SCE 5764 – Engenharia de Software

Qualidade de Software e Atividades de Verificação e Validação

Profa. Ellen Francine Barbosa
{francine}@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação — ICMC/USP

Roteiro

- Casos de Falhas em Sistemas
- Qualidade de Software
- Atividades de V&V
- Revisões de Software
- Teste de Software
- Conclusão

Casos de Falhas em Sistemas I

- **Software**
- Fator chave que diferencia produtos modernos.
  - Sistemas de transportes, médicos, telecomunicações, militares, processos industriais, entretenimento, ...
- Presente no dia-a-dia das pessoas.
  - Importante que esteja correto!!!

Casos de Falhas em Sistemas II

- Por que devemos nos preocupar com a **qualidade** do software?

Trem "Fantasma"

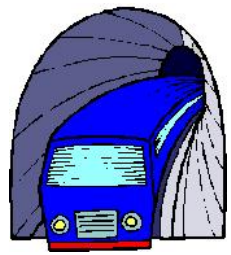
- Em 1995 registraram-se falhas nos sensores das linhas férreas situadas em túneis nos canais ingleses.
- Os trens paravam devido à suspeita de outro trem na linha.
- A causa disso era a névoa de água salgada que "confundia" os sensores.

- Motivo da falha:
  - Provavelmente os requisitos do sistema não levavam em conta a névoa de água salgada como um possível evento.
  - Testes e validação apropriados poderiam detectar o problema.

Desastre no Metrô de Nova Iorque

- Em 1995 um trem bateu atrás de outro, matando o maquinista e ferindo 54 pessoas.
- A distância entre os sinais (projetado em 1918) era menor que a distância de parada necessária para os trens atuais (maiores, mais pesados e mais rápidos). Os trens foram atualizados sem modificação no sistema de controle.

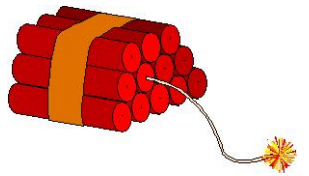
- Motivo da falha:
  - Atualização de uma parte do sistema sem validar o sistema como um todo.



O Foguete Espacial Challenger

- A explosão a bordo do Challenger é uma das falhas mais notáveis da tecnologia moderna.
- Em 1986, após 73 segundos do seu lançamento, uma explosão envolveu o foguete matando 7 astronautas.
- As investigações concluíram que o problema estava em algumas juntas do motor que não estavam projetadas para a temperatura e pressão ocorrida.

- Motivo da falha:
  - A especificação da pressão não estava de acordo com os requisitos do sistema.
  - Os testes realizados foram inapropriados para detectar a falha.



O Veículo Espacial Ariane 5

- Em 1996, o veículo espacial Ariane 5 saiu do curso e explodiu segundos após o seu lançamento.
- Levou uma década de desenvolvimento e custou 7 bilhões de dólares.
- Os testes insuficientes em componentes reutilizados do veículo Ariane 4 foram a causa do acidente.

- Motivo da falha:
  - Erro de software no cálculo da velocidade horizontal do foguete.
    - A variável que armazenava este valor tinha 64 bits (floating point) e foi erroneamente modificada para 16 bits (signed integer).
    - O valor era maior que 32.767 (maior inteiro), gerando uma falha de conversão!!!

Casos de Falhas em Sistemas

- Por que é difícil construir software com qualidade?
- Existe algum mecanismo para garantir a qualidade do software? Como utilizá-lo eficientemente?

Engenharia de Software

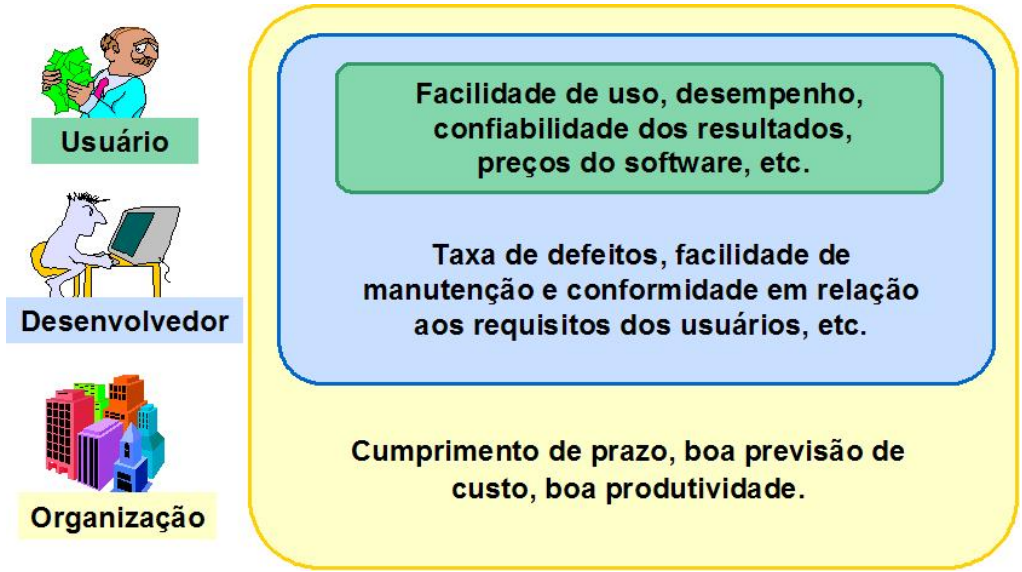
Conjunto de princípios, métodos e técnicas que tratam o software como produto de engenharia que requer planejamento, projeto, implementação e manutenção.

- Objetivo Principal
  - Produzir software de **alta qualidade**, com um baixo custo.

Qualidade de Software

Qualidade é a totalidade de características e critérios de um produto ou serviço que exercem suas habilidades para satisfazer às necessidades declaradas ou envolvidas.

- Visões de Qualidade de Software



Garantia de Qualidade (SQA – Software Quality Assurance) I

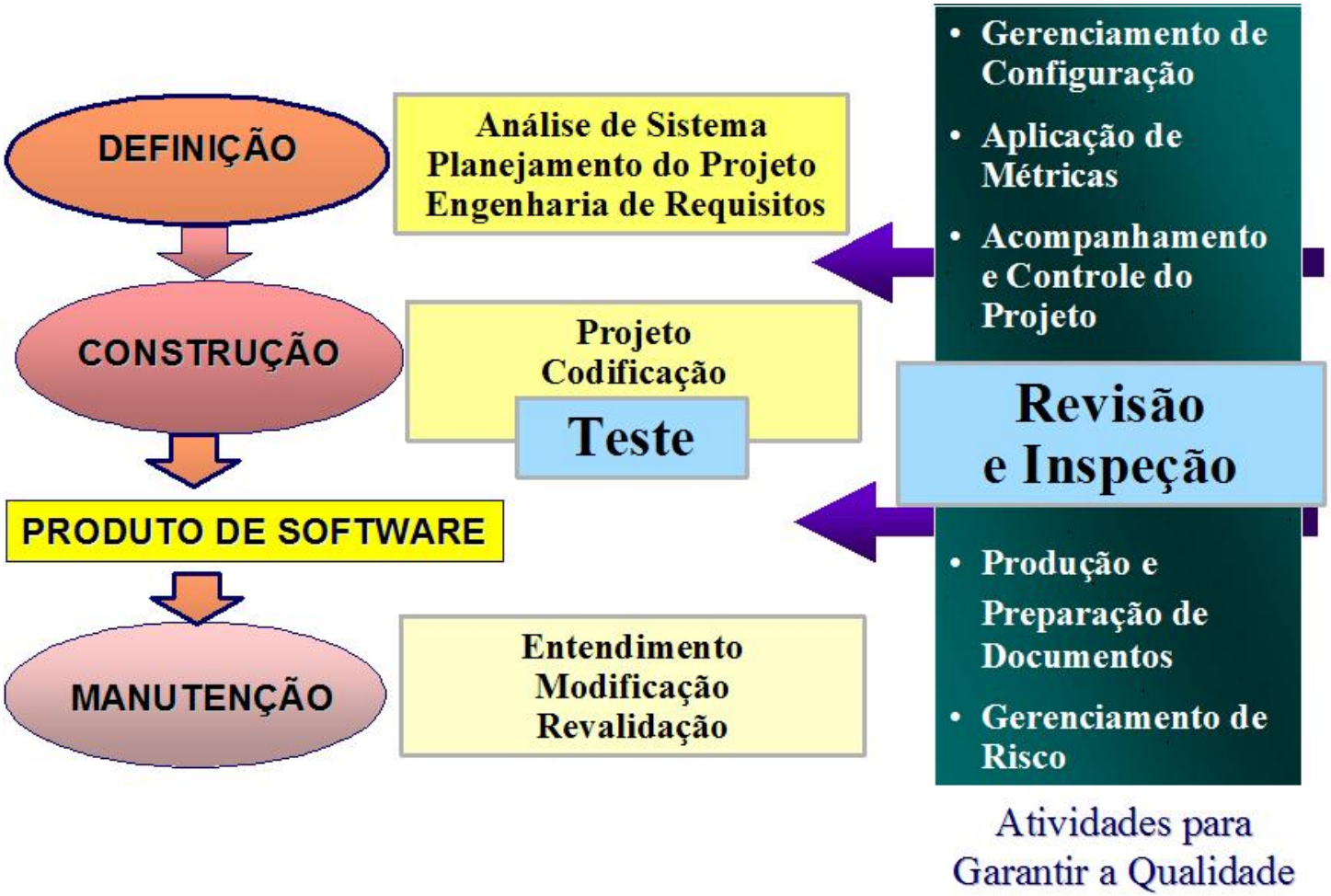
- Conjunto de atividades técnicas aplicadas durante todo o processo de desenvolvimento.
  - Garantir que tanto o processo de desenvolvimento quanto o produto de software atinjam os níveis de qualidade especificados.

- Atividades de SQA
  - Aplicação de métodos técnicos.
    - Ajudar o analista a conseguir uma especificação de qualidade.
    - Ajudar o projetista a desenvolver um projeto de qualidade.
  - Realização de revisões.
    - Avaliar a qualidade da especificação, do projeto, do código, ...

Garantia de Qualidade (SQA – Software Quality Assurance) II

- Atividades de teste de software.
  - Ajudar a garantir que a detecção de erros seja efetiva.
- Aplicação de padrões e procedimentos formais.
  - Garantir que estes sejam seguidos durante o desenvolvimento.
- Processo de controle de mudanças.
  - Atividade associada ao gerenciamento de configuração de software.
- Mecanismos de medição.
  - Apoio no acompanhamento da qualidade de software.
  - Avaliar o impacto de mudanças metodológicas e procedimentais.
- Anotação e manutenção de registros.
  - Procedimentos para coleta e disseminação de informações de garantia de qualidade.

SQA no Processo de Desenvolvimento de Software I



SQA no Processo de Desenvolvimento de Software II

- Aspectos Positivos
  - O software terá menos defeitos latentes.
  - Maior confiabilidade resultará em maior satisfação do cliente.
  - O custo do ciclo de vida global do software é reduzido.
  - Os custos de manutenção podem ser reduzidos.
- Aspectos Negativos
  - Difícil de ser instituída em pequenas empresas.
  - Representa uma mudança cultural.
    - Mudança nunca é fácil!!!

Atividades de V&V I

- Dentre as atividades de SQA estão as atividades de verificação e validação de software.
- O objetivo é minimizar a ocorrência de erros e riscos associados.
  - Detectar a presença de erros nos produtos de software.

Atividades de V&V II

- Verificação
  - Assegurar consistência, completitude e corretitude do produto em cada fase e entre fases consecutivas do ciclo de vida.

Estamos construindo corretamente o produto?

  - Assegurar que o produto, ou uma determinada função do mesmo, esteja sendo implementado corretamente.
    - Verifica-se inclusive se os métodos e processos de desenvolvimento foram adequadamente aplicados.

Atividades de V&V III

- Validação
  - Assegurar que o produto sendo desenvolvido corresponde ao produto correto, conforme os requisitos do usuário.

Estamos construindo o produto certo?

Atividades de V&V IV

- V&V abrangem um amplo conjunto de atividades de SQA:
  - Revisões técnicas formais
  - Auditoria de qualidade e configuração
  - Monitoramento de desempenho
  - Simulação
  - Estudo de viabilidade
  - Revisão da documentação
  - Revisão da base de dados
  - Testes
- V&V envolvem atividades de análise estática e de análise dinâmica.

Análise Estática

- Não requerem a execução propriamente dita do produto.
- Podem ser aplicadas em qualquer produto intermediário do processo de desenvolvimento.
  - Documento de requisitos, diagramas de projeto, código-fonte, planos de teste, ...
- As revisões são o exemplo mais clássico de análise estática.
  - **Inspeção**
  - Walkthrough
  - Peer Review

Análise Dinâmica

- Requerem a execução do produto.
  - Código ou quaisquer outras representações executáveis do sistema.
- Exemplos de atividades que constituem uma análise dinâmica do produto:
  - **Teste de Software**
  - Simulação

Revisões de Software I

- Meio efetivo para melhorar a qualidade de software.
  - Filtro para o processo de Engenharia de Software.
- Podem ser aplicadas em vários pontos durante o desenvolvimento do software.
- Maneira de usar a diversidade de um grupo de pessoas para:
  - Apontar melhorias necessárias ao produto.
  - Confirmar as partes de um produto em que uma melhoria não é desejada ou não é necessária.
  - Realizar um trabalho técnico de qualidade mais uniforme de forma a torná-lo mais administrável.

Objetivo das Revisões de Software I

- Encontrar erros durante o processo de desenvolvimento, de forma que eles não se transformem em defeitos depois da entrega do software.
- Descoberta precoce dos erros.
  - Melhoria da qualidade já nas primeiras fases do processo de desenvolvimento.
  - Aumento da produtividade e diminuição dos custos.
    - Erros são detectados quando sua correção é mais barata.

Objetivo das Revisões de Software II

- É uma oportunidade de treinamento.
  - Aprender por experiência.
  - Participantes aprendem as razões e padrões em descobrir erros.
  - Participantes aprendem bons padrões de desenvolvimento de software.
- Com o decorrer do tempo....
  - A revisão auxilia os participantes a desenvolver produtos mais fáceis de entender e de manter.

Tipos de Revisões

- Discussão informal de um problema técnico.
- Apresentação do projeto de software para uma audiência de clientes, administradores e pessoal técnico.
- Revisões Técnicas Formais (RTF), as quais incluem avaliações técnicas do software realizadas em pequenos grupos.

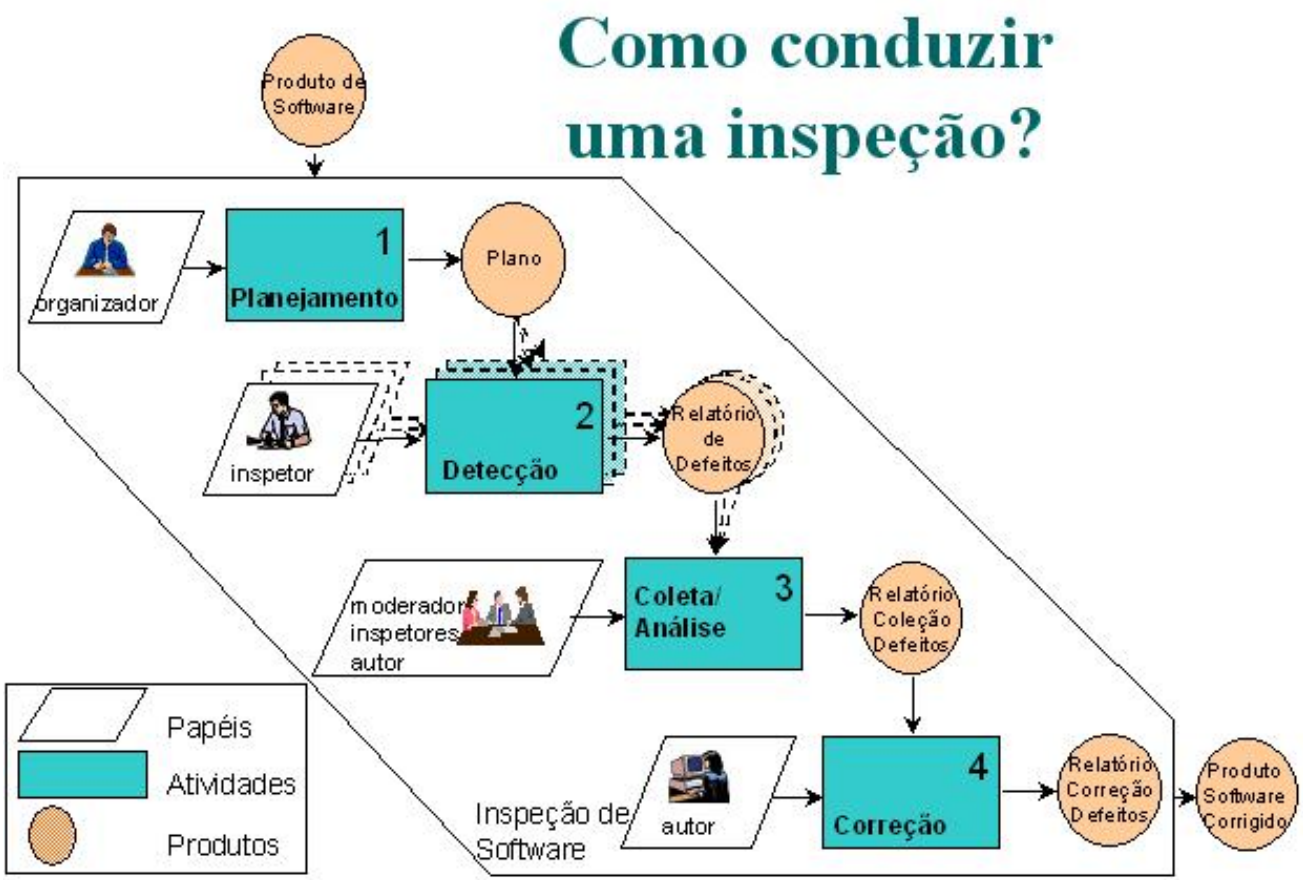
Revisões Técnicas Formais

- Métodos de RTF:
  - Inspeção
  - Walkthrough
  - Peer-Review

Inspeção de Software

- Método de análise estática para verificar a qualidade de um produto de software.
- Pode-se inspecionar tanto produtos de software como também projetos de software.
  - Diferencial está na seleção dos aspectos que devem ser considerados durante a revisão.
- **Inspeção em Documentos de Requisitos**.
- **Inspeção em Código-fonte**.

Etapas da Inspeção



Vantagens da Inspeção

- Detecção antecipada de defeitos (inspeção de requisitos).
- Aprende-se pela experiência.
  - Participantes aprendem os padrões e o raciocínio utilizado na detecção de defeitos.
  - Participantes aprendem bons padrões de desenvolvimento.
- A longo prazo...
  - A inspeção convence os participantes a desenvolverem produtos mais compreensíveis e mais fáceis de manter.

As inspeções ajudam a integrar o processo de **prevenção** de defeitos com o processo de **detecção** de defeitos.

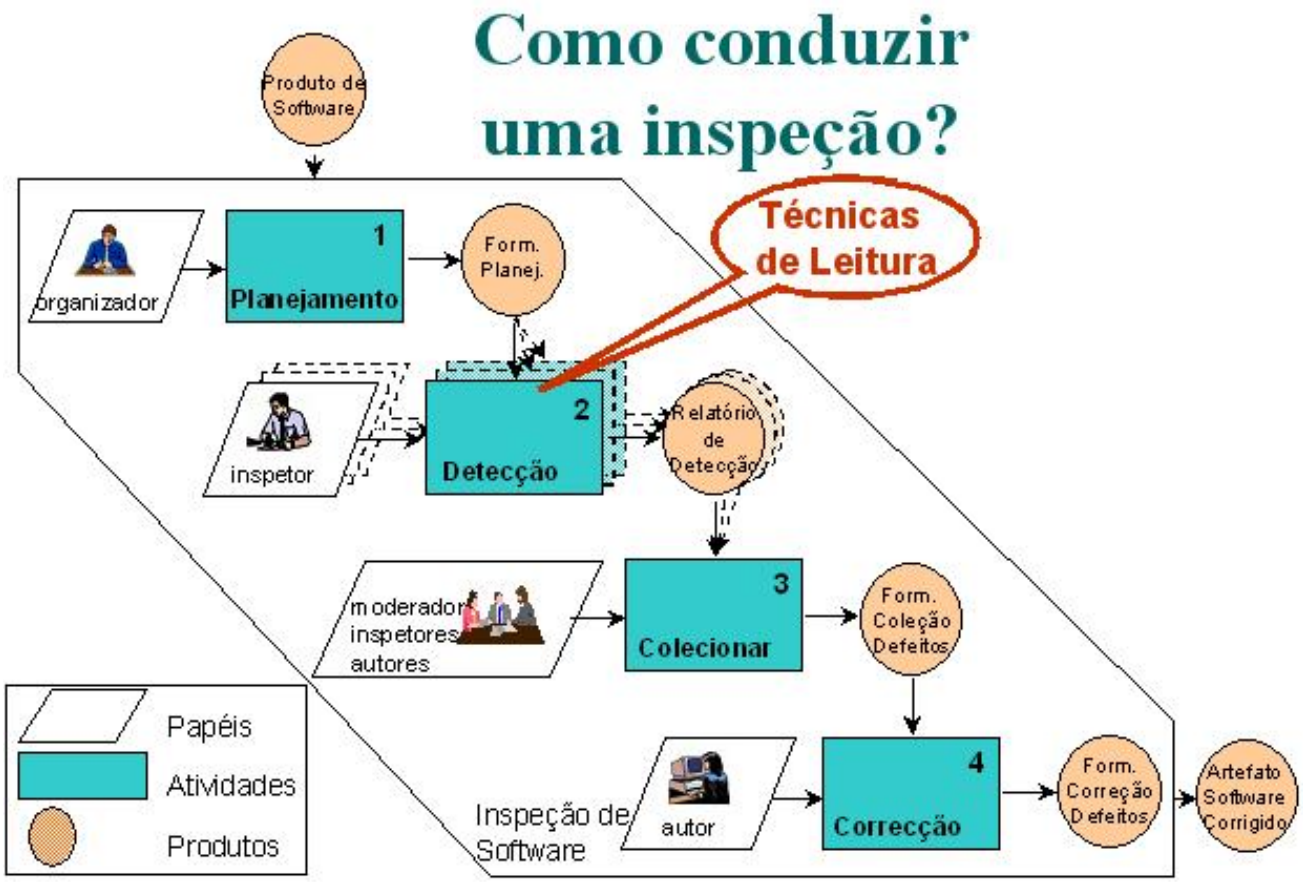
Técnicas de Leitura para Inspeção I

- Como detectar defeitos?
  - Lendo o documento.
  - Entendendo o que o documento descreve.
  - Verificando as propriedades de qualidade requeridas.
- Problema:
  - **Em geral não se sabe como fazer a leitura de um documento!!!**
- Razão:
  - Em geral, os desenvolvedores aprendem a escrever documento de requisitos, código, projeto, mas não aprendem a fazer uma leitura adequada dos mesmos.

Técnicas de Leitura para Inspeção II

- Solução:
  - Fornecer **técnicas de leitura** bem definidas.
- Benefícios:
  - Aumenta a relação custo/benefício das inspeções.
  - Fornece modelos para escrever documentos com maior qualidade.
  - Reduz a subjetividade nos resultados da inspeção.

Técnicas de Leitura para Inspeção III



Técnicas de Leitura para Inspeção IV

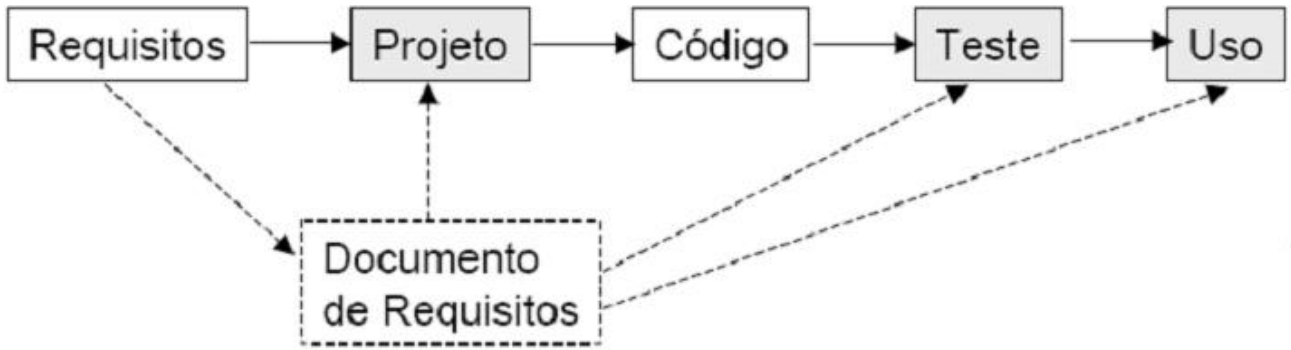
- O que é uma técnica de leitura?
  - Conjunto de instruções fornecido ao revisor dizendo como ler e o quê procurar no produto de software.
- Algumas técnicas de leitura:
  - Ad-hoc
  - Checklist
  - Leitura Baseada em Perspectiva (PBR)
  - Stepwise Abstraction

Técnica de Leitura Baseada em Perspectiva (PBR) I

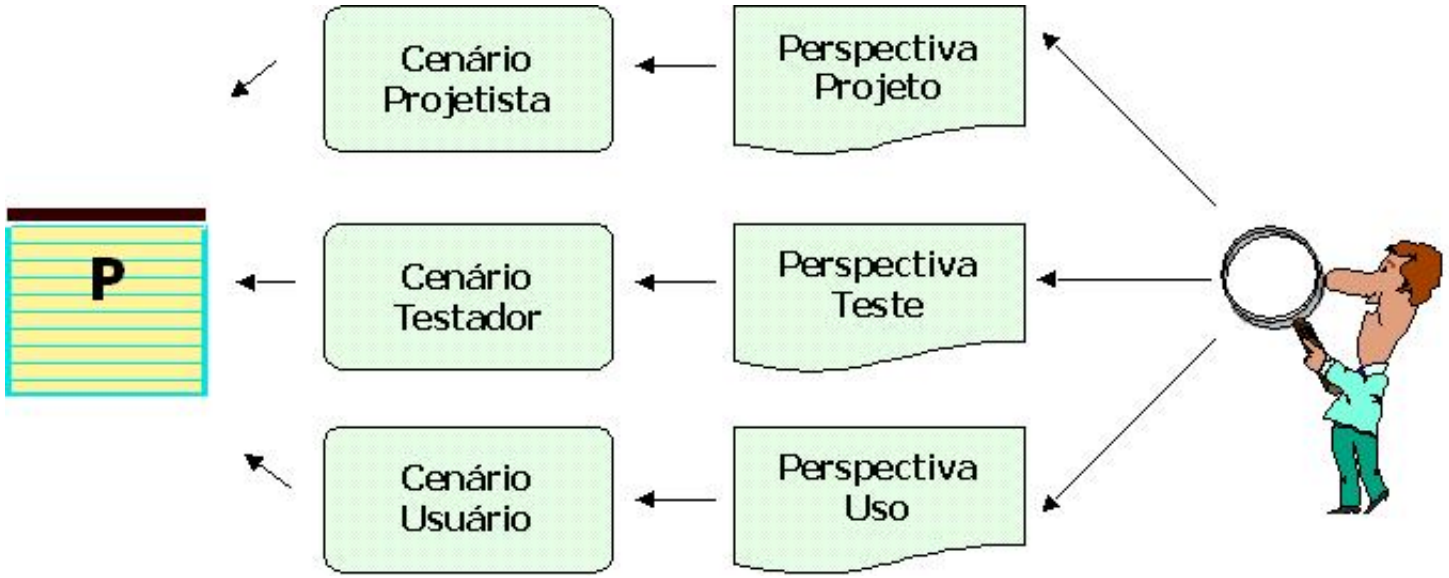
- Técnica de leitura proposta para detectar defeitos em especificações de requisitos.
- Faz com que cada revisor se torne responsável por uma perspectiva em particular.
- Possibilita que o revisor melhore sua experiência em diferentes aspectos do documento de requisitos.
- Assegura que perspectivas importantes sejam contempladas.
- Cada revisor lê o Documento de Requisitos com olhos diferentes.
- Benefícios:
  - Determina uma responsabilidade específica para cada revisor.
  - Melhora a cobertura de defeitos.

Técnica de Leitura Baseada em Perspectiva (PBR) II

- Várias leituras podem ser feitas no Documento de Requisitos.
  - O projetista que usa o DR para gerar o projeto do sistema.
  - O testador que, com base no DR, deve gerar casos de teste para testar o sistema quando este estiver implementado.
  - O usuário que verifica se o DR está capturando toda funcionalidade que ele deseja para o sistema.



Técnica de Leitura Baseada em Perspectiva (PBR) III



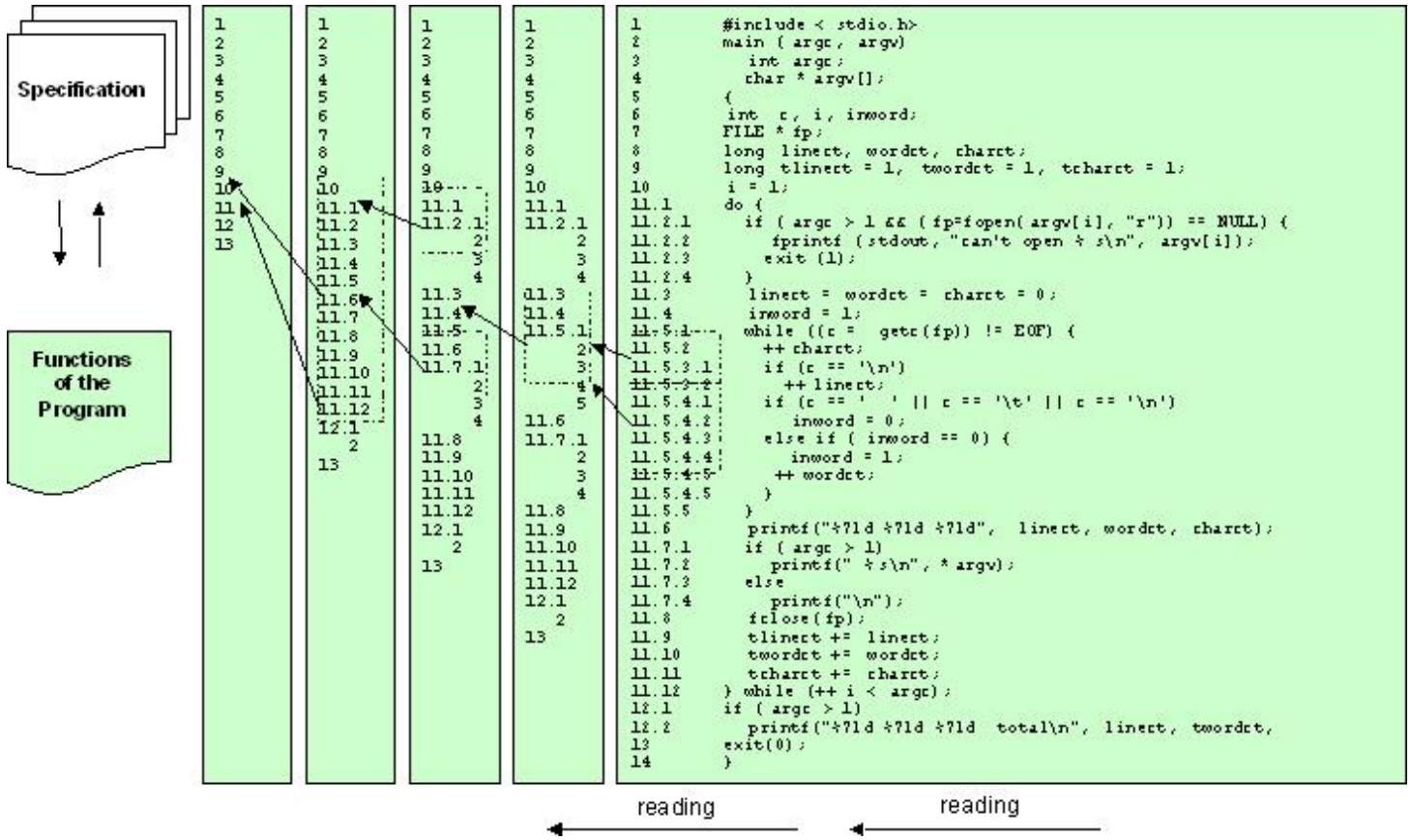
Técnica de Leitura Stepwise Abstraction I

- Técnica de leitura utilizada para detectar defeitos em código-fonte.
  - Code-Reading
- Consiste em desenvolver abstrações funcionais a partir do código-fonte, para determinar a funcionalidade do programa.

Técnica de Leitura Stepwise Abstraction II

- Os revisores identificam subprogramas no código-fonte e escrevem suas próprias especificações para os subprogramas (abstrações de funcionalidade).
  - As abstrações construídas devem ser combinadas em uma abstração mais geral, até que se tenha capturado a função completa do programa.
- Inconsistências são detectadas comparando a especificação original com a especificação construída por meio das abstrações.

Técnica de Leitura Stepwise Abstraction III



Walkthrough

- Procedimento similar ao procedimento para condução de uma inspeção.
- A diferença fundamental está na maneira como a sessão de revisão é conduzida.
  - Em vez de ler o programa ou checar os erros por meio de um *checklist*, os participantes simulam sua execução.
  - Papel adicional: testador.
    - Elaborar um pequeno conjunto de casos de teste (em papel).
    - Monitorar e controlar os resultados obtidos.

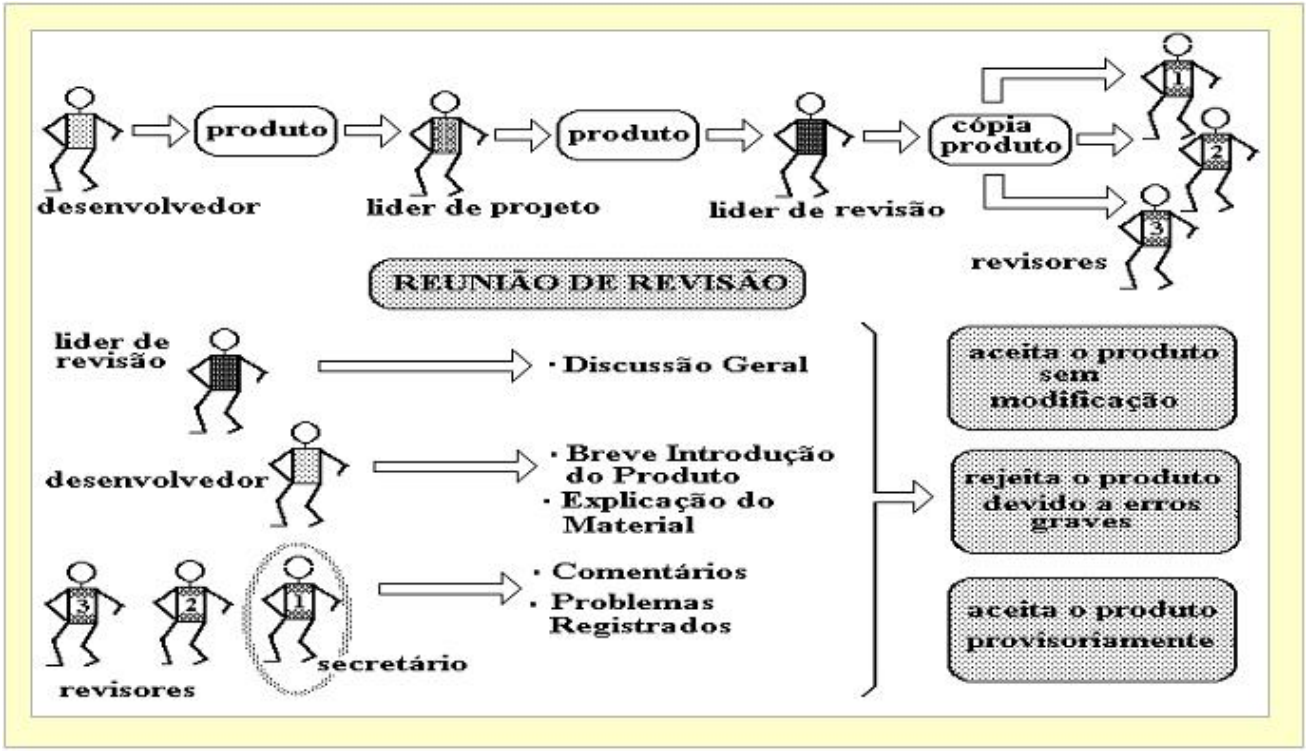
Peer-Review

- Conduzida por pares de programadores.
  - Mesmo nível de conhecimento.
- Aplicada ao código.
- Reuniões com duração de 1 a 2 horas.
  - Somente um programa ou parte dele (rotinas) deve ser revisado.
- Resultados são publicados em um relatório informal.
  - Não faz parte da documentação oficial do projeto.

Reunião de Revisão Técnica I

- Independentemente do formato da RTF, toda reunião de revisão deve seguir as seguintes recomendações:
  - Envolver de 3 a 5 pessoas.
  - Deve haver uma preparação para a reunião.
    - A preparação não deve exigir mais de 2 horas de trabalho de cada pessoa.
  - A reunião deve durar menos de 2 horas.
  - Deve-se focalizar uma parte específica do software.
    - Maior probabilidade de descobrir erros.

Reunião de Revisão Técnica II



Diretrizes para Revisão

- Revise o produto, não o produtor.
- Fixe e mantenha uma agenda.
- Limite o debate e a refutação.
- Relacione as áreas problemáticas.
- Faça anotações por escrito.
- Limite o número de participantes e insista em uma preparação antecipada.
- Desenvolva uma lista de conferência (checklist) para cada produto que provavelmente será revisto.
- Atribua recursos e uma programação de tempo para as revisões.
- Realize um treinamento significativo para todos os revisores.
- Reveja suas antigas revisões.

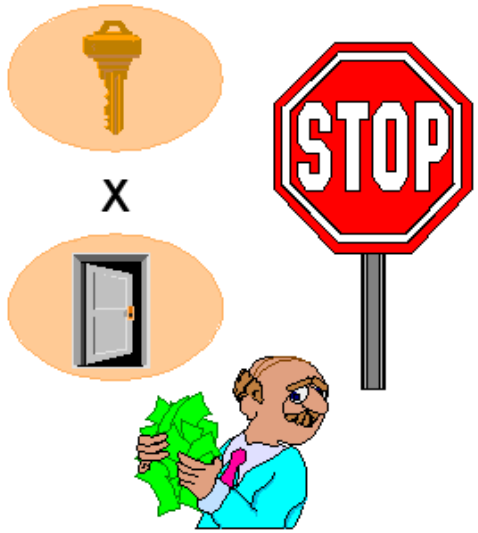
Teste de Software

- Análise dinâmica do produto de software.
  - Processo de executar o software de modo controlado, observando seu comportamento em relação aos requisitos especificados.
- Processo de executar um programa com a intenção de encontrar erros.
  - O teste bem sucedido é aquele que consegue determinar casos de teste que resultem na falha do programa sendo analisado.

Teste de Software

- Como testar esse produto?
  - Que casos de teste utilizar?
  - Como garantir que o produto não contém erros? Critérios...
  - Como decidir se o produto foi suficientemente testado?

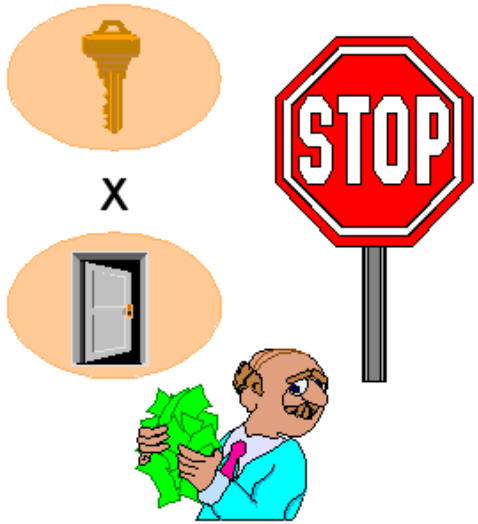
O programa *Identifier* determina se um identificador é válido ou não. Um identificador válido deve começar com uma letra, conter apenas letras ou dígitos e ter no mínimo um caractere e no máximo seis caracteres de comprimento.



Teste de Software

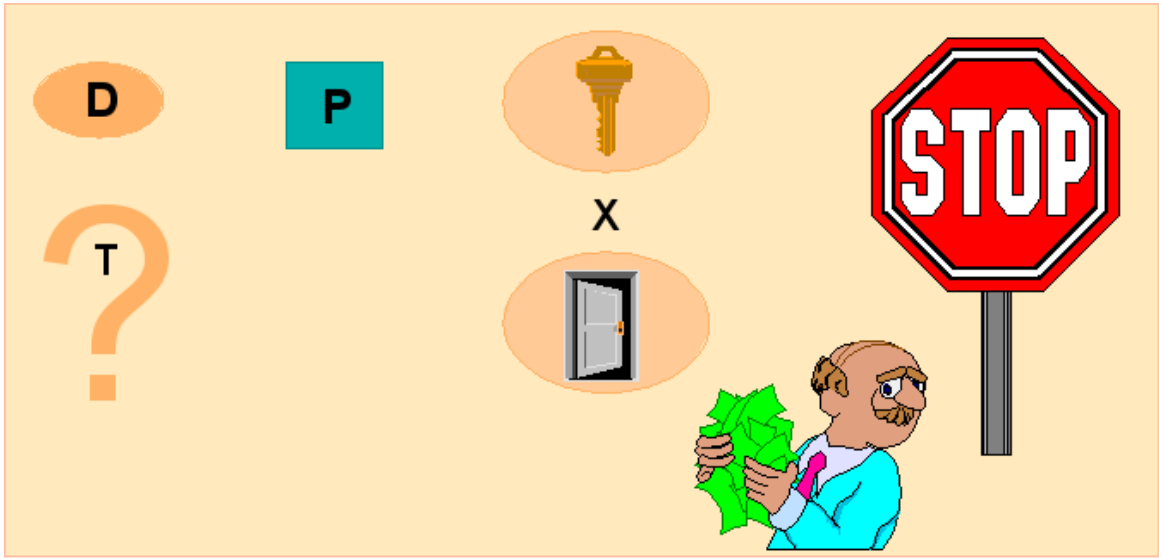
- Como testar esse produto?
  - Que casos de teste utilizar?
  - Como garantir que o produto não contém erros? Critérios...
  - Como decidir se o produto foi suficientemente testado?


```
main () {
    int   valor, num, fat;
    fat= 1;
    scanf("%d",&valor);
    num = valor;
    if (num >= 0) {
      while (num > 1) {
        fat = fat * num;
        num--; }
      printf("%d\n",fat); }
    else
      printf("Erro!\n"); }
```



Objetivos do Teste

- Revelar a presença de erros.



- Inexistência de erro:
  - Software é de alta qualidade?
  - Conjunto de casos de teste T é de baixa qualidade?

Objetivos do Teste

- Revelar a presença de erros.

Através da atividade de teste pode-se detectar a **presença de erros** em um programa, embora não se possa concluir a ausência deles.

- Quando o processo de teste não detecta erros...
  - O teste pode não ter sido suficientemente completo para revelar os erros existentes.
  - Não se pode afirmar que o programa esteja isento de erros e, portanto, correto.

Objetivos do Teste

- Verificar se o software executa conforme especificado na documentação do projeto.
- Fornecer uma avaliação da qualidade do software.

Apesar de não ser possível, por meio de testes, provar que um programa está correto, estes contribuem para **aumentar a confiança** de que o software desempenha as funções especificadas.

Desafios do Teste

- Alguém já testou algum produto de software?

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.
  - Requisitos do software inexistentes ou que mudam rapidamente.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.
  - Requisitos do software inexistentes ou que mudam rapidamente.
  - Não há tempo para o teste exaustivo.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.
  - Requisitos do software inexistentes ou que mudam rapidamente.
  - Não há tempo para o teste exaustivo.
  - Não há treinamento no processo de teste.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.
  - Requisitos do software inexistentes ou que mudam rapidamente.
  - Não há tempo para o teste exaustivo.
  - Não há treinamento no processo de teste.
  - Não há ferramenta de apoio.

Desafios do Teste

- Alguém já testou algum produto de software?
- Quais foram os maiores desafios?
- Alguns problemas comuns...
  - Não há tempo suficiente para o teste.
  - Muitas combinações de entrada para serem exercitadas.
  - Dificuldade em determinar os resultados esperados para cada caso de teste.
  - Requisitos do software inexistentes ou que mudam rapidamente.
  - Não há tempo para o teste exaustivo.
  - Não há treinamento no processo de teste.
  - Não há ferramenta de apoio.
  - Gerentes que desconhecem teste ou que não se preocupam com qualidade.

Limitações do Teste

- Observe o exemplo a seguir:

```
1  int blech(int j) {
2      j = j - 1;  // deveria ser j = j + 1
3      j = j / 30000;
4      return j;
5  }
```

Código-fonte: `blech.java`

- Considerando o tipo inteiro com 16 bits (2 bytes), o menor valor possível seria -32.768 e o maior seria 32.767, resultando em 65.536 valores diferentes possíveis.
- Haverá tempo suficiente para se criar 65.536 casos de teste? E se os programas forem maiores? Quantos casos de teste serão necessários?

Limitações do Teste

```
1  int blech(int j) {
2      j = j - 1;  // deveria ser j = j + 1
3      j = j / 30000;
4      return j;
5  }
```

Código-fonte: `blech.java`

- Quais valores escolher?

Entrada (j)	Saída Esperada	Saída Obtida
1	0	0
42	0	0
20000	0	0
-20000	0	0

- Quais valores de entrada revelam o erro?

Limitações do Teste

- Nenhum dos casos de testes anteriores revelou o erro.

Limitações do Teste

- Nenhum dos casos de testes anteriores revelou o erro.
- Somente quatro valores do intervalo de entrada válido revelam o erro:

Entrada (j)	Saída Esperada	Saída Obtida
-30000	0	-1
-29999	0	-1
30000	1	0
29999	1	0

Limitações do Teste

- Nenhum dos casos de testes anteriores revelou o erro.
- Somente quatro valores do intervalo de entrada válido revelam o erro:

Entrada (j)	Saída Esperada	Saída Obtida
-30000	0	-1
-29999	0	-1
30000	1	0
29999	1	0

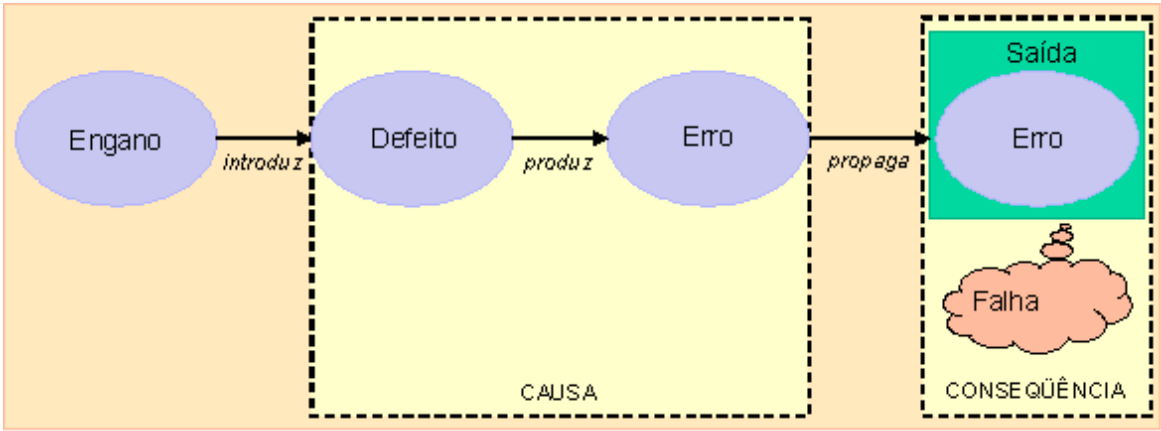
- Qual a chance de tais valores serem selecionados???

Limitações do Teste

- Corretitude de Programas
- Equivalência de Programas
- Executabilidade
- Correção Coincidente

Terminologia de Erros

- Engano x Defeito x Erro x Falha (Padrão IEEE 610.12-1990)
  - Um engano introduz um defeito no software.
  - O defeito, quando ativado, pode produzir um erro.
  - O erro, se propagado até a saída do software, constitui uma falha.



Engano x Defeito x Erro x Falha

- Considere o comando $(z = y + x)$ substituído por engano pelo comando $(z = y - x)$.
- Se o defeito introduzido for ativado com $(x = 0)$:
  - Nenhum valor incorreto para a variável z é produzido.
  - O defeito é ativado mas não produz um erro e, conseqüentemente, não ocorre uma falha.
- Para qualquer outro valor de x $(x \neq 0)$:
  - A ativação do defeito produz um erro na variável z.
  - Tal erro, se propagado até a saída, caracteriza uma falha.

Caso de Teste

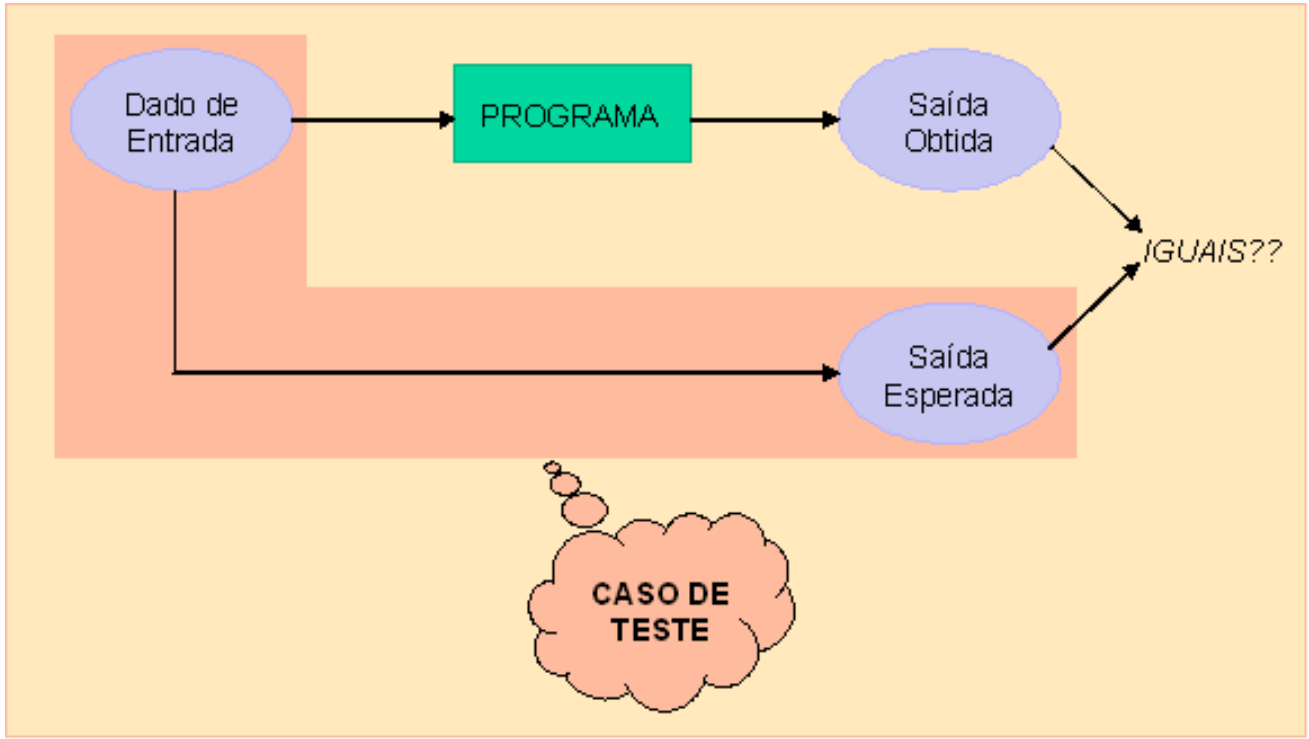
- O segredo do sucesso do teste está no projeto dos casos de teste.
- Um bom caso de teste tem alta probabilidade de revelar um erro ainda não descoberto.
- Casos de teste também podem revelar especificações incompletas ou ambíguas.

Partes de um Caso de Teste

Par ordenado (d; S(d)) no qual d é um elemento pertencente ao domínio de entrada D e S(d) corresponde à sua respectiva saída esperada.

- Dado de entrada (*inputs*).
  - Dado necessário para uma execução do programa.
- Saída esperada (*outputs*).
  - Resultado de uma execução do programa.
  - Pressupõe-se a existência de um oráculo de teste.

Caso de Teste



Ordem de Execução

- Existem dois estilos de projeto de casos de teste relacionados com a ordem de execução:
  - Casos de teste em cascata.
  - Casos de teste independentes.

Casos de Teste em Cascata

- Os casos de teste devem ser executados um após o outro, em uma ordem específica.
- O estado do sistema deixado pelo primeiro caso de teste é reaproveitado pelo segundo e assim sucessivamente.
- Vantagem: Casos de testes tendem a ser pequenos e simples. Fáceis de serem projetados, criados e mantidos.
- Desvantagem: Se um caso de teste falhar, os casos de teste subseqüentes também podem falhar.

Casos de Teste Independentes

- Cada caso de teste é inteiramente auto-contido.
- Vantagem: Casos de teste podem ser executados em qualquer ordem.
- Desvantagem: Casos de teste tendem a ser grandes e complexos, mais difíceis de serem projetados, criados e mantidos.

Técnicas de Teste

- Diferentes tipos de teste podem ser utilizados para verificar se um programa comporta-se como especificado.
- Basicamente, os testes podem ser classificados em teste caixa-preta (*black-box testing*), teste caixa-branca (*white-box testing*) e teste baseado em erros (*error-based testing*).
- Esses tipos de teste correspondem às chamadas técnicas de teste.

Técnicas de Teste

- As técnicas de teste são definidas conforme o tipo de informação utilizada para realizar o teste.
- Contemplam diferentes perspectivas do software: aspecto complementar!!!!
  - Técnica Caixa-Preta
    - Os testes são baseados exclusivamente na especificação de requisitos do programa.
    - Nenhum conhecimento de como o programa está implementado é requerido.
  - Técnica Caixa-Branca
    - Os testes são baseados na estrutura interna do programa, ou seja, na implementação do mesmo.
  - Técnica Baseada em Erros
    - Os requisitos de teste são derivados a partir dos erros mais freqüentes cometidos durante o processo de desenvolvimento do software.

Critérios de Teste

- Maneira sistemática e planejada para conduzir os testes.
- Fornece indicações a respeito de quais casos de teste utilizar de modo a aumentar as chances de revelar erros no programa.
- Quando erros não forem revelados...
  - Estabelecer um nível elevado de confiança na correção do programa.

Critérios de Teste

- Propriedades Mínimas:
  1. Garantir, do ponto de vista de fluxo de controle, a cobertura de todos os desvios condicionais.
  2. Requerer, do ponto de vista de fluxo de dados, ao menos um uso de todo resultado computacional.
  3. Requerer um conjunto de casos de teste finito.

Requisitos de Teste

- Cada critério estabelece um conjunto de requisitos de teste específico.
- Casos de teste são selecionados de modo a satisfazer os requisitos estabelecidos pelo critério em questão.
- Dados um programa P, um conjunto de casos de teste T e um critério C, diz-se que:
  - T é C-adequado para o teste de P se T preencher os requisitos de teste estabelecidos pelo critério C.

Critérios da Técnica Funcional

- Particionamento em Classes de Equivalência
- Análise do Valor Limite
- Grafo de Causa-Efeito

Critérios da Técnica Estrutural

- Baseados em Complexidade
  - Critério de McCabe (teste do caminho base).
- Baseados em Fluxo de Controle
  - Todos-Nós
  - Todas-Arestas
  - Todos-Caminhos: Simples, Completo, Livre de Laço, ...
- Baseados em Fluxo de Dados
  - Critérios de Rapps & Weyuker
    - Todas-Defs, Todos-Usos, Todos-P-Usos, e outros.
  - Critérios Potenciais-Usos (Maldonado)
    - Todos-Potenciais-Usos, Todos-Potenciais-Usos/DU, e outros.

Critérios da Técnica Baseada em Erros

- Semeadura de Erros
- Teste de Mutação
  - Análise de Mutantes (unidade)
  - Mutação de Interface (integração)

Critérios de Teste

Critério de Geração

Procedimento para **escolher** casos de teste para o teste de P.

- T é C-adequado por construção.

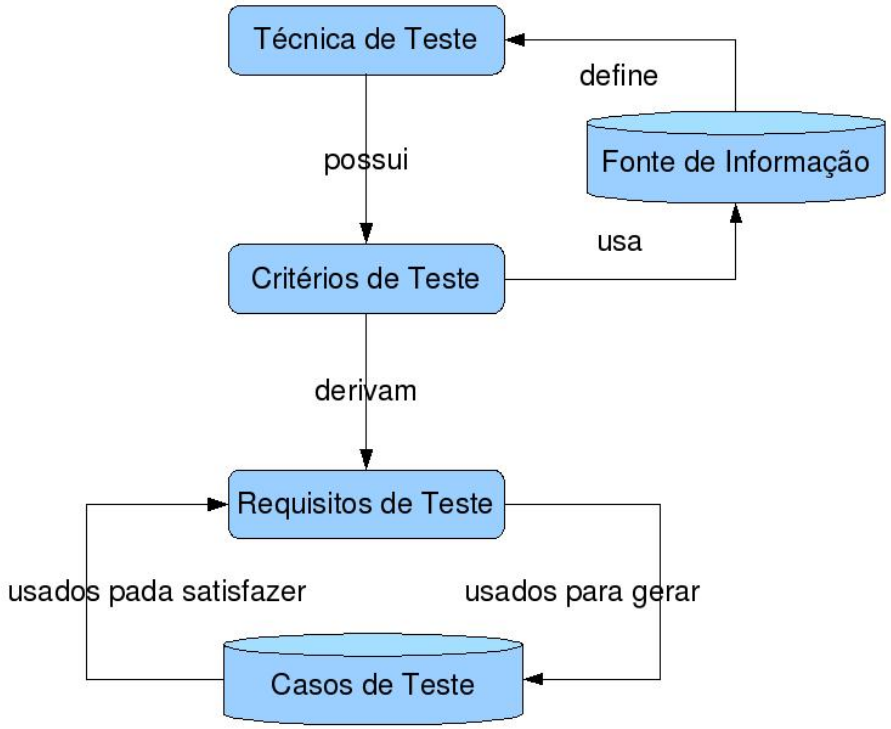
Critério de Adequação

Predicado para **avaliar** um conjunto de casos de teste T no teste do programa P.

- Um critério de adequação C é utilizado para verificar se o conjunto de casos de teste T satisfaz os requisitos de teste estabelecidos pelo critério.

Técnicas, Critérios e Requisitos de Teste

- Termos utilizados na área de teste e seus relacionamentos.



Etapas de Teste

- A atividade de teste envolve basicamente quatro etapas:
  - Planejamento
  - Projeto de casos de teste
  - Execução do programa
  - Análise de resultados

Planejamento

- São estimados os recursos necessários e definidas as estratégias, métodos e técnicas de teste a serem utilizadas.
- Plano de Teste
  - Apresenta o planejamento para a execução do teste, incluindo:
    - Abrangência
    - Abordagem
    - Recursos e requisitos do ambiente
    - Cronogramas
  - Identifica os itens e as funcionalidades a serem testados.
  - Identifica as tarefas de teste a serem realizadas e os riscos associados com a atividade de teste.

Projeto de Casos de Teste

- São elaborados os casos de teste com os quais o programa deve ser testado.
  - É uma das etapas fundamentais da atividade de teste.
  - Pode ser tão difícil quanto o projeto do próprio produto a ser testado.
  - É um dos melhores mecanismos para prevenção de defeitos.
  - É tão eficaz em identificar erros quanto a execução dos casos de teste projetados.
  - Casos de teste elaborados nessa etapa possuem forte dependência com relação ao critério de teste utilizado.

Execução do Programa

- O programa é executado com os casos de teste elaborados.
  - Fazer registros cronológicos de detalhes relevantes relacionados com a execução dos testes.
  - Documentar qualquer evento que ocorra durante a atividade de teste e que requeira análise posterior.

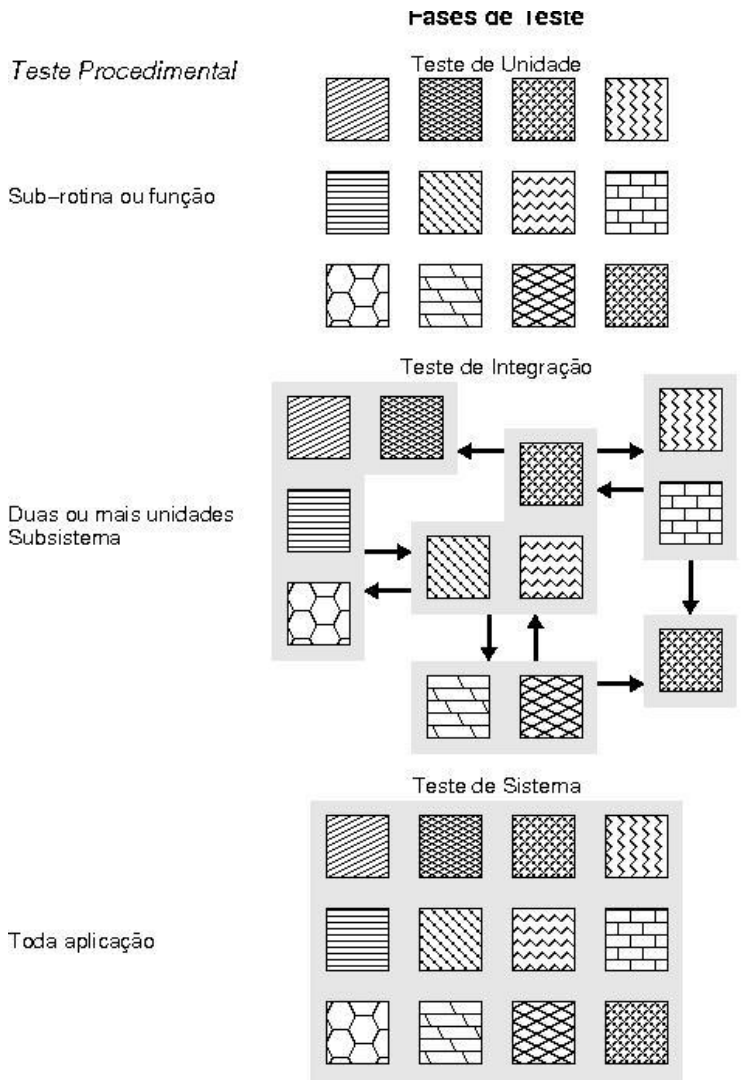
Análise dos Resultados

- O comportamento do programa é avaliado em relação aos casos de teste a fim de determinar se o mesmo está correto ou não.
  - Fornecer avaliações com base nos resultados observados.

Fases de Teste

- Assim como outras atividades de Engenharia de Software, a atividade de teste também é dividida em fases.
  - Cada fase aborda diferentes tipos de erros e aspectos do software.
- Conceito de dividir para conquistar.
- Objetivo: minimizar a complexidade na condução dos testes.
- Idéia: Iniciar os testes a partir da menor unidade executável até atingir o programa como um todo.
  - Teste de Unidade
  - Teste de Integração
  - Teste de Sistema

Fases de Teste



Teste de Unidade

- Concentra esforços nas unidades de projeto do software a fim de verificar se estas funcionam adequadamente.
- Cada unidade do programa é explorada separadamente, procurando-se identificar erros de lógica e de implementação nas mesmas.

Teste de Unidade

- Unidade (Módulo)
  - Componente de software que não pode ser dividido.
  - Menor parte funcional de um programa que pode ser executada.
- Diferentes linguagens possuem unidades diferentes.
  - Pascal e C possuem procedimentos ou funções.
  - Java e C++ possuem métodos (ou classes???).
  - Basic e COBOL pode ser o programa todo.

Teste de Unidade

- Como testar uma unidade que depende de outra para ser executada?

Teste de Unidade

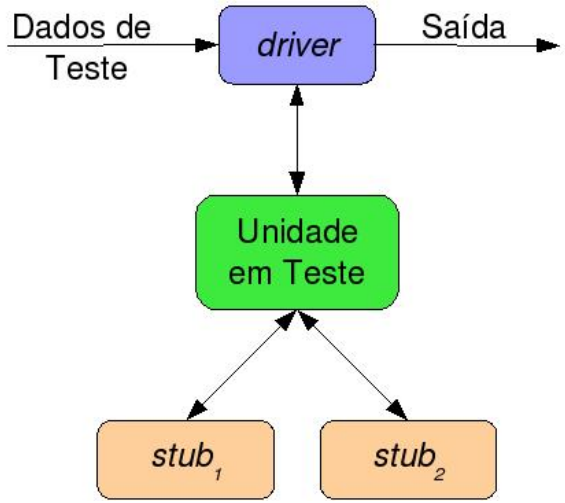
- Como testar uma unidade que depende de outra para ser executada?
- Como testar uma unidade que precisa receber dados de outra unidade para ser executada?

Drivers e Stubs

- Para auxiliar no teste de unidade, em geral, são necessários *drivers* e *stubs*.
- O *driver* é responsável por fornecer para uma dada unidade os dados necessários para que ela possa ser executada e, posteriormente, apresentar os resultados ao testador.
- O *stub* serve para simular o comportamento de uma unidade que ainda não foi desenvolvida, mas da qual a unidade em teste depende.

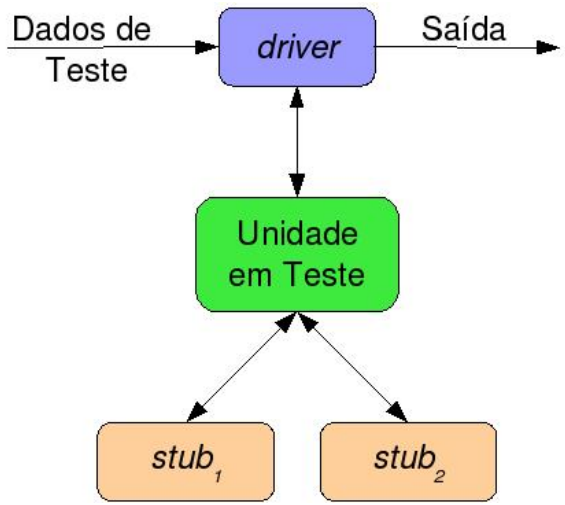
Drivers e Stubs

- Considere F uma dada unidade a ser testada.
- *Driver*
  - Responsável por ativar e coordenar o teste de F, recebendo os dados de teste fornecidos pelo testador, passando tais dados na forma de parâmetros para F, coletando os resultados relevantes produzidos por F, e apresentando-os para o testador.



Drivers e Stubs

- Considere F uma dada unidade a ser testada.
- *Stub*
  - Unidade que substitui, durante o teste, uma unidade utilizada (chamada) por F.
  - Em geral, simula o comportamento da unidade chamada por F com o mínimo de computação ou manipulação de dados.

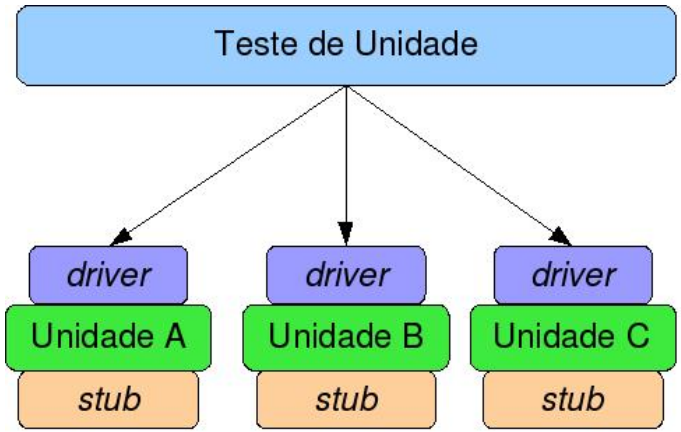


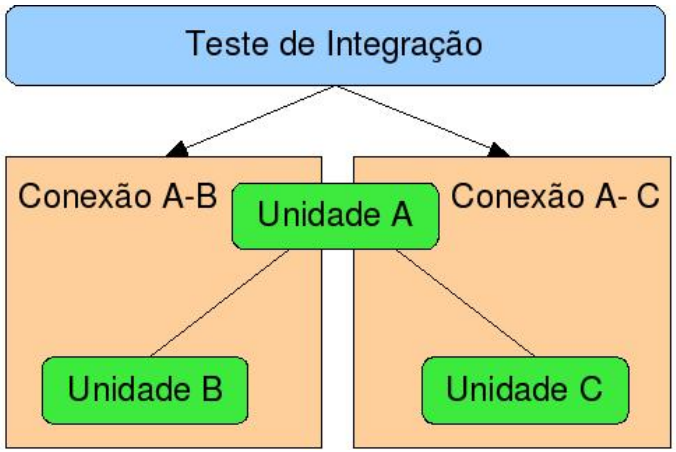
Teste de Integração

- Atividade sistemática aplicada durante a integração da estrutura do programa.
- Visa a identificar erros associados às interfaces entre os módulos do software.
- O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.
  - Verificar se as unidades testadas individualmente comunicam-se conforme desejado.

Teste de Integração

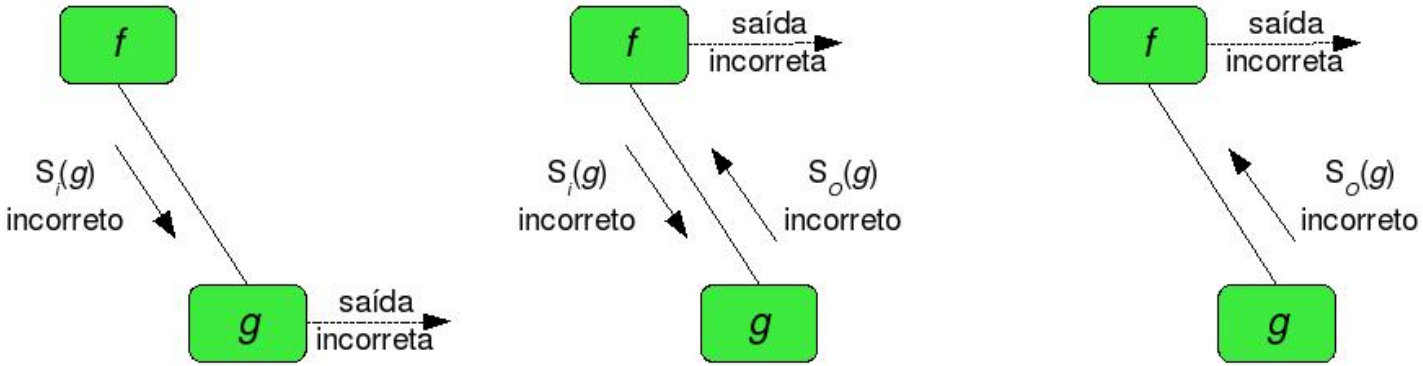
- Por que testar a integração entre unidades se as mesmas, em isolado, funcionam corretamente?
  - Dados podem se perder na interface das unidades.
  - Variáveis globais podem sofrer alterações indesejadas.





Teste de Integração

- Tipos de erros de integração:



Teste de Sistema

- Visa a identificar erros de funções e características de desempenho que não estejam de acordo com a especificação.
- O objetivo é assegurar que o software e os demais elementos que compõem o sistema (por exemplo, hardware, banco de dados, sistema operacional) combinam-se adequadamente e que a função/desempenho global desejada é obtida.

Concluindo...

- Qualidade de software.
- V&V envolvem atividades de análise estática e de análise dinâmica.
- As inspeções são exemplos clássicos de análise estática.
- Os testes são exemplos clássicos de análise dinâmica.
- Inspeção e Teste são técnicas de V&V complementares. Ambas devem ser usadas!!!