



Teste de Software

Professor Luis Rivero

Material Adaptado do Material dos profs.
Ricardo de Almeida Falbo e Marcio Delamaro

V&V: Estática x Dinâmica

- As atividades de V&V costumam ser divididas em estáticas e dinâmicas.
- As estáticas não requerem a execução ou mesmo a existência de um programa ou modelo executável para serem realizadas.
- As dinâmicas se baseiam na execução de um programa ou modelo (Delamaro et al., 2007).

2

Teste de Software

- É o processo de executar um programa com o objetivo de encontrar defeitos (Myers, 1979).
- É, portanto, uma atividade de V&V dinâmica.
- Do ponto de vista psicológico, o teste de software é uma atividade com um certo viés destrutivo, ao contrário de outras atividades do processo de software.

3

Perspectiva de Teste

- Bons testadores necessitam de um conjunto especial de habilidades. Um testador deve abordar um software com a atitude de questionar tudo sobre ele (McGregor e Sykes, 2001).
- A perspectiva de teste é, um modo de olhar qualquer produto de desenvolvimento e questionar a sua validade.
- Habilidades requeridas na perspectiva de teste:
 - Querer prova de qualidade,
 - Não fazer suposições,
 - Não deixar passar áreas importantes,
 - Procurar ser reproduzível.

4



Perspectiva de Teste

- A perspectiva de teste requer que um fragmento de software demonstre não apenas que ele executa de acordo com o especificado, mas que executa apenas o especificado (McGregor e Sykes, 2001).
- O software faz o que deveria fazer e somente isso?

5



Teste de Software

- Executa-se um programa ou modelo utilizando algumas entradas em particular e verificar-se se seu comportamento está de acordo com o esperado.
- Caso a execução apresente algum resultado não especificado, um defeito foi identificado.
- Os dados da execução podem servir como fonte para a localização e correção de defeitos, mas teste não é depuração (Delamaro et al., 2007).

6



Conceitos

- Seja **P** um programa a ser testado.
- O **domínio de entrada** de **P** (**D(P)**) é o conjunto de todos os valores possíveis que podem ser utilizados para executar **P**.
- Um **dado de teste** para **P** é um elemento de **D(P)**.
- O **domínio de saída** de **P** é o conjunto de todos os possíveis resultados produzidos por **P**.
- Um **caso de teste** de **P** é um par formado por um dado de teste mais o resultado esperado para a execução de **P** com aquele dado de teste.
- Ao conjunto de todos os casos de teste usados durante uma determinada atividade de teste dá-se o nome de **conjunto de casos de teste** ou **conjunto de teste** (Delamaro et al., 2007).

7

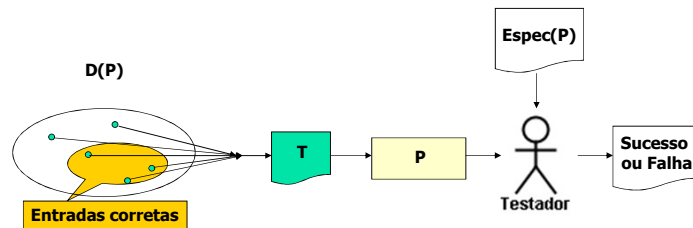


Cenário Típico da Atividade de Teste

- Definido um conjunto de casos de teste **T**, executa-se **P** com **T** e verificam-se os resultados obtidos.
- Se os resultados obtidos coincidem com os resultados esperados, então nenhum defeito foi identificado ("O software passou no teste").
- Se, para algum caso de teste, o resultado obtido difere do esperado, então um defeito foi detectado ("O software não passou no teste").
- De maneira geral, fica por conta do testador, baseado na especificação do programa, decidir sobre a correção da execução (Delamaro et al., 2007).

8

Cenário Típico da Atividade de Teste



9

Teste Exaustivo

- Para se poder garantir que **P** não contém defeitos, **P** deveria ser executado com todos os elementos de **D(P)**.
- Seja um programa **exp** com a seguinte especificação: $\text{exp}(\text{int } x, \text{int } y) = x^y$
 - **D(exp)** = todos os pares de números inteiros (x,y) passíveis de representação.
 - Cardinalidade (#) de **D(exp)** = $2^n * 2^n$, onde n = número de bits usado para representar um inteiro.
 - Em uma arquitetura de 32 bits, **#(D(exp))** = 2^{64}
 - Se cada teste puder ser executado em 1 milissegundo, seriam necessários aproximadamente 5,85 milhões de séculos para executar todos os testes.
- Logo, em geral, teste exaustivo não é viável.

10

Teste de Software

- Ao se testar **P**, devem-se selecionar alguns pontos específicos de **D(P)** para executar.
- Portanto, testes podem mostrar apenas a presença de defeitos, mas não a ausência deles.
- Um aspecto crucial para o sucesso na atividade de teste é a escolha correta dos casos de teste. Um teste bem-sucedido identifica defeitos que ainda não foram detectados.
- Um bom caso de teste é aquele que tem alta probabilidade de encontrar um defeito ainda não descoberto.

11

Teste de Software

- A escolha de casos de teste passa pela identificação de subdomínios de teste.
- Um subdomínio de teste é um subconjunto de **D(P)** que contém dados de teste semelhantes, ou seja, que se comportam do mesmo modo; por isso, basta executar **P** com um deles.
- Fazendo-se isso com todos os subdomínios de **D(P)**, consegue-se um conjunto de teste **T** bastante reduzido em relação a **D(P)**, mas que, de certa maneira, representa cada um de seus elementos (Delamaro et al., 2007).

12



Teste de Software

- Existem duas maneiras de se selecionar elementos de cada um dos subdomínios de teste (Delamaro et al., 2007):
 - Teste Aleatório: um grande número de casos de teste é selecionado aleatoriamente, de modo que, probabilisticamente, se tenha uma boa chance de ter todos os subdomínios representados em **T**.
 - Teste de Subdomínios ou Teste de Partição: procura-se estabelecer quais são os subdomínios a serem utilizados e, então, selecionam-se os casos de teste em cada subdomínio.

13



Teste de Subdomínios

- A identificação dos subdomínios é feita com base em critérios de teste.
- Dependendo dos critérios estabelecidos, são obtidos subdomínios diferentes.
- Tipos principais de critérios de teste:
 - Funcionais
 - Estruturais
 - Baseados em Modelos
- O que diferencia cada um deles é o tipo de informação utilizada para estabelecer os subdomínios (Delamaro et al., 2007) → Técnicas de Teste.

14



NÍVEIS DO TESTE (MOMENTO DE APLICAÇÃO)

15



Fases de Teste

- A atividade de teste é dividida em fases com objetivos distintos. De uma forma geral, pode-se estabelecer como fases (Delamaro et al., 2007):
 - Teste de Unidade
 - Teste de Integração
 - Teste de Sistema
 - Teste de Regressão

16



Teste de Unidade

- Tem como foco as menores unidades de um programa.
- Uma unidade é um componente de software que não pode ser subdividido.
- Nesta fase esperam-se encontrar defeitos relacionados a algoritmos incorretos ou mal implementados, estruturas de dados incorretas ou simples erros de programação.
- Pode ser aplicado à medida que ocorre a implementação das unidades e pode ser realizado pelo próprio desenvolvedor (Delamaro et al., 2007).

17



Teste de Integração

- Deve ser realizado após serem testadas as unidades individualmente.
- A ênfase é colocada na construção da estrutura do sistema.
- Deve-se verificar se as partes, quando colocadas para trabalhar juntas, não conduzem a erros.
- Requer grande conhecimento das estruturas internas do sistema e, por isso, geralmente é executado pela própria equipe de desenvolvimento (Delamaro et al., 2007).
- Todas as técnicas de teste se aplicam, com destaque para o teste funcional.

18



Teste de Sistema

- Uma vez integradas todas as partes, inicia-se o teste de sistema.
- Quando realizado por uma equipe de teste, o objetivo é verificar se as funcionalidades especificadas na especificação de requisitos foram corretamente implementadas.
- Quando realizado por usuários, o objetivo é validar o sistema (Teste de Aceitação).
- É uma boa prática que essa fase seja realizada por testadores independentes.
- Tipicamente, aplica-se teste funcional.

19



Teste de Regressão

- A cada novo módulo adicionado ou alteração, o software se modifica. Essas modificações podem introduzir novos defeitos, inclusive em funções que previamente funcionavam corretamente.
- Assim, é necessário verificar se as alterações efetuadas estão corretas e, portanto, deve-se reexecutar algum subconjunto de testes que já foi conduzido para garantir que as modificações não estão propagando efeitos colaterais indesejados (Pressman, 2006).

20



Teste de Sanidade (Bônus)

- O significado de "debugging" e "testing" inclui esforços de detectar, localizar, identificar e corrigir falhas.
- A distinção entre as duas atividades estava na definição do seu sucesso (rodar ou resolver problemas)
- O modelo de destruição reagrupa estas atividades para que a detecção da falha ocorra no "testing" e a localização do defeito, identificação e correção ocorra no "debugging".
- Em muitas organizações, o processo de "debugging" é chamado de "teste de sanidade".

Gelperin, D., & Hetzel, B. (1988). The growth of software testing. *Communications of the ACM*, 31(6), 687-695.

21



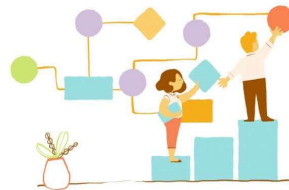
Hora de Treinar

22



Em que momento testar?

- Escolha um modelo de ciclo de vida de desenvolvimento de software (não pode ser tradicional/cascata nem modelo V).
- Em que etapas do processo escolhido você poderia aplicar os níveis de teste de software (Teste de Unidade, Teste de Integração, Teste de Sistema e Teste de Regressão)?



23



PROCESSO DE TESTE DE SOFTWARE

24



Processo de Teste

- O processo de teste pode ser definido como um processo separado, mas intimamente ligado, ao processo de desenvolvimento. Isso porque eles têm metas e medidas de sucesso diferentes.
- Por exemplo, quanto menor a taxa de defeitos (razão entre o nº de casos de teste que falham pelo total de casos de teste), mais bem sucedido é considerado o processo de desenvolvimento. Por outro lado, quanto maior a taxa de defeitos, considera-se mais bem sucedido o processo de teste (McGregor e Sykes, 2001).

25



Processo de Teste

- Independentemente da fase de teste, o processo de teste inclui as seguintes atividades:
 - Planejamento
 - Análise de Requisitos de Teste
 - Projeto de Casos de Teste
 - Implementação de Casos de Teste
 - Execução
 - Análise

26



Planejamento de Teste

- Questões importantes:
 - Quem deve realizar os testes?
 - O que testar? Que partes devem ser mais cuidadosamente testadas? (Análise de Requisitos de Teste)
 - Quanto teste é adequado?
 - Quando testar?
 - Como o teste deve ser realizado?

27



Quem deve realizar os testes?

- Desenvolvedor:
 - Código é resultado de seu trabalho. Logo, procurar defeitos é, de certo modo, a destruição do mesmo, e o próprio desenvolvedor não tem motivação psicológica para projetar casos de teste que demonstrem que seu produto tem defeitos (dissonância cognitiva).
 - Dificilmente o desenvolvedor consegue criar um caso de teste que rompa com a lógica de funcionamento de seu próprio código (Koscianski e Soares, 2006).
- Testador Independente:
 - Assume a "perspectiva de teste".
 - Pode representar papéis distintos (testador de unidade, de integração e de sistema).
 - Se responsável por todos os testes, em todas as fases, pode tornar o processo lento e pouco produtivo.

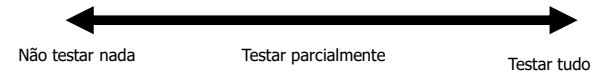
28

Quem deve realizar os testes?

- Compartilhamento de Responsabilidades:
 - Divisão por Fases:
 - Desenvolvedores testam unidades, muitas vezes em paralelo com a implementação das mesmas.
 - Testadores independentes testam integração e sistema.
 - Divisão por Atividades do Processo de Teste:
 - Testadores independentes são responsáveis pelo Planejamento, Análise e Projeto de Casos de Teste
 - Desenvolvedores são responsáveis pela construção e execução dos casos de teste.

29

O que testar?



30

O que testar?

- Princípio de Pareto adaptado ao teste: 20% dos componentes de software concentram 80% dos defeitos.
- Esse princípio sugere que os esforços devem ser concentrados nas partes mais importantes e/ou frágeis.
- Um bom teste é ao mesmo tempo econômico e encontra o máximo de defeitos (Koscianski e Soares, 2006).
- Análise de Requisitos de Teste: Quais as partes mais importantes? Quais as mais frágeis? Enfim, que partes devem ser mais cuidadosamente testadas?

31

O que testar?

- Exemplo - Teste de Unidade: testar todas as unidades? Testar unidades reutilizadas de outros projetos?
- Utilizar uma abordagem sistemática para selecionar o subconjunto de componentes a ser testado.
- Algumas diretrizes (McGregor e Sykes, 2001):
 - Pode não ser necessário testar unidades reutilizadas de outros projetos ou de bibliotecas.
 - Algumas unidades não são fáceis de serem testadas individualmente, requerendo *drivers* e/ou *stubs* complexos.
 - Concentrar esforços em usos prováveis.

32

Quanto teste é adequado?

A horizontal double-headed arrow represents a spectrum of testing. The left end is labeled 'Nenhum teste' and the right end is labeled 'Teste Exaustivo'.

33

Quanto teste é adequado?

- É impossível testar tudo.
- Testes podem somente mostrar a presença de erros, mas não a sua ausência.
- Cobertura de Teste pode ser medida pelo menos de duas maneiras (McGregor e Sykes, 2001):
 - Em relação a requisitos: quanto dos requisitos especificados foi testado? Considerar requisitos funcionais (casos de uso, fluxos de eventos) e não funcionais.
 - Em relação à execução de linhas de código (teste de caminhos possíveis).

34

Quanto teste é adequado?


- O esforço de teste deve ser compensatório, ou seja, deve haver um balanceamento entre tempo/custo do teste e a quantidade de defeitos encontrados.
- O número de defeitos encontrados segue uma curva logarítmica, ou seja, embora ainda possam existir defeitos, o esforço para encontrá-los passa a ser muito grande, fazendo com que a atividade de teste comece a ser percebida como muito custosa (Koscianski e Soares, 2006).

35

Quando testar?

A horizontal double-headed arrow represents a spectrum of when to test. The left end is labeled 'Todo dia' and the right end is labeled 'Testar no final'. In the middle, below the arrow, is the text 'Na medida em que os componentes são desenvolvidos'.


36



Quando testar?

- Teste como uma atividade do processo de software (testar no final) x Teste como um processo paralelo ao processo de desenvolvimento (teste todo dia)
- Revisões e testes são atividades complementares.
- Planejar, analisar e projetar testes à medida que o processo de desenvolvimento progride.
- Utilizar informações do ciclo de vida (incrementos) e dos requisitos (casos de uso) para definir o cronograma de testes.


37




Quando testar?

- Cronograma de teste
 - Teste de Unidade: depende da produção das unidades pelo processo de desenvolvimento.
 - Teste de Integração: depende do ciclo de vida (incrementos / iterações) e da estrutura (subsistemas / casos de uso). Pode ser programado em intervalos específicos.
 - Teste de Sistema: sua execução deve ocorrer nas entregas e, portanto, depende do ciclo de vida.

38




Como testar?



Apenas com o conhecimento da implementação (como)	Apenas com o conhecimento da especificação (o quê)	Usando o conhecimento da especificação e da implementação (o quê e como)
---	--	--

39



Como testar?


- Definir técnicas de teste para cada fase (funcional, estrutural, baseado em modelos, baseado em defeitos).
- Veremos a definição nas próximas aulas...

40




Hora de Treinar

41



Vamos planejar um teste?


- Considere o seguinte cenário.

“Os restaurantes da cidade precisam de um serviço para facilitar o delivery de comida. Para isso contrataram a sua startup para desenvolver um aplicativo móvel que permita a postagem de cardápio, realização de pedidos e pagamentos. Na sua startup os funcionários estão capacitados para desempenhar papéis diferentes (programadores, analistas, etc.).

Espera-se que um cliente faça login no aplicativo e seja capaz de informar sua localização e buscar os restaurantes próximos a ele ou que entreguem na sua residência. Um cliente pode fazer um pedido e depois realizar o pagamento do mesmo. Um ponto a ser destacado é que clientes que tenham realizado mais que dez compras no aplicativo poderão aplicar descontos, embora isto não ocorra frequentemente. Espera-se que o sistema facilite a vida dos usuários no consumo em restaurantes.


Todas as funcionalidades estão sendo desenvolvidas e cada membro do time ficou responsável por desenvolver uma funcionalidade. Na última reunião, os funcionários indicaram que concluiriam o desenvolvimento das funcionalidades nesse mesmo dia. A data de entrega do sistema está muito próxima (uma semana) e a equipe decidiu que era necessário testar o aplicativo como um todo.”

42




Vamos planejar um teste?

- Considerando o cenário descrito.
- Defina:
 - Quem deve realizar os testes? (Papeis)
 - O que testar? (Funcionalidades – Nada/Tudo)
 - Quanto teste é adequado? (Nada/Teste Exaustivo)
 - Quando testar? (Nunca/Toda Hora)



43



TÉCNICAS DE TESTE

44



TESTE FUNCIONAL (CAIXA PRETA)

45



Teste Funcional

- São testes definidos de acordo com os requisitos funcionais do software.
- Como não há conhecimento sobre a operação interna do programa, o analista concentra-se nas funções que o software contemplará.
- Baseado na especificação determina-se as saídas que são esperadas para um determinado conjunto de dados.
- Esse tipo de teste reflete a ótica do usuário sem levar em conta os detalhes de sua construção
- Comparado a outros modelos, sua concepção é relativamente simples.

46



Teste Funcional

- O programa ou sistema é considerado uma caixa-preta.
- Para testá-lo, são fornecidas entradas e avaliadas as saídas geradas para verificar se estão em conformidade com os objetivos especificados.
- Estabelece critérios para particionar o domínio de entrada em subdomínios, a partir dos quais serão definidos os casos de teste (Delamaro et al., 2007).

47



Teste Funcional

- Alguns critérios:
 - Particionamento de Equivalência
 - Análise de Valor Limite
 - Teste Funcional Sistemático
- Todos os critérios das técnicas funcionais baseiam-se apenas na especificação do produto testado e, portanto, o teste funcional depende fortemente da qualidade da especificação de requisitos.
- Podem ser aplicados em todas as fases de teste e são independentes de paradigma, pois não levam em consideração a implementação.

48



TESTE FUNCIONAL (PARTICIONAMENTO DE EQUIVALÊNCIA)

49



Particionamento de Equivalência

- Divide o domínio de entrada em classes de equivalência, de acordo com a especificação do programa.
- Uma classe de equivalência representa um conjunto de estados válidos ou inválidos para as condições de entrada.
- Caso as classes de equivalência se sobreponham ou os elementos de uma mesma classe não se comportem da mesma maneira, elas devem ser revistas, a fim de torná-las distintas (Delamaro et al., 2007).

50



Particionamento de Equivalência

- Uma vez identificadas as classes de equivalência, devem-se definir os casos de teste, escolhendo um elemento de cada classe.
- Qualquer elemento da classe pode ser considerado um representante desta e, portanto, basta ter/definir um caso de teste por classe de equivalência (Delamaro et al., 2007).


51



Particionamento de Equivalência

- Passos:
 - Identificar as classes de equivalência, tomando por base a especificação e, sobretudo, as entradas e as saídas;
 - Gerar casos de teste selecionando um elemento de cada classe, de modo a ter o menor número possível de casos de teste (Delamaro et al., 2007).


52



Exemplo 1: Programa Fibonacci

- Especificação:
 - Fluxo Principal: O usuário informa um número n , inteiro maior do que zero. O n -ésimo termo da série de Fibonacci é calculado e apresentado. A série de Fibonacci tem a seguinte formação:
 - O dois primeiros termos da série são iguais a 1.
 - O n -ésimo termo ($n > 2$) é calculado da seguinte forma: $F_n = F_{n-1} + F_{n-2}$.
 - Portanto, a série de Fibonacci tem a seguinte forma: 1, 1, 2, 3, 5, 8, 13, 21, 34...
 - Fluxo de Exceção: O número informado é menor do que 1.
 - Uma mensagem de erro é exibida indicando que o número deve ser maior do que 0.
 - Fluxo de Exceção: O valor informado não é um número inteiro.
 - Uma mensagem de erro é exibida indicando que um número inteiro positivo deve ser informado.


53



Exemplo 1: Programa Fibonacci

- Entrada:
- Saídas possíveis:


54



Exemplo 1: Programa Fibonacci

- Entrada:
 - Número inteiro n .
- Saídas possíveis:
 - n -ésimo termo da série de Fibonacci.
 - Mensagem de erro informando que n deve ser maior do que 0.
 - Mensagem de erro informando que o valor informado deve ser um número inteiro positivo.

55



Exemplo 1: Programa Fibonacci

- Classes de Equivalência:

Entrada	Classes de Equivalência Válidas	Classes de Equivalência Inválidas
n	n é um número inteiro positivo.	n é um número inteiro menor ou igual a 0. n não é um número inteiro.

56

Exemplo 1: Programa Fibonacci

- Casos de Teste:

n	Saída Esperada
6	8
-2	Erro: n deve ser maior do que 0.
a	Erro: Entre com um número inteiro positivo.

57

Hora de treinar!



Exercício 1

Considere o seguinte cenário de uso de um sistema:

Um sistema de cálculo de imposto de renda foi instalado para ajudar as pessoas a verificar quanto irão pagar para o governo federal. Uma pessoa pode verificar quanto deve pagar ao inserir no sistema o valor do seu salário. O sistema verifica conforme a tabela abaixo e calcula o valor a ser descontado no importo de renda. Após os cálculos, o sistema fornece a seguinte mensagem: “Você deve pagar R\$ XXX,XX de imposto de renda”, substituindo o valor XXX,XX pelo valor calculado, caso haja imposto devido. Ou pode responder simplesmente: “Você está isento =)”, se a pessoa não tiver ganho o suficiente para caracterizar dívida.

Aliquota de IR	Tabela corrigida
Isento	Até R\$ 3.556,56
7,5%	De R\$ 3.556,57 até R\$ 5.280,09
15,0%	De R\$ 5.280,10 até R\$ 7.073,23
22,5%	De R\$ 7.073,24 até R\$ 8.837,92
27,5%	Acima de R\$ 8.837,92

O sistema não aceita valores negativos ou 0. Caso algum destes valores seja informado, o sistema apresenta a mensagem: “Este valor não é valido para o cálculo.” Além disso, se um usuário digitar caracteres não numéricos, o sistema responde: “Erro! Insira números!”

Exercício 2: Programa Cadeia de Caracteres

- Especificação:
 - Fluxo Principal: O usuário informa uma cadeia de caracteres contendo de 1 a 20 caracteres e um caractere a ser procurado. O sistema informa a posição na cadeia em que o caractere foi encontrado pela primeira vez.
 - Fluxo Alternativo: O caractere informado não está presente na cadeia previamente informada.
 - Uma mensagem é exibida informando que o caractere não pertence à cadeia informada.
 - Fluxo de Exceção: A cadeia informada está vazia ou seu tamanho é maior do que 20.
 - Uma mensagem de erro é exibida indicando que a cadeia de caracteres deve ter tamanho de 1 a 20.
 - Fluxo de Exceção: Não é informado um caractere a ser procurado ou é informado mais de um caractere.
 - Uma mensagem de erro é exibida indicando que um caractere a ser procurado deve ser informado.

60

Exercício 2: Programa Cadeia de Caracteres

- Entradas:
- Saídas possíveis:

61

Exercício 2: Programa Cadeia de Caracteres

- Entradas:
 - Cadeia de caracteres
 - Caractere a ser procurado
- Saídas possíveis:
 - A posição da primeira ocorrência do caractere a ser procurado na cadeia.
 - Mensagem informando que o caractere não pertence à cadeia informada.
 - Mensagem de erro informando que a cadeia é inválida.
 - Mensagem de erro informando que o caractere a ser procurado é inválido.

62

Exercício 2: Programa Cadeia de Caracteres

- Classes de Equivalência:

Entrada	Classes de Equivalência Válidas	Classes de Equivalência Inválidas
Cadeia de caracteres	Qualquer cadeia de tamanho T , tal que $1 \leq T \leq 20$.	Qualquer cadeia de tamanho T , tal que $T < 1$ ou $T > 20$.
Caractere a ser procurado	Caractere que pertence à cadeia.	Caractere não informado.
	Caractere que não pertence à cadeia.	Mais de um caractere é informado.

63

Exercício 2: Programa Cadeia de Caracteres

- Casos de Teste:

Cadeia de Caracteres	Caractere a ser procurado	Saída Esperada
abc	b	2
abc	d	O caractere não pertence à cadeia informada.
abcdefghijklm nopqrstuvwxyz	<<qualquer>>	Erro: Cadeia inválida.
abc	xyz	Erro: Mais de um caractere informado.
abc	<<caractere não informado>>	Erro: Nenhum caractere informado.

64



Análise de Valor Limite

- Critério usado em conjunto com o particionamento de equivalência.
- Premissa: casos de teste que exploram condições limites têm maior probabilidade de encontrar defeitos.
- Tais condições estão exatamente sobre ou imediatamente acima ou abaixo dos limitantes das classes de equivalência (Delamaro et al., 2007).

65



Análise de Valor Limite

- Recomendações:
 - Condição de entrada especificando um intervalo de valores: definir casos de teste para os limites desse intervalo e para valores imediatamente subsequentes, acima e abaixo, que explorem as classes vizinhas.
 - Condição de entrada especificando uma quantidade de valores: definir casos de teste com nenhum valor de entrada, somente um valor, com a quantidade máxima de valores e com a quantidade máxima de valores + 1.
 - Aplicar as recomendações acima para condições de saída, quando couber.
 - Se a entrada ou a saída for um conjunto ordenado, deve ser dada atenção especial aos primeiro e último elementos desse conjunto.

66



Exemplo 1: Programa Fibonacci

- Casos de Teste adicionais:

n	Saída Esperada
1	1
2	1
0	Erro: n deve ser maior do que 0.

67




Exercício 2: Programa Cadeia de Caracteres

- Casos de Teste adicionais:

Cadeia de Caracteres	Caractere a ser procurado	Saída Esperada
	<<qualquer>>	Erro: Cadeia inválida. Entre com uma cadeia contendo de 1 a 20 caracteres.
abcdefghijklm nopqrstu	<<qualquer>>	Erro: Cadeia inválida. Entre com uma cadeia contendo de 1 a 20 caracteres.
a	a	1
a	x	O caractere não pertence à cadeia informada.

68

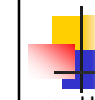


Exercício 2: Programa Cadeia de Caracteres

- Casos de Teste:


Cadeia de Caracteres	Caractere a ser procurado	Saída Esperada
abcdefghijklm nopqrst	a	1
abcdefghijklm nopqrst	t	20
abc	xy	Erro: Mais de um caractere informado.

69




Exercício 3

- Um sistema de radar audita a velocidade dos veículos, emitindo multa se necessário. Caso um veículo ultrapasse 60 km/h, emite-se um registro de multa. O valor da multa é de R\$ 200,00 mais R\$ 3,00 para cada 1 km/h acima do limite. O sistema não deve permitir entrada inválida ($v < 0$ ou $v \neq \text{num}$).
- Descreva o caso de uso para este sistema.



Especifique os casos de teste usando particionamento de equivalência e análise do valor limite.


72



Grafo Causa Efeito

- Esse critério de teste verifica o efeito combinado de dados de entrada.
- As causas (condições de entrada) e os efeitos (ações) são identificados e combinados em um grafo a partir do qual é montada uma tabela de decisão
- A partir da tabela são derivados os casos de teste e as saídas.

71



Passos

- Dividir a especificação do software em partes, pois a construção do grafo para grandes especificações torna-se bastante complexa. Identificar as causas e efeitos na especificação.
- As causas correspondem às condições de entrada, ou uma particular classe de equivalência, estímulos, ou qualquer coisa que provoque uma resposta do sistema em teste. Os efeitos correspondem às saídas, mudanças no estado do sistema ou qualquer resposta observável.
- Uma vez identificados, a cada um deve ser atribuído um único número.

72

Passos

- Analisar a semântica da especificação e transformar em um grafo booleano – o Grafo Causa-Efeito – que liga as causas e os efeitos.
- Adicionar anotações ao grafo, as quais descrevem combinações das causas e efeitos que são impossíveis devido a restrições sintáticas ou do ambiente.
- Converter o grafo em uma tabela de decisão, na qual cada coluna representa um caso de teste
- Converter as colunas da tabela de decisão em casos de teste.

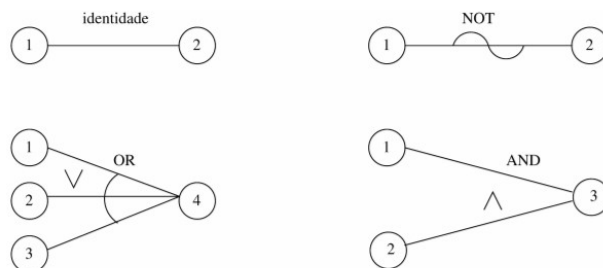
73

Notação

- Cada nó tem o valor 0 ou 1
- Valor 1 indica que aquela determinado estado existe (é verdadeiro)
- Valor 0 indica que estado não é verdadeiro
- Função identidade: se nó "1" é 1, então nó "2" é 1; senão nó "2" é 0.
- Função not: se nó "1" é 1, então nó "2" é 0; senão nó "2" é 1.
- Função or: se nó "1" ou "2" ou "n" é 1, então nó "4" é 1; senão nó "4" é 0.
- Função and: se ambos nós "1" e "2" são 1, então nó "3" é 1; senão nó "3" é 0.

74

Notação



75

Exemplo

- Sistema Imprime Mensagens
- Um programa lê dois caracteres e, de acordo com eles, mensagens serão impressas. O primeiro caractere deve ser um "A" ou um "B". Já o segundo caractere deve ser um dígito. Nessa situação, o sistema informa a seguinte mensagem "Operação Realizada". Se o primeiro caractere é incorreto, enviar a mensagem: "Caractere não é letra válida". Se o segundo caractere é incorreto, enviar a mensagem "Caractere não é um número"

76

Definindo Causas e Efeitos

Causas

- 1 - caractere na coluna 1 é "A"
- 2 - caractere na coluna 1 é "B"
- 3 - caractere na coluna 2 é um dígito

Efeitos

- 70 - "Operação Realizada"
- 71 - "Caractere não é letra válida"
- 72 - "Caractere não é um número"

77

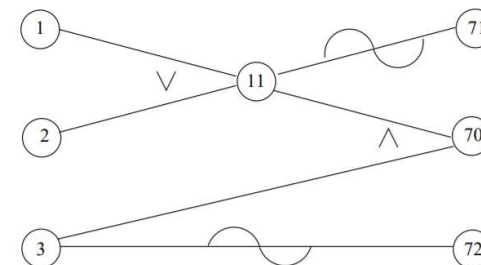
O Grafo

Causas

- 1 - caractere na coluna 1 é "A"
- 2 - caractere na coluna 1 é "B"
- 3 - caractere na coluna 2 é um dígito

Efeitos

- 70 - "Operação Realizada"
- 71 - "Caractere não é letra válida"
- 72 - "Caractere não é um número"



78

Gerando Casos de Teste

- O próximo passo é estudar sistematicamente o grafo e construir uma tabela de decisão.
- Selecionar um efeito para estar com valor 1.
- Rastrear o grafo para trás, encontrando todas as combinações de causas (sujeitas a restrições) que fazem com que esse efeito seja 1.
- Criar uma coluna na tabela de decisão para cada combinação de causa.
- Determinar, para cada combinação, os estados de todos os outros efeitos, anotando na tabela.

79

Gerando Casos de Teste

- Ao executar o passo 2, fazer as seguintes considerações:
- Quando o nó for do tipo OR e a saída deva ser 1, nunca atribuir mais de uma entrada com valor 1 simultaneamente. O objetivo disso é evitar que alguns erros não sejam detectados pelo fato de uma causa mascarar outra.
- Quando o nó for do tipo AND e a saída deva ser 0, todas as combinações de entrada que levem à saída 0 devem ser enumeradas. No entanto, se a situação é tal que uma entrada é 0 e uma ou mais das outras entradas é 1, não é necessário enumerar todas as condições em que as outras entradas sejam iguais a 1.

80



Gerando Casos de Teste

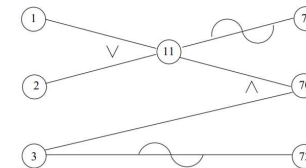
- Ao executar o passo 2, fazer as seguintes considerações:
- Quando o nó for do tipo AND e a saída deva ser 0, somente uma condição em que todas as entradas sejam 0 precisa ser enumerada. (Se esse AND estiver no meio do grafo, de forma que suas entradas estejam vindo de outros nós intermediários, pode ocorrer um número excessivamente grande de situações nas quais todas as entradas sejam 0.)

81



Gerando Casos de Teste

		Casos de Teste				
	Nós	T1	T2	T3	T4	T4
Causas	1	0	1	0	0	1
	2	0	0	1	1	0
	3	-	1	1	0	0
Efeitos	70	0	1	1	0	0
	71	1	0	0	0	0
	72	0	0	0	1	1



82



Hora de treinar!



Exercício 1

Um programa lê dois números inteiros. O primeiro número deve ter valores entre 3 e 4. O segundo número deve ter valores entre 1 e 2. O programa, então, apresenta o mês correspondente ao primeiro número inserido em português (se o segundo número for 1) ou em inglês (se o segundo número for 2). Caso seja digitado um valor inválido para o número 1, deverá ser exibida uma mensagem informando "Não existe mês com esse número". Caso seja digitado um valor inválido para o número 2, o sistema exibe: "Não existe esta opção de língua".



Possível Solução

Causas

- C1) Número 1 = 3
- C2) Número 1 = 4
- C3) Número 2 = 1
- C4) Número 2 = 2

Efeitos

- E1) March
- E2) April
- E3) Março
- E4) Abril
- E5) "Não existe mês com esse número"
- E6) "Não existe esta opção de língua"

85

Possível Solução

Causas

- C1) Número 1 = 3
- C2) Número 1 = 4
- C3) Número 2 = 1
- C4) Número 2 = 2

Efeitos

- E1) Março
- E2) Abril
- E3) March
- E4) April
- E5) "Não existe mês com esse número"
- E6) "Não existe esta opção de língua"

86

Possível Solução

		Casos de Teste						
		Nós	T1	T2	T3	T4	T5	T6
Causas	C1	0	-	1	0	1	0	
	C2	0	-	0	1	0	1	
	C3	-	0	1	1	0	0	
	C4	-	0	0	0	1	1	
Efeitos	E1	0	0	1	0	0	0	
	E2	0	0	0	1	0	0	
	E3	0	0	0	0	1	0	
	E4	0	0	0	0	0	1	
	E5	1	0	0	0	0	0	
	E6	0	1	0	0	0	0	

87

Exercício 1 - Modificado

Um programa lê dois números inteiros. O primeiro número deve ter valores entre 1 e 2. O segundo número deve ter valores entre 1 e 2. O programa, então, apresenta o mês correspondente ao primeiro número inserido em português (se o segundo número for 1) ou em inglês (se o segundo número for 2). Caso seja digitado um valor fora desse intervalo para o número 1, deverá ser exibida uma mensagem informando "Não existe mês com esse número". Caso seja digitado um número fora do intervalo para o número 2, o sistema exibe: "Não existe esta opção de língua". Adicionalmente, o sistema apresenta a seguinte mensagem caso um valor não numérico seja inserido: "Algum valor informado não é número."

Possível Solução

Causas

- 1 - Número 1 = 1
- 2 - Número 1 = 2
- 3 - Número 1 < 1
- 4 - Número 1 > 2
- 5 - Número 1 != Número
- 6 - Número 2 = 1
- 7 - Número 2 = 2
- 8 - Número 2 < 1
- 9 - Número 2 > 2
- 10 - Número 2 != Número

89

Possível Solução

Efeitos

- 20 - Mês em português
- 21 - Mês em inglês
- 22 - "Não existe mês com esse número"
- 23 - "Não existe esta opção de língua"
- 24 - "Algum valor informado não é número."

90

Exercício 2


Considere um sistema bancário que trate somente duas transações: Depósito em conta e Saque de conta. O sistema utiliza dados como: opção escolhida, valor a ser sacado ou depositado e conta da qual será feito o saque ou na qual será feito o depósito. Se o comando é depósito, o nº da conta e valor são válidos então a quantia é depositada. Se o comando é saque, o nº da conta é válido e a quantia é válida então a quantia é sacada. Se o comando ou nº da conta ou a quantia for inválido então exibir mensagem de erro apropriada.

Crie o grafo causa-efeito e a tabela contendo os casos de teste para tal grafo.

Exercício 3

- "Em um programa de compras pela Internet, a cobrança ou não do frete é definida seguindo a regra: Se o valor da compra for maior que R\$ 60,00 e foram comprados menos que 3 produtos, o frete é gratuito. Caso contrário, o frete deverá ser cobrado."






Teste Funcional Sistemático

- Combina os critérios de Particionamento de Equivalência e Análise de Valor Limite.
- Uma vez que os domínios de entrada e de saída tenham sido particionados, este critério requer ao menos dois casos de teste de cada partição para minimizar o problema de defeitos coincidentes que mascaram falhas (Delamaro et al., 2007).


93



Teste Funcional Sistemático

- Algumas diretrizes adicionais:
 - Valores numéricos – Domínio de Entrada
 - Discretos: testar todos os valores
 - Intervalo de valores: testar os extremos e um valor no interior do intervalo
 - Valores numéricos – Domínio de Saída
 - Discretos: gerar cada um dos valores
 - Intervalo de valores: gerar cada um dos extremos e ao menos um valor no interior do intervalo
 - Valores ilegais: também devem ser incluídos nos casos de teste para se verificar se o software os rejeita. Diretrizes do critério de análise de valor limite devem ser empregadas.


94



Teste Funcional Sistemático

- Algumas diretrizes adicionais:
 - Tipos de valores diferentes
 - Explorar tipos ilegais que podem ser interpretados como valores válidos (p.ex., valor real para um campo inteiro).
 - Explorar entradas válidas, mas que podem ser interpretadas como tipos ilegais (p.ex., números em campos que requerem caracteres).
 - Explorar limites da representação binária dos dados (p.ex., para campos inteiros de 16 bits, selecionar os valores -32768 e +32767).
 - Testar ao menos dois elementos de uma mesma classe de equivalência para minimizar o problema de defeitos coincidentes que mascaram falhas.

95



Exemplo 1: Programa Cadeia de Caracteres

- Casos de Teste adicionais:

Cadeia de Caracteres	Caractere a ser procurado	Saída Esperada
!	` `	O caractere não pertence à cadeia informada.
!"#\$%&()*+` /01234567	!	1
	&	6
	`	11
	6	19
	7	20

96



Exemplo 2: Programa Fibonacci

- Casos de Teste:

n	Saída Esperada
1.0	Erro: Entre com um número inteiro positivo.
-1	Erro: n deve ser maior do que 0.
xyz	Erro: Entre com um número inteiro positivo.
23	28657
24	46368
11	89

97



Teste Funcional: Considerações Finais

- Como os critérios funcionais se baseiam apenas na especificação, não podem assegurar que partes críticas e essenciais do código tenham sido cobertas.
- Além disso, os próprios critérios apresentam limitações.
- Assim, é fundamental que outras técnicas sejam aplicadas em conjunto, para que o software seja explorado por diferentes pontos de vista (Delamaro et al., 2007).

98



Referências

- Delamaro, M.E., Maldonado, J.C., Jino, M., Introdução ao Teste de Software, Série Campus – SBC, Editora Campus, 2007.