefícios

- Custos
  - O tempo necessário para executar as revisões;
  - O esforço necessário para obter e analisar métricas;
  - A melhoria do processo;

- Benefícios
  - Prazos menores (devido a remoção eficiente de bugs);
  - Períodos de teste menores e custos menores de teste;
  - Produtividade do desenvolvedor;
  - Qualidade melhorada do produto (redução de custos no futuro);

Revisões de todos os gêneros são comprovadamente técnicas de alto retorno para a melhoria da qualidade !

Teste Estático e Dinâmico – Similaridades

- Busca identificar defeitos;
- Trabalha melhor quando um amplo conjunto transversal de stakeholders estiver envolvido;
- Poupa tempo de dinheiro da empresa.

Teste Estático e Dinâmico – Diferenças

- Cada técnica pode encontrar diferentes tipos de defeitos de modo mais efetivo e eficiente;
- Técnicas estáticas encontram **defeitos** ao invés de **falhas**.

Tipos de Revisão

- **Informal**
  - Nenhum processo formal é usado, mesmo assim é útil, baixo custo e popular;
- **Técnica**
  - Processo documentado e definido de remoção de defeitos, envolvendo pares e especialistas técnicos, mas não gerentes;
- **Acompanhamento (Walkthroughs)**
  - O **autor** "conduz" seus pares através do documento ou código;
- **Inspeções**
  - Um moderador treinado (que não seja o autor) lidera a equipe de inspeção (com papéis definidos) através de um processo formal de inspeção (regras, listas de checagem), que inclui obtenção de métricas de remoção de defeitos.

Norma IEEE 1028 – Revisões de Software

1. Visão Geral
2. Referências
3. Definições
4. Revisões Gerenciais
5. Revisões Técnicas
6. Inspeções
7. Acompanhamento
8. Auditorias

IEEE

Benefícios da Análise Estática

- Detecção de bugs mais cedo e com baixo custo (antes do início da execução dos testes);
- Manutenibilidade melhorada do código e do projeto;
- Prevenção de defeitos baseada nas métricas obtidas;
- Localização de bugs que o teste dinâmico pode não encontrar;
- Avisos sobre onde agrupamentos de bugs podem existir, devido a programação perigosa, alta complexidade, tec.
- Detecção de dependências e inconsistências nos modelos de software.

Bugs típicos encontrados na Análise Estática

- Referenciar uma variável com um **valor indefinido**;
- Interface inconsistente entre módulos e componentes;
- Variáveis que **nunca** são usadas;
- Código inatingível (**dead code**);
- Violações das **normas** de programação;
- Vulnerabilidades de segurança;
- Violações de **sintaxe** do código e modelos de software.