

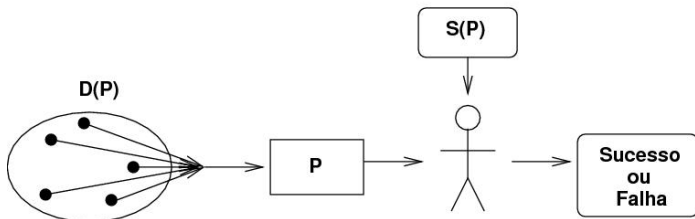
SSC721 – Teste e Inspeção de Software

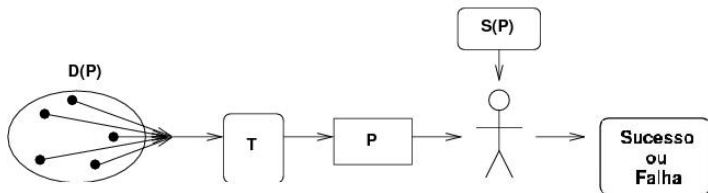
Técnica de Teste Funcional

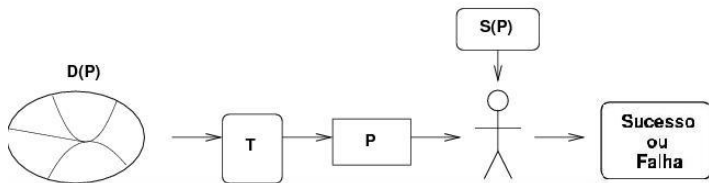
Profa. Ellen Francine Barbosa
francine@icmc.usp.br

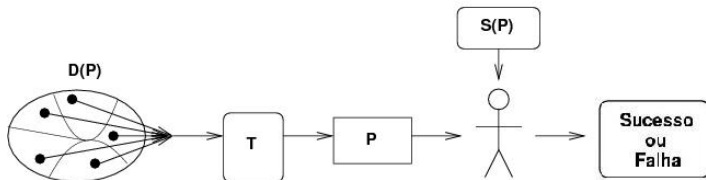
Instituto de Ciências Matemáticas e de Computação — ICMC/USP

- Técnica Funcional
- Particionamento em Classes de Equivalência
- Análise do Valor Limite
- Conclusão











Aula Anterior...

SSC721 – Teste e
Inspeção de Software

Aula Anterior

Técnica Funcional

Particionamento em
Classes de Equivalência

Análise do Valor Limite

Conclusão

Como computar domínios?

- Diferentes **técnicas** e **critérios** de teste existem para auxiliar na atividade de teste.
 - Basicamente, os testes podem ser classificados em **teste caixa-preta** (teste funcional) ou **teste caixa-branca** (teste estrutural).
 - Contemplam diferentes perspectivas do software: **aspecto complementar!!!!**

- Também conhecida como **Técnica Caixa-Preta**
 - Considera o produto em teste como uma caixa da qual só se conhece a entrada e a saída (sem conhecimento da parte interna).
- Baseia-se na **especificação do software** para derivar os requisitos de teste.
 - Aborda o software de um ponto de vista macroscópico.
 - Não se preocupa com detalhes de implementação.



- Conjunto de teste é **pequeno**.
- Número de subdomínios é pequeno.
- Uma vez definidos os subdomínios, pode-se assumir, com alguma segurança, que **qualquer elemento da classe** pode ser considerado.
- Se um elemento detectar um defeito, qualquer outro também detecta; se não detectar, os outros também não detectam.
- Particionar o **domínio de saída** também é válido.

- Década de 70.
- Métodos para especificar sistemas como Análise e Projeto Estruturados.
- Eram mencionados, embora não diretamente, aspectos de validação do sistema com relação à **satisfação dos seus requisitos funcionais**.

- Diversos critérios funcionais são definidos.
- Critérios mais conhecidos:
 - **Particionamento em Classes 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.

- Passos básicos para aplicar um critério de teste funcional:
 - A **especificação de requisitos** é analisada.
 - Entradas **válidas** são escolhidas para determinar se o produto em teste comporta-se corretamente.
 - Entradas **inválidas** são escolhidas para verificar se estas são detectadas e manipuladas adequadamente.
 - Os **casos de testes** são construídos (saídas são determinadas para cada entrada).
 - O conjunto de teste é executado e as saídas obtidas são comparadas com as saídas esperadas.
 - Um **relatório** é gerado para avaliar o resultado dos testes.

- Por ser independente da implementação, critérios da técnica funcional podem ser utilizados em todas as **fases** de teste.
- A complexidade de aplicação aumenta em cada fase.



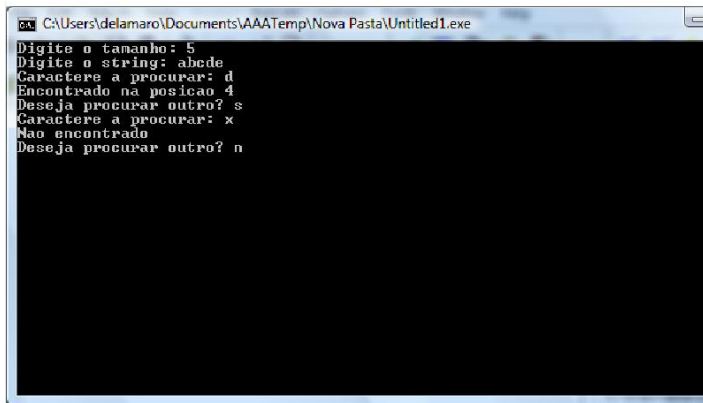
- Independente do **paradigma de programação** utilizado.
- Eficaz em detectar determinados **tipos de defeitos**.
 - Funcionalidade ausente, por exemplo.

- Dependente de uma **boa especificação** o que, em geral, não é bem feito.
- Não é possível garantir que partes **essenciais** ou **críticas** do software sejam executadas.
- Ruim quando se tem entradas simples mas **processamento complexo**.

- Critério utilizado para reduzir o número de casos de teste, procurando garantir uma **boa cobertura** do código do produto em teste.
- Empregado **intuitivamente** pelos programadores mesmo sem conhecer o critério.

- Qualquer valor **dentro do intervalo** tem a mesma importância, ou seja, qualquer valor escolhido é adequado.
 - O mesmo se aplica para os demais intervalos de dados.
- Tais intervalos determinam o que é chamado de **Classes de Equivalência**.
- Qualquer valor no intervalo de uma classe é considerado **equivalente** em termos de teste.
 - Se um caso de teste de uma classe de equivalência revela um erro, qualquer caso de teste da mesma classe também revelaria e vice-versa.

O programa solicita do usuário um inteiro positivo no intervalo entre 1 e 20 e, então, lê uma cadeia de caracteres desse comprimento. Após isso, o programa solicita um caracter e retorna a posição na cadeia em que o caracter é encontrado pela primeira vez ou uma mensagem indicando que o caracter não está presente na cadeia. O usuário tem a opção de procurar por vários caracteres.



```
C:\Users\delamaro\Documents\AAATemp\Nova Pasta\Untitled1.exe
Digite o tamanho: 5
Digite o string: abcde
Caractere a procurar: d
Encontrado na posicao 4
Deseja procurar outro? s
Caractere a procurar: x
Nao encontrado
Deseja procurar outro? n
```

- Duas etapas:
 - Identificar classes de equivalência.
 - Definir os casos de teste para cobrir essas classes.

- Identificar classes de equivalência:
 - Identificar **condições de entrada** relevantes.
 - **Particionar** cada condição em dois ou mais grupos.
 - Para ajudar na identificação das partições, pode-se observar a especificação procurando termos como “intervalo” e “conjunto” ou palavras similares que indiquem que os dados são processados da mesma forma.

- Classes de Equivalência

Condição de entrada	Classes de equivalência válidas	Classes de equivalência inválidas

- Identificando as entradas...
 - O tamanho da cadeia (**T**)

- Identificando as entradas...
 - O tamanho da cadeia (**T**)
 - A cadeia de caracteres (**CC**)

- Identificando as entradas...
 - O tamanho da cadeia (**T**)
 - A cadeia de caracteres (**CC**)
 - O caractere a ser procurado (**C**)

- Identificando as entradas...
 - O tamanho da cadeia (**T**)
 - A cadeia de caracteres (**CC**)
 - O caractere a ser procurado (**C**)
 - A opção por procurar mais caracteres (**O**)

Identificando as classes – Guidelines

Uma condição de entrada estabelece um intervalo de valores.

- *O número de itens pode variar de 1 a 999.*

Identificando as classes – Guidelines

Uma condição de entrada estabelece um intervalo de valores.

- *O número de itens pode variar de 1 a 999.*
- **Uma classe válida:**

Identificando as classes – Guidelines

Uma condição de entrada estabelece um intervalo de valores.

- *O número de itens pode variar de 1 a 999.*
- Uma classe **válida**:
 - $1 \leq \text{número de itens} \leq 999$

Identificando as classes – Guidelines

Uma condição de entrada estabelece um intervalo de valores.

- *O número de itens pode variar de 1 a 999.*
- Uma classe **válida**:
 - $1 \leq \text{número de itens} \leq 999$
- **Doas classes inválidas:**

Identificando as classes – Guidelines

Uma condição de entrada estabelece um intervalo de valores.

- *O número de itens pode variar de 1 a 999.*
- Uma classe **válida**:
 - $1 \leq \text{número de itens} \leq 999$
- Duas classes **inválidas**:
 - $1 > \text{número de itens}$

Identificando as classes – Guidelines

Uma condição de entrada estabelece um intervalo de valores.

- *O número de itens pode variar de 1 a 999.*
- Uma classe **válida**:
 - $1 \leq \text{número de itens} \leq 999$
- Duas classes **inválidas**:
 - $1 > \text{número de itens}$
 - $999 < \text{número de itens}$

Identificando as classes – Guidelines

A condição de entrada estabelece uma quantidade de valores.

- *Para um automóvel, de 1 a 6 proprietários podem ser relacionados.*

Identificando as classes – Guidelines

A condição de entrada estabelece uma quantidade de valores.

- *Para um automóvel, de 1 a 6 proprietários podem ser relacionados.*
- Uma classe válida:

Identificando as classes – Guidelines

A condição de entrada estabelece uma quantidade de valores.

- *Para um automóvel, de 1 a 6 proprietários podem ser relacionados.*
- Uma classe **válida**:
 - De 1 a 6 proprietários.

Identificando as classes – Guidelines

A condição de entrada estabelece uma quantidade de valores.

- *Para um automóvel, de 1 a 6 proprietários podem ser relacionados.*
- Uma classe **válida**:
 - De 1 a 6 proprietários.
- **Dois classes inválidas:**

Identificando as classes – Guidelines

A condição de entrada estabelece uma quantidade de valores.

- *Para um automóvel, de 1 a 6 proprietários podem ser relacionados.*
- Uma classe **válida**:
 - De 1 a 6 proprietários.
- Duas classes **inválidas**:
 - Nenhum proprietário.

Identificando as classes – Guidelines

A condição de entrada estabelece uma quantidade de valores.

- *Para um automóvel, de 1 a 6 proprietários podem ser relacionados.*
- Uma classe **válida**:
 - De 1 a 6 proprietários.
- Duas classes **inválidas**:
 - Nenhum proprietário.
 - **Mais do que 6 proprietários.**

Identificando as classes – Guidelines

A condição de entrada especifica um conjunto de valores que (acredita-se) devem ser tratados de maneiras diversas.

- *Tipo de veículo deve ser: ônibus, caminhão, automóvel ou motocicleta.*

Identificando as classes – Guidelines

A condição de entrada especifica um conjunto de valores que (acredita-se) devem ser tratados de maneiras diversas.

- *Tipo de veículo deve ser: ônibus, caminhão, automóvel ou motocicleta.*
- Uma classe válida para cada valor.

Identificando as classes – Guidelines

A condição de entrada especifica um conjunto de valores que (acredita-se) devem ser tratados de maneiras diversas.

- *Tipo de veículo deve ser: ônibus, caminhão, automóvel ou motocicleta.*
- Uma classe **válida** para cada valor.
- Uma classe **inválida**:

Identificando as classes – Guidelines

A condição de entrada especifica um conjunto de valores que (acredita-se) devem ser tratados de maneiras diversas.

- *Tipo de veículo deve ser: ônibus, caminhão, automóvel ou motocicleta.*
- Uma classe **válida** para cada valor.
- Uma classe **inválida**:
 - **Trailer.**

Identificando as classes – Guidelines

Condição de entrada determina uma condição do tipo
“tem que ser”.

- *Primeiro caractere do identificador tem que ser uma letra.*

Identificando as classes – Guidelines

Condição de entrada determina uma condição do tipo “tem que ser”.

- *Primeiro caractere do identificador tem que ser uma letra.*
- Uma classe válida:

Identificando as classes – Guidelines

Condição de entrada determina uma condição do tipo “tem que ser”.

- *Primeiro caractere do identificador tem que ser uma letra.*
- Uma classe **válida**:
 - **Primeiro caractere é uma letra.**

Identificando as classes – Guidelines

Condição de entrada determina uma condição do tipo “tem que ser”.

- *Primeiro caractere do identificador tem que ser uma letra.*
- Uma classe **válida**:
 - Primeiro caractere é uma letra.
- Uma classe **inválida**:

Identificando as classes – Guidelines

Condição de entrada determina uma condição do tipo “tem que ser”.

- *Primeiro caractere do identificador tem que ser uma letra.*
- Uma classe **válida**:
 - Primeiro caractere é uma letra.
- Uma classe **inválida**:
 - Primeiro caractere é um dígito.

Identificando as classes – Guidelines

Se existe alguma razão para acreditar que o programa não trata elementos de uma classe de maneira uniforme, a classe deve ser dividida em classes menores.



Cadeia: T

SSC721 – Teste e
Inspeção de Software

Aula Anterior

Técnica Funcional

Particionamento em
Classes de Equivalência

Exemplo

Aplicabilidade e Limitações

Análise do Valor Limite

Conclusão

- Tamanho da cadeia de caracteres.

- Tamanho da cadeia de caracteres.
- Enquadra-se na primeira guideline: valores num intervalo.

- Tamanho da cadeia de caracteres.
- Enquadra-se na primeira guideline: valores num intervalo.
- Uma classe válida: de 1 a 20.

- Tamanho da cadeia de caracteres.
- Enquadra-se na primeira guideline: valores num intervalo.
- Uma classe válida: de 1 a 20.
- Duas classes inválidas: menor que 1, maior que 20.



Cadeia: CC

SSC721 – Teste e
Inspeção de Software

Aula Anterior

Técnica Funcional

Particionamento em
Classes de Equivalência

Exemplo

Aplicabilidade e Limitações

Análise do Valor Limite

Conclusão

- Valor da cadeia de caracteres.

- Valor da cadeia de caracteres.
- O valor em si não determina comportamentos diferentes do programa, apenas o seu tamanho.

- Valor da cadeia de caracteres.
- O valor em si não determina comportamentos diferentes do programa, apenas o seu tamanho.
 - **abcde**

- Valor da cadeia de caracteres.
- O valor em si não determina comportamentos diferentes do programa, apenas o seu tamanho.
 - abcde
 - 8y4e*

- Valor da cadeia de caracteres.
- O valor em si não determina comportamentos diferentes do programa, apenas o seu tamanho.
 - abcde
 - 8y4e*
- Assim, não é necessário usar essa variável como condição de entrada.



Cadeia: 0

SSC721 – Teste e
Inspeção de Software

Aula Anterior

Técnica Funcional

Particionamento em
Classes de Equivalência

Exemplo

Aplicabilidade e Limitações

Análise do Valor Limite

Conclusão

- Opção de continuar ou não.

- Opção de continuar ou não.
- Enquadra-se na terceira guideline: conjunto de valores possíveis.

- Opção de continuar ou não.
- Enquadra-se na terceira guideline: conjunto de valores possíveis.
- Duas classes válidas: **s** ou **n**.

- Opção de continuar ou não.
- Enquadra-se na terceira guideline: conjunto de valores possíveis.
- Duas classes válidas: **s** ou **n**.
- Uma classe inválida: **w**.



Cadeia: C

SSC721 – Teste e
Inspeção de Software

Aula Anterior

Técnica Funcional

Particionamento em
Classes de Equivalência

Exemplo

Aplicabilidade e Limitações

Análise do Valor Limite

Conclusão

- Caractere a ser procurado.

- Caractere a ser procurado.
- Enquadra-se também na terceira guideline.

- Caractere a ser procurado.
- Enquadra-se também na terceira guideline.
- Duas classes válidas: C pertence à string e C não pertence à string.

- Caractere a ser procurado.
- Enquadra-se também na terceira guideline.
- Duas classes válidas: C pertence à string e C não pertence à string.
- Nenhuma classe inválida.

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$	$T < 1$ e $T > 20$
O	S N	Outro
C	Pertence Não pertence	

- Definir casos de teste:
 - Próximo passo é usar as classes para definir os casos de teste necessários.
 - Regras a seguir:
 - Atribuir um número para cada classe.

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$ (1)	$T < 1$ (2) e $T > 20$ (3)
O	S (4) N (5)	Outro (6)
C	Pertence (7) Não pertence (8)	

- Definir casos de teste:
 - Próximo passo é usar as classes para definir os casos de teste necessários.
 - Regras a seguir:
 - Definir casos de teste para cobrir o maior número de **classes válidas** possíveis, até que todas as classes válidas sejam cobertas.
 - Para cada **classe inválida**, projetar um caso de teste específico.

- Definir casos de teste:
 - Entradas inválidas podem **mascarar** ou **extrapolar** outras entradas inválidas.
 - Por exemplo: **T** = 34 e **O** = w provavelmente não irá exercitar a classe inválida para a condição **O**.

• $T = 3$ $CC = abc$ $C = c$ $O = s$ $C = k$ $O = n$

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$ (1)	$T < 1$ (2) e $T > 20$ (3)
O	S (4) N (5)	Outro (6)
C	Pertence (7) Não pertence (8)	

- $T = 3$ $CC = abc$ $C = c$ $O = s$ $C = k$ $O = n$
- Saída: Encontrado na posição 3; Não encontrado.

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$ (1)	$T < 1$ (2) e $T > 20$ (3)
O	S (4) N (5)	Outro (6)
C	Pertence (7) Não pertence (8)	

- **T = 3 CC = abc C = c O = s C = k O = n**
- Saída: Encontrado na posição 3; Não encontrado.
- **Classes cobertas: 1, 4, 5, 7, 8 (todas as válidas).**

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$ (1)	$T < 1$ (2) e $T > 20$ (3)
O	S (4) N (5)	Outro (6)
C	Pertence (7) Não pertence (8)	

• $T = -3$

- $T = -3$
- Saída: Tamanho inválido

- $T = -3$
- Saída: Tamanho inválido
- Classe coberta: 2

- $T = -3$
- Saída: Tamanho inválido
- Classe coberta: 2
- $T = 34$

- $T = -3$
- Saída: Tamanho inválido
- Classe coberta: 2
- $T = 34$
- Saída: Tamanho inválido

- $T = -3$
 - Saída: Tamanho inválido
 - Classe coberta: 2
-
- $T = 34$
 - Saída: Tamanho inválido
 - Classe coberta: 3

- $T = -3$
- Saída: Tamanho inválido
- Classe coberta: 2
- $T = 34$
- Saída: Tamanho inválido
- Classe coberta: 3
- $T = 3$ $CC = abc$ $C = c$ $O = w$

- $T = -3$
- Saída: Tamanho inválido
- Classe coberta: 2
- $T = 34$
- Saída: Tamanho inválido
- Classe coberta: 3
- $T = 3$ $CC = abc$ $C = c$ $O = w$
- Saída: Responda com s ou n.

- $T = -3$
- Saída: Tamanho inválido
- Classe coberta: 2
- $T = 34$
- Saída: Tamanho inválido
- Classe coberta: 3
- $T = 3$ $CC = abc$ $C = c$ $O = w$
- Saída: Responda com s ou n.
- Classe coberta: 6

- Poderíamos pensar numa condição de entrada como:
Número de vezes que um caractere é procurado no string.
- Nesse caso, como ficaria a tabela de classes?
Modifique-a.
- Defina os casos de teste para cobrir essas novas classes.



- Reduz significativamente o **número de casos de teste** em relação ao teste exaustivo.
- Mais adequado para o teste de produtos com domínios de entrada divididos em **intervalos** ou conjuntos.
- Assume que os valores dentro da mesma classe são **equivalentes** (isso nem sempre é verdade!).
 - Importante empregar outros critérios de teste!!
- Aplicável em todas as fases de teste: unidade, integração e sistema.

- Exemplo: Parte de um Sistema de Recursos Humanos que determina contratações com base na idade dos candidatos.

0 – 16	Não empregar.
16 – 18	Pode ser empregado tempo parcial.
18 – 55	Pode ser empregado tempo integral.
55 – 99	Não empregar.

- Como derivar os **casos de teste**?

- Considere que o módulo que resolve o problema anterior tenha sido implementado como se segue:

```
1  if (idade == 0)  empregar = "NAO";
2  if (idade == 1)  empregar = "NAO";
3  ...
4  if (idade == 15) empregar = "NAO";
5  if (idade == 16) empregar = "PAR";
6  if (idade == 17) empregar = "PAR";
7  if (idade == 18) empregar = "INT";
8  if (idade == 19) empregar = "INT";
9  ...
10 if (idade == 53) empregar = "INT";
11 if (idade == 54) empregar = "INT";
12 if (idade == 55) empregar = "NAO";
13 if (idade == 56) empregar = "NAO";
14 ...
15 if (idade == 98) empregar = "NAO";
16 if (idade == 99) empregar = "NAO";
```

- Neste caso, a única forma de testá-lo adequadamente seria executar o módulo com valores de idade de **0..99**.
- Caso haja tempo suficiente, esse é o melhor teste a ser realizado!
- O problema é que da forma como o código anterior foi implementado, a execução de um dado caso de teste não diz nada a respeito da execução do próximo.

Considere agora uma outra implementação (bem melhor!!) do mesmo problema:

```
1  if (idade >= 0 && idade <= 16)
2      empregar = "NAO";
3  if (idade >= 16 && idade <= 18)
4      empregar = "PAR";
5  if (idade >= 18 && idade <= 55)
6      empregar = "INT";
7  if (idade >= 55 && idade <= 99)
8      empregar = "NAO";
```

- Dada essa implementação, fica claro que não é necessário testar para todos os valores 0, 1, 2, ..., 14, 15 e 16, por exemplo.
- Apenas um **conjunto de valores** precisa ser testado.
 - Quais seriam esses valores?

- Complementa o critério Particionamento de Equivalência, exigindo casos de teste nos **limites (fronteiras)** de cada classe de equivalência.

- É mais proveitoso explorar **condições limites**.
 - Aquelas que estão sobre, acima e abaixo dos limites das classes de equivalência.
- Assim, em vez de selecionar um caso de teste qualquer, deve-se tomar um ou mais casos de teste de modo que cada **limitante** da classe seja testado.
- Além disso, deve-se considerar também o **domínio de saída** para derivar casos de teste (classes de equivalência de saída).

Guidelines

- Se uma condição de entrada define um intervalo de valores: casos de teste nos limites do intervalo e logo além dos limites (casos inválidos).
 - *Se o intervalo for -1.00 a 1.00 , devemos tomar -1.00 , 1.00 , -1.01 e 1.01 .*
- Se especifica uma quantidade de valores, escolher casos de teste nos limites, uma unidade acima e uma abaixo.
 - *Se um arquivo de entrada deve conter de 1 a 255 registros, casos de teste devem contemplar 0, 1, 255 e 256 registros.*

Guidelines

- Aplicar a 1a. regra para o domínio de saída.
- *Por exemplo, um programa deve calcular a dedução que uma empresa tem sobre um determinado imposto.*
- *Diz a especificação que essa dedução é de, no mínimo, \$0,00 e no máximo \$1.165,25.*
 - Aplicar entrada que cause \$0,00 de dedução e outra que cause \$1.165,25.
 - Tente achar casos de teste que causem dedução negativa ou maior que \$1.165,25.
- Nem sempre é possível achar casos de teste além dos limites.

Guidelines

- Aplicar 2a. regra para o domínio de saída.
- *Por exemplo, um sistema de recuperação de informação mostra no máximo 4 abstracts para cada consulta.*
 - Achar casos de teste que façam com que 0, 1 e 4 abstracts sejam mostrados.
 - Tentar caso de teste que erroneamente mostre 5 registros.

Guidelines

- Se a entrada ou a saída for um conjunto ordenado (um arquivo sequencial ou uma lista ligada), dar atenção ao primeiro e último elemento.
- Use a sua criatividade para definir outras condições limites.

Particionamento em Classes de Equivalência

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$ (1)	$T < 1$ (2) e $T > 20$ (3)
O	S (4) N (5)	Outro (6)
C	Pertence (7) Não pertence (8)	

Análise do Valor Limite

Variável de E/S	Classes de equivalência válidas	Classes de equivalência inválidas
T	$1 \leq T \leq 20$ (1)	$T < 1$ (2) e $T > 20$ (3)
O	S (4) N (5)	Outro (6)
C	Pertence (7) Não pertence (8)	
Posição	$1 \leq pos \leq 20$ (9)	$pos < 1$ (10) $pos > 20$ (11)

• $T = 1 \text{ CC} = a \text{ C} = a \text{ O} = s \text{ C} = k \text{ O} = n$

- $T = 1 \text{ CC} = a \text{ C} = a \text{ O} = s \text{ C} = k \text{ O} = n$
- Saída: Encontrado na posição 1; Não encontrado.

- $T = 1$ $CC = a$ $C = a$ $O = s$ $C = k$ $O = n$
- Saída: Encontrado na posição 1; Não encontrado.
- $T = 20$ $CC = abcdefghijklmnopqrst$ $C = t$ $O = n$

- $T = 1 \quad CC = a \quad C = a \quad O = s \quad C = k \quad O = n$
- Saída: Encontrado na posição 1; Não encontrado.
- $T = 20 \quad CC = \text{abcdefghijklmnopqrst} \quad C = t \quad O = n$
- Saída: Encontrado na posição 20.

- $T = 1 \quad CC = a \quad C = a \quad O = s \quad C = k \quad O = n$
- Saída: Encontrado na posição 1; Não encontrado.
- $T = 20 \quad CC = \text{abcdefghijklmnopqrst} \quad C = t \quad O = n$
- Saída: Encontrado na posição 20.
- $T = 0$

- $T = 1$ $CC = a$ $C = a$ $O = s$ $C = k$ $O = n$
- Saída: Encontrado na posição 1; Não encontrado.
- $T = 20$ $CC = abcdefghijklmnopqrst$ $C = t$ $O = n$
- Saída: Encontrado na posição 20.
- $T = 0$
- Saída: Tamanho inválido.

• $T = 21$

- $T = 21$
- Saída: Tamanho inválido.

- $T = 21$
- Saída: Tamanho inválido.
- $T = 3$ $CC = abc$ $C = c$ $O = w$

- $T = 21$
- Saída: Tamanho inválido.
- $T = 3$ $CC = abc$ $C = c$ $O = w$
- Saída: Responda com s ou n.

- Considerando o exemplo utilizado anteriormente:

0 – 16	Não empregar.
16 – 18	Pode ser empregado tempo parcial.
18 – 55	Pode ser empregado tempo integral.
55 – 99	Não empregar.

- Observe que os limites aparecem em duas classes de equivalência (16 por exemplo).

- As condições anteriores, na verdade, deveriam ser escritas como:

$0 \leq idade < 16$	Não empregar.
$16 \leq idade < 18$	Pode ser empregado tempo parcial.
$18 \leq idade < 55$	Pode ser empregado tempo integral.
$55 \leq idade < 99$	Não empregar.

ou

$0 \leq idade \leq 15$	Não empregar.
$16 \leq idade \leq 17$	Pode ser empregado tempo parcial.
$18 \leq idade \leq 54$	Pode ser empregado tempo integral.
$55 \leq idade \leq 99$	Não empregar.

- Na primeira regra, 16 não deve ser incluído.
- Na segunda, 16 pode ser empregado em tempo parcial.

A implementação a seguir implementa as regras anteriores:

```
1 if (idade >= 0 && idade < 16)
2     empregar = "NAO";
3 if (idade >= 16 && idade < 18)
4     empregar = "PAR";
5 if (idade >= 18 && idade < 55)
6     empregar = "INT";
7 if (idade >= 55 && idade < 99)
8     empregar = "NAO";
```

- Valores limites a serem considerados: $\{-1, 0\}$, $\{15, 16, 17, 18, 19\}$, $\{54, 55, 56\}$ e $\{99, 100\}$

- Reduz significativamente o **número de casos de teste** em relação ao teste exaustivo.
- Mais adequado para o teste de produtos com domínios de entrada divididos em **intervalos** ou conjuntos.
- Aplicável em todas as fases de teste: unidade, integração e sistema.

- A técnica funcional pode ser utilizada em **todas** as fases de teste.
- Independe do paradigma de programação utilizado.
- Eficaz em detectar determinados tipos de erros.
 - Por exemplo: Funcionalidade ausente.
- Dependente de uma **boa** especificação de requisitos.
 - Especificações descritivas e não formais.
 - Requisitos imprecisos e informais.
- Dificuldade em **quantificar** a atividade de teste.
- Não é possível garantir que partes essenciais ou críticas do software sejam executadas.
- Dificuldade de **automação**: em geral, a aplicação é manual.

SSC721 – Teste e
Inspeção de Software

Aula Anterior

Técnica Funcional

Particionamento em
Classes de Equivalência

Análise do Valor Limite

Conclusão

