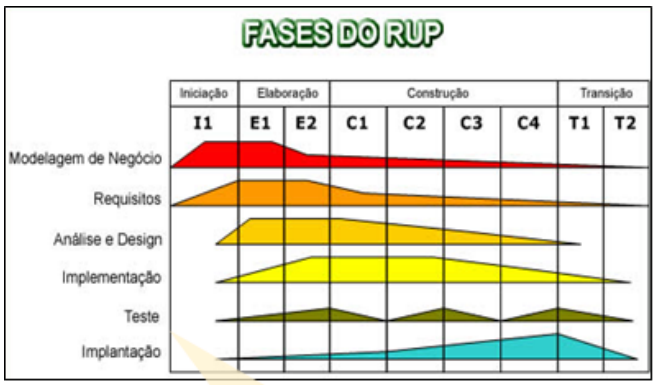
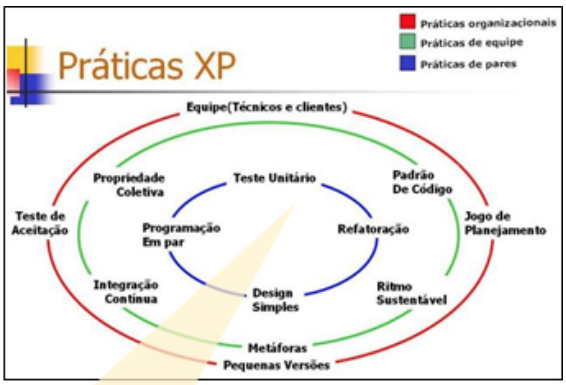
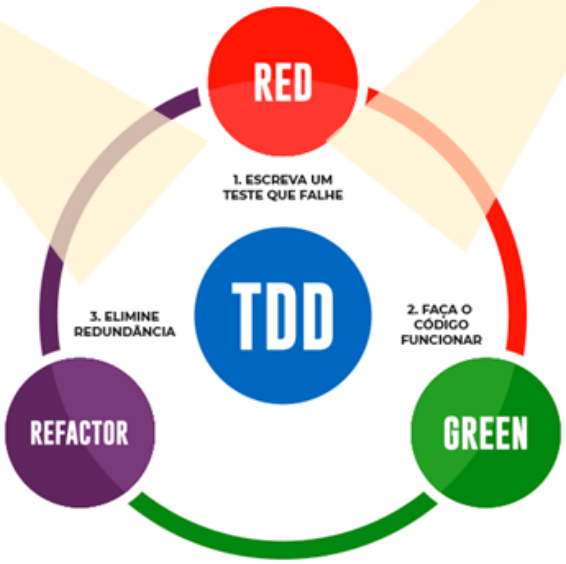
Artigo
Yuri Marx Perei... · Abr. 5, 2022 10min de leitura

Open Exchange

Testes Unitários: Qualidade do seu Código ObjectScript







Nas principais metodologias de desenvolvimento de software sempre há um capítulo dedicado aos testes. Trata-se de uma abordagem obrigatória para alcançar qualidade nas entregas de forma contínua.

Existem dois tipos de teste:

1. **Teste Caixa Branca**: são testes que examinam a qualidade do código-fonte e das funcionalidades da aplicação. Neste tipo de teste temos:
    1. **Análise estática**: são utilizadas soluções de análise estática (não há execução da funcionalidade no momento do teste) do código-fonte, onde são avaliadas padrões de nomenclatura, identação, variáveis declaradas e não utilizadas, índice de acoplamento entre componentes, dentre outros critérios definidos dentro das soluções de análise. Geralmente, dentro do ambiente de desenvolvimento são apontadas as linhas de código-fonte com problemas de qualidade, permitindo ao desenvolvedor atuar para resolver o problema.
    2. **Teste unitário**: são criadas classes de teste (Test Unit - Teste Unitário), uma para cada tópico de funcionalidade (Manter Clientes, Cobrança de Transações, etc.), agrupadas em Suites de teste (conjunto de Test Units), uma suite para cada módulo da aplicação (Módulo de Cadastro, Módulo de Vendas, etc.) ou para a aplicação como um todo. Geralmente, é emitido um Relatório em HTML com os resultados encontrados.

2. **Teste Caixa Preta:** são testes realizados no binário da aplicação, sem acesso ao código de fonte da aplicação. Neste tipo temos:
    1. **Teste de Desempenho**: são definidos scripts de carga de dados e de execução das principais funcionalidades para avaliar se a aplicação e o ambiente de execução irão suportar a quantidade de transações e usuários previstos.
    2. **Teste de Penetração - PenTest**: são empregadas soluções de análise de vulnerabilidades dos endereços IP e portas da aplicação, dos produtos que hospedam a solução (servidor web, de aplicação e de banco de dados) e os endpoints da aplicação (API, Serviços Web, Diretórios de Troca de Arquivos e demais pontos de interação). Geralmente, é emitido um Relatório em PDF com as vulnerabilidades encontradas.
    3. **Teste Funcional**: analistas e testadores escrevem e executam diversos cenários de testes a partir das interfaces visuais da aplicação, visando identificar falhas funcionais (funcionalidade com erro sob o ponto de vista do usuário final). Hoje já existem soluções que também automatizam estes testes a partir das interfaces visuais.

Um objetivo importante da Disciplina de Testes é encontrar o ponto ideal de cobertura dos testes (funcionalidades cobertas por testes). Nos projetos que estou envolvido, o objetivo é alcançar ao menos 75% das funcionalidades. O grande segredo é concentrar seus testes na camada de negócio da aplicação, pois é lá onde se encontram as regras de negócio e a implementação dos requisitos funcionais entregues ao usuário final. Diferentes interfaces visuais e pontos de integração sempre acabam por requisitar a camada de negócio. Desta forma, ao automatizar os testes da camada funcional, a cobertura dos testes terá um índice relavante.

Este artigo apresenta como automatizar os testes unitários da camada de negócio da sua aplicação IRIS, com o objetivo de demonstrar como alcançar bons índices de cobertura de testes e boa qualidade nas aplicações IRIS entregues ao usuário final.

Baixe a aplicação de exemplo a ser testada

Vá ate https://openexchange.intersystems.com/package/global-mindmap e clique no botão Github. Siga os passos a seguir:

1. Faça o Clone/git pull do repositório dentro do diretório de sua escolha

```
$ git clone https://github.com/yurimarx/global-mindmap.git
```

2. Faça o docker build dentro do diretório da aplicação clonada:

```
$ docker-compose build
```

3. Execute o container IRIS:

```
$ docker-compose up -d
```

Veja a aplicação funcionando

- No seu ambiente: http://localhost:3000
- No YouTube: https://www.youtube.com/watch?v=M6EupvAATro

Agora vamos aos testes

A classe de negócio

Classe de negócio alvo do teste

A classe de negócio possui 3 funcionalidades para serem testadas:

1. **StoreMindmapNode**: funcionalidade utilizada para gravar os dados de um nó de mapa mental no banco de dados na forma de globais.
2. **GetMindmap**: funcionalidade utilizada para retornar todos os nós do mapa mental armazenados em globais, ou seja o mapa mental como um todo.
3. **DeleteMindmapNode**: funcionalidade utilizada para excluir um nó do mapa mental do banco de dados (exclui a global que armazena o nó).

Criando a Suite de Teste - Agrupador dos Casos de Teste

Basta criar um Diretório na raiz do diretório src, com o nome do suite de testes, neste exemplo o nome criado foi UnitTests.

Criando os Casos de Teste do Suite de Testes

É necessário criar classes de teste que herdem de %UnitTest.TestCase. O nome da classe, por boa prática, deve ser NomeDaClasseASerTestada + Test. No nosso exemplo a classe se chama GlobalMindMapServiceTest. Veja o conteúdo da classe:

Classe de Caso de Teste

Perceba que para cada funcionalidade a ser testada, temos um Method com a seguinte convenção de nome: Test + NomeDoMetodoASerTestado. É importante que cada método de teste faça todo o necessário para que o teste possa ser executado. No teste de Delete, por exemplo, primeiro é criado um registro, que será utilizado para excluir e assim testar se a exclusão está funcionando.

Para cada Method deve existir ao menos um chamada para $$$Assert.... Esta macro é a execução e registro da condição do teste em si. No nosso exemplo, utilizamos $$$AssertEquals em todos os métodos. Neste caso, se a condição do teste for igual ao resultado da execução do método de negócio, o teste é marcado como bem sucedido, senão é marcado como mal sucedido. Há várias opções de Assert, veja mais detalhes em https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls....

Também possível é utilizar o Method OnBeforeAllTests para criar os dados e condições necessárias aos testes e utilizar OnAfterAllTests para limpar os dados e os ambientes para o estado original.

Realizando a execução dos testes

Com a suite e os casos de teste prontos está na hora de realizar os testes. Para isto é necessário acessar o Terminal, no namespace onde se encontram as classes, veja os passos para o exemplo deste artigo:

1. No Terminal, namespace IRISAPP, é necessário ir para o namespace que contém as classes de negócio e teste:

```
USER>zn "IRISAPP"
```

2. É necessário apontar a pasta onde se encontra a pasta de Suite de teste (no exemplo é a pasta UnitTests dentro de src)

```
IRISAPP>Set ^UnitTestRoot="/opt/irisbuild/src"
```

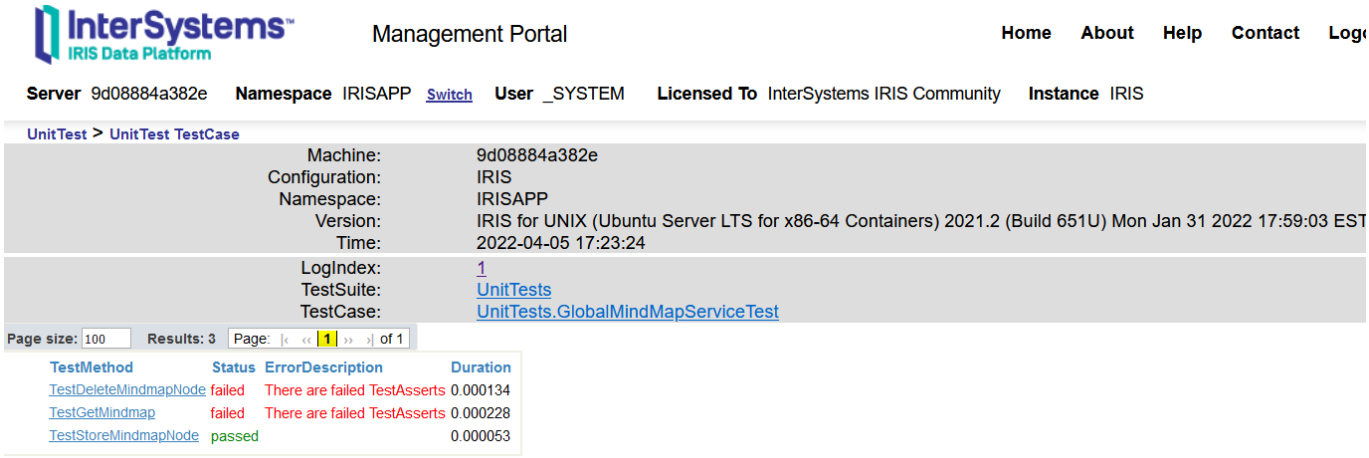
3. Execute os testes unitários

```
IRISAPP>do ##class(%UnitTest.Manager).RunTest("UnitTests")
```

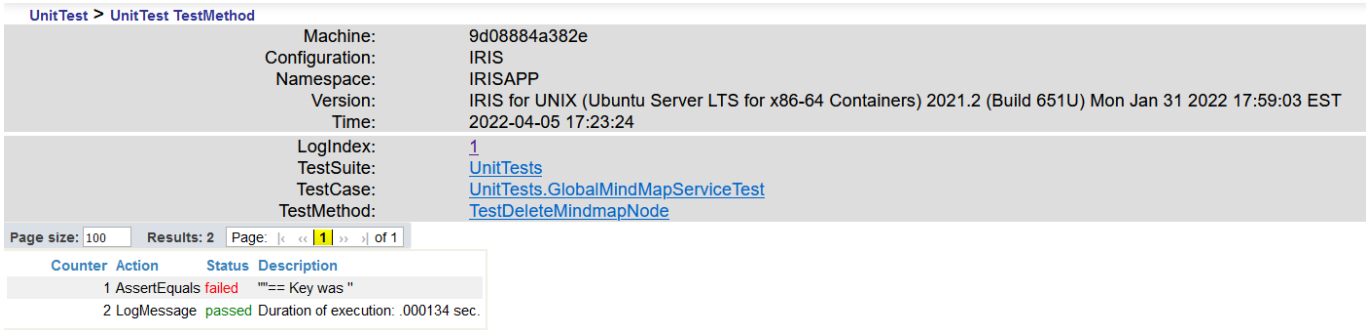
4. Confira os resultados em http://localhost:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=1&$NAMESPACE=IRISAPP

Analisando os resultados

O relatório do passo anterior retorna neste layout:



É possível saber o que passou (verde) e o que falhou (vermelho). Ao clicar no vermelho, é possível saber onde ocorreu o erro:



Basta agora agir nas correções.

Se gostou da aplicação de exemplo, vote nela no Global Contest.

#Testes #InterSystems IRIS
Confira o aplicativo relacionado no InterSystems Open Exchange

URL de origem:https://pt.community.intersystems.com/post/testes-unit%C3%A1rios-qualidade-do-seu-c%C3%B3digo-objectscript