Teste de unidade

Teste de unidade é toda a aplicação de teste nas assinaturas de entrada e saída de um sistema. Consiste em validar dados válidos e inválidos via I/O (entrada/saída) sendo aplicado por desenvolvedores ou analistas de teste.

Uma unidade é a menor parte testável de um programa de computador. Em programação procedural, uma unidade pode ser uma função individual ou um procedimento. Idealmente, cada teste de unidade é independente dos demais, o que possibilita ao programador testar cada módulo isoladamente.

Relação de conceitos de testes de unidade:

I/O Input Output (Entrada e Saída): são todas as entradas e saídas existentes na programação.

Válidos

São entradas e saídas de dados comuns ao sistema e pertencem ao processo normal. Não apresentam tratamento além do normal já programado. No caso de retorno deverá seguir os padrões estabelecidos e não permitir retornos fora das regras especificadas.

Inválidos

São entradas e saídas de dados não comuns ao sistema. Apresentam tratamento para validar o tipo de dado inválido ou situação. Pode apresentar até dois retornos, uma mensagem para um log no sistema e uma mensagem com formatação e escrita adequada ao usuário.

> Ex.: Dividir (x int,y int)=z int .

Caso tenhamos x=1 e y=0, z será um valor com erro e deverá retornar uma mensagem ao usuário, avisando que a operação é inválida. Caso a expressão seja um dado comum do sistema, a autorização para tal validação deverá ser do usuário, pois faz parte do conjunto de regras de negócio. Não existe retorno inválido sem um tratamento. O tratamento genérico será apenas para condições não visíveis na regra e uso do sistema.

Domínio

Pode ser um campo, uma assinatura, um I/O, ou qualquer tipo de local que receba valores externos ao sistema. Todo domínio deve realizar consistências de dados válidos e inválidos. Um domínio só permite dados com a formatação igual ao que será armazenado.

> Ex.: Campo DDD deverá permitir números de até quatro casas não negativas ou a base de dados deve impedir a entrada de valores inválidos.

Receber e guardar o mesmo tipo de dado, o tamanho do campo que recebe os dados deve ser menor ou igual ao campo que irá armazenar os dados (em raros casos os campos de armazenamento são menores que os de exibição).

Em suma, domínio é o tipo de valor válido para cada campo. Como exemplo podemos citar: Campo nome: Dominio = tipo: string; tamanho:50. Ao aplicarmos o particionamento por equivalência e a análise por valor limite, poderemos criar as seguintes classes de testes.

Particionamento por Equivalência: campo nome:

- valor em branco; Cenário Negativo
- valor > 50; Cenário Negativo
- qualquer valor de 1 a 50; Cenário Positivo

Análise por Valor Limite:

campo nome: valor em branco; valores 49,50,51; Usamos um valor exatamente inferior e exatamente posterior ao valor do campo, devido ao fato dos erros aparecerem nas fronteiras da aplicação.

Tipos de valores

Existem diversos formatos de valores e cada um possui uma regra própria além das regras da região em que será usado.

Data

Deve validar anos iguais à faixa de datas existente na base. Deve ser feita conforme o padrão da região de uso. Validar o maior e o menor dia de todos os meses. Deve estar preparado para anos bissextos. Em caso de cálculo de feriados deve apresentar todos eles corretamente conforme as regiões definidas pelo usuário. Validar meses e dias invalidados conforme regra (X,X-1,X+1,(X+Y)/2,Y,Y-1,Y+1) sendo X o maior número e Y o menor. Verificar quantos anos o sistema deverá tratar anteriores e posteriores.

Números

Devem ser tratados no formato correto e com as regras para a região especificadas pelo usuário. Validar valores negativos ou sinais. Testar sempre o valor (um acima do maior, o maior, um abaixo do maior, alguns intermediários, um acima do menor, o menor, um abaixo do menor, e alguns valores inválidos) (X,X-1,X+1,(X+Y)/2,Y,Y-1,Y+1) sendo X o maior número e Y o menor número. Tratar valores fracionados conforme a quantidade de casas validadas pela empresa.

E-mail

Todo e-mail válido possui um "@" e um ".", sendo que o "@" nunca está no começo ou no fim, o "." deve estar sempre entre o "@" e o fim, o "." não pode estar junto com o "@", e antes do "@" não pode haver caracteres especiais. Um e-mail dever ter no mínimo 5 caracteres contando o "@" e o ".". Ex.: nome@dominio.com

Senha

Deve atender as normas da empresa. Não deve ser exibida durante a digitação; somente com caractere representativo. Deve ser confirmada a digitação. Não deve permitir ser copiada ou colada.

Domínio

Existem diversas formas para validar domínio e e-mail válido, porém devem ser aprovadas pela empresa, pois consomem comunicação com outros sites.

Campos especiais CPF, CNPJ, CEP, CNS, etc.

- Validar a formatação na entrada dos dados e na armazenagem.
- Devem atender as regras de cálculo existentes.
- Não podem ser dados viciados.
- Campos dependentes DDI-DDD-TEL e similares.
- Validar o preenchimento de ambos em validação única.
- Validar a formatação.
- Campos pré-preenchidos.
- Validar a formatação.
- Testar todas as regras de preenchimento.
- Força dados inválidos ou repetir o preenchimento.

Valores para teste

Existem diversos tipos de dados válidos que se tornam inválidos conforme a linguagem usada.

Sintaxes da linguagem

Muitas linguagens podem ter problemas ao tratar valores do tipo objeto, string e outros. Tais linguagens têm tratamentos especiais para estes valores para assim evitar erros ou em alguns casos não tem tratamento pois o erro é na codificação.

Erro na linguagem

Um exemplo claro de erro na linguagem são sistemas feitos na própria base de dados que não permitem pesquisas usando termos como else, while, for por serem palavras reservadas da linguagem. Atualmente a maior parte dos sistemas não apresenta erros com termos reservados.

Alteração de valores

Algumas linguagens, ao receberem determinados valores, alteram seus dados internos com tratamentos automáticos. No geral elas possuem travas para estes tratamentos. Exemplos de valores: 0x00, 00FF, 1.1, 1,1, 1^2, etc.

Interpretação de valores

Algumas linguagens interpretam diretamente valores sem tratar, passando eles para outra aplicação em uso. Este erro ocorre geralmente quando uma outra linguagem é usada como intermediária. Exemplo: em HTML (ASP, JSP, CGI, PHP e outras), quando passamos uma estrutura HTML(TAGs) para a página e esta estrutura é apresentada novamente, porém no corpo do HTML podemos ter dados incorporados que mudam o resultado apresentado. Este tipo de erro é chamado de inject. Outro exemplo de inject é o SQL em junção com outras linguagens, recebendo diretamente linhas de comando, podendo alterar pesquisas ou apagar tabelas de dados inteiras. Ex.: HTML(<TAGs>,javascript) SQL(términos de linhas seguidas de comandos (1;' while 1=1 print 1)). Estes erros ocorrem pois temos uma segunda linguagem que faz o intermédio entre os comandos. Para tratar estes tipos de erros, devemos sempre conhecer bem as linguagens usadas, evitando transferir direto dados do usuário que acabem por virar comandos.

Cálculos mesclados

Ficar atento aos valores que são permitidos pois o usuário pode acabar por inserir dados inválidos. Excreções como (-,(),{},[],',^,x,*,/,. e outras) podem mudar totalmente o valor que era esperado, podendo até colocar valores além dos permitidos pela especificação do sistema. Ponto flutuante ou Estouros. Erros de ponto flutuante dificilmente são identificados durante a especificação de um sistema ou acabam por ocorrer com a maturidade da empresa e seus dados. O que acarreta tais erros é: Não validar os dados corretamente. Não planejar bem o crescimento dos dados. Não ter uma área de armazenamento maior que a área de entrada de dados. Na época da inflação eram comuns sistemas terem um campo de valor de 9 casas e uma área de armazenamento com 12 casas. Outro erro comum que permite ocorrer estouro é uso de objeto e matrizes sem estudo, qualquer laço entre eles ou o próprio crescimento descontrolado.

Formatações de região ou não

É bom sempre forçar o sistema a usar uma região única caso não se tratem todos os valores com os quais ele pode trabalhar. O não tratamento da região pode tanto mudar o valor de um pagamento como alterar a data de uma cobrança. Caso o cliente não defina uma região devemos cobrar dele tal definição e devemos

sempre estar atentos aos padrões já existentes, evitando misturar dados como DD/MM com MM/DD e 1.100 com 1,1.

Medidas

Um dado extremamente importante são as medidas. Elas devem ser sempre iguais no sistema inteiro ou serem tratadas para isso. É extremamente recomendado usar uma medida única no sistema evitando assim erros como mandar um foguete para um planeta e errarmos ele por 65.000km (caso da NASA de um sistema que misturava dados em polegadas e dados em metros). Os erros de medidas não se aplicam somente a fórmulas diferentes, mas também a tamanhos (ex.: 100cm=1m=0.001km).

Data

A formatação de data é o erro mais comum da maioria dos sistemas. Muitas vezes o sistema foi todo desenvolvido para dd/mm/yyyy, porém, ao ser implantado, descobre-se que a data da base é mm/dd/yyyy. Para evitar este tipo de erro deve ser sempre requisitado ao usuário o seguinte: 1°: qual a configuração do servidor em que será usado o sistema e qual seu idioma. 2°: qual a configuração dos clientes que usam o sistema e quais os seus idiomas. Com estas informações é possível validar para qual data devemos converter os dados ou qual data devemos desenvolver.

Navegação

Pode ser tanto a navegação sobre o sistema ou a navegação entre sistemas. Sendo que somente a navegação dentro do bloco é validada como teste de unidade. Os itens Interna ao Sistema e Entre Sistemas são testes de integração.

Local

Navegação entre os campos ou funcionalidades da tela. Deve sempre atender a ordem: esquerda para direita, de cima para baixo. Caso o usuário use outro padrão, ele deve ter especificado previamente, pois está fora dos padrões de mercado. A navegação local deve também validar os valores entre campos e seus resultados. Atentar para alteração de valores em campos diferentes do usado. Algumas regras podem forçar as alterações de campos secundários. Devem-se validar todas as interações que geram estas alterações com dados válidos e inválidos.

Interna ao sistema. Teste Integrado

Consiste na navegação entre as telas do sistema ou suas estruturas internas. Deve seguir o padrão definido pelo usuário. Devemos atentar para alterações entre telas e todas as suas iterações (válidas e inválidas). Devemos sempre listar as funcionalidades de navegação em um objeto único. Ex.: Apontar o objeto que controla o botão (x) (o objeto do (alt+f4)) para o mesmo objeto que controla o fechar do menu. A navegação interna deve ser sempre bem mapeada e deve ser tratada por quem libera uma funcionalidade

ou um responsável por integrar ela ao sistema. Muitos sistemas travam pois suas funções ficam congestionadas entre elas por não serem tratadas com sinalização e eventos de integração (como funções de fechar, funções de novo, funções de próxima e anterior). Esse tipo de teste também é conhecido como Teste de Instrumentação.

Entre sistemas. Teste Integrado

Quando temos a integração entre sistemas deve ser programada sempre a prioridade entre eles, quem tem prioridade de gravar ou apagar um dado. Deve ser validada a chamada entre eles para se evitar congestionamento. Atualização de dados. Retorno e saída de dados. Formatações de entrada e saída.

Dicas para teste de unidade

Sete regras,[1][2] ou boas práticas, para implementação de testes de unidade:

1. Escreva o teste primeiro
2. Nunca inicie com um teste que será bem sucedido
3. Comece com valores nulos, ou algo que não funcione
4. Não fique com medo de fazer algo trivial para fazer o teste funcionar
5. Desacoplamento e testabilidade andam de mãos dadas
6. Utilize teste de mock
7. Divida o seu teste em 3 partes, Preparação, Ação e Verificação (em inglês: Arrange, Act and Assert)[3]

Existem outras listas de boas práticas para implementar teste de unidade que podem ser encontradas na comunidade.

Ver também

- Teste de software
- Gestão da qualidade
- ISO
- BIA
- Underwriters Laboratories

Referências

1. «The Six Rules of Unit Testing» (http://testingreflections.com/node/403)
2. «Charles' Six Rules of Unit Testing» (https://fishbowl.pastiche.org/2002/07/25/charles_six_rules_of_unit_testing/). *The Fishbowl: Charles Miller's Weblog* (em inglês). Consultado em 3 de setembro de 2021
3. «Arrange Act Assert» (http://wiki.c2.com/?ArrangeActAssert). *wiki.c2.com*. Consultado em 3 de setembro de 2021