Resumo - Teste de caixa preta

4 - Projeto de Caso de teste

Introdução

A consideração mais importante no teste de programa é o design e a criação de casos de teste eficazes.
O design do caso de teste é tão importante porque o teste completo é impossível. Dito de outra forma, um teste de qualquer programa deve ser necessariamente incompleto. A estratégia óbvia, então, é tentar fazer os testes o mais completos possível.
Em geral, a metodologia menos eficaz de todas é o teste de entrada aleatória - o processo de testar um programa selecionando, aleatoriamente, algum subconjunto de todos os valores de entrada possíveis.

Caixa preta:

- Particionamento equivalente
- Análise de valor de limite
- Gráficos de causa e efeito
- Erro ao adivinhar

Teste de caixa preta

Conforme discutimos no capítulo 2, o teste de caixa preta (orientado por dados ou orientado por entradas e saídas) é baseado nas especificações do programa. O objetivo é encontrar áreas em que o programa não se comporte de acordo com suas especificações.

Particionamento equivalente

Uma maneira de localizar esse subconjunto é perceber que um caso de teste bem selecionado também deve ter duas outras propriedades:
Reduz, em mais de um, o número de outros casos de teste que devem ser desenvolvidos para atingir algum objetivo pré definido de teste “razoável”.
Abrange um grande conjunto de outros casos de teste possíveis. Ou seja, ele nos diz algo sobre a presença ou ausência de erros além desse conjunto específico de valores.

A primeira implica que cada caso de teste deve invocar considerações de entrada quando possível para minimizar o número total de casos de teste necessários. A segunda implica que você deve tentar particionar o domínio de entrada de um programa em um número finito de classes de equivalência tal que você possa razoavelmente supor (mas, é claro, não ter certeza absoluta) que um teste de um valor representativo de cada classe é equivalente a um teste de qualquer outros valor. Ou seja, se um caso de teste em uma classe de equivalência detecta um erro, espera-se que todos os outros casos de teste em uma classe de equivalência

encontrem o mesmo erro. Por outro lado, se um caso de teste não detectar um erro, esperaríamos que nenhum outro caso de teste na classe de equivalência se enquadra em outra classe de equivalência, uma vez que as classes de equivalência podem se sobrepor.

A segunda consideração é usada para desenvolver um conjunto de condições “interessantes” a serem testadas. A primeira consideração é então usada para desenvolver um conjunto mínimo de casos de teste cobrindo essas condições.

Identificando as classes de equivalência
As classes de equivalência são identificadas tomando cada condição de entrada (geralmente uma sentença ou frase na especificação) e dividindo-a em dois ou mais grupos.

Dada uma entrada ou condição externa, identificar as classes de equivalência é em grande parte um processo heurístico. Siga as orientações:

1. Se uma condição de entrada especificar um intervalo de valores (por exemplo, “a contagem de itens pode ser de 1 a 999”), identifique uma classe de equivalência válida (1< contagem de itens < 999) e duas classes de equivalência inválidas (contagem de itens <1 e contagem de itens > 999).
2. Se uma condição de entrada especificar o número de valores (por exemplo, "de um a seis proprietários podem ser listados para o automóvel"), identifique uma classe de equivalência válida e duas classes de equivalência inválidas (nenhum proprietário e mais de seis proprietários) .
3. Se uma condição de entrada especifica um conjunto de valores de entrada, e há motivos para acreditar que o programa trata cada um de forma diferente ("tipo de veículo deve ser ÔNIBUS, CAMINHÃO, TÁXI, PASSAGEIRO ou MOTO"), identificar uma classe de equivalência válida para cada uma e uma classe de equivalência inválida ("TRAILER", por exemplo).
4. Se uma condição de entrada especificar uma situação "obrigatória", como "o primeiro caractere do identificador deve ser uma letra", identifique uma classe de equivalência válida (é uma letra) e uma classe de equivalência inválida (não é uma carta).

Se houver alguma razão para acreditar que o programa não trata os elementos de uma classe de equivalência de forma idêntica, divida a classe de equivalência em classes de equivalência menores. Ilustramos um exemplo desse processo em breve.

Identificando os casos de teste
O processo é como se segue:

- Atribua um número único a cada classe de equivalência.

- Até que todas as classes de equivalência válidas tenham sido cobertas (incorporadas em) casos de teste, escreva um novo caso de teste cobrindo o maior número possível de classes de equivalência válidas descobertas.
- Até que seus casos de teste tenham coberto todas as classes de equivalência inválidas, escreva um caso de teste que cubra uma, e apenas uma, das classes de equivalência inválidas descobertas.

A razão pela qual os casos de teste individuais cobrem casos inválidos é que certas verificações de entrada errônea mascaram ou substituem outras verificações de entrada errôneas.

Um exemplo:
Uma instrução DIMENSION é usada para especificar as dimensões das matrizes.
O primeiro passo é identificar as condições de entrada e, a partir delas, localizar as classes de equivalência. Estes são tabulados na Tabela 4.1. Os números na tabela são identificadores únicos das classes de equivalência.
O próximo passo é escrever um caso de teste cobrindo uma ou mais classes de equivalência válidas. Por exemplo, o caso de teste DIMENSION(2)
abrange as classes 1,4,7,10,12,15,24,28,27 e 43.

O próximo passo é elaborar um ou mais casos de teste cobrindo o restante das classes de equivalência válidas. Um caso de teste do formulário:

DIMENSÃO A 12345 (I,9,J4XXXX,65535,1,KLM,
X,1000, BBB(-65534:100,0:1000,10:10, I:65535)

abrange as demais classes. As classes de equivalência de entrada inválidas e um caso de teste representando cada um, são:

Embora o particionamento de equivalência seja muito superior a uma seleção aleatória de casos de teste, ele ainda apresenta deficiências. Ele ignora certos tipos de casos de teste de alto rendimento, por exemplo. As próximas duas metodologias, análise de valor limite e gráficos de causa e efeito, cobrem muitas dessas deficiências.

Análise de valor limite
A experiência mostra que os casos de teste que exploram as condições de contorno têm um retorno maior do que os casos de teste que não exploram. As condições de fronteira são aquelas situações diretamente sobre, acima e abaixo das bordas das classes de equivalência de entrada e classes de equivalência de saída. A análise de valor limite difere da partição de equivalência em dois aspectos:

1. Em vez de selecionar qualquer elemento em uma classe de equivalência como sendo representativo, a análise de valor de contorno requer que um ou

mais elementos sejam selecionados de modo que cada aresta da classe de equivalência seja objeto de um teste.

2. Em vez de apenas focar a atenção nas condições de entrada (espaço de entrada), os casos de teste também são derivados considerando o espaço de resultado (classes de equivalência de saída).

É difícil apresentar um "livro de receitas" para a análise de valor de fronteira, pois requer um grau de criatividade e uma certa especialização em relação ao problema em questão.

No entanto, algumas diretrizes gerais são necessárias:

1. Se uma condição de entrada especificar um intervalo de valores, escreva casos de teste para as extremidades do intervalo e casos de teste de entrada inválida para situações logo além das extremidades. Por exemplo, se o domínio válido de um valor de entrada for –1,0 a 1,0, escreva casos de teste para as situações –1,0, 1,0, –1,001 e 1,001.
2. Se uma condição de entrada especificar um número de valores, escreva casos de teste para o número mínimo e máximo de valores e um abaixo e além desses valores. Por exemplo, se um arquivo de entrada pode conter de 1 a 255 registros, escreva casos de teste para 0, 1, 255 e 256 registros.
3. Use a diretriz 1 para cada condição de saída. Por exemplo, se um programa de folha de pagamento calcular a dedução mensal do FICA e se o mínimo for $0,00 e o máximo for $1.165,25, escreva casos de teste que causem a dedução de $0,00 e $1.165,25. Além disso, veja se é possível inventar casos de teste que possam causar uma dedução negativa ou uma dedução de mais de $1.165,25.

Observe que é importante examinar os limites do espaço de resultados porque nem sempre os limites dos domínios de entrada representam o mesmo conjunto de circunstâncias que os limites dos intervalos de saída (por exemplo, considere uma subrotina senoidal). . Além disso, nem sempre é possível gerar um resultado fora do intervalo de saída; no entanto, vale a pena considerar a possibilidade.

4. Use a diretriz 2 para cada condição de saída. Se um sistema de recuperação de informações exibe os resumos mais relevantes com base em uma solicitação de entrada, mas nunca mais de quatro resumos, escreva casos de teste de modo que o programa exiba zero, um e quatro resumos e escreva um caso de teste que possa causar o programa exibir erroneamente cinco resumos.
5. Se a entrada ou saída de um programa for um conjunto ordenado (um arquivo sequencial, por exemplo, ou uma lista linear ou uma tabela), concentre a atenção no primeiro e no último elemento do conjunto.
6. Além disso, use sua criatividade para procurar outros limites de condições.

Portanto, a diferença importante entre a análise de valor de contorno e a partição de equivalência é que a análise de valor de contorno explora situações dentro e ao redor das bordas das partições de equivalência.

Os quatro registro de saída são:

1. Um relatório, ordenado por identificador de aluno, mostrando a nota de cada aluno (porcentagem de acertos) e classificação
2. Um relatório semelhante, mas classificado por grau.
3. Um relatório indicando a média, mediana e desvio padrão dos graus.
4. Um relatório, ordenado por número de pergunta, mostrando a porcentagem de idade dos alunos que responderam corretamente a cada pergunta.

A primeira condição de entrada de limite é um arquivo de entrada vazio. A segunda condição de entrada é o registro de título; condições de contorno são registro de título ausente e os títulos mais curtos e mais longos possíveis. Nas próximas condições de entrada são a presença de registros de respostas corretas e a campo de número de perguntas no primeiro registro de resposta. A classe de equivalência, pois o número de perguntas não é de 1 a 999, porque algo especial acontece a cada múltiplo de 50 (ou seja, são necessários vários registros). Uma partição razoável disso em classes de equivalência é 1–50 e 51–999. Portanto, precisamos de casos de teste em que o campo de número de perguntas seja definido como 0, 1, 50, 51 e 999. Isso cobre a maioria das condições de limite para o número de registros de respostas corretas; no entanto, três situações mais interessantes são a ausência de registros de respostas e ter um registro de resposta a mais e um a menos.

Representação gráfica de causa e efeito

Um ponto fraco da análise de valor limite e partição de equivalência é que eles não exploram combinações de circunstâncias de entrada. O teste de combinações de entrada não é uma tarefa simples porque mesmo se você particionar por equivalência às condições de entrada, o número de combinações geralmente é astronômico.

A representação gráfica de causa-efeito auxilia na seleção, de forma sistemática, de um conjunto de casos de teste de alto rendimento. Tem um efeito colateral benéfico ao apontar incompletude e ambiguidades na especificação.

Um grafo causa-efeito é uma linguagem formal na qual uma linguagem natural especificada é traduzida. O gráfico na verdade é um circuito lógico digital (um rede lógica combinatória), mas em vez de notação eletrônica padrão , uma notação um pouco mais simples é usada. Nenhum conhecimento de eletrônica é necessário além da compreensão da lógica booleana (ou seja, dos operadores lógicos e, ou, e não).

O processo a seguir é usado para derivar casos de teste:

1. A especificação é dividida em partes viáveis. Isso é necessário porque os gráficos de causa e efeito tornam-se difíceis de manejar quando usados em especificações grandes. Por exemplo, ao testar um sistema de comércio eletrônico, uma solução viável peça pode ser a especificação para escolher e verificar um único item colocado em um carrinho de compras. Ao testar um design de página da Web, você pode testar uma única árvore de menu ou até mesmo uma sequência de navegação menos complexa.
2. As causas e efeitos na especificação são identificados. Uma causa é uma condição de entrada distinta ou uma classe de equivalência de condições de entrada. Um efeito é uma condição de saída ou uma transformação do sistema (um efeito prolongado que uma entrada tem no estado do programa ou sistema). Por exemplo, se uma transação fizer com que um arquivo ou registro de banco de dados seja atualizado, a alteração é uma transformação do sistema; uma confirmação mensagem seria uma condição de saída.

Você identifica causas e efeitos lendo a palavra de especificação por palavra e sublinhando palavras ou frases que descrevem causas e efeitos. Uma vez identificados, cada causa e efeito recebe um número único.

3. O conteúdo semântico da especificação é analisado e transformado em um gráfico booleano ligando as causas e os efeitos. Isto é o gráfico causa-efeito.

4. O gráfico é anotado com restrições que descrevem combinações de causas e/ou efeitos que são impossíveis por causa da sintaxe ou do ambiente e restrições mentais.

5. Ao rastrear metodicamente as condições de estado no gráfico, você converte o gráfico em uma tabela de decisão de entrada limitada. Cada coluna do tabela representa um caso de teste.

6. As colunas na tabela de decisão são convertidas em casos de teste.

A função identidade afirma que se a é 1, b é 1; senão b é 0.
A função not afirma que se a é 1, b é 0, senão b é 1.
A função ou afirma que se a ou b ou c é 1, d é 1; senão d é 0.
A função and afirma que se a e b são 1, c é 1; senão c é 0.
As duas últimas funções (ou e e) podem ter qualquer número de entradas.

Você deve confirmar que o gráfico representa a especificação definindo todos os estados possíveis das causas e verificando se os efeitos estão definidos com os valores corretos.

Para explicar isso, a notação na Figura 4.8 é usada. A restrição E afirma que deve ser sempre verdade que, no máximo, um de a e b pode ser 1 (a e b não podem ser 1 simultaneamente). A restrição I afirma que pelo menos um de a, b e c deve ser sempre 1 (a, b e c não podem ser 0 simultaneamente). A restrição O afirma que um, e apenas um, de a e b deve ser 1. A restrição R afirma que para a ser 1, b deve ser 1 (ou seja, é impossível que a seja 1 e b seja 0).

Frequentemente há a necessidade de uma restrição entre os efeitos. O M contra strain na Figura 4.9 afirma que se o efeito a é 1, o efeito b é forçado a 0.

Para ilustrar como o gráfico de causa e efeito é usado para derivar casos de teste, use a seguinte especificação para um comando de depuração em um sistema interativo.

O comando DISPLAY é usado para visualizar a partir de uma janela de terminal o conteúdo dos locais de memória. A sintaxe do comando é mostrada em Figura 4.11. Os colchetes representam operandos opcionais alternativos. Letras maiúsculas representam palavras-chave de operandos. As letras minúsculas representam valores dos operandos (os valores reais devem ser substituídos). Os operandos sublinhados representam os valores padrão (ou seja, o valor usado quando o operando é omitido).

O primeiro operando (hexloc1) especifica o endereço do primeiro byte cujo conteúdo deve ser exibido. O endereço pode ter de um a seis dígitos hexadecimais (0–9, A–F) de comprimento. Se não for especificado, o endereço 0 é assumido. O endereço deve estar dentro do intervalo de memória real da máquina.

O segundo operando especifica a quantidade de memória a ser exibida. Se hexloc2 for especificado, ele define o endereço do último byte no intervalo de locais a serem exibidos. Pode ter de um a seis dígitos hexadecimais de comprimento. O endereço deve ser maior ou igual ao endereço inicial (hexloc1). Além disso, hexloc2 deve estar dentro do intervalo de memória real da máquina. Se o END for especificado, a memória será exibida até o último byte real na máquina. Se for especificado bytecount , define o número de bytes de memória a serem exibidos (começando com o local especificado em hexloc1). O operando bytecount é um inteiro hexadecimal (um a seis dígitos). A soma de bytecount e hexloc1 não deve exceder o tamanho real da memória mais 1 e bytecount deve ter um valor de pelo menos 1.

O primeiro passo é uma análise cuidadosa da especificação para identificar as causas e efeitos. As causas são as seguintes:

1. O primeiro operando está presente.
2. O operando hexloc1 contém apenas dígitos hexadecimais.
3. O operando hexloc1 contém de um a seis caracteres.
4. O operando hexloc1 está dentro da faixa de memória real da máquina.
5. O segundo operando é END.
6. O segundo operando é hexloc.
7. O segundo operando é bytecount.
8. O segundo operando é omitido.
9. O operando hexloc2 contém apenas dígitos hexadecimais.
10. O operando hexloc2 contém de um a seis caracteres.
11. O operando hexloc2 está dentro da faixa de memória real da máquina.
12. O operando hexloc2 é maior ou igual ao operando hexloc1.
13. O operando bytecount contém apenas dígitos hexadecimais.
14. O operando bytecount contém de um a seis caracteres.
15. bytecount þ hexloc1 <¼ tamanho da memória þ
1. 16. bytecount >¼ 1.
17. A faixa especificada é grande o suficiente para exigir várias linhas de saída.
18. O início do intervalo não cai em um limite de palavra.

Os efeitos são os seguintes:

91. A mensagem M1 é exibida.
92. A mensagem M2 é exibida.
93. A mensagem M3 é exibida.
94. A memória é exibida em uma linha.
95. A memória é exibida em várias linhas.
96. O primeiro byte do intervalo exibido cai em um limite de palavra.
97. O primeiro byte do intervalo exibido não cai em um limite de palavra.

O próximo passo é o desenvolvimento do gráfico. Os nós de causa são listados verticalmente no lado esquerdo da folha de papel; os nós de efeito são listados verticalmente no lado direito. O conteúdo semântico da especificação é cuidadosamente analisado para interligar as causas e os efeitos (ou seja, para mostrar em que condições um efeito está presente).

O próximo passo é a geração de uma tabela de decisão de entrada limitada. Para leitores familiarizados com tabelas de decisão, as causas são as condições e os efeitos são as ações. O procedimento utilizado é o seguinte:

1. Selecione um efeito para ser o estado atual (1).
2. Voltando ao gráfico, encontre todas as combinações de causas (sujeitas às restrições) que definirão esse efeito como 1.
3. Crie uma coluna na tabela de decisão para cada combinação de causas.

Para cada combinação, determine os estados de todos os outros efeitos e coloque-os em cada coluna.

Na execução da etapa 2, as considerações são as seguintes:

1. Ao rastrear de volta através de um nó ou cuja saída deve ser 1, nunca defina mais de uma entrada para ou como 1 simultaneamente. Isso é chamado de sensibilização de caminho. Seu objetivo é evitar a falha na detecção de certos erros por causa de uma causa mascarando outra causa.
2. Ao rastrear de volta através de um nó e cuja saída deve ser 0, todas as combinações de entradas que levam à saída 0 devem, é claro, ser enumeradas. No entanto, se você estiver explorando a situação em que uma entrada é 0 e uma ou mais das outras são 1, não é necessário enumerar todas as condições sob as quais as outras entradas podem ser 1.
3. Ao rastrear de volta através de um nó e cuja saída deve ser 0, apenas uma condição em que todas as entradas são zero precisa ser enumerada. (Se está no meio do gráfico de tal forma que suas entradas vêm de outros nós intermediários, pode haver um número excessivamente grande de situações em que todas as suas entradas são 0.)

Essas considerações podem parecer caprichosas, mas têm um propósito importante: diminuir os efeitos combinados do gráfico. Eles eliminam situações que tendem a ser casos de teste de baixo rendimento. Se casos de teste de baixo rendimento não forem eliminados, um grande gráfico de causa e efeito produzirá um State seja 0. A consideração 3 afirma que devemos listar apenas uma circunstância onde os nós 5 e 6 são 0. A consideração 2 afirma que para o estado onde o nó 5 é 1, em vez de enumerar todas as maneiras possíveis que o nó 5 número de casos de teste. Se o número de casos de teste for muito grande para ser prático,você selecionará algum subconjunto, mas não há garantia de que o baixo rendimento nos casos de teste serão eliminados. Por isso, é melhor eliminá-los durante a análise do gráfico.

Comentários

A representação gráfica de causa-efeito é um método sistemático de gerar casos representando combinações de condições. A alternativa seria fazer uma seleção ad hoc de combinações; mas ao fazê-lo, é provável que você ignoraria muitos dos casos de teste "interessantes" identificados pelo gráfico de causa e efeito.

Como o gráfico de causa e efeito requer a tradução de uma especificação em uma rede lógica booleana, ela oferece uma perspectiva diferente e uma visão adicional da especificação. Na verdade, o desenvolvimento de uma causa de gráfico de efeito é uma boa maneira de descobrir ambiguidades e incompletude nas especificações.

Além disso, o gráfico causa-efeito não explora adequadamente as condições de contorno.

O problema em fazer isso, no entanto, é que complica tremendamente o gráfico e leva a um número excessivamente grande de casos de teste. Por esta razão, é melhor considerar uma análise de valor de limite separada.
Como o gráfico de causa-efeito nos dá margem de manobra na seleção de valores específicos para operandos, as condições de contorno podem ser combinadas nos casos de teste derivados do gráfico de causa-efeito.

A identificação da saída esperada de cada caso de teste é uma parte inerente da técnica (cada coluna na tabela de decisão indica os efeitos esperados).

O aspecto mais difícil da técnica é a conversão do gráfico em tabela de decisão. Esse processo é algorítmico, o que implica que você pode automatizá-lo escrevendo um programa; existem vários programas comerciais para ajudar na conversão.

Erro ao adivinhar
Uma explicação para isso é que essas pessoas estão praticando - subconscientemente com mais frequência do que não - uma técnica de projeto de caso de teste que poderia ser chamada de adivinhação de erros.

A ideia básica é enumerar uma lista de possíveis erros ou situações propensas a erros e então escrever casos de teste com base na lista. Por exemplo, a presença do valor 0 na entrada de um programa é uma situação propensa a erros. Portanto, você pode escrever casos de teste para os quais valores de entrada específicos têm um valor 0 e para os quais valores de saída específicos são forçados a 0. Além disso, onde um número variável de entradas ou saídas pode estar presente (por exemplo, o número de entradas em um lista a ser pesquisada), os casos de "nenhum" e "um" (por exemplo, lista vazia, lista contendo apenas uma entrada) são situações propensas a erros. Outra ideia é identificar casos de teste associados a suposições que o programador possa ter feito ao ler a especificação.

Em outras palavras, você enumera os casos especiais que podem ter sido negligenciados quando o programa foi projetado. Se estiver testando uma sub-rotina de pesquisa binária, você pode tentar as situações em que: (1) há apenas uma entrada na tabela que está sendo pesquisada; (2) o tamanho da tabela é uma potência de 2 (por exemplo, 16); e (3) o tamanho da tabela é um menor e um maior que uma potência de 2 (por exemplo, 15 ou 17).

A estratégia
A razão para combiná-los já deve ser óbvia: cada um contribui com um conjunto específico de casos de teste úteis, mas nenhum deles por si só contribui com um conjunto completo de casos de teste.

Uma estratégia razoável é a seguinte:

1. Se a especificação contiver combinações de condições de entrada, inicia Testes de adivinhação de erros que vêm à mente para o comando DISPLAY do com gráficos de causa e efeito.
2. Em qualquer caso, use a análise de valor limite. Lembre-se de que esta é uma análise dos limites de entrada e saída. A análise do valor limite produz um conjunto de condições de teste suplementares, mas, conforme observado na seção sobre gráficos de causa e efeito, muitas ou todas elas podem ser incorporadas aos testes de causa e efeito.
3. Identifique as classes de equivalência válidas e inválidas para a entrada e saída e complemente os casos de teste identificados acima, se necessário.
4. Use a técnica de adivinhação de erros para adicionar casos de teste adicionais.
5. Examine a lógica do programa em relação ao conjunto de casos de teste. Use o critério de cobertura de decisão, cobertura de condição, idade de cobertura de decisão/condição ou critério de cobertura de várias condições (sendo o último o mais completo). Se o critério de cobertura não tiver sido atendido pelos casos de teste identificados nas quatro etapas anteriores, e se atender ao critério não for impossível (ou seja, certas combinações de condições podem ser impossíveis de criar devido à natureza do programa), adicione casos de teste suficientes para fazer com que o critério seja satisfeito.

Resumo

As técnicas de projeto de caso de teste discutidas neste capítulo incluem:

- Cobertura lógica: Testes que exercitam todos os resultados do ponto de decisão pelo menos uma vez e garantem que todas as instruções ou pontos de entrada sejam executados pelo menos uma vez.
- Particionamento equivalente: Define condições ou classes de erro para ajudar a reduzir o número de testes finitos. Assume que um teste de um valor representativo dentro de uma classe também testa todos os valores ou condições dentro dessa classe.
- Análise de valor limite: Testa cada condição de aresta de uma classe de equivalência; também considera classes de equivalência de saída, bem como classes de entrada.
- Gráficos de causa-efeito: Produz representações gráficas booleanas de resultados de casos de teste em potencial para auxiliar na seleção eficiente e completa de casos de teste.
- Erro ao adivinhar: Produz casos de teste com base no conhecimento intuitivo e especializado dos membros da equipe de teste para definir possíveis erros de software para facilitar o design eficiente do caso de teste.

Testes extensivos e aprofundados não são fáceis; nem o projeto de caso de teste mais extenso garantirá que todos os erros sejam descobertos.

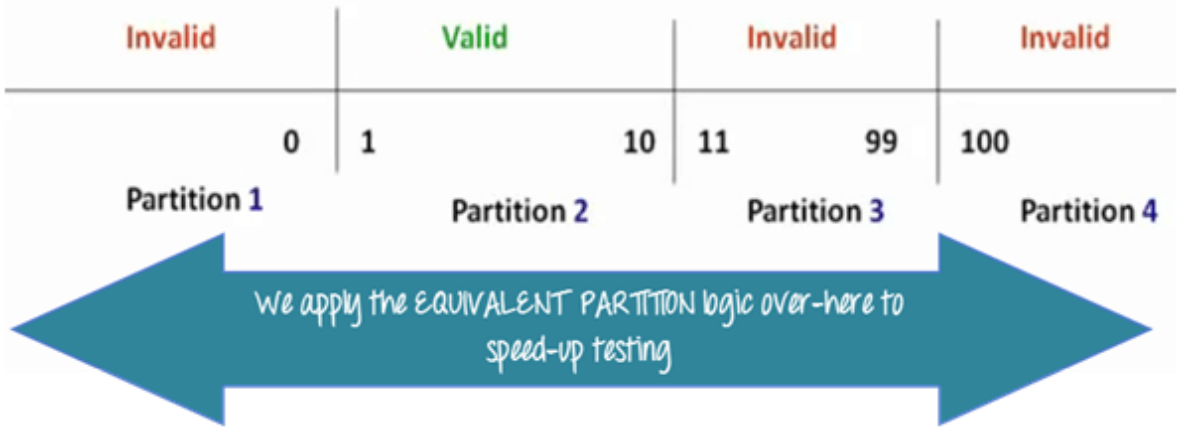
Exemplo 1: Particionamento de equivalência

- Vamos considerar o comportamento da Caixa de Texto Pedir Pizza Abaixo
- Valores de pizza de 1 a 10 são considerados válidos. Uma mensagem de sucesso é exibida.
- Enquanto os valores de 11 a 99 são considerados inválidos para pedido e uma mensagem de erro aparecerá, **"Somente 10 Pizzas podem ser pedidas"**

Aqui está a condição de teste

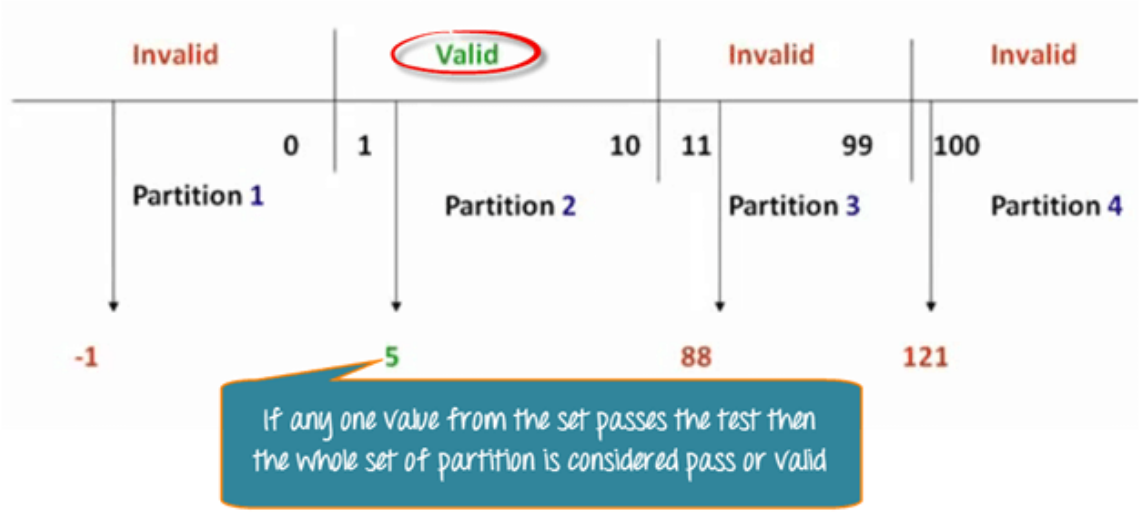
1. Qualquer número maior de 10 inseridos no campo Pedir Pizza (digamos 11) é considerado inválido.
2. Qualquer número menor que 1 que seja 0 ou inferior, é considerado inválido.
3. Os números de 1 a 10 são considerados válidos
4. Qualquer número de 3 dígitos diz que -100 é inválido.

Não podemos testar todos os valores possíveis porque se feito, o número de casos de teste será maior que 100. Para resolver esse problema, usamos a hipótese de particionamento de equivalência onde dividimos os valores possíveis de tickets em grupos ou conjuntos conforme mostrado abaixo onde o sistema comportamento pode ser considerado o mesmo.



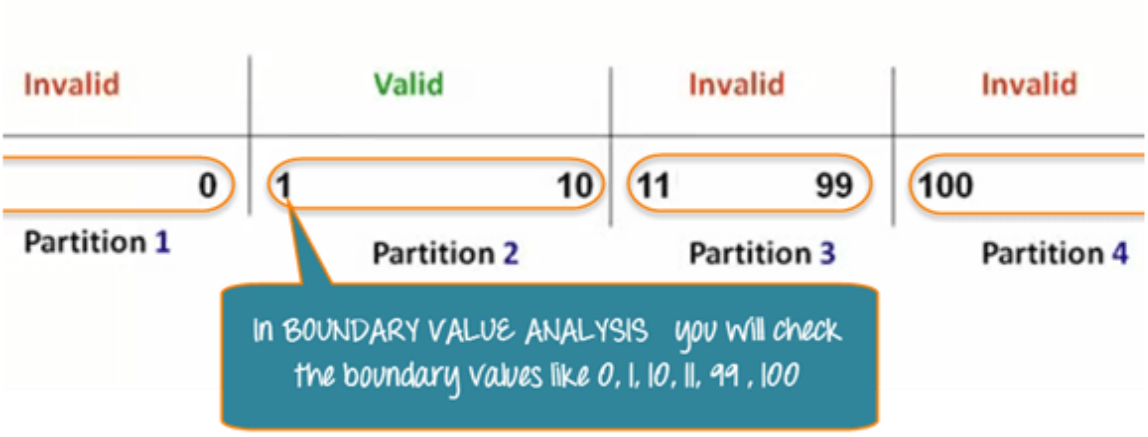
Os conjuntos divididos são chamados de Partições de Equivalência ou Classes de Equivalência. Em seguida, escolhemos apenas um valor de cada partição para teste. A hipótese por trás dessa técnica é **que, se uma condição/valor em uma**

partição passar, todas as outras também passarão . Da mesma forma **, se uma condição em uma partição falhar, todas as outras condições nessa partição falharão** .



Exemplo 2: Análise do valor limite

Análise de valor de limite - na análise de valor de limite, você testa limites entre partições de equivalência.



Em nosso exemplo anterior de particionamento de equivalência, em vez de verificar um valor para cada partição, você verificará os valores nas partições como 0, 1, 10, 11 e assim por diante. Como você pode observar, você testa valores em **limites válidos e inválidos**. A análise de valor de limite também é chamada **de verificação de intervalo**.