XVII Simpósio Brasileiro de Engenharia de Software

Teste de Software: Teoria e Prática

José Carlos Maldonado
Ellen Francine Barbosa

{jcmaldon, francine}@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – São Carlos

Manaus (AM), Outubro de 2003

Teste de Software: Teoria e Prática

Cenário de Aplicação
Transferência Tecnológica
Garantia de Qualidade
Direções para Teste
Conceitos Básicos
Estudos Teóricos e Empíricos
Teste de Software
Ferramentas de Teste
Etapas de Teste
Tipos de Aplicações
Fases de Teste
Técnicas de Teste
Modelos Subjacentes
Exemplo Identifier
Exercício Fatorial
Texto Complementar
Coweb

Cenário de Aplicação

- **Evolução Histórica de Protocolos**
  (Era uma vez....)
  - Século 2 a.C.: Comunicação com tochas!!
  - 1837: Telégrafos (1ª. patente)
    - Uso no controle metroviário
      - Muitos acidentes (homem ↔ equipamentos)
    - 1851: London & Paris (*Stock Exchange*)
    - 1875: 200.000 milhas de linhas de telégrafo
      - Código Morse
  - 1850-1950: Telefone e Rádio
  - 1946: Eniac
  - 1950: Automação da execução de protocolos
  - 1956: Longas Distâncias
  - 1960:
    - Satélites
    - Sistemas de Reserva Aérea (SABRE)

Cenário de Aplicação

- **Protocolos de Comunicação**
  - Conjunto de regras que governam a interação de processos concorrentes em sistemas distribuídos.
  - Aplicações
    - Sistemas Operacionais
    - Redes de Computadores
    - Transmissão de Dados
    - ...
  - Problema no Projeto de Protocolos
    - Estabelecer acordo sobre o uso de recursos compartilhados.
      - Tarefa árdua!!
    - Elementos
      - Serviços
      - Hipóteses sobre o ambiente
      - Vocabulário de Mensagens
      - Codificação
      - Regras de Procedimento

Ocorrências inesperadas de eventos
⇩
Falhas no Protocolo

Túnel de Clayton

- **Composto por 2 trilhos, um em cada direção:**
  - Permitia-se apenas um trem por trilho no interior do túnel.
- **Em cada extremidade do túnel:**
  - Um operador de sinais, 24 horas por dia.
  - Semáforos para garantir que um trem só passaria quando o sinal estivesse verde.
    - O sinal ficaria vermelho após a passagem do trem.
    - O sinal retornaria a verde somente com intervenção humana.
      - O operador podia mudar o sinal para verde novamente, certificando-se de que o trem que entrou por um lado do túnel realmente saiu pelo outro lado.

Túnel de Clayton

- **Outras medidas de segurança:**
  - Telégrafo de agulha
    - Enviava mensagens pré-definidas entre os operadores do túnel:
      - "trem no túnel"
      - "túnel está livre"
      - "o trem saiu do túnel?"
  - Bandeiras de sinalização manual

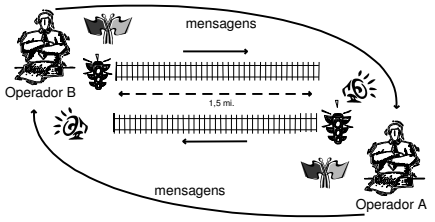
Túnel de Clayton

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem

- **Funcionamento**
  - Quando um trem entrasse no túnel, o operador transmitiria o código "trem no túnel".
    - Se um semáforo falhasse ao mostrar o sinal vermelho quando um trem passasse, uma sirene era tocada. Então o operador podia usar bandeiras verdes e vermelhas para controlar o tráfego.
  - Quando o trem saísse do túnel, o operador do outro lado responderia com o código "túnel está livre".
    - Ao receber essa mensagem, o primeiro operador podia mudar o sinal de entrada para permitir a entrada do próximo trem.
  - Havia um terceiro código para que um operador perguntasse ao outro se "o trem saiu do túnel?".
  - A presença de dois operadores garantia que o túnel fosse usado com segurança, mesmo se o semáforo de um dos lados do túnel não funcionasse direito.

Complementar Exercício

Representação do Túnel de Clayton



Complementar

Túnel de Clayton

- **No século XIX, um grave acidente entre dois trens ocorreu no Túnel de Clayton.**
  - Na colisão morreram 21 pessoas e 176 ficaram feridas.
- **Apesar de todas as medidas de segurança existentes no túnel, ocorreu o acidente…**
- **Qual teria sido a cauda do acidente?**
- **Como o acidente poderia ter sido evitado?**

Coweb

Exercício

Túnel de Clayton: Fatos

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem

- **Um trem passou no semáforo A e o sinal não ficou vermelho, mas a sirene avisou o operador A.**

  Este imediatamente transmitiu a mensagem "trem no túnel" para o operador B e depois foi buscar a bandeira para avisar o próximo trem. Mas …

  Um segundo trem estava muito rápido e já havia passado pelo sinal verde. O maquinista porém, observou a bandeira vermelha quando entrava pelo túnel. Um terceiro trem foi avisado a tempo e não entrou no túnel.

  O operador A voltou para a cabine e enviou outra mensagem "trem no túnel" para indicar que havia dois trens no túnel.

Complementar

Túnel de Clayton: Fatos

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem

O protocolo de comunicação não contava com esse evento, de modo que o significado de duas mensagens "trem no túnel" subseqüentes não estava especificado. Porém, como era improvável que o segundo trem alcançasse o primeiro, nenhum problema real existiria.

O único problema para o operador A era saber do operador B quando os dois trens tivessem saído do túnel, de modo que o terceiro trem pudesse entrar.

Para alertar o operador B do problema, o operador A transmitiu a única mensagem apropriada que possuía: "o trem saiu do túnel?".

O operador B, após ver o primeiro trem sair do túnel, respondeu "túnel está livre".

Complementar

Túnel de Clayton: Fatos

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem

O operador A não sabia se devia esperar por duas mensagens "túnel está livre" ou se a mensagem que ele havia recebido informava que os dois trens tinham saído do túnel. Decidiu que ambos os trens tinham deixado o túnel e permitiu que o terceiro trem entrasse.

O maquinista do segundo trem, entretanto, havia visto a bandeira vermelha e parou o trem completamente no meio do túnel, decidindo sair com o trem de marcha ré, provocando o acidente.

Complementar

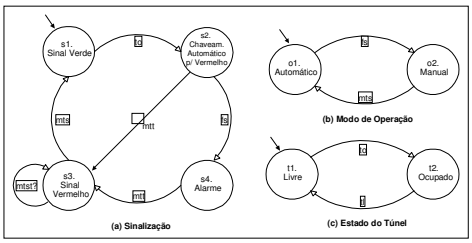
Túnel de Clayton: Modelagem

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem
Modelagem com MEF´s
Modelagem com Statecharts

- **Características**
  - Sinalização
    - Alarme
    - Chaveamento
    - Sinal verde
    - Sinal vermelho
  - Modo de operação
    - Automático
    - Manual
  - Estado do túnel
    - Livre
    - Ocupado

Túnel de Clayton: Modelagem com MEF´s

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem
Modelagem com MEF´s
Modelagem com Statecharts

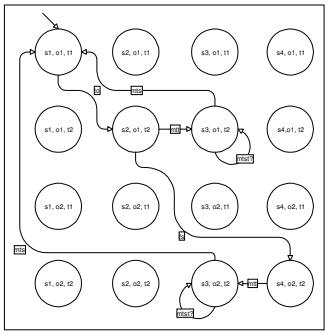


Legenda

fs: falha no sistema
to: trem entra no túnel
tl: trem sai do túnel
mtt: mensagem trem entrou no túnel
mts: mensagem trem saiu do túnel
mtst?: mensagem perguntando se o trem saiu do túnel

Túnel de Clayton: Modelagem com MEF´s

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem
Modelagem com MEF´s
Modelagem com Statecharts

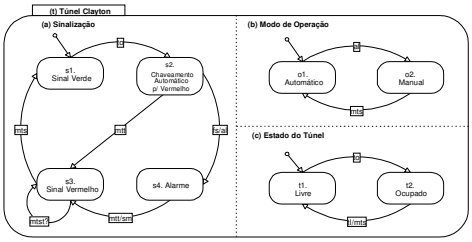


Legenda

fs: falha no sistema
to: trem entra no túnel
mtt: mensagem trem entrou no túnel
mts: mensagem trem saiu do túnel
mtst?: mensagem perguntando se o trem saiu do túnel

Túnel de Clayton: Modelagem com Statecharts

Cenário de Aplicação
Túnel de Clayton
Túnel de Clayton: Fatos
Túnel de Clayton: Modelagem
Modelagem com MEF´s
Modelagem com Statecharts

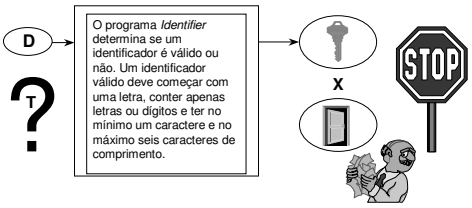


Legenda

al: alarme
fs: falha no sistema
to: trem entra no túnel
tl: trem sai do túnel
sm: sinalização manual
mtt: mensagem trem entrou no túnel
mts: mensagem trem saiu do túnel
mtst?: mensagem perguntando se o trem saiu do túnel

Teste de Software

- **Como testar esse produto?**



Que casos de teste utilizar?

Como garantir que o produto não contém erros? Critérios...

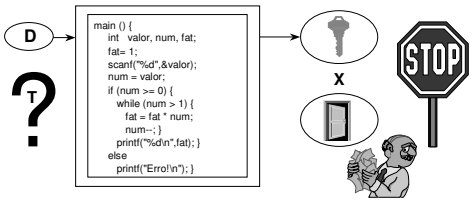
Como decidir se o produto foi suficientemente testado?

Coweb

Exercício

Teste de Software

- **Como testar esse produto?**



Que casos de teste utilizar?

Como garantir que o produto não contém erros? Critérios...

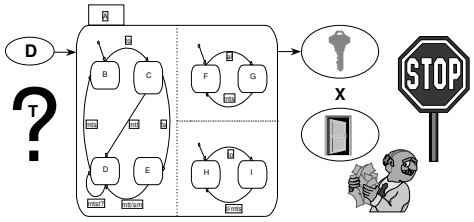
Como decidir se o produto foi suficientemente testado?

Coweb

Exercício

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?

Coweb

Exercício

Teste de Software

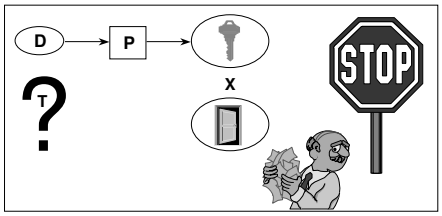
Teste de Software
Objetivos
Limitações

- **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.

Objetivos do Teste

Teste de Software
Objetivos
Limitações

- **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

Teste de Software
Objetivos
Limitações

- **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

Teste de Software
Objetivos
Limitações

- **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.

Limitações do Teste

Teste de Software
Objetivos
Limitações
Corretitude de Programas
Equivalência de Programas
Executabilidade
Correção Coincidente

- **Corretitude**
  - Não existe um algoritmo de teste de propósito geral para provar a corretitude de um programa.
    - Necessidade de um oráculo de teste
- **Equivalência**
  - Dados dois programas, se eles são equivalentes.
  - Dados dois caminhos (seqüências de comandos) de um programa ou de programas diferentes, se eles computam a mesma função.
- **Executabilidade**
  - Dado um caminho (seqüência de comandos), se existe um dado de entrada que leve à sua execução.
- **Correção Coincidente**

Corretitude de Programas

Teste de Software
Objetivos
Limitações
Corretitude de Programas
Oráculo de Teste
Equivalência de Programas
Executabilidade
Correção Coincidente

- **Diz-se que um programa *P* com domínio de entrada *D* é correto com respeito a uma especificação *S* se, para qualquer item de dado $d \in D$, $S(d) = P^*(d)$.**

Ou seja…

Se o comportamento do programa está de acordo com o comportamento esperado para todos os itens de dado pertencentes ao seu domínio de entrada.

Oráculo de Teste

Teste de Software
Objetivos
Limitações
Corretitude de Programas
Oráculo de Teste
Equivalência de Programas
Executabilidade
Correção Coincidente

- **Testador ou algum outro mecanismo que possa determinar, para qualquer item de dado *d* pertencente ao domínio de entrada *D*, se $S(d) = P^*(d)$ dentro de limites de tempo e esforços razoáveis.**

Ou seja...

Um oráculo é responsável por decidir se os valores de saída resultantes da execução do programa são corretos.

Equivalência de Programas

Teste de Software
Objetivos
Limitações
Corretitude de Programas
Equivalência de Programas
Executabilidade
Correção Coincidente

- **Dados dois programas P_1 e P_2 com domínio de entrada *D*, diz-se que P_1 e P_2 são equivalentes se, para qualquer item de dado $d \in D$, $P_1^*(d) = P_2^*(d)$.**

Ou seja…

Se o comportamento de ambos os programas é idêntico para todos os itens de dado pertencentes ao domínio de entrada.

Mutante Equivalente

Executabilidade

Teste de Software
Objetivos
Limitações
Corretitude de Programas
Equivalência de Programas
Executabilidade
Correção Coincidente

- **Dado um programa *P* com domínio de entrada *D*, diz-se que um caminho (seqüência de comandos) é executável se existe ao menos um item de dado $d \in D$ que leve à execução desse caminho.**
- **Do contrário, diz-se que o caminho é não-executável.**

Grafo de Programa
Exemplo Exercício

Executabilidade

- **Caminho Não-Executável (seqüência de comandos em destaque)**

```
{
    char  achar;
    int length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
       length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
       if (!(valid_f(achar)))
          valid_id = 0;
       length++;
       achar = fgetc (stdin);
    }
    if (valid_id && (length >= 1) && (length < 6))
       printf ("Valido\n");
    else
       printf ("Invalido\n");
}
```

Programa Identifier (função main)

Exemplo Identifier
Exemplo

Executabilidade

- **Determine uma seqüência de comandos que represente um caminho não-executável.**

```
{
    char  achar;
    int length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
       length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
       if (!(valid_f(achar)))
          valid_id = 0;
       length++;
       achar = fgetc (stdin);
    }
    if (valid_id && (length >= 1) && (length < 6))
       printf ("Valido\n");
    else
       printf ("Invalido\n");
}
```

Programa Identifier (função main)

Coweb
Exercício

Correção Coincidente

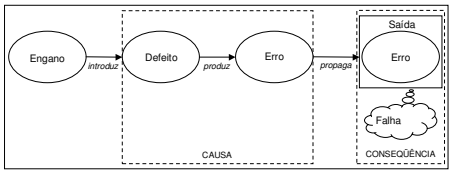
Teste de Software
Objetivos
Limitações
Corretitude de Programas
Equivalência de Programas
Executabilidade
Correção Coincidente

- **O programa pode apresentar, coincidentemente, um resultado correto para um particular item de dado $d \in D$, satisfazendo a um requisito de teste e não revelando a presença do erro.**
  - Entretanto, se escolhido um outro dado de entrada, o resultado obtido seria incorreto e a presença do erro seria identificada.

Terminologia

Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Falha
Caso de Teste
Critério de Teste

- **Padronização da terminologia de ES:**
  - 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.

Exemplo Exercício

Engano (*Mistake*)

Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Falha
Caso de Teste
Critério de Teste

- **Ação humana que introduz um defeito no software.**

Exemplo

Engano

Ação incorreta tomada pelo programador.

Exemplo

Defeito (*Fault*)

Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Falha
Caso de Teste
Critério de Teste

- **Passo, processo ou definição de dados incorreta, incompleta, ausente ou extra que, ao ser executada, pode produzir um erro no programa.**

Exemplo

Defeito

Instrução ou comando incorreto.

Exemplo

Erro (*Error*)

Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Erro Computacional
Erro de Domínio
Falha
Caso de Teste
Critério de Teste

- **Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado na execução do programa, intermediário ou final, que seja inconsistente.**

Erro Computacional

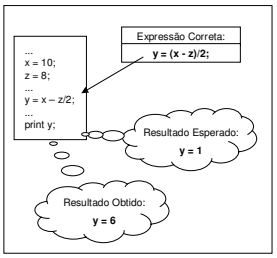
Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Erro Computacional
Erro de Domínio
Falha
Caso de Teste
Critério de Teste

- **Um erro computacional provoca uma computação incorreta mas a seqüência de comandos executada é igual à seqüência esperada.**

Exemplo

Erro Computacional

A atribuição de um valor incorreto a uma variável do programa por uma expressão aritmética correta corresponde a um erro computacional.



Exemplo

Erro de Domínio

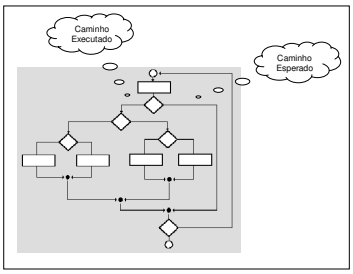
Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Erro Computacional
Erro de Domínio
Falha
Caso de Teste
Critério de Teste

- **Um erro de domínio faz com que a seqüência de comandos executada seja diferente da seqüência esperada, ou seja, uma seqüência errada é selecionada.**
- **Causas**
  - Erros de domínio podem ser gerados por um erro computacional ou uma condição incorreta de um comando de decisão.

Exemplo

Erro de Domínio

A seleção de saídas incorretas do comando de decisão devido a operadores relacionais incorretos corresponde a um erro de domínio.



Exemplo

Falha (*Failure*)

Conceitos Básicos
Terminologia
Engano
Defeito
Erro
Falha
Caso de Teste
Critério de Teste

- **Evento notável em que o sistema viola suas especificações, ou seja, é a produção de uma saída incorreta com relação à especificação.**

Engano × Defeito × Erro × 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.

Exemplo

Engano × Defeito × Erro × Falha

- **Contraste e relacione os termos Engano, Defeito, Erro e Falha.**
  - Forneça exemplos que fundamentem suas idéias.

Coweb

Exercício

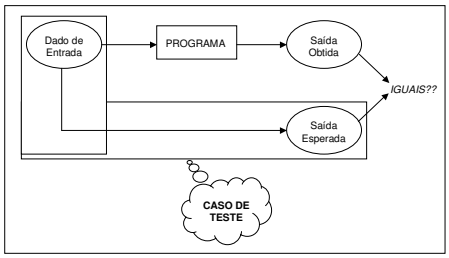
Caso de Teste

Conceitos Básicos
Terminologia
Caso de Teste
Critério 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
    - Dado necessário para uma execução do programa.
  - Saída esperada
    - Resultado de uma execução do programa.
    - Pressupõe-se a existência de um oráculo de teste.

Caso de Teste

Conceitos Básicos
Terminologia
Caso de Teste
Critério 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.

Critério de Teste

Conceitos Básicos
Terminologia
Caso de Teste
Critério de Teste
Critério de Teste: Propriedades Mínimas
Requisito de Teste
Critério de Adequação
Critério de Geração

- **Maneira sistemática e planejada para conduzir os testes.**
- **Fornecer 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.

Critério de Teste: Propriedades Mínimas

Conceitos Básicos
Terminologia
Caso de Teste
Critério de Teste
Critério de Teste: Propriedades Mínimas
Requisito de Teste
Critério de Adequação
Critério de Geração

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.**

Requisito de Teste

Conceitos Básicos
Terminologia
Caso de Teste
Critério de Teste
Critério de Teste: Propriedades Mínimas
Requisito de Teste
Critério de Adequação
Critério de Geração
Técnicas 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*.

Critério de Adequação

Conceitos Básicos
Terminologia
Caso de Teste
Critério de Teste
Critério de Teste: Propriedades Mínimas
Requisito de Teste
Critério de Adequação
Análise de Cobertura
Critério de Geração
Técnicas de Teste

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.**

Análise de Cobertura

Conceitos Básicos
Terminologia
Caso de Teste
Critério de Teste
Critério de Teste: Propriedades Mínimas
Requisito de Teste
Critério de Adequação
Análise de Cobertura
Critério de Geração
Técnicas de Teste

- **Consiste em determinar o percentual de elementos estabelecidos por um critério de teste que foram executados pelo conjunto de casos de teste.**
- **Com essas informações...**
  - O conjunto de casos de teste pode ser melhorado para que os elementos ainda não abordados sejam testados.
    - Adição de novos casos de teste ao conjunto.

Critério de Geração

Conceitos Básicos
Terminologia
Caso de Teste
Critério de Teste
Critério de Teste: Propriedades Mínimas
Requisito de Teste
Critério de Adequação
Critério de Geração
Técnicas de Teste

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

- ***T* é *C*-adequado "por construção".**

Etapas de Teste

Etapas de Teste
Planejamento
Projeto
Execução
Análise
Norma IEEE 829

- **A atividade de teste envolve basicamente quatro etapas:**
  - Planejamento
  - Projeto de casos de teste
  - Execução do programa
  - Análise de resultados
- **Documentação de Teste**
  - Uma série de documentos pode ser produzida a cada etapa de teste, servindo como apoio às etapas subseqüentes.
    - Norma IEEE Std 829-1998

Planejamento

Etapas de Teste
Planejamento
Plano de Teste
Projeto
Execução
Análise
Norma IEEE 829

- **Nessa etapa são estimados os recursos necessários e definidas as estratégias, métodos e técnicas de teste a serem utilizadas.**
- **De acordo com a Norma IEEE 829, devem ser produzidos os seguintes documentos:**
  - Plano de Teste

Teste de Software: Teoria e Prática

Plano de Teste

Etapas de Teste
- Planejamento
  - Plano de Teste
- Projeto
- Execução
- Análise
- Norma IEEE 829

- **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.**

Complementar

Teste de Software: Teoria e Prática

Projeto de Casos de Teste

Etapas de Teste
- Planejamento
- Projeto
  - Especificação de Projeto de Teste
  - Especificação de Caso de Teste
  - Especificação de Procedimento de Teste
- Execução
- Análise
- Norma IEEE 829

- **Nessa etapa são elaborados os casos de teste com os quais o programa deve ser testado.**
- **De acordo com a Norma IEEE 829, devem ser produzidos os seguintes documentos:**
  - Especificação de Projeto de Teste
  - Especificação de Caso de Teste
  - Especificação de Procedimento de Teste

Complementar

Aspectos do Projeto de Casos de Teste

- **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 a 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.**

Complementar

Teste de Software: Teoria e Prática

Especificação de Projeto de Teste

Etapas de Teste
- Planejamento
- Projeto
  - Especificação de Projeto de Teste
  - Especificação de Caso de Teste
  - Especificação de Procedimento de Teste
- Execução
- Análise
- Norma IEEE 829

- **Refina a abordagem apresentada no Plano de Teste.**
- **Identifica as funcionalidades e características a serem testadas pelo projeto e por seus testes associados.**
- **Identifica os casos e procedimentos de teste.**
- **Determina os critérios de aprovação e de reprovação.**

Complementar 1 Complementar 2

Teste de Software: Teoria e Prática

Especificação de Caso de Teste

Etapas de Teste
- Planejamento
- Projeto
  - Especificação de Projeto de Teste
  - Especificação de Caso de Teste
  - Especificação de Procedimento de Teste
- Execução
- Análise
- Norma IEEE 829

- **Define um caso de teste, incluindo:**
  - Dados de entrada
  - Resultados esperados
  - Ações e condições gerais para a execução do teste

Complementar 1 Complementar 2

Teste de Software: Teoria e Prática

Especificação de Procedimento de Teste

Etapas de Teste
- Planejamento
- Projeto
  - Especificação de Projeto de Teste
  - Especificação de Caso de Teste
  - Especificação de Procedimento de Teste
- Execução
- Análise
- Norma IEEE 829

- **Especifica os passos para executar um conjunto de casos de teste.**

Complementar 1 Complementar 2

Estruturação da Norma IEEE 829

- **Separação da Especificação de Caso de Teste em relação à Especificação de Projeto de Teste:**
  - Permitir seu uso em mais de um projeto de teste.
  - Permitir sua reutilização no teste de outros produtos.
- **Separação da Especificação de Procedimento de Teste em relação à Especificação de Caso de Teste:**
  - Um procedimento de teste deve possuir somente passos simples, não devendo conter outros detalhes.

Complementar

Teste de Software: Teoria e Prática

Etapas de Teste
- Planejamento
- Projeto
- Execução
  - Diário de Teste
  - Relatório de Incidente de Teste
  - Relatório de Encaminhamento de Item de Teste
- Análise
- Norma IEEE 829

Execução do Programa

- **Nessa etapa o programa é executado com os casos de teste elaborados.**
- **De acordo com a Norma IEEE 829, devem ser produzidos os seguintes documentos:**
  - Diário de Teste
  - Relatório de Incidente de Teste
  - Relatório de Encaminhamento de Item de Teste

Teste de Software: Teoria e Prática

Etapas de Teste
- Planejamento
- Projeto
- Execução
  - Diário de Teste
  - Relatório de Incidente de Teste
  - Relatório de Encaminhamento de Item de Teste
- Análise
- Norma IEEE 829

Diário de Teste

- **Apresenta registros cronológicos de detalhes relevantes relacionados com a execução dos testes.**

Complementar

Teste de Software: Teoria e Prática

Etapas de Teste
- Planejamento
- Projeto
- Execução
  - Diário de Teste
  - Relatório de Incidente de Teste
  - Relatório de Encaminhamento de Item de Teste
- Análise
- Norma IEEE 829

Relatório de Incidente de Teste

- **Documenta qualquer evento que ocorra durante a atividade de teste e que requeira análise posterior.**

Complementar

Teste de Software: Teoria e Prática

Etapas de Teste
- Planejamento
- Projeto
- Execução
  - Diário de Teste
  - Relatório de Incidente de Teste
  - Relatório de Encaminhamento de Item de Teste
- Análise
- Norma IEEE 829

Relatório de Encaminhamento de Item de Teste

- **Identifica os item encaminhados para teste no caso de equipes distintas serem responsáveis pelas tarefas de desenvolvimento e de teste.**

Complementar

Teste de Software: Teoria e Prática

Etapas de Teste
- Planejamento
- Projeto
- Execução
- Análise
  - Relatório-Resumo de Teste
- Norma IEEE 829

Análise dos Resultados

- **Nessa etapa avalia-se o comportamento do programa em relação aos casos de teste a fim de determinar se o mesmo está correto ou não.**
- **De acordo com a Norma IEEE 829, devem ser produzidos os seguintes documentos:**
  - Relatório-Resumo de Teste

Teste de Software: Teoria e Prática

Relatório-Resumo de Teste

Etapas de Teste
- Planejamento
- Projeto
- Execução
- Análise
  - Relatório-Resumo de Teste
- Norma IEEE 829

- **Apresenta de forma resumida os resultados das atividades de teste associadas com uma ou mais especificações de projeto de teste.**
- **Fornece avaliações com base nos resultados observados.**

Complementar

Teste de Software: Teoria e Prática

Norma IEEE Std 829-1998

Etapas de Teste
- Planejamento
- Projeto
- Execução
- Análise
- Norma IEEE 829

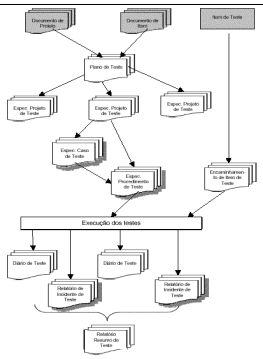
- **Padrão para a documentação de teste de software.**
- **Independente de técnicas, métodos, estratégias, recursos e ferramentas de teste.**
- **Independente do tipo de produto testado.**
- **Descreve um conjunto de 8 documentos básicos para o teste de um produto, cobrindo as tarefas de:**
  - Planejamento dos Testes
  - Especificação dos Testes
  - Relato dos Testes

Complementar

Norma IEEE Std 829-1998

- **Relacionamento entre os documentos de teste estabelecidos pela Norma IEEE 829.**

(Itens com fundo cinza não foram definidos pela norma.)



Fonte: CenPRA, 2003

Complementar

Teste de Software: Teoria e Prática

Fases de Teste

Fases de Teste
- Teste de Unidade
- Teste de Integração
- Teste de Sistema

- **Maneira prática de minimizar a complexidade na condução dos testes.**
  - Teste de Unidade
  - Teste de Integração
  - Teste de Sistema
- **Cada fase aborda diferentes tipos de erros e aspectos do software.**
  - Aplicação de técnicas e critérios de teste distintos no estabelecimento de estratégias adequadas de seleção de casos de teste e medidas de cobertura.

Teste de Software: Teoria e Prática

Teste de Unidade

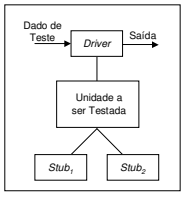
Fases de Teste
- Teste de Unidade
- Teste de Integração
- Teste de Sistema

- **Concentra esforços nas unidades de projeto do software a fim de verificar se estas funcionam adequadamente.**
  - Unidade (Módulo)
    - Componente de software que não pode ser dividido.
      - Menor parte funcional de um programa que pode ser executada.
- **Cada módulo do programa é explorado separadamente, procurando-se identificar erros de lógica e de implementação nos mesmos.**

Complementar

Drivers e *Stubs*

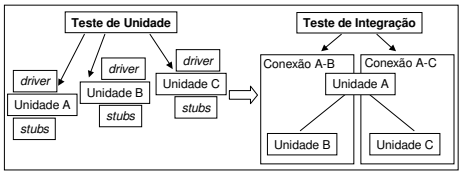
- **Considere *F* uma dada unidade a ser testada.**
- ***Driver***
  - Responsável por ativar e coordenar o teste de *F*, recebendo os dados de teste fornecidos pelo testador, passando tais dados na forma de parâmetros para *F*, coletando os resultados relevantes produzidos por *F*, e apresentando-os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



Complementar

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.



Complementar Exercício

Teste de Integração

- **Por que um programa construído a partir de unidades que individualmente trabalham corretamente, supondo que todas foram submetidas ao teste de unidade, não funcionaria adequadamente?**

Coweb

Exercício

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 e banco de dados) combinam-se adequadamente e que a função/desempenho global desejada é obtida.**

Técnicas de Teste

- **Definem quais tipos de informação serão utilizados para estabelecer os requisitos de teste.**
  - Contemplam diferentes perspectivas do software.
    - Aspecto complementar
- **Técnica Funcional (Caixa-Preta)**
- **Técnica Estrutural (Caixa-Branca)**
- **Técnica Baseada em Erros**
- **...**

Critério de Teste

Técnica Funcional

- **Baseia-se na especificação do software para derivar os requisitos de teste.**
- **Aborda o software de um ponto de vista macroscópico.**
  - Sem se preocupar com detalhes de implementação.
- **Passos Principais**
  - Identificar as funções que o software deve realizar.
    - Especificação
  - Criar casos de teste capazes de checar se essas funções estão sendo executadas corretamente.

Técnica Funcional

- **Problemas**
  - Dificuldade em quantificar a atividade de teste
    - Não se pode garantir que partes essenciais ou críticas do software foram executadas
  - Especificações descritivas e não formais
    - Requisitos imprecisos e informais
  - Dificuldade de automatização
    - Aplicação manual

Critérios Funcionais

- **Particionamento de Equivalência**
  - Divide o domínio de entrada (e de saída) de um programa em classes de equivalência, a partir das quais derivam-se os casos de teste.
- **Análise do Valor Limite**
  - Complementa o critério Particionamento de Equivalência, exigindo casos de teste nos limites (fronteiras) de cada classe de equivalência.
- **Grafo de Causa-Efeito**
  - Verifica o efeito combinado de dados de entrada.
    - Causas (condições de entrada) e efeitos (ações) são identificados e combinados em um grafo.
      - Tabela de decisão → Casos de Teste

Particionamento de Equivalência

- **Divide o domínio de entrada (e de saída) do programa em classes de dados.**
  - Classes de equivalência
    - Em princípio, todos os elementos de uma classe devem se comportar de maneira equivalente.
- **Os casos de teste são derivados a partir das classes de equivalência.**
- **Passos**
  - Identificar classes de equivalência
    - Condições de entrada
    - Classes válidas e inválidas
  - Definir os casos de teste
    - Enumeram-se as classes de equivalência.
    - Casos de teste para as classes válidas.
    - Casos de teste para as classes inválidas.

Exemplo Exercício

Particionamento de Equivalência

- **Programa Identifier: Classes Válidas e Inválidas**

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho *t* do identificador	$1 \le t \le 6$ (1)	$t > 6$ (2) $t < 1$ (3)
Primeiro caractere *c* é uma letra	Sim (4)	Não (5)
Só contém caracteres válidos	Sim (6)	Não (7)

- **Conjunto de Casos de Teste**

T_0 = {(a1,Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)}
(1, 4, 6) (5) (7) (2)

Exemplo Identifier
Complementar
Exemplo

Técnica Estrutural

- **Baseia-se no conhecimento da estrutura interna (implementação) do programa.**
  - Teste dos detalhes procedimentais.
- **A maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de programa.**
- **Problemas**
  - Impossibilidade, em geral, de determinar se um caminho é executável ou não.
    - Intervenção do testador.

Grafo de Programa

Critérios Baseados em Complexidade

- **Utilizam informações sobre a complexidade do programa para derivar os requisitos de teste.**
- **Critério de McCabe**
  - Utiliza a complexidade ciclomática do grafo de programa para derivar os requisitos de teste.
  - Requer que um conjunto de caminhos linearmente independentes do grafo de programa seja executado.

Critérios Baseados em Fluxo de Controle

- **Utilizam características de controle da execução do programa para determinar os requisitos de teste.**
  - Comandos
  - Desvios de Execução
- **Critérios Todos-Nós**
- **Critério Todos-Arcos**
- **Critério Todos-Caminhos**

Critério Todos-Nós

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Todos-Nós
Todos-Arcos
Critérios Baseados em Fluxo de Dados
Técnica Baseada em Erros

- **Exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de programa.**

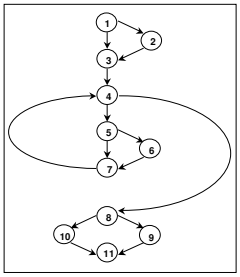
Ou seja…

Requer que cada comando do programa seja executado pelo menos uma vez.

Exemplo

Critério Todos-Nós

- **Elementos Requeridos:**
  - Nós
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11



Exemplo Identifier
Complementar
Exemplo

Critério Todos-Arcos

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Todos-Nós
Todos-Arcos
Critérios Baseados em Fluxo de Dados
Técnica Baseada em Erros

- **Exige que a execução do programa passe, ao menos uma vez, em cada arco do grafo de programa.**

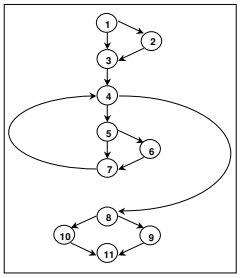
Ou seja…

Requer que cada desvio de fluxo de controle do programa seja exercitado pelo menos uma vez.

Exemplo

Critério Todos-Arcos

- **Elementos Requeridos:**
  - Arcos
    - Arcos primitivos
      <1,2>, <1,3>, <5,6>, <5,7>, <8,9>, <8,10>



Exemplo Identifier
Complementar
Exemplo

Critérios Baseados em Fluxo de Dados

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Critérios Baseados em Fluxo de Dados
Rapps & Weyuker
Potenciais-Usos
Técnica Baseada em Erros

- **Utilizam informações do fluxo de dados do programa para determinar os requisitos de teste.**
  - Exploram as interações que envolvem definições de variáveis e referências a tais definições.
- **Critérios de Rapps & Weyuker**
- **Critérios Potenciais-Usos**
- **...**

Critérios de Rapps & Weyuker

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Critérios Baseados em Fluxo de Dados
Rapps & Weyuker
Todas-Defs
Todos-Usos
Potenciais-Usos
Técnica Baseada em Erros

- **Utilizam o Grafo Def-Uso (*Def-Use Graph*) para derivar os requisitos de teste.**

Grafo Def-Uso:
Grafo de Programa + Definição e Uso de Variáveis

- **Critério Todas-Definições**
- **Critério Todos-Usos**
- **...**

Grafo Def-Uso

Critério Todas-Definições

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Critérios Baseados em Fluxo de Dados
Rapps & Weyuker
Todas-Defs
Todos-Usos
Potenciais-Usos
Técnica Baseada em Erros

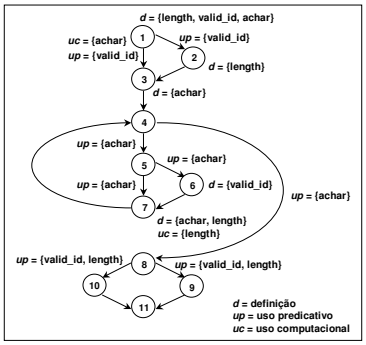
Requer que cada definição de variável seja exercitada pelo menos uma vez, não importa se por um c-uso ou por um p-uso.

Exemplo

Critério Todas-Definições

- **Definição de *length* no nó 1**

(1,3,4,5,7)
(1,3,4,8,9)
(1,3,4,8,10)
(1,3,4,5,6,7)
...



Exemplo Identifier
Complementar
Exemplo

Critério Todos-Usos

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Critérios Baseados em Fluxo de Dados
Rapps & Weyuker
Todas-Defs
Todos-Usos
Potenciais-Usos
Técnica Baseada em Erros

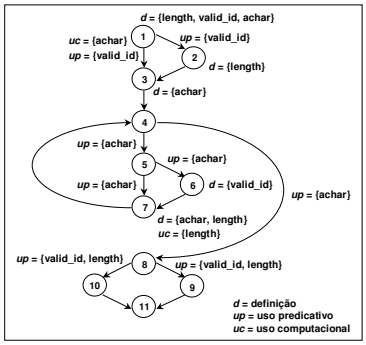
Requer que todas as associações entre uma definição de variável e seus subseqüentes usos sejam exercitadas pelos casos de teste, através de pelo menos um caminho livre de definição.

Exemplo

Critério Todos-Usos

- **Definição de length no nó 1**

(1,7, length)
(1, (8,9), length)
(1, (8,10), length)
...



Exemplo Identifier
Complementar
Exemplo

Critérios Potenciais-Usos

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Critérios Baseados em Fluxo de Dados
Rapps & Weyuker
Potenciais-Usos
Todos-Potenciais-Usos
Técnica Baseada em Erros

- **Utilizam o Grafo Def para derivar os requisitos de teste.**

Grafo Def-Uso:
Grafo de Programa + Definição de Variáveis

- **Os elementos requeridos são caracterizados independentemente da ocorrência explícita de uma referência (um uso) a uma determinada definição.**
  - Potencial-Associação (Potencial-Uso)
- **Critério Todos-Potenciais-Usos**
- **Critério Todos-Potenciais-Usos/DU**
- **...**

Grafo Def

Critério Todos-Potenciais-Usos

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Critérios Baseados em Complexidade
Critérios Baseados em Fluxo de Controle
Critérios Baseados em Fluxo de Dados
Rapps & Weyuker
Potenciais-Usos
Todos-Potenciais-Usos
Técnica Baseada em Erros

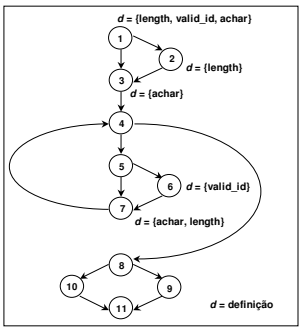
Requer, para todo nó *i* e para toda variável *x*, para a qual existe uma definição em *i*, que pelo menos um caminho livre de definição com relação à variável *x* do nó *i* para todo nó e para todo arco possível de ser alcançado a partir de *i* por um caminho livre de definição com relação a x seja exercitado.

Exemplo Exercício

Critério Todos-Potenciais-Usos

- **Elementos Requeridos:**
  - Associações

<1,2,{length}>
<1,(1,3), {valid_id}>
<2,(8,9), {length}>
<3,(8,10), {achar}>
...



Critério Todos-Potenciais-Usos

Associações Requeridas	T_0	T_1	T_2	Associações Requeridas	T_0	T_1	T_2
1) <1,(6,7),{ *length* }>		✓		17) <2,(6,7),{ *length* }>	✓		
2) <1,(1,3),{ *achar, length, valid_id* }>	✓			18) <2,(5,6),{ *length* }>	✓		
3) <1,(8,10),{ *length, valid_id* }>		✓		19) <3,(8,10),{ *achar* }>		✓	
4) <1,(8,10),{ *valid_id* }>	✓			20) <3,(8,9),{ *achar* }>		✓	
5) <1,(8,9),{ *length, valid_id* }>	*	*	*	21) <3,(5,7),{ *achar* }>	✓		
6) <1,(8,9),{ *valid_id* }>	✓			22) <3,(6,7),{ *achar* }>	✓		
7) <1,(7,4),{ *valid_id* }>	✓			23) <3,(5,6),{ *achar* }>	✓		
8) <1,(5,7),{ *length, valid_id* }>	✓			24) <6,(8,10),{ *valid_id* }>	✓		
9) <1,(5,7),{ *valid_id* }>	✓			25) <6,(8,9),{ *valid_id* }>	*	*	*
10) <1,(5,6),{ *length, valid_id* }>		✓		26) <6,(5,7),{ *valid_id* }>	✓		
11) <1,(5,6),{ *valid_id* }>	✓			27) <6,(5,6),{ *valid_id* }>			✓
12) <1,(2,3),{ *achar, valid_id* }>	✓			28) <7,(8,10),{ *achar, length* }>	✓		
13) <1,(1,2),{ *achar, length, valid_id* }>	✓			29) <7,(8,9),{ *achar, length* }>	✓		
14) <2,(8,10),{ *length* }>	*	*	*	30) <7,(5,7),{ *achar, length* }>	✓		
15) <2,(8,9),{ *length* }>		✓		31) <7,(6,7),{ *achar, length* }>			✓
16) <2,(5,7),{ *length* }>	✓			32) <7,(5,6),{ *achar, length* }>			✓

- **Conjunto de Casos de Teste**

T_0 = { (a1, Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)}
T_1 = T_0 ∪ {(1#, Inválido), (%, Inválido), (c, Válido)}
T_2 = T_1 ∪ {(#-%, Inválido)}

Técnica Baseada em Erros

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes

- **Os requisitos de teste são derivados a partir dos erros mais freqüentes cometidos durante o processo de desenvolvimento do software**
- **Critério Semeadura de Erros**
- **Teste de Mutação**
  - Análise de Mutantes (unidade)
  - Mutação de Interface (integração)

Critério Análise de Mutantes

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas

- **Consiste na introdução de pequenos desvios sintáticos no programa em teste, os quais são responsáveis por modelar erros freqüentes de desenvolvimento.**
  - Encoraja o testador a construir casos de testes capazes de demonstrar que tais transformações resultam em programas semanticamente incorretos.
    - Casos de teste que evidenciem as diferenças de comportamento entre o programa original (em teste) e os programas modificados.

Critério Análise de Mutantes

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas

- **Hipótese do Programador Competente**

Programadores experientes escrevem programas corretos ou muito próximos do correto.

- **Efeito de Acoplamento**

Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros complexos.

Operador de Mutação

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas

- **Entende-se por operador de mutação as regras que definem as alterações a serem aplicadas ao programa *P*, dando origem a programas similares.**
- **Seleção dos operadores de mutação:**
  - Abrangente
    - Capaz de modelar a maior parte dos erros
  - Pequena cardinalidade
    - Problemas de custo
      - Quanto maior o número de operadores utilizados, maior o número de mutantes gerados.

Operador de Mutação

Mutação de Comandos	
SSDL	Retira um comando de cada vez do programa
SWDD	Troca o comando *while* por *do-while*
SMTC	Interrompe a execução do laço após duas execuções

Mutação de Operadores	
ORRN	Troca operador relacional por operador relacional
OLBN	Troca operador lógico por operador *bitwise*
OASN	Troca operador aritmético por operador de deslocamento

Mutação de Constantes	
Ccsr	Troca referências escalares por constantes
Cccr	Troca constante por constante

Mutação de Variáveis	
VTWD	Troca referência escalar pelo sucessor e predecessor
VDTR	Requer valor negativo, positivo e zero para cada referência escalar

Exemplo

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Mutante Morto
Mutante Equivalente
Mutante Error-Revealing
Escore de Mutação
Aplicação
Abordagens Alternativas

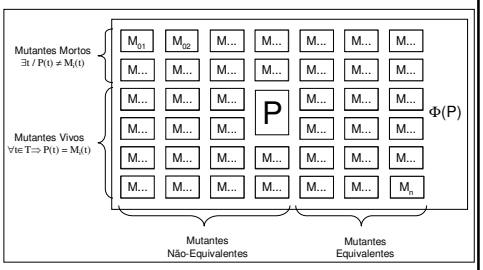
Mutante

- **Programa modificado, resultante da aplicação dos operadores de mutação sobre o programa original.**
- **Assumindo a validade do Efeito de Acoplamento, apenas uma mutação de cada vez é aplicada ao programa em teste, ou seja, cada mutante contém apenas uma transformação sintática.**
  - Observa-se, entretanto, que *k* transformações sintáticas podem ser introduzidas no programa.
    - *k*-mutante

Exemplo

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Mutante Morto
Mutante Equivalente
Mutante Error-Revealing
Escore de Mutação
Aplicação
Abordagens Alternativas

Mutante



Exemplo

Mutante

- **Troca do operador relacional " < " pelo operador relacional " <= "**

```
{
    char  achar;
    int length, valid_id;
    length = 0;
    printf ("Identificador: ");
    achar = fgetc (stdin);
    valid_id = valid_s(achar);
    if (valid_id)
       length = 1;
    achar = fgetc (stdin);
    while (achar != '\n')
    {
       if (!(valid_f(achar)))
          valid_id = 0;
       length++;
       achar = fgetc (stdin);
    }
    if (valid_id * (length >= 1) && (length <= 6))
       printf ("Valido\n");
    else
       printf ("Invalido\n");
}
```

Programa Identifier (função main)

Exemplo Identifier
Complementar
Exemplo

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Mutante Morto
Mutante Equivalente
Mutante Error-Revealing
Escore de Mutação
Aplicação
Abordagens Alternativas

Mutante Morto

- **Para algum caso de teste o resultado do mutante e o do programa original diferem entre si.**

Exemplo 1 Exemplo 2

Mutante Morto

- **Considere o programa Fatorial:**

```
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"); }
```

Programa Fatorial

```
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"); }
```

Mutante Morto

- Suponha que a variável *num* assuma o valor 3. Executando o programa original com esse valor, o resultado obtido é 6.
- Por outro lado, executando o programa mutante, obtém-se uma mensagem de erro. Nesse caso, diz-se que o mutante foi morto pelo caso de teste (3,6).

Exemplo

Mutante Morto

- **Mutante morto com o caso de teste (*[, Inválido*)**

```
                 :
                 :
int valid_s(char ch)
{
   if(((ch >= 'A') &&
       (ch <= 'z')) ||
      ((ch >= 'a') &&
      ((ch <= 'z')))
   {
      return (1);
   }
   else
   {
      return (0);
   }
}
                 :
                 :
```

Programa Identifier (função valid_s)

Exemplo Identifier

Complementar

Exemplo

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Mutante Morto
Mutante Equivalente
Mutante Error-Revealing
Escore de Mutação
Aplicação
Abordagens Alternativas

Mutante Equivalente

- **O mutante e o programa original apresentam sempre o mesmo resultado, para qualquer caso de teste pertencente ao domínio de entrada.**

Equivalência de Programas

Exemplo 1 Exemplo 2

Mutante Equivalente

- **Considere o programa Fatorial:**

```
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"); }
```

Programa Fatorial

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

Programa Equivalente

- A troca do comando if (num >= 0) pelo comando if (valor >= 0), não altera os resultados produzidos pelo programa, que continua comportando-se conforme o esperado.

Exemplo

Mutante Equivalente

- **Troca do operador lógico " && " pelo operador aritmético " * "**

```
{
   char  achar;
   int length, valid_id;
   length = 0;
   printf ("Identificador: ");
   achar = fgetc (stdin);
   valid_id = valid_s(achar);
   if (valid_id)
      length = 1;
   achar = fgetc (stdin);
   while (achar != '\n')
   {
      if (!(valid_f(achar)))
         valid_id = 0;
      length++;
      achar = fgetc (stdin);
   }
   if (valid_id * (length >= 1) && (length < 6))
      printf ("Valido\n");
   else
      printf ("Invalido\n");
}
```

Programa Identifier (função main)

Exemplo Identifier

Complementar

Exemplo

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Mutante Morto
Mutante Equivalente
Mutante Error-Revealing
Escore de Mutação
Aplicação
Abordagens Alternativas

Mutante Error-Revealing

- **Um mutante é dito ser *error-revealing* se para qualquer caso de teste *t* tal que $P^*(t) \neq M^*(t)$ pudermos concluir que $P^*(t)$ não está de acordo com o resultado esperado, ou seja, revela a presença de um erro.**
  - Ou seja ...

Para qualquer caso de teste que diferencie o comportamento do mutante em relação ao programa original, também é possível concluir que o comportamento do programa original não está de acordo com o resultado esperado.

Exemplo 1 Exemplo 2

Mutante Error-Revealing

- **Mutante gerado pelo operador VTWD**

```
{
   char  achar;
   int length, valid_id;
   length = 0;
   printf ("Identificador: ");
   achar = fgetc (stdin);
   valid_id = valid_s(achar);
   if (valid_id)
      length = 1;
   achar = fgetc (stdin);
   while (achar != '\n')
   {
      if (!(valid_f(achar)))
         valid_id = 0;
      length++;
      achar = fgetc (stdin);
   }
   if (valid_id * (length >= 1) && (PRED(length) < 6))
      printf ("Valido\n");
   else
      printf ("Invalido\n");
}
```

Programa Identifier (função main)

Exemplo Identifier

Complementar

Exemplo

Mutante Error-Revealing

- **Mutante gerado pelo operador ORRN**

```
{
     char  achar;
     int length, valid_id;
     length = 0;
     printf ("Identificador: ");
     achar = fgetc (stdin);
     valid_id = valid_s(achar);
     if (valid_id)
        length = 1;
     achar = fgetc (stdin);
     while (achar != '\n')
     {
        if (!(valid_f(achar)))
           valid_id = 0;
        length++;
        achar = fgetc (stdin);
     }
     if (valid_id * (length >= 1) && (length <= 6))
        printf ("Valido\n");
     else
        printf ("Invalido\n");
}
```

t = (ABCDEF, Válido)}
Saída obtida = Inválido

Programa Identifier (função main)

Programa Identifier: Versão Corrigida

```
{
   char  achar;
   int length, valid_id;
   length = 0;
   printf ("Identificador: ");
   achar = fgetc (stdin);
   valid_id = valid_s(achar);
   if (valid_id)
      length = 1;
   achar = fgetc (stdin);
   while (achar != '\n')
   {
      if (!(valid_f(achar)))
         valid_id = 0;
      length++;
      achar = fgetc (stdin);
   }
   if (valid_id * (length >= 1) && (length <= 6))
      printf ("Valido\n");
   else
      printf ("Invalido\n");
}
```

Programa Identifier (função main)

Escore de Mutação

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas

- **Relação entre o número de mutantes mortos e o número de mutantes gerados, descartados os equivalentes.**
  - Medida objetiva a respeito do nível de confiança da adequação dos casos de teste utilizados.
  - Varia no intervalo entre 0 e 1 sendo que, quanto maior o escore mais adequado é o conjunto de casos de teste.

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

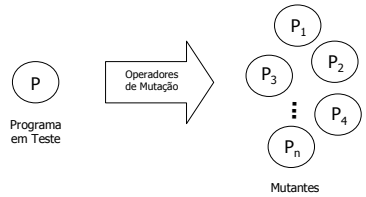
  - *DM*(*P*, *T*): total de mutantes mortos pelo conjunto de casos de teste *T*;
  - *M*(*P*): total de mutantes gerados a partir do programa *P*; e
  - *EM*(*P*): total de mutantes equivalentes ao programa *P*.

Análise de Mutantes: Aplicação

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas

1 - Geração de Mutantes

Para modelar os desvios sintáticos mais comuns, operadores de mutação são aplicados a um programa, transformando-o em programas similares: mutantes.



Análise de Mutantes: Aplicação

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas

2 - Execução do Programa

- Execução do programa com os casos de teste

3 - Execução dos Mutantes

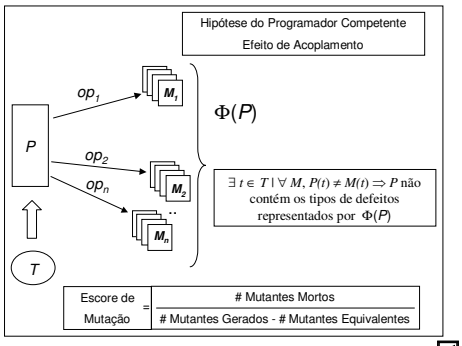
- Execução dos mutantes com os casos de teste
  - Mutante morto
  - Mutante vivo

4 - Análise dos Mutantes Vivos

- Mutante equivalente
- Inclusão de novos casos de teste
- Escore de mutação

Análise de Mutantes: Aplicação

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas



Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas
Mutação Aleatória
Mutação Seletiva
Mutação Restrita

Abordagens Alternativas

- **Critério Análise de Mutantes**
  - Alta eficácia em revelar a presença de erros
  - Alto custo de aplicação
    - Equivalência entre programas
    - Grande número de mutantes gerados e que precisam ser executados
- **Abordagens Alternativas**
  - Mutação Aleatória
  - Mutação Seletiva
  - Mutação Restrita
    - Conjunto Essencial de Operadores de Mutação

Viabilizar a aplicação do critério em ambientes reais de desenvolvimento de software.

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas
Mutação Aleatória
Mutação Seletiva
Mutação Restrita

Mutação Aleatória

- **Apenas uma porcentagem dos mutantes gerados a partir de cada operador é considerada.**

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas
Mutação Aleatória
Mutação Seletiva
Mutação Restrita

Mutação Seletiva

- **Os operadores de mutação responsáveis pelo maior número de mutantes não são aplicados.**

Técnicas de Teste
Técnica Funcional
Técnica Estrutural
Técnica Baseada em Erros
Análise de Mutantes
Operador de Mutação
Mutante
Escore de Mutação
Aplicação
Abordagens Alternativas
Mutação Aleatória
Mutação Seletiva
Mutação Restrita

Mutação Restrita

- **Operadores de mutação específicos são selecionados para serem utilizados na geração dos mutantes.**

Seja *OP* o conjunto total de operadores de mutação: $\forall$ subconjunto $SC \in 2^{(OP)}$ constitui um critério de mutação restrito.

- **Conjunto Essencial de Operadores de Mutação**
  - Subconjunto *SC* tal que:

$\Uparrow$ Escore
$\Downarrow$ Custo $\approx$ Eficácia

Ferramentas de Teste

Ferramentas de Teste

Para a aplicação efetiva de um critério de teste faz-se necessário o uso de ferramentas automatizadas que apóiem a aplicação desse critério.

- Contribuem para reduzir as falhas produzidas pela intervenção humana.
  - Aumento da qualidade e produtividade da atividade de teste.
  - Aumento da confiabilidade do software.
- Facilitam a condução de estudos comparativos entre critérios.

Ferramentas de Teste

Ferramentas de Teste

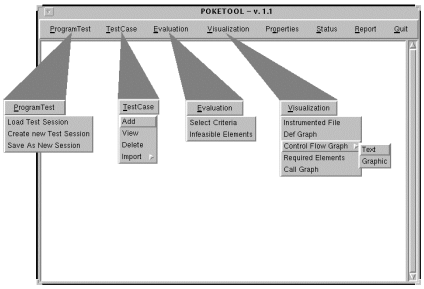
- **Interface Gráfica**
  - Mais fácil
  - Constante interação com o testador
- **Scripts**
  - Possibilitam a condução de uma sessão de teste de modo programado.
  - Domínio dos conceitos e critérios utilizados e dos programas que compõem as ferramentas.

Ferramenta PokeTool

- **Critérios Baseados em Fluxo de Controle**
- **Critérios de Rapps e Weyuker**
- **Critérios Potenciais-Usos**
- **Linguagem C**
- **Características**
  - Sessão de teste
  - Importação de casos de teste
  - Inserção e remoção de casos de teste dinamicamente
  - Casos de teste podem ser habilitados ou desabilitados
  - Geração de relatórios

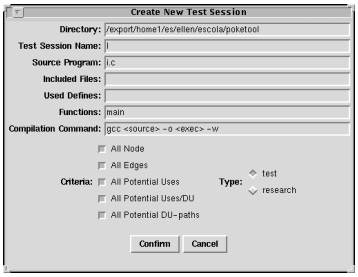
Ferramenta PokeTool

- **Interface Gráfica**



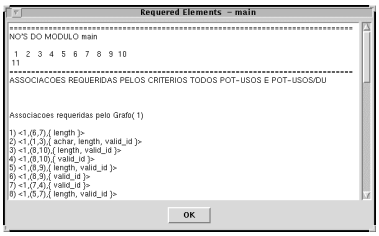
Ferramenta PokeTool

- **Sessão de Teste**



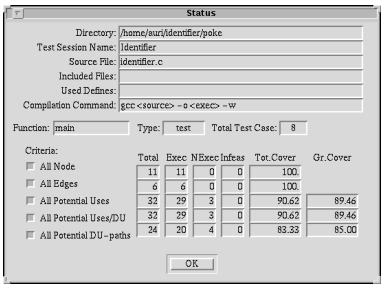
Ferramenta PokeTool

- **Elementos Requeridos**



Ferramenta PokeTool

- **Relatórios de Teste**

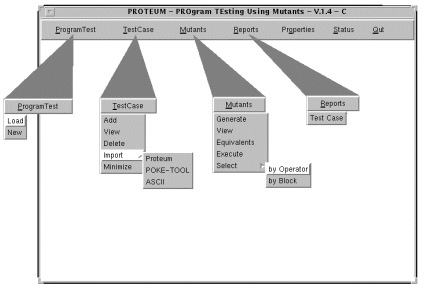


Ferramenta Proteum

- **Critério Análise de Mutantes**
- **Linguagem C**
- **Características**
  - Sessão de teste
  - Importação de casos de teste
  - Inserção e remoção de casos de teste dinamicamente
  - Casos de teste podem ser habilitados ou desabilitados
  - Seleção dos operadores a serem utilizados
    - 71 operadores: comandos, operadores, variáveis e constantes
  - Geração de relatórios

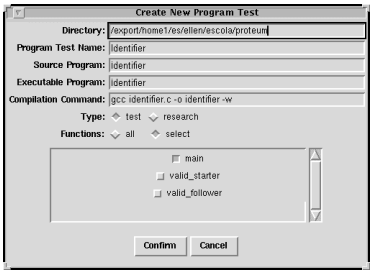
Ferramenta Proteum

- **Interface Gráfica**



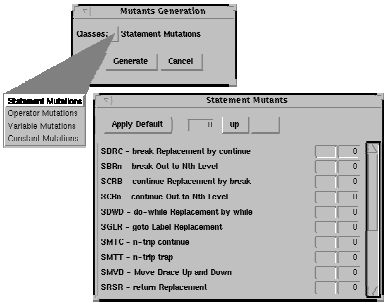
Ferramenta Proteum

- **Sessão de Teste**



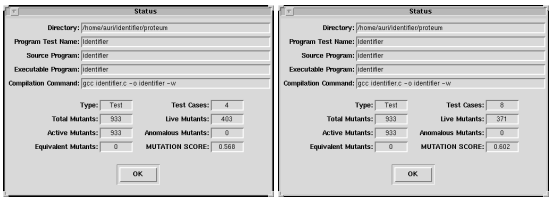
Ferramenta Proteum

- **Geração de Mutantes**



Ferramenta Proteum

- **Relatórios de Teste**
  - *Status* após T_0 (a) e T_1 e T_2 (b)

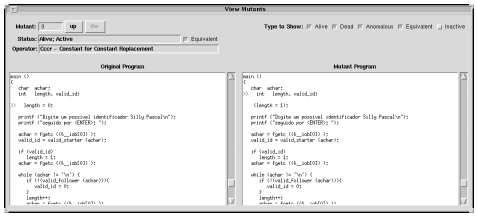


(a) (b)

T_0 = { (a1, Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)}
$T_1 = T_0 \cup$ {(1#, Inválido), (%, Inválido), (c, Válido)}
$T_2 = T_1 \cup$ {(#-%, Inválido)}

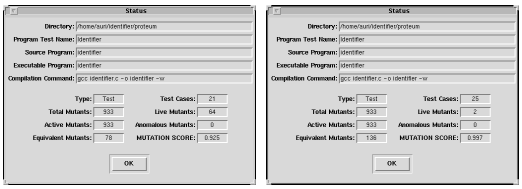
Ferramenta Proteum

- **Visualização de Mutantes**



Ferramenta Proteum

- **Relatórios de Teste**
  - *Status* após T_3 (a) e T_4 (b)

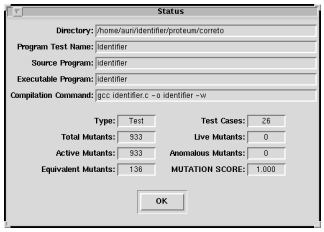


(a) (b)

$T_3 = T_2 \cup$ {(zzz, Válido), (aA, Válido), (A1234, Válido), (ZZZ, Válido),
(AAA, Válido), (aa09, Válido), ([, Inválido), ({, Inválido),
(x/, Inválido), (x:, Inválido), (x18, Válido), (x[[, Inválido), (x{{, Inválido)}
$T_4 = T_3 \cup$ { (@, Inválido), (`, Inválido), (x@, Inválido), (x`, Inválido)}

Ferramenta Proteum

- **Relatórios de Teste**
  - *Status* após T_5 no programa corrigido



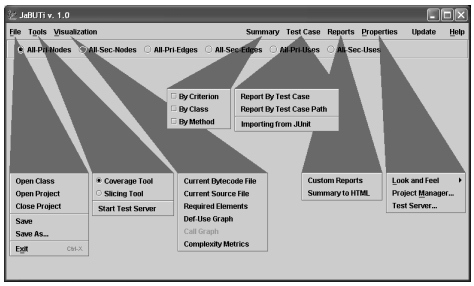
$T_5 = T_4 \cup \{(ABCDEF, Válido)\}$

Ferramenta JaBUTi

- **Teste OO**
  - Todos-nós
  - Todos-arcos
  - Todos-arcos-primários
  - Todos-arcos-secundários
  - Todos-usos
  - Todos-usos-primários
  - Todos-arcos-secundários
- **Java Bytecode**
- **Características**
  - Projeto de teste
  - Importação de casos de teste
  - Inserção e remoção de casos de teste dinamicamente
  - Casos de teste podem ser habilitados ou desabilitados
  - Geração de relatórios
    - Resumo
    - Relatório por Critério
    - Relatório por Classe
    - Relatório por Caso de Teste

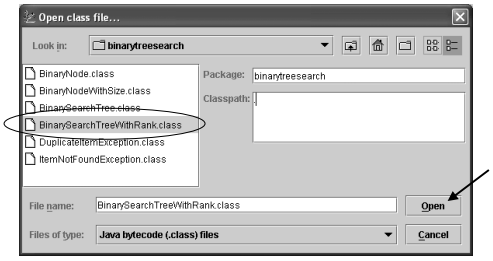
Ferramenta JaBUTi

- **Tela Principal**



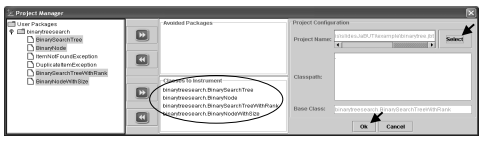
Ferramenta JaBUTi

- **Projeto de Teste**
  - Identificação da classe base



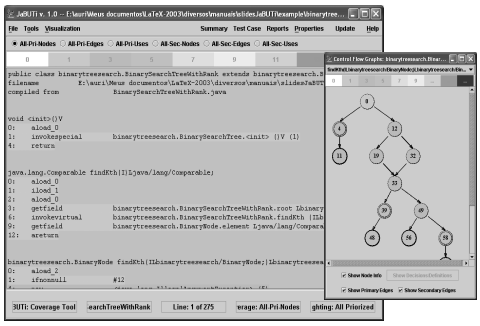
Ferramenta JaBUTi

- **Projeto de Teste**
  - Seleção das classes a serem instrumentadas



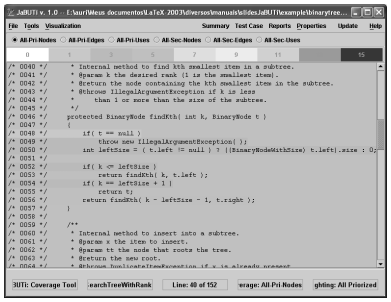
Ferramenta JaBUTi

- **Bytecode e CFG**



Ferramenta JaBUTi

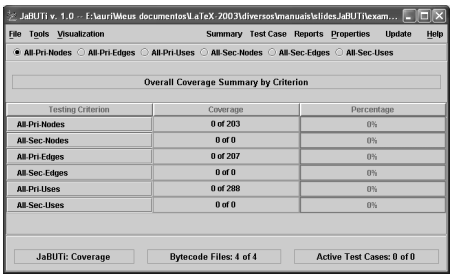
- **Código-Fonte**



Exemplo

Ferramenta JaBUTi

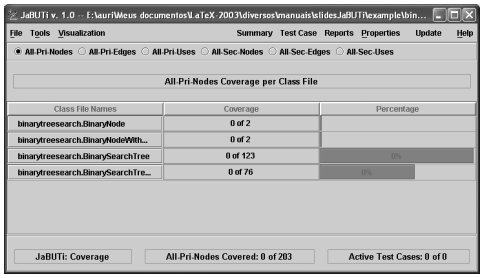
- **Relatórios**
  - Resumo Inicial por Critério



Exemplo

Ferramenta JaBUTi

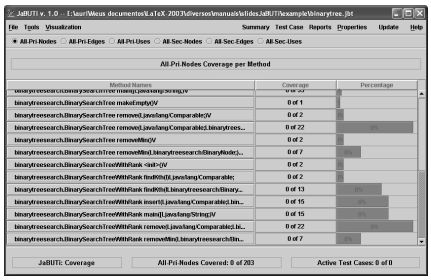
- **Relatórios**
  - Resumo Inicial por Classe



Exemplo

Ferramenta JaBUTi

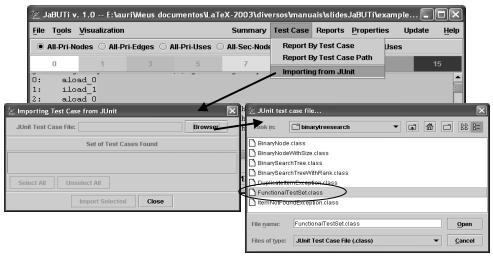
- **Relatórios**
  - Resumo Inicial por Método



Exemplo

Ferramenta JaBUTi

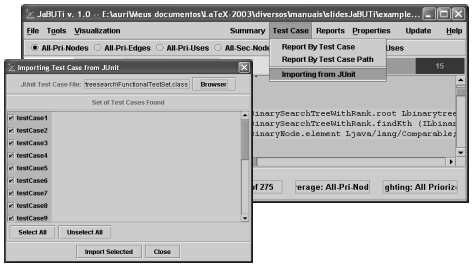
- **Importação de Casos de Teste**



Exemplo

Ferramenta JaBUTi

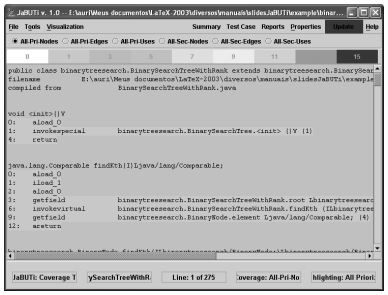
- **Importação de Casos de Teste**



Exemplo

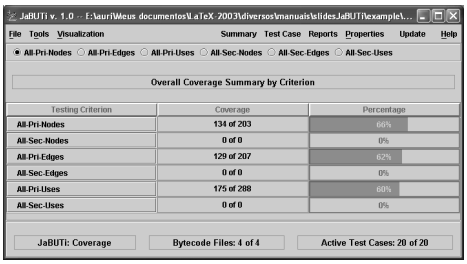
Ferramenta JaBUTi

- **Atualização da Cobertura**



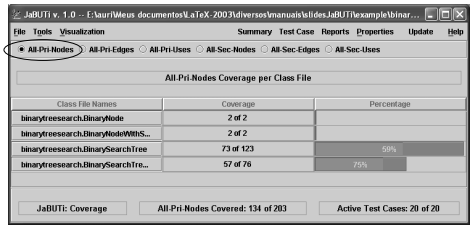
Ferramenta JaBUTi

- **Relatório por Critério de Teste**



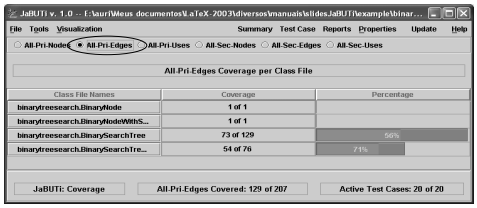
Ferramenta JaBUTi

- **Relatório por Classe**
  - All-Prim-Nodes



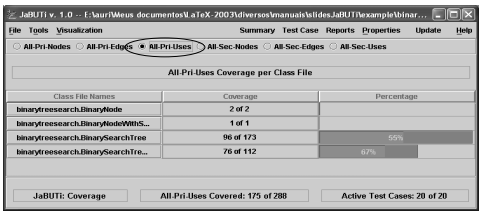
Ferramenta JaBUTi

- **Relatório por Classe**
  - All-Prim-Edges



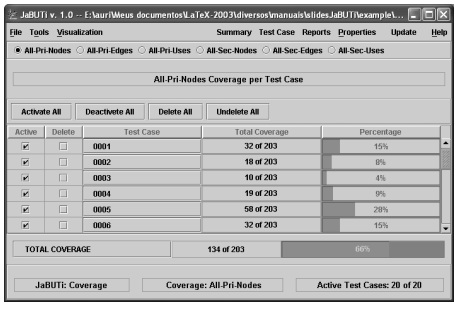
Ferramenta JaBUTi

- **Relatório por Classe**
  - All-Prim-Uses



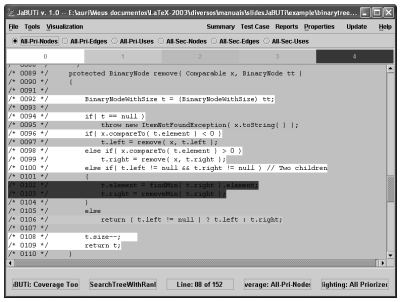
Ferramenta JaBUTi

- **Relatório por Caso de Teste**



Ferramenta JaBUTi

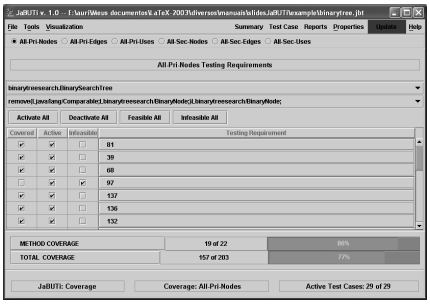
- **Dicas**



Exemplo

Ferramenta JaBUTi

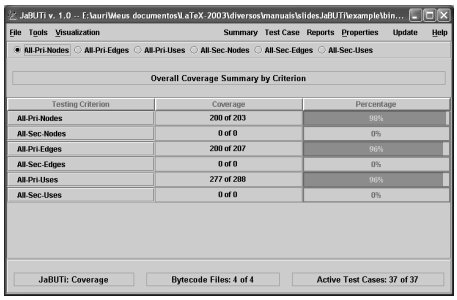
- **Identificação de Requisitos Não-Executáveis**



Exemplo

Ferramenta JaBUTi

- **Relatório por Critério de Teste**



Exemplo

Garantia de Qualidade

Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Atividades de V&V

- **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 V&V**
  - Verificação e Validação

Qualidade

Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Atividades de V&V

"Qualidade é a totalidade de características e critérios de um produto ou serviço que exercem suas habilidades para satisfazer às necessidades declaradas ou envolvidas." [ISO9126 1994]

- **Conformidade com requisitos funcionais e de desempenho, padrões de desenvolvimento documentados, e características implícitas esperadas de todo software profissionalmente desenvolvido.**

Qualidade

Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Atividades de V&V

- Os requisitos de software são a base a partir da qual a qualidade é medida.
  - A falta de conformidade aos requisitos significa falta de qualidade.
- Padrões especificados definem um conjunto de critérios de desenvolvimento que orientam a maneira segundo a qual o software passa pelo trabalho de engenharia.
  - Se os critérios não forem seguidos, o resultado quase que seguramente será a falta de qualidade; e
- Existe ainda um conjunto de requisitos implícitos que freqüentemente não são mencionados na especificação.
  - Por exemplo, o desejo de uma boa manutenibilidade.

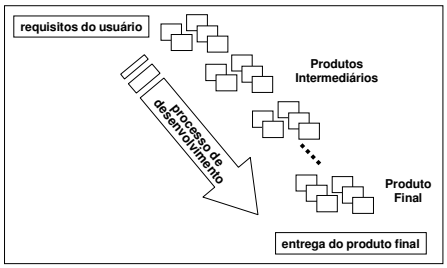
Qualidade de Produto

- **Norma ISO/IEC 9126**
  - Padronização mundial para a qualidade do produto.
  - Características
    - Funcionalidade: Satisfaz as necessidades?
    - Confiabilidade: É imune a falhas?
    - Usabilidade: É fácil de usar?
    - Eficiência: É rápido e "enxuto"?
    - Manutenibilidade: É fácil de modificar?
    - Portabilidade: É fácil de usar em outro ambiente?

Teste: Atividade relevante para avaliação da característica Funcionalidade.

Qualidade dos Produtos Intermediários

- **Cada produto intermediário tem certos atributos de qualidade que afetam a qualidade do produto intermediário da próxima fase e assim, afetam a qualidade do produto final.**

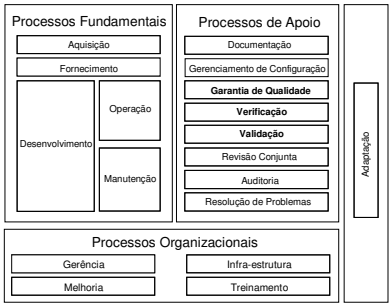


Processo de Software

- **Consiste em uma série de atividades, práticas, eventos, ferramentas e métodos que garantem, técnica e administrativamente que o software pode ser desenvolvido com qualidade e de maneira organizada, disciplinada e previsível.**

Qualidade de Processo

- **Norma ISO/IEC 12207**

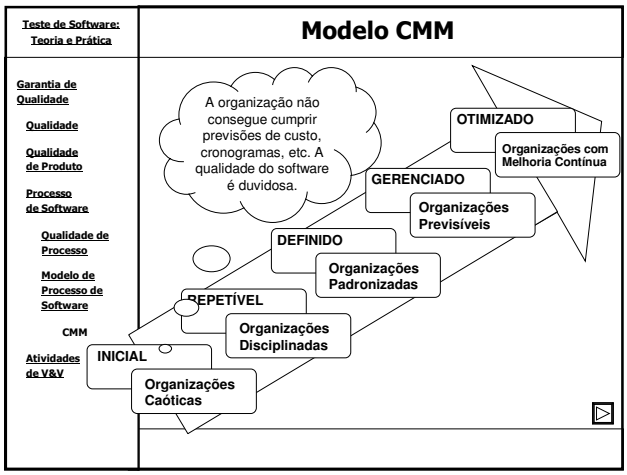


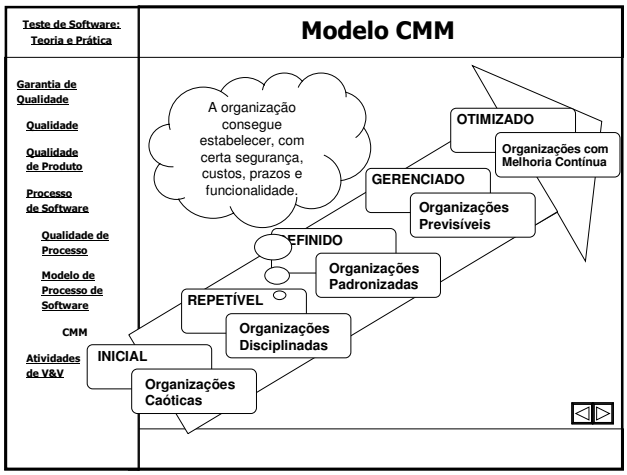
Modelo de Processo de Software

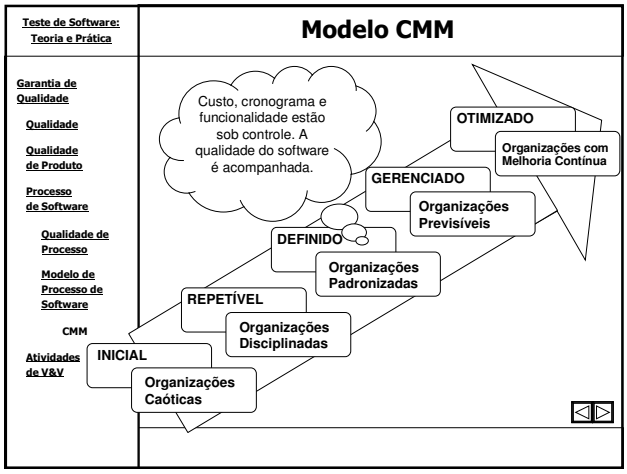
- **Modelo de Processo de Software**
  - Procura descrever formalmente e de maneira organizada todas as atividades que devem ser seguidas para a obtenção segura de um produto de software.
  - Padrões relacionados a Processo de Software
    - CMM
    - SPICE
    - Bootstrap

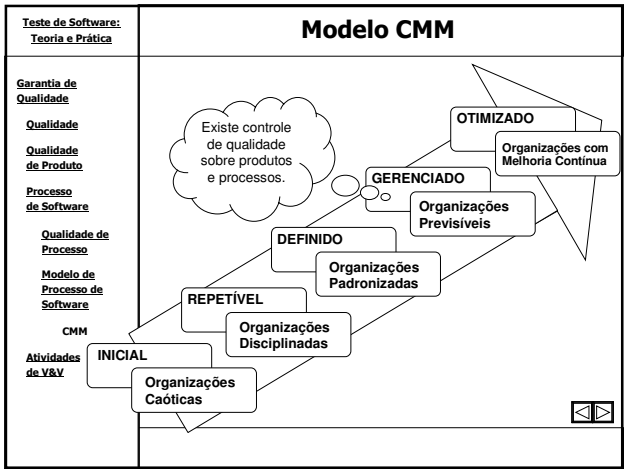
Modelo CMM (Capability Maturity Model)

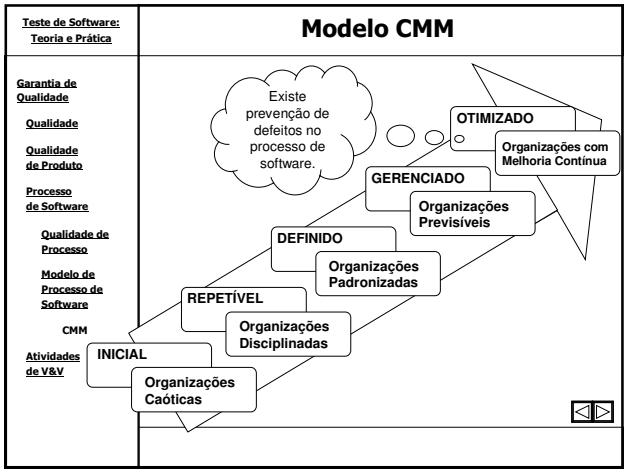
- **Modelo de referência para a qualidade de processo de produção de software.**
- **Através de um processo formal de avaliação, uma organização é classificada em um "nível de maturidade" .**
  - O nível de maturidade indica em que medida os processos da organização são maduros.
  - Quanto maior o nível de maturidade, melhores e mais maduros são os processos.
- **Cada nível de maturidade possui um grupo de atividades correlatas – áreas-chave do processo (KPA´s)**
  - Visam a atingir as metas consideradas importantes na implementação da competência do processo.

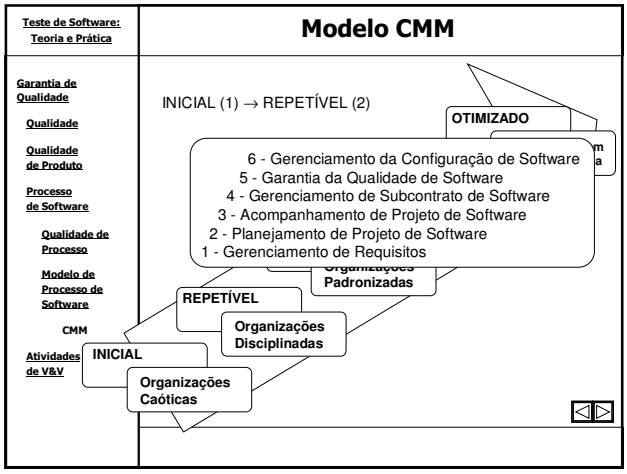

Teste de Software: Teoria e Prática
Modelo CMM
Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Qualidade de Processo
Modelo de Processo de Software
CMM
Atividades de V&V
A organização não consegue cumprir previsões de custo, cronogramas, etc. A qualidade do software é duvidosa.
OTIMIZADO
Organizações com Melhoria Contínua
GERENCIADO
Organizações Previsíveis
DEFINIDO
Organizações Padronizadas
REPETÍVEL
Organizações Disciplinadas
INICIAL
Organizações Caóticas


Teste de Software: Teoria e Prática
Modelo CMM
Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Qualidade de Processo
Modelo de Processo de Software
CMM
Atividades de V&V
A organização consegue estabelecer, com certa segurança, custos, prazos e funcionalidade.
OTIMIZADO
Organizações com Melhoria Contínua
GERENCIADO
Organizações Previsíveis
DEFINIDO
Organizações Padronizadas
REPETÍVEL
Organizações Disciplinadas
INICIAL
Organizações Caóticas


Teste de Software: Teoria e Prática
Modelo CMM
Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Qualidade de Processo
Modelo de Processo de Software
CMM
Atividades de V&V
Custo, cronograma e funcionalidade estão sob controle. A qualidade do software é acompanhada.
OTIMIZADO
Organizações com Melhoria Contínua
GERENCIADO
Organizações Previsíveis
DEFINIDO
Organizações Padronizadas
REPETÍVEL
Organizações Disciplinadas
INICIAL
Organizações Caóticas


Teste de Software: Teoria e Prática
Modelo CMM
Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Qualidade de Processo
Modelo de Processo de Software
CMM
Atividades de V&V
Existe controle de qualidade sobre produtos e processos.
OTIMIZADO
Organizações com Melhoria Contínua
GERENCIADO
Organizações Previsíveis
DEFINIDO
Organizações Padronizadas
REPETÍVEL
Organizações Disciplinadas
INICIAL
Organizações Caóticas


Teste de Software: Teoria e Prática
Modelo CMM
Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Qualidade de Processo
Modelo de Processo de Software
CMM
Atividades de V&V
Existe prevenção de defeitos no processo de software.
OTIMIZADO
Organizações com Melhoria Contínua
GERENCIADO
Organizações Previsíveis
DEFINIDO
Organizações Padronizadas
REPETÍVEL
Organizações Disciplinadas
INICIAL
Organizações Caóticas


Teste de Software: Teoria e Prática
Modelo CMM
Garantia de Qualidade
Qualidade
Qualidade de Produto
Processo de Software
Qualidade de Processo
Modelo de Processo de Software
CMM
Atividades de V&V
INICIAL (1) → REPETÍVEL (2)
OTIMIZADO
6 - Gerenciamento da Configuração de Software
5 - Garantia da Qualidade de Software
4 - Gerenciamento de Subcontrato de Software
3 - Acompanhamento de Projeto de Software
2 - Planejamento de Projeto de Software
1 - Gerenciamento de Requisitos
Padronizadas
REPETÍVEL
Organizações Disciplinadas
INICIAL
Organizações Caóticas

Modelo CMM

INICIAL (1) → REPETÍVEL (2)

OTIMIZADO

6 - Gerenciamento da Configuração de Software
⇨ 5 - Garantia da Qualidade de Software

Esta kpa tem o duplo objetivo de assegurar a qualidade tanto do produto sendo construído (o software) quanto do processo de sua construção.

Em termos de produto, isso significa que o CMM exige que a organização defina métodos, técnicas e padrões para verificação da qualidade, estabelecendo, por exemplo, quais os **tipos de teste** que deverão ser realizados, o uso de **revisões**, **inspeções**, etc.

Já em relação ao processo, esta kpa procura garantir que todas as outras kpas estejam sendo respeitadas, garantindo, por exemplo, que o gerente de projetos esteja fazendo estimativas corretamente usando o modelo definido para a organização.

INICIAL

Caóticas

Modelo CMM

REPETÍVEL (2) → DEFINIDO (3)

OTIMIZADO

7- Revisões (peer review)
6- Coordenação Intergrupos
5- Engenharia de Produto de Software
4- Gerenciamento de Software Integrado
3- Programa de Treinamento
2- Definição do Processo da Organização
1- Foco no Processo da Organização

Padronizadas

REPETÍVEL

Organizações Disciplinadas

INICIAL

Organizações Caóticas

Modelo CMM

REPETÍVEL (2) → DEFINIDO (3)

OTIMIZADO

7- Revisões (peer review)
6- Coordenação Intergrupos
⇨ 5- Engenharia de Produto de Software

Esta kpa procura garantir que o processo de software definido seja executado de modo que sejam produzidos produtos corretos e consistentes de forma eficaz e eficiente.

Padronizadas

REPETÍVEL

Organizações Disciplinadas

INICIAL

Organizações Caóticas

Teste: Atividade essencial para ascensão ao Nível 3 do Modelo CMM.

Atividades de V&V

- **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

- **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

- **Envolvem atividades de análise estática e de análise dinâmica.**
- **Análise Estática**
  - Revisões Técnicas
  - Inspeções
  - *Model Checking*
  - *Peer Review*

Atividades Complementares

- **Análise Dinâmica**
  - Simulações
  - Teste

Teste de MEF´s

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO

- **Modelo Subjacente:**
  - Árvore de Alcançabilidade
- **Critérios Definidos:**
  - Critério TT
  - Critério "n-switch set cover"
    - n-switch: n+1 transições consecutivas
  - Critério DS
  - Critério UIO
  - Critério W
  - Critério Wp

Teste de Statecharts

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO

- **Modelo Subjacente:**
  - Árvore de Alcançabilidade
- **Família de Critérios FCCS:**
  - Abordam aspectos de fluxo de controle
    - Todas-Configurações
    - Todas-Transições
    - Todos-Caminhos
    - Todos-Caminhos-k-C_0-Configurações
    - Todos-Caminhos-k-Configurações
    - Todos-Caminhos-com-um-Laço
    - Todos-Caminhos-Simples
    - Todos-Caminhos-livre-Laço
    - Todas-Reação-em-Cadeia
    - Todas-Configurações-História

Teste de Programas Concorrentes

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO

- **Modelos Subjacentes:**
  - Grafo de Fluxo de Programa
  - Grafo de Sincronização
  - Grafo de Concorrência
- **Critérios Definidos:**
  - Teste Análise de Caminhos
  - Abordagem de Taylor et al.
    - Todos-caminhos-concorrência
    - Todas-histórias-concorrência
    - Todos-arcos-entre-estados-concorrência
    - Todos-estados-concorrência
    - Todas-possíveis-sincronizações

Teste de Programas OO

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

- **Principais Características**
  - Classes, Objetos, Herança
  - Polimorfismo e Acoplamento Dinâmico
- **Estilo de Programação**
  - Modularidade
  - Objetos
  - Reuso por meio de herança
  - Complexas interações entre classes
  - Uso extensivo de bibliotecas de classes.

Fases de Teste

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

- **As fases de teste são independentes do paradigma de desenvolvimento de software utilizado, seja ele procedimental ou orientado a objetos.**

Menor Unidade: Método		
Fase	Teste Procedimental	Teste OO
Unidade	Intraprocedimental	Intra-Método
Integração	Interprocedimental	Inter-Método
		Intra-Classe
		Inter-Classe
Sistema	Toda Aplicação	Toda Aplicação

Menor Unidade: Classe		
Fase	Teste Procedimental	Teste OO
Unidade	Intraprocedimental	Intra-Método
		Inter-Método
		Intra-Classe
Integração	Interprocedimental	Inter-Classe
Sistema	Toda Aplicação	Toda Aplicação

Teste Baseado em Especificação

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

- **Análise do Valor Limite**
- **Particionamento de Equivalência**
- **Método Partição-Categoria**

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

Teste Baseado em Estados

- **O comportamento dos objetos é representado por meio de uma máquina de estado ou diagrama de transição de estado.**
- **A partir da máquina de estado finito é derivada a árvore de transição de estado.**
- **Cada caminho completo ou parcial na árvore dá origem a um caso de teste.**
  - missing transitions, incorrect transitions, incorrect output actions, incorrect states.

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

Teste Baseado em Erros

- **Técnica Hazard and Operability Studies (HAZOP)**
  - Determinar um conjunto de operadores de mutação para a linguagem Java (Intra-Classe).
- **Delamaro *et al*. definiram um conjunto de operadores de mutação que modelam erros de concorrência em programas Java.**

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

Teste Baseado em Fluxo de Dados

- **Teste de Fluxo de Dados em Classes**
  - Grafo de Fluxo de Controle de Classe
  - Diferentes níveis de teste são considerados:
    - Teste Intra-Método
    - Teste Inter-Método
    - Teste Intra-Classe
    - Teste Inter-Classe

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

Teste Baseado em Fluxo de Dados

- **Estratégia de Teste Incremental Hierárquica**
  - Reaproveitar casos de teste da super-classe, reduzindo os custos no teste das subclasses.
  - Teste da super-classe ou classe-base
    - Teste intra-método
      - Construção do GFC de cada método
  - Teste intra-classe e inter-classe
      - Construção do GFCC de cada classe
  - História do teste é registrada
  - Teste da subclasse

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

Teste Baseado em Fluxo de Dados

- **Teste Intra-Método**
  - ASTOOT e TACCLE
  - Critérios para o Teste de Construções de Tratamento de Exceção
  - OMEN (Object Manipulations in addition to using Escape iNformation)
  - Análise de Dependência em Bytecode
  - Análise Estática em Bytecode

Tipos de Aplicações
Teste de MEF´s
Teste de Statecharts
Teste de Programas Concorrentes
Teste de Programas OO
Fases de Teste
Teste Baseado em Especificação
Teste Baseado em Estados
Teste Baseado em Erros
Teste Baseado em Fluxo de Dados

Teste Baseado em Fluxo de Dados

- **Análise Estática em Bytecode**
  - Modelos Subjacentes
    - Instruction CFG
    - Data-Flow Instruction CFG
    - Data-Flow Block Graph (BG)
  - Critérios de Teste: Fluxo de Dados
    - Todos-nós
    - Todos-arcos
    - Todos-arcos-primários
    - Todos-arcos-secundários
  - Critérios de Teste: Fluxo de Controle
    - Todos-usos
    - Todos-usos-primários
    - Todos-arcos-secundários

Máquina de Estados Finitos

- **Uma MEF *M* é uma classe de sistemas com:**
  - Alfabeto de entrada $X = \{x_1, x_2, x_3, ..., x_n\}$
  - Alfabeto de saída $Z = \{z_1, z_2, z_3, ..., z_q\}$
  - Conjunto de estados $S = \{s_1, s_2, s_3, ..., s_n\}$
  - Par de funções de caracterização f_z e f_s dados por:
    - zv = fz (sv ,xv)
    - sv+1 = fs (sv xv)

Onde...
- xv - símbolo de entrada
- zv - símbolo de saída
- sv - estado da MEF M no instante tv

Teste de Software: Teoria e Prática

Máquina de Estados Finitos

Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

- **Uma MEF consiste de:**
  - Um conjunto finito de estados: Q
  - Um conjunto finito de entradas: I
  - Uma função de transição $\delta : Q \times I \rightarrow Q$ ou $\delta : Q \times I \rightarrow Q \times O$, sendo O um conjunto finito de símbolos de saída.

  (δ pode ser uma função parcial, ou seja, pode ser indefinida para alguns valores de seu domínio).

Teste de Software: Teoria e Prática

Máquina de Estados Finitos

Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

- **Representação**

Grafo Direcionado

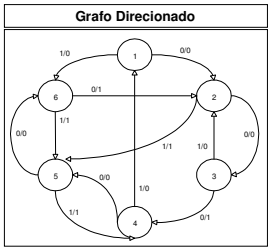


Tabela de Transição

Sv \ Xv	0	1	0	1
1	0	0	2	6
2	0	1	3	5
3	1	0	4	2
4	0	0	5	1
5	0	1	6	4
6	1	1	2	5

Teste de Software: Teoria e Prática

Máquina de Estados Finitos

Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

- **Propriedades**
  - Completamente Especificada
    - Se para cada estado da máquina, existe uma transição para cada símbolo de entrada.
  - Minimal
    - Se nenhum par de estados da máquina é equivalente.
  - Determinística
    - Se para cada entrada existe no máximo uma transição definida em cada estado da máquina.
  - Máquina de Mealy
    - Quando produz uma saída para cada transição da máquina.
  - Fortemente Conectada
    - Se para todo par de estados (g_i, g_j) da máquina, existe uma seqüência de símbolos de entrada que leve g_i até g_j.

Teste de Software: Teoria e Prática

Statecharts

Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

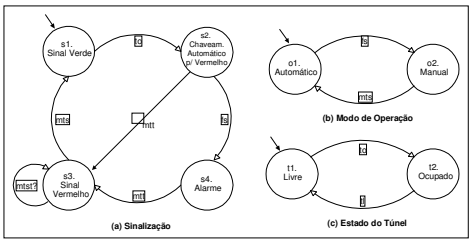
- **Extensão das MEF´s**
- **Sintaxe e Semântica bem definidas.**
  - Semântica dada por uma definição formal das mudanças que ocorrem no sistema como uma reação aos estímulos externos.
  - Semântica baseada em uma seqüência de instantes de tempo $\{\sigma\}_{\delta \geq 0}$.
- **Aspectos essenciais para a especificação e projeto de sistemas complexos:**
  - Hierarquia (decomposição)
  - Concorrência (ortogonalidade)
  - Comunicação (*broadcasting*)

Teste de Software: Teoria e Prática

Statecharts

Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

- **Representação**



Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Grafo Def-Uso
Grafo Def
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização
Executabilidade
Técnica Estrutural

Grafo de Programa

- **Seja um grafo de programa *G = (N, E, s)* onde *N* representa o conjunto de nós, *E* o conjunto de arcos, e *s* o nó de entrada.**
- **Caminho**

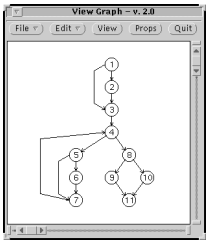
Um caminho é uma seqüência finita de nós $(n_1, n_2, ..., n_k)$, $k \geq 2$, tal que existe um arco de n_i para $n_i + 1$, para $i = 1, 2, ..., k - 1$.

- Caminho Simples
  - Todos os nós que compõem esse caminho, exceto possivelmente o primeiro e o último, são distintos.
- Caminho Livre de Laço
  - Todos os nós são distintos.
- Caminho Completo
  - O primeiro nó é o nó de entrada e o último nó é o nó de saída do grafo *G*.

Exemplo Exercício

Grafo de Programa

- Programa *Identifier*
  - Nó: 1, 2, 3, ...
  - Arco: <1,2>, <1,3>, ...
  - Arcos Primitivos:
    <1,2>,<1,3>,<5,6>,<5,7>,
    <8,9>,<8,10>
  - Caminho
    - Simples: (2,3,4,5,6,7)
    - Completo: (1,2,3,4,5,7,4,8,9,11)



Exemplo Identifier
Complementar
Exemplo

Implementação do Programa Identifier

```
/* 01 */     {
/* 01 */         char  achar;
/* 01 */         int length, valid_id;
/* 01 */         length = 0;
/* 01 */         printf ("Identificador: ");
/* 01 */         achar = fgetc (stdin);
/* 01 */         valid_id = valid_s(achar);
/* 01 */         if (valid_id)
/* 02 */            length = 1;
/* 03 */         achar = fgetc (stdin);
/* 04 */         while (achar != '\n')
/* 05 */         {
/* 05 */            if (!(valid_f(achar)))
/* 06 */               valid_id = 0;
/* 07 */            length++;
/* 07 */            achar = fgetc (stdin);
/* 07 */         }
/* 08 */         if (valid_id && (length >= 1) && (length < 6))
/* 09 */            printf ("Valido\n");
/* 10 */         else
/* 10 */            printf ("Invalido\n");
/* 11 */     }
```

Programa Identifier (função main)

Exemplo Identifier
Complementar

Especificação do Programa Identifier

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

- Função valid_s(): determina se o primeiro caracter é válido.
- Função valid_f(): determina se o próximo caracter é válido.

- Identificadores Válidos
  abc12
  C4d5
  dcdf
- Identificadores Inválidos
  cont*1
  1soma
  a123456

Exemplo Identifier
Complementar

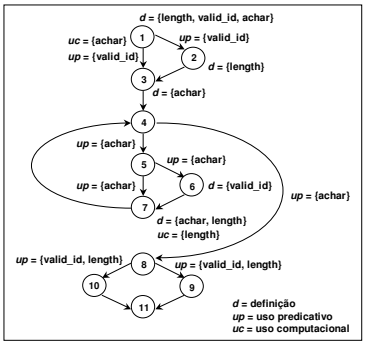
Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Grafo Def-Uso
Grafo Def
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização
Rapps & Weyuker

Grafo Def-Uso

- **Informações a respeito do fluxo de dados do programa.**
  - Extensão do grafo de programa
- **Definição**
  - Atribuição de um valor a uma variável
- **Uso**
  - Predicativo
    - A variável é utilizada em uma condição.
  - Computacional
    - A variável é utilizada em uma computação.

Exemplo

Grafo Def-Uso



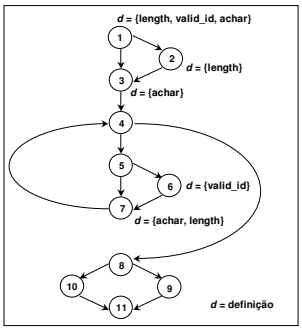
Exemplo Identifier
Complementar
Exemplo

Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Grafo Def-Uso
Grafo Def
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

Grafo Def

- **Associa-se a cada nó do grafo informações a respeito das definições que ocorrem nesses nós.**
  - Extensão do grafo de programa

Grafo Def

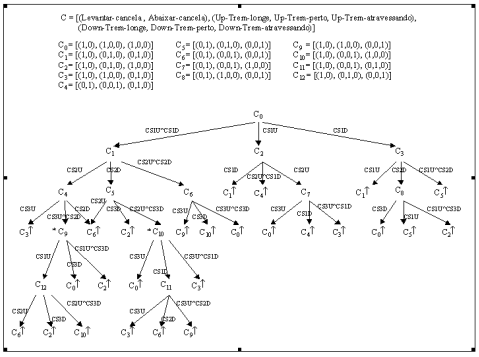


Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

Árvore de Alcançabilidade

- **Permite a análise de algumas propriedades de um Statecharts:**
  - Alcançabilidade
  - Reiniciabilidade
  - *Deadlock*
  - Uso de transições
  - Seqüência de eventos válidos

Árvore de Alcançabilidade



Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

Grafo de Fluxo de Tarefa

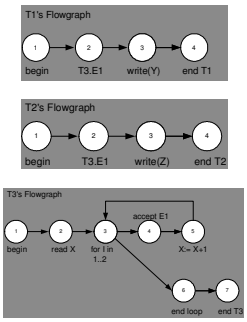
- **Seja *P* um programa concorrente consistindo de n tarefas T_1, T_2, ..., T_n.**
  - T_i pode ser representada como um grafo de fluxo $G_i = (N_i, E_i)$
    - N_i é conjunto de nós
    - E_i é conjunto de arcos
  - Caminho da tarefa
    - Seqüência de nós n_0, n_1, n_2, ..., n_m, onde n_0 é o nó inicial e n_m é o nó final.
    - Um caminho representa uma possível seqüência de execução da tarefa T_i.
    - *C*-caminho é uma *n*-tupla (P_1, P_2, ..., P_n) onde, para cada *i*, P_i é um caminho da tarefa T_i.
      - Executável
      - Não-Executável

Grafo de Fluxo de Tarefa

```
task body T1 is
      Y: INTEGER;
begin
      T3.E1(Y);
      write(Y);
end T1;
```

```
task body T2 is
      Z: INTEGER;
begin
      T3.E1(Z);
      write(Z);
end T2;
```

```
task body T3 is
      X: INTEGER;
begin
      read X;
      for i in 1..2 loop
        accept E1(T: out INTEGER) do
            X:=X+1;
            T:=X;
        end E1;
        end loop;
end T3;
```



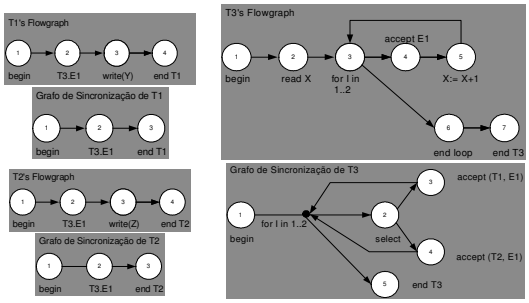
Modelos Subjacentes
MEF´s
Statecharts
Grafo de Programa
Árvore de Alcançabilidade
Grafo de Fluxo de Tarefa
Grafo de Sincronização

Grafo de Sincronização

- **Caracteriza cada comportamento de execução ao longo de um *C*-caminho.**
- **Sincronização**
  - *select*
  - *entry call*
  - *accept*
- **$RG_i = (RN_i, RE_i)$ para cada tarefa T_i**
  - RN_i e RE_i são obtidos do grafo de fluxo de tarefa G_i de T_i

Exemplo

Grafo de Sincronização



Exemplo

Estudos Teóricos e Empíricos
Estudos Teóricos
Estudos Empíricos

Estudos Teóricos e Empíricos

- **Diversidade de Critérios de Teste**

Qual critério utilizar a fim de obter a melhor relação custo/benefício?

- **Estudos Teóricos e Empíricos**
  - Avaliar e comparar critérios de teste

Estudos Teóricos e Empíricos
Estudos Teóricos
Estudos Empíricos

Estudos Teóricos

- **Relação de Inclusão**
  - Estabelece uma ordem parcial entre os critérios, caracterizando uma hierarquia entre eles.

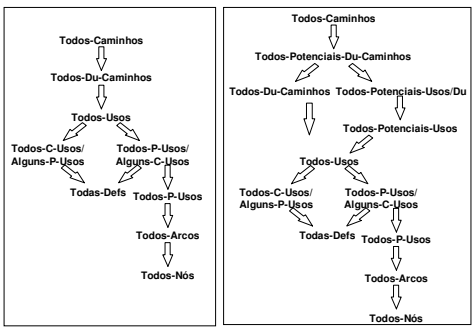
Diz-se que um critério C_1 inclui um critério C_2 se para qualquer programa *P* e qualquer conjunto de casos de teste T_1 C_1-adequado, T_1 for também C_2-adequado e existir um programa *P* e um conjunto T_2 C_2-adequado que não seja C_1-adequado.

- **Complexidade**
  - Número máximo de casos de teste exigidos por um critério de teste, no pior caso.

Exemplo 1 Exemplo 2

Estudos Teóricos

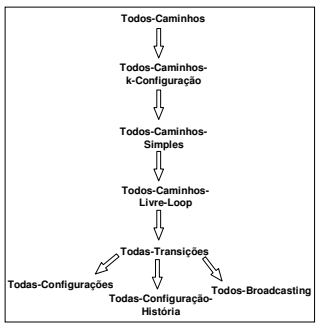
- **Hierarquia entre Critérios Estruturais**



Exemplo

Estudos Teóricos

- **Hierarquia entre Critérios FCCS**



Exemplo

Teste de Software: Teoria e Prática

Estudos Teóricos e Empíricos

Estudos Teóricos

Estudos Empíricos

Estudos Empíricos

- **Custo**
  - Esforço necessário para que um critério seja utilizado.
- **Eficácia**
  - Capacidade de um critério em revelar um maior número de erros em relação a outro.
- **Dificuldade de satisfação (*strength*)**
  - Capacidade de satisfazer-se um critério tendo satisfeito outro.

Exemplo 1 Exemplo 2 Exemplo 3

Estudos Empíricos

- **Avaliação Empírica de Técnicas de VV&T**
  - Comparar técnicas de VV&T quanto à eficácia e eficiência.
- ***Code Reading* (*Stepwise Abstraction*)**
- **Técnica Funcional**
  - Particionamento de Equivalência
  - Análise do Valor-Limite
- **Técnica Estrutural**
  - Todos-Nós

Complementar

Exemplo

Estudos Empíricos

- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Média de defeitos isolados por participante

Technique	Average	Standard Deviation	Number of Subjects
Code Reading	3,92	1,83	12
Functional Testing	2,58	0,90	12
Incremental Testing	2,58	1,73	12

Exemplo

Estudos Empíricos

- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Eficácia e eficiência – Defeitos

Technique	Average Percentage of Faults Isolated (of total possible)	Fault-Isolation Rate (hours)
Code Reading	47,45 %	1,35
Functional Testing	31,25 %	0,95
Incremental Testing	31,25 %	0,69

Exemplo

Estudos Empíricos

- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Média de falhas observadas por participante

Technique	Average	Standard Deviation	Number of Subjects
Code Reading	2,92	1,62	12
Functional Testing	1,58	1,78	12
Incremental Testing	1,92	1,78	12

Exemplo

Estudos Empíricos

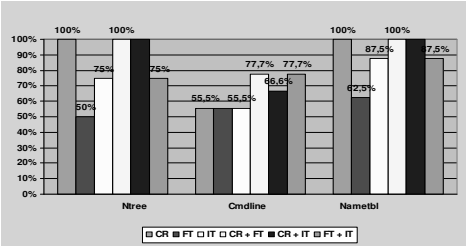
- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Eficácia e eficiência – Falhas

Technique	Average Percentage of Faults Isolated (of total possible)	Fault-Isolation Rate (hours)
Code Reading	35,30 %	1,14
Functional Testing	19,44 %	1,23
Incremental Testing	26,16 %	0,75

Exemplo

Estudos Empíricos

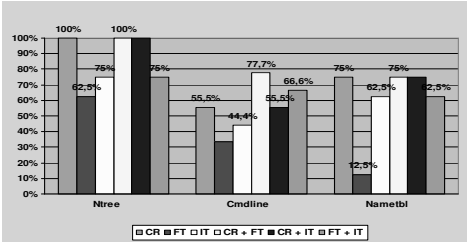
- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Aspectos Complementares – Defeitos



Exemplo

Estudos Empíricos

- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Aspectos Complementares – Falhas



Exemplo

Estudos Empíricos

- **Avaliação Empírica de Técnicas de VV&T**
  - Principais Resultados
    - Falhas
      - Code Reading mostrou-se mais efetivo.
      - Técnica Funcional mostrou-se mais eficiente.
      - Em média, 26.96% das falhas foram detectadas.
    - Defeitos
      - Code Reading mostrou-se mais efetivo.
      - Nenhuma correlação significativa entre a experiência dos participantes e os resultados obtidos.

A combinação das técnicas pode melhorar os resultados quanto à observação de falhas e detecção de defeitos.

Exemplo

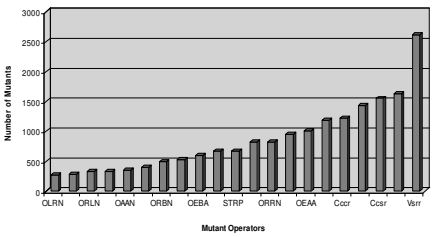
Estudos Empíricos

- **Avaliação Empírica de Critérios de Teste**
- **Estudo de Mathur, Wong e Offutt**
  - Análise de Mutantes × Mutação Aleatória (10%) × Mutação Restrita × Todos-Usos
    - Custo (número de casos de teste)
      - Análise de Mutantes, Todos-Usos, Mutação Aleatória e Mutação Restrita
    - Eficácia
      - Análise de Mutantes, Mutação Restrita, Todos-Usos e Mutação Aleatória

Exemplo

Estudos Empíricos

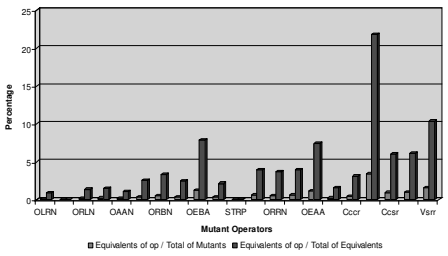
- **Conjunto Essencial de Operadores de Mutação**
  - Custo dos operadores de mutação quanto ao número de mutantes gerados



Exemplo

Estudos Empíricos

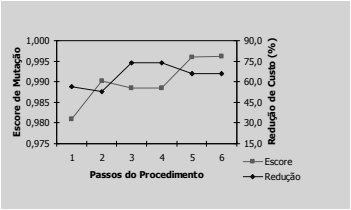
- **Conjunto Essencial de Operadores de Mutação**
  - Custo dos operadores de mutação quanto ao número de mutantes equivalentes.



Exemplo

Estudos Empíricos

Passo	Conjunto Essencial Preliminar	Escore	Redução
1	{Ccsr, Vsrr, CRCR, VTWD, Cccr, ORRN}	0.98087	56.234
2	{Ccsr, Vsrr, CRCR, VTWD, Cccr, ORRN, SSDL}	0.99014	52.879
3	{Ccsr, VTWD, Cccr, ORRN, SSDL}	0.98839	73.980
4	{SSDL, ORRN, VTWD, Cccr, Ccsr}	0.98839	73.980
5	{SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}	0.99592	65.988
6	{SWDD, SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}	0.99616	65.899



Exemplo

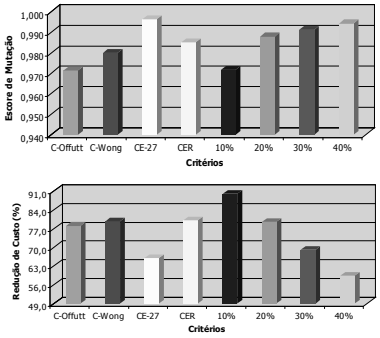
Estudos Empíricos

- **Conjunto Essencial de Operadores de Mutação**
  - Critérios Restritos e Aleatórios × Conjuntos Essenciais

Critério	Escore de Mutação	Redução de Custo (%)
C-Offutt	0.97143	78.115
C-Wong	0.97979	79.738
CE-27	0.99616	65.899
CER	0.98505	80.031
Mutação 10% Aleatória	0.97160	90.038
Mutação 20% Aleatória	0.98791	79.554
Mutação 30% Aleatória	0.99120	69.006
Mutação 40% Aleatória	0.99420	59.352

Exemplo

Estudos Empíricos



Exemplo

Teste de Software: Teoria e Prática

Direções para Teste

Direções para a Área de Teste

- **Pesquisas Fundamentais**
  - Teste de sistemas baseado em componentes
  - Teste de artefatos baseados em projeto
  - Teste de regressão
  - Estudo demonstrando a eficácia das técnicas de teste
  - Desenvolvimento de processos de teste efetivos
  - Uso de artefatos (informação) de teste
  - Outras abordagens de teste
- **Métodos e Ferramentas de Teste**
- **Estudos Empíricos**

Teste de Software: Teoria e Prática

Direções para Teste

Direções para a Área de Teste

- **Teste é uma atividade indispensável dentro do processo de desenvolvimento de software.**
- **Crescimento significativo na demanda por pessoal capacitado, sobretudo no ambiente industrial.**

Teste de Software: Teoria e Prática

Direções para Teste

Direções para a Área de Teste

- **O meio acadêmico não tem respondido satisfatoriamente. Algumas iniciativas:**
  - *George Mason University*
  - *Kansas State University*
  - *Purdue University*
  - ICMC-USP
- **No contexto nacional ...**
  - Mais de 600 cursos de Computação/Informática no país
    - Poucos possuem uma disciplina especificamente voltada ao ensino dos conceitos relacionados à atividade de teste.
    - Subsídios ao Ensino Integrado de Teste de Software e de Fundamentos Básicos de Programação (ICMC-USP)

Direções para a Área de Teste

Direções para Teste

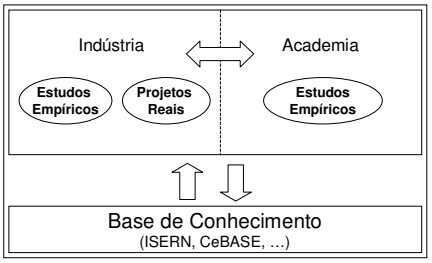
- **Contribuições**
  - Estudos Teóricos
    - Critérios Potenciais-Usos
    - *Pair-Wise Data Flow*
    - Critério Mutação de Interface
    - Análise de Mutantes × Sistemas Reativos
  - Desenvolvimento de Ferramentas
    - *Proteum*, *Proteum/IM* e *Proteum/RS*
    - *PokeTool*
    - *MGASET*
    - *JaBUTi* e *JaBUTi/MA*
    - *CATSDL*
  - Estudos Empíricos

Complementar

Direções para a Área de Teste

Direções para Teste

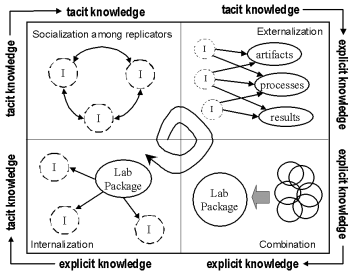
- **Necessidade de Cooperação Indústria-Academia**



Direções para a Área de Teste

Direções para Teste

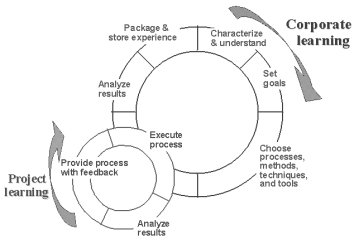
- ***Experimentation Knowledge Sharing Model***



Direções para a Área de Teste

Direções para Teste

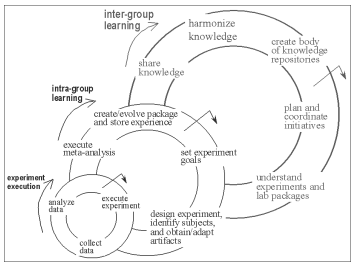
- ***QIP – Quality Improvement Paradigm***



Direções para a Área de Teste

Direções para Teste

- ***EIP – Experimentation Improvement Paradigm***



Especificação do Programa Fatorial

O programa *fatorial.c* calcula o fatorial de um número inteiro entre 0 e 12 (no ambiente UNIX).
Caso um número fornecido esteja fora desse intervalo, o programa produz uma mensagem de erro.

Exercício Fatorial

Complementar

Particionamento de Equivalência

- Identificar as classes de equivalências válidas e inválidas para o programa **fatorial.c.**

Coweb

Particionamento de Equivalência

- **Programa Fatorial: Classes Válidas e Inválidas**

Condições de Entrada	Classes Válidas	Classes Inválidas	
Número Inteiro	$0 \leq valor \leq 12$ (1)	$valor < 0$ (2)	$valor > 12$ (3)
...	...	...	

- **Conjunto de Casos de Teste**

T_0 = {(3,6), (-1, Erro), (30, Erro)}		
(1)	(2)	(3)

Implementação do Programa Fatorial

```
main () {
   int   valor, numero;

   printf("Entre com um numero inteiro: ");
   scanf("%d",&valor);

   numero = valor;

   if ((numero >= 0) && (numero <= 12))
      printf("O fatorial de %d e %d.\n", valor, fat(numero) );
   else
      printf("Erro: o numero deve estar entre [0-12]!\n");
}


int fat (n)
int n; {
   int fat;

   fat = 1;

   while (n >= 1) {
         fat = fat * n;
         n--;
   }
   return (fat);
}
```


Grafo de Programa

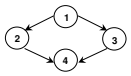
- Gerar o grafo de programa para o programa **fatorial.c**.
  - Função *main()*
  - Função *fat(n)*

Coweb

Grafo de Programa

```
main ()
/*  1 */        {
/*  1 */           int   valor, numero;
/*  1 */           printf("Entre com um numero inteiro: ");
/*  1 */           scanf("%d",&valor);
/*  1 */           numero = valor;
/*  1 */           if((numero >= 0) && (numero <= 12))
/*  2 */           {
/*  2 */              printf("O fatorial de %d e %d.\n", valor, fat(numero) );
/*  2 */           }
/*  3 */           else
/*  3 */           {
/*  3 */              printf("Erro: o numero deve estar entre [0-12]!\n");
/*  3 */           }
/*  4 */        }
```

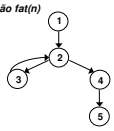
Função main()



Grafo de Programa

```
int fat (n)
int n;
/*  1 */        {
/*  1 */           int fat;
/*  1 */           fat = 1;
/*  2 */           while(n >= 1)
/*  3 */           {
/*  3 */              fat = fat * n;
/*  3 */              n--;
/*  3 */           }
/*  4 */           return (fat);
/*  5 */        }
```

Função fat(n)

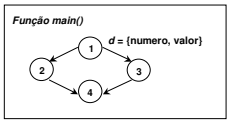


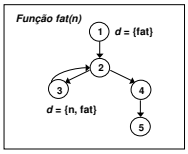
Grafo Def

- Gerar o Grafo Def para o programa **fatorial.c**.
  - Função *main()*
  - Função *fat(n)*

Coweb

Grafo-Def





Critério Todos-Potenciais-Usos

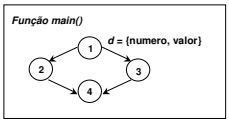
- Identificar algumas associações do critério Todos-Potenciais-Usos para o programa **fatorial.c**.
  - Função *main()*
  - Função *fat(n)*

Coweb

Critério Todos-Potenciais-Usos

- **Elementos Requeridos:**
  - Associações

<1,(1,3), {valor, numero}>
<1,(1,2), {valor, numero}>
. . .

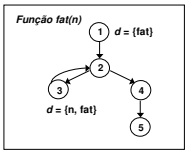


T_0 = {(3,6), (-1, Erro), (30, Erro)}

Critério Todos-Potenciais-Usos

- **Elementos Requeridos:**
  - Associações

<1,(2,4), {fat}>
<1,(2,3), {fat}>
<3,(2,4), {n, fat}>
<3,(2,3), {n, fat}>
. . .



T_0 = {(3,6), (-1, Erro), (30, Erro)}
$T_1 = T_0 \cup$ {(0,1)}

Critério Análise de Mutantes

- Gerar alguns mutantes para o programa **fatorial.c**.
  - Função *main()*
  - Função *fat(n)*

Coweb

Critério Análise de Mutantes

$T_0 = \{(3,6), (-1, Erro), (30, Erro)\}$ $T_1 = T_0 \cup \{(0,1)\}$

```
main()
                    ...
/*  1 */            numero = valor;
/*  1 */            if((numero >= 0) && (numero <= 12))
                    ...
```

P: {(3,6)}
M: {(3,1)}

```
main()
                    ...
/*  1 */            numero = 0;
/*  1 */            if((numero >= 0) && (numero <= 12))
                    ...
```

P: {(3,6)}
M: {(3,2)}

```
main()
                    ...
/*  1 */            numero = valor-1;
/*  1 */            if((numero >= 0) && (numero <= 12))
                    ...
```

Solução

Critério Análise de Mutantes

```
main()
                    ...
/*  1 */            numero = valor;
/*  1 */            if((numero >= 0) && (numero <= 12))
                    ...
```

Mutante Equivalente

```
main()
                    ...
/*  1 */            numero = valor;
/*  1 */            if((valor >= 0) && (numero <= 12))
                    ...
```

P: {(0,1)}
M: {(0,Erro)}

```
main()
                    ...
/*  1 */            numero = valor;
/*  1 */            if((numero > 0) && (numero <= 12))
                    ...
```

P: {(-1,Erro)}
M: {(-1,1)}

```
main()
                    ...
/*  1 */            numero = valor;
/*  1 */            if((numero >= 0) || (numero <= 12))
                    ...
```

Solução

Critério Análise de Mutantes

```
int fat (n)
                    ...
/*  1 */            fat = 1;
/*  2 */            while(n >= 1)
/*  3 */            {
/*  3 */               fat = fat * n;
/*  3 */               n--;
/*  3 */            }
                    ...
```

Mutante Equivalente

```
int fat (n)
                    ...
/*  1 */            fat = 1;
/*  2 */            while(n > 1)
/*  3 */            {
/*  3 */               fat = fat * n;
/*  3 */               n--;
/*  3 */            }
                    ...
```

```
int fat (n)
                    ...
/*  1 */            fat = n;
/*  2 */            while(n >= 1)
/*  3 */            {
/*  3 */               fat = fat * n;
/*  3 */               n--;
/*  3 */            }
                    ...
```

P: {(3,6)} M: {(3,18)}

```
int fat (n)
                    ...
/*  1 */            fat = 1;
/*  2 */            while(n >= 1)
/*  3 */            {
/*  3 */               fat = fat + n;
/*  3 */               n--;
/*  3 */            }
                    ...
```

P: {(3,6)} M: {(3,7)}

Solução

CoWeb

- ***Collaborative WebSite***
  - Ferramenta de edição colaborativa baseada na web *(Georgia Tech)*
  - Principais Usos
    - Distribuição da Informação
    - Criação colaborativa de artefatos (projetos, seminários, artigos, experimentos, ...)
    - Discussão e revisão

Coweb

Complementar

CoWeb

Coweb

Complementar