Testes Funcionais de Software para Empresas Ágeis

curso gratuito

técnicas de teste funcional

PROF. THIAGO DELGADO PINTO

thiago_dp (at) yahoo (dot) com (dot) br

versão: 2018.05.25

objetivos

revisar **técnicas básicas** para melhorar os testes produzidos

facilitar a compreensão de *algumas* técnicas usadas por Concordia

http://concordialang.org

→ voltado para testes **funcionais**

não cobre técnicas de teste exploratório ou de sistema, por exemplo

técnicas básicas

aleatório

teste aleatório

seleciona entradas aleatórias, de acordo com o domínio

- *exemplo*: no domínio entre 1 e 100, escolher valores como 17, 63 e 82

entradas podem diferir daquelas usadas por testadores

- exercitando novos caminhos
- e com isso revelar defeitos

uso recomendado junto a outras práticas

adivinhação

adivinhação

geralmente baseada em **experiência prévia** do testador com **domínio**

ou quando há alguma **indicação** de **problemas** em **potencial**

exemplos

- divisão por zero
- caracteres problemáticos
- extrapolação de limites de string ou array

se possível, deve ser documentada para uso futuro

particionamento em classes de equivalência

particionamento em classes de equivalência

cria **partições** no domínio de entrada ou de saída

as chamadas *classes de equivalência*

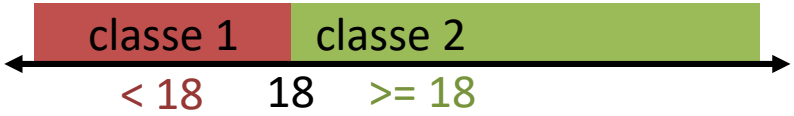
a **equivalência** se refere ao fato de a técnica assumir que todos os membros de certa classe são **processados** pelo sistema da **mesma forma**

sendo, portanto, equivalentes para o sistema

testar um **membro** da classe **equivale** a testar **qualquer outro dela**

particionamento em classes de equivalência

exemplo: "usuário deve ter 18 anos ou mais"



se um **membro** da classe é capaz de **revelar** certo **defeito**, **qualquer outro dela** é **capaz de relevar** o **mesmo defeito**

idem se **não for capaz** de revelar

- a não ser que haja subclasses
- discussão em Beizer (1990)*

*Beizer, B. **Software Testing Techniques**, second edition, Van Nostrand Reinhold, New York, 1990.

pce – vantagens

elimina a necessidade de **teste exaustivo**, que não é viável

guia o testador a **selecionar** um **subconjunto de entradas** de teste com **alta probabilidade** de encontrar um **defeito**

permite ao testador **cobrir** um **largo domínio** de entrada/saída com um **pequeno subconjunto** da classe de equivalência escolhida

pce – escolha das classes

Myers (2004) propõe escolher **condições de entrada "interessantes"**

vêm de uma **descrição** na **especificação do software** a ser testado

testador e **analista** interagem para **desenvolver**...
- conjunto **verificável** de requisitos
- especificação (completa e correta) de **entradas** e **saídas**

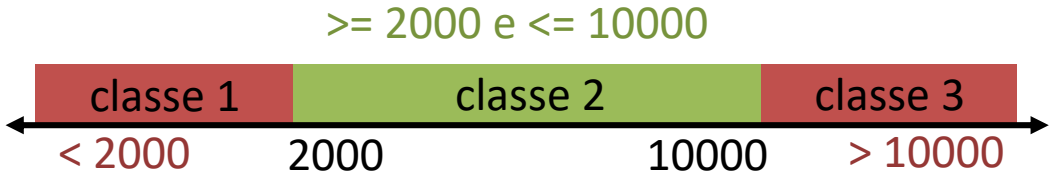
testador identifica **condições** e cria **classes**

como no exemplo anterior → "usuário deve ter 18 anos ou mais" → < 18 e >= 18

daí desenvolve **Casos de Teste** para **cobrir classes** identificadas

faixa de valores → uma *classe* para válidos e duas para inválidos

exemplo: "salário deve estar entre 2.000 e 10.000 reais"



classe 1 – inválidos: qualquer < 2000

classe 2 – válidos: qualquer >= 2000 e <= 10000

classe 3 – inválidos: qualquer > 10000

conjunto de valores → uma *classe* para válidos e uma para inválidos

exemplo: "nível de acesso deve ser Administrador, Gerente, Vendedor ou Estoquista"

*classe 1 - **válidos**: "Administrador", "Gerente", "Vendedor", "Estoquista"*

*classe 2 - **inválidos**: qualquer outro, como "Visitante", "XjNq6", ou ""*

condição especial → uma *classe* para válidos e uma para inválidos

exemplo: "código de barras deve começar com 789"

*classe 1 – **válido** – exemplos: "7891234567890", "789", ...*

*classe 2 – **inválidos** – exemplos: "6781234567890", "z78", "", ...*

exemplo: "código de barras deve começar com 789 e ter 13 dígitos numéricos"

tem **2 condições** – porém **válidos** deve respeitar **ambas**

*classe 1 – **válido -** exemplos: "7891234567890", "7890000000000", ...*

*classe 2 – **inválidos com 13 dígitos numéricos** – exemplos: "6781234567890", "0000000000000", ...*

*classe 3 – **inválidos que começam com 789** – exemplos: "789", "789@zw$", ...*

*classe 4 – **inválidos que não começam com 789 nem têm 13 dígitos numéricos** – exemplos: "", "@zw$", ...*

subcasos → quando certos valores funcionam de modo diferente

não seguem o mesmo padrão dos demais

exemplo: "CPF deve ter 11 dígitos numéricos e ser válido segundo o cálculo ..."

porém, há casos conhecidos que fogem a regra, como ter todos os números repetidos (ex.: "00000000000", "11111111111", ..., "99999999999")

subcaso deve ser tratado com nova(a) classe(s)

ex.:

classe 1 – válidos – exemplos: "35005215093", "95411615020", ...
classe 2 – inválidos que passam no cálculo – exemplos: "00000000000", ...
classe 3 – inválidos que não passam no cálculo – exemplos: "35005215095", ...
classe 4 – inválidos que não passam no formato – exemplos: "U4_%#", ...

análise de valor limite

análise de valor limite

sugere que o testador selecione **valores próximos aos limites**

por muitos **defeitos** ocorrerem **nos limites**, ou **acima**, ou **abaixo**

complementa a técnica de particionamento em **classes de equiv**.

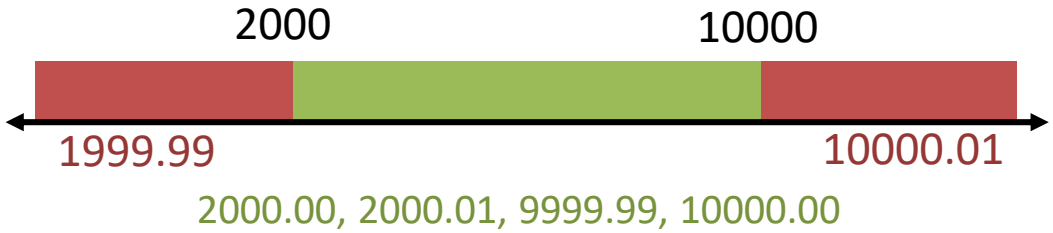
pode atuar no limite das classes

exemplo: "usuário deve ter 18 anos ou mais"

17 (abaixo), 18 (limite) e 19 (acima) podem ser incluídos no teste

faixa de valores → válidos para os limítrofes e inválidos para abaixo e acima

exemplo: "salário deve estar entre 2.000 e 10.000 reais"



válidos: `2000.00, 2000.01, 9999.99 e 10000.00`

inválidos: `1999.99 e 10000.01`

conjunto de valores → foco no primeiro e último

exemplo: "nível de acesso deve ser Administrador, Gerente, Vendedor ou Estoquista"

válidos *– exemplos: "Administrador", "Estoquista"*

inválidos *– exemplos: "Visitante", "XjNq6", ou ""*

combinando técnicas

pce + avl + aleatório – exemplo

"salário deve estar entre 2.000 e 10.000 reais" → faixa de valores

pce + aleatório:

- caso 1 – pce classe 1 – inválido → < 2000.00 → aleatório → 653.11
- caso 2 – pce classe 2 – válido → >= 2000.00 e <= 10000.00 → aleatório → 7885.90
- caso 3 – pce classe 3 – inválido → > 10000.00 → aleatório → 224.020.15

avl:

- caso 1 – inválido – 1999.99
- caso 2 – válido – 2000.00
- caso 3 – válido – 2000.01
- caso 4 – válido – 9999.99
- caso 5 – válido – 10000.00
- caso 6 – inválido – 10000.01

pce + avl + aleatório – exemplo

"*salário deve estar entre 2.000 e 10.000 reais*" → faixa de valores

caso	origem	expectativa	valor
1	pce classe 1 (< 2000) + aleatório	inválido	653.11
2	pce classe 2 (>= 2000.00 e <= 10000.00) + aleatório	válido	7885.90
3	pce classe 1 (> 10000) + aleatório	inválido	224.020.15
4	avl caso 1	inválido	1999.99
5	avl caso 2	válido	2000.00
6	avl caso 3	válido	2000.01
7	avl caso 4	válido	9999.99
8	avl caso 5	válido	10000.00
9	avl caso 6	inválido	10000.01

exercício

aplique técnicas de teste combinadas para formular dados de teste para o seguinte requisito:

"na campanha de vacinação desse mês, uma pessoa está apta a ser vacinada se tiver entre 2 e 8 anos, ou ter mais de 65 anos."

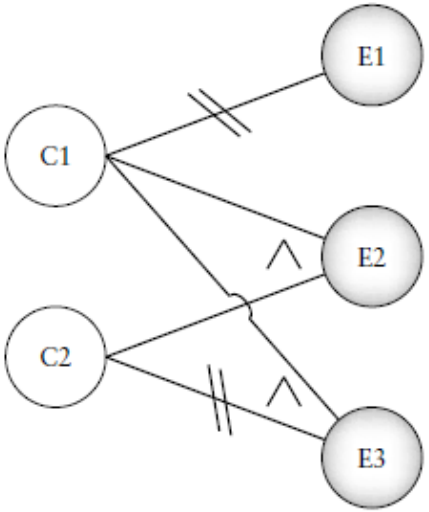
outras técnicas

- visão geral somente -

fica para pesquisa

grafo de causa-e-efeito

notação gráfica



vantagens

- permite cobrir combinações de casos de teste
- ajuda a evidenciar inconsistências na especificação

desvantagens

- pode se tornar difícil de ler
- para coisas complexas, pode ser difícil de construir e consumir tempo

*Imagem de Burnstein (2004), p. 104

tabela de decisão

vantagens

- permite cobrir combinações de casos de teste
- ajuda a evidenciar inconsistências na especificação
- fácil de construir e revisar

desvantagens

- pode ficar grande

	T1	T2	T3
C1	1	1	0
C2	1	0	-
E1	0	0	1
E2	1	0	0
E3	0	1	0

diagrama de transição de estados

notação gráfica

representa **estados do sistema**, suas **transições** e **condições**

vantagens

permite detectar defeitos não revelados por entradas/saídas

como os representados por PCE, AVL, grafo de causa-e-efeito e tabela de dec.

desvantagens

notação requer algum treinamento

pode se tornar complexo

ex.: aninhamento de diferentes níveis de abstração

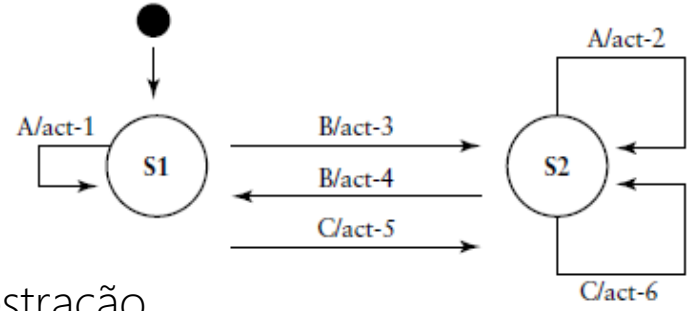


*Imagem de Burnstein (2004), p. 107

tabela de transição de estados

vantagens

- permite detectar defeitos não revelados por entradas/saídas
- notação fácil de construir

desvantagens

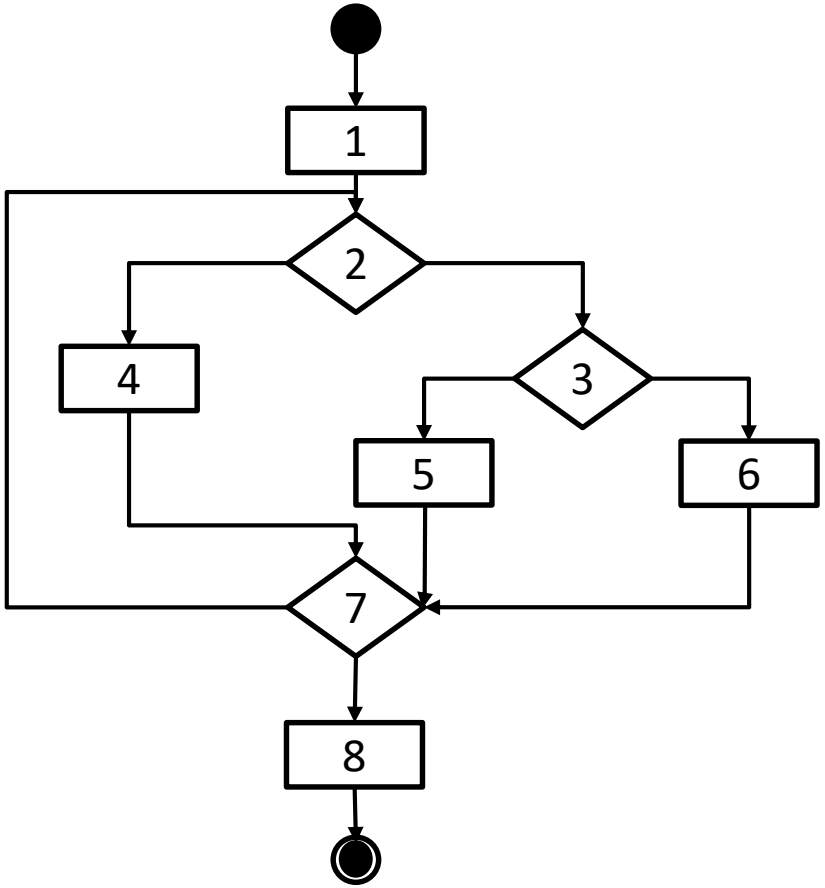
- mais difícil de compreender transições que em diagrama
- pode ficar grande

	S1	**S2**
Input A	S1 (act-1)	S2 (act-2)
Input B	S2 (act-3)	S1 (act-4)
Input C	S2 (act-5)	S2 (act-6)

teste combinatório

- introdução rápida -

software e explosão combinatória



exemplo

valores

parâmetros

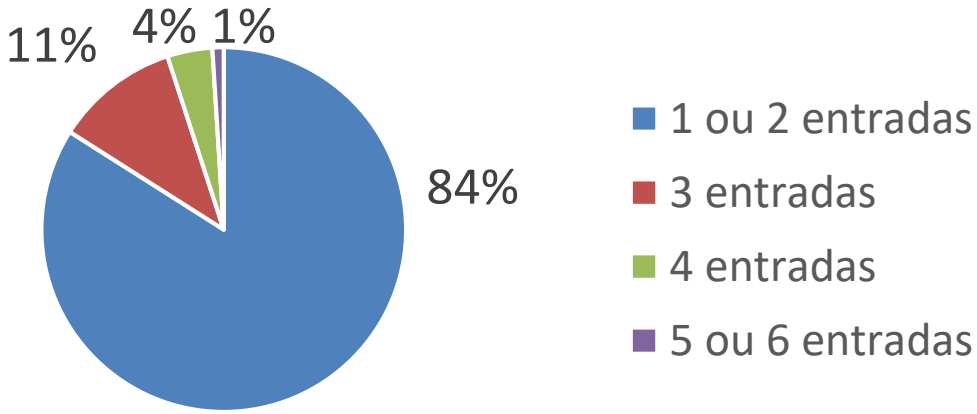
Type	Single, Spanned, Stripped, Mirror, RAID-5
Format Method	Quick, Slow
File System	FAT32, NTFS, Ext4, ReiserFS
Cluster Size	512, 1024, 2048, 4096, 8192, 16384
Compression	On, Off

todas as combinações: `5 x 2 x 4 x 6 x 2 = 480 testes`

*ignorando que podem haver restrições entre valores (*i.e.*, combinações inválidas)

constatação

Quantas combinações de entradas (parâmetros) são necessárias para detectar defeitos em produção?



WALLACE, D. R. & KUHN, D. R. **Failure modes in medical device software: an analysis of 15 years of recall data**. International Journal of Reliability, Quality and Safety Engineering, 8(4), 2001.

KUHN, D. R. & REILLY, M. J. **An investigation of the applicability of design of experiments to software testing**. In Proceedings of the 27th NASA/IEEE Software Engineering Workshop, NASA Goddard Space Flight Center, 2002.

KUHN, D. R.; WALLACE, D. R; GALLO JR, A. M.. **Software Fault Interactions and Implications for Software Testing**. IEEE Transactions on Software Engineering. pp. 418-421. Volume 30 Issue 6, June 2004.

BELL, K Z. **Optimizing Effectiveness and Efficiency of Software Testing: A Hybrid Approach**. PhD Thesis. North Carolina State University, 2006.

combinação *n-wise*

Exemplo:

parâmetros e valores

A	*X, Y*
B	*W, Z*
C	*I, J, K*

2 x 2 x 3 = 12 testes

100% cobertura

cada valor aparece pelo menos uma vez

1-wise

#	A	B	C
1	*X*	*W*	*I*
2	*Y*	*Z*	*J*
3	*X*	*W*	*K*

3 testes 25% cobertura

cada par de valores aparece pelo menos uma vez

2-wise

#	A	B	C	(A C)
1	*X*	*W*	*I*	X + I
2	*X*	*W*	*K*	X + K
3	*X*	*Z*	*J*	X + J
4	*Y*	*Z*	*I*	Y + I
5	*Y*	*Z*	*J*	Y + J
6	*Y*	*Z*	*K*	Y + K
7	*Y*	*W*	*J*	

7 testes ~58% cobertura

cada trio de valores aparece pelo menos uma vez*

3-wise

#	A	B	C
1	*X*	*W*	*I*
2	*X*	*W*	*J*
3	*X*	*W*	*K*
4	*X*	*Z*	*I*
5	*X*	*Z*	*J*
6	*X*	*Z*	*K*
7	*Y*	*W*	*I*
8	*Y*	*W*	*J*
9	*Y*	*W*	*K*
10	*Y*	*Z*	*I*
11	*Y*	*Z*	*J*
12	*Y*	*Z*	*K*

12 testes 100% cobertura

*Como há 3 parâmetros, a *3-wise* é igual a combinação total.

voltando ao exemplo

Type	Single, Spanned, Stripped, Mirror, RAID-5
Format Method	Quick, Slow
File System	FAT32, NTFS, Ext4, ReiserFS
Cluster Size	512, 1024, 2048, 4096, 8192, 16384
Compression	On, Off

5 x 2 x 4 x 6 x 2 = **480 testes**

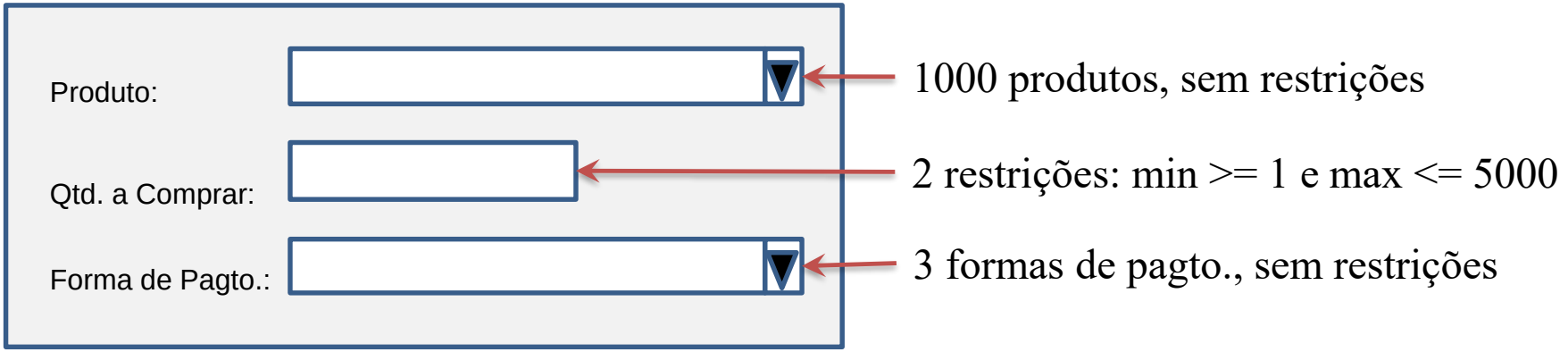
	1-wise	*2-wise*	*3-wise*	*4-wise*	*5-wise (tudo)*
testes	**6**	**30**	**121**	**249**	**480**

Cob. ~1% ~6% ~25% ~52% 100%

~85% defeitos

redução baseada em modelo – Concordia

Pedido de compra de produto → somente casos válidos (positivos), por simplificação



exaustivo = 1000 x 2 x 3 = 6000 testes **1-wise** = 1000 testes

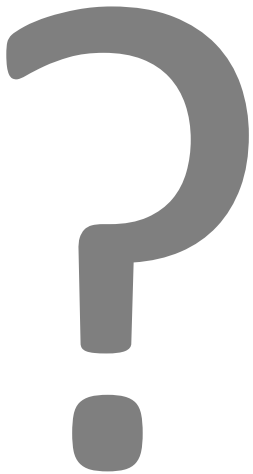
Suposições (*casos válidos*):

1) Basta 1 exemplar (aleatório) de uma entrada sem restrições → não influencia lógica da aplicação;

2) Cada restrição deve ser coberta pelos valores limítrofes → melhoria da cobertura relevante.

proposto (básico) = 1 + 4 + 1 = 6 testes

(*4 é referente a*: =1, >1, =5000, <5000)



perguntas

referências

Burnstein, Ilene. **Practical Software Testing**. 2003. Springer.

Myers, Glenford J. **The Art of Software Testing**. 2nd edition. 2004. John Wiley & Sons.