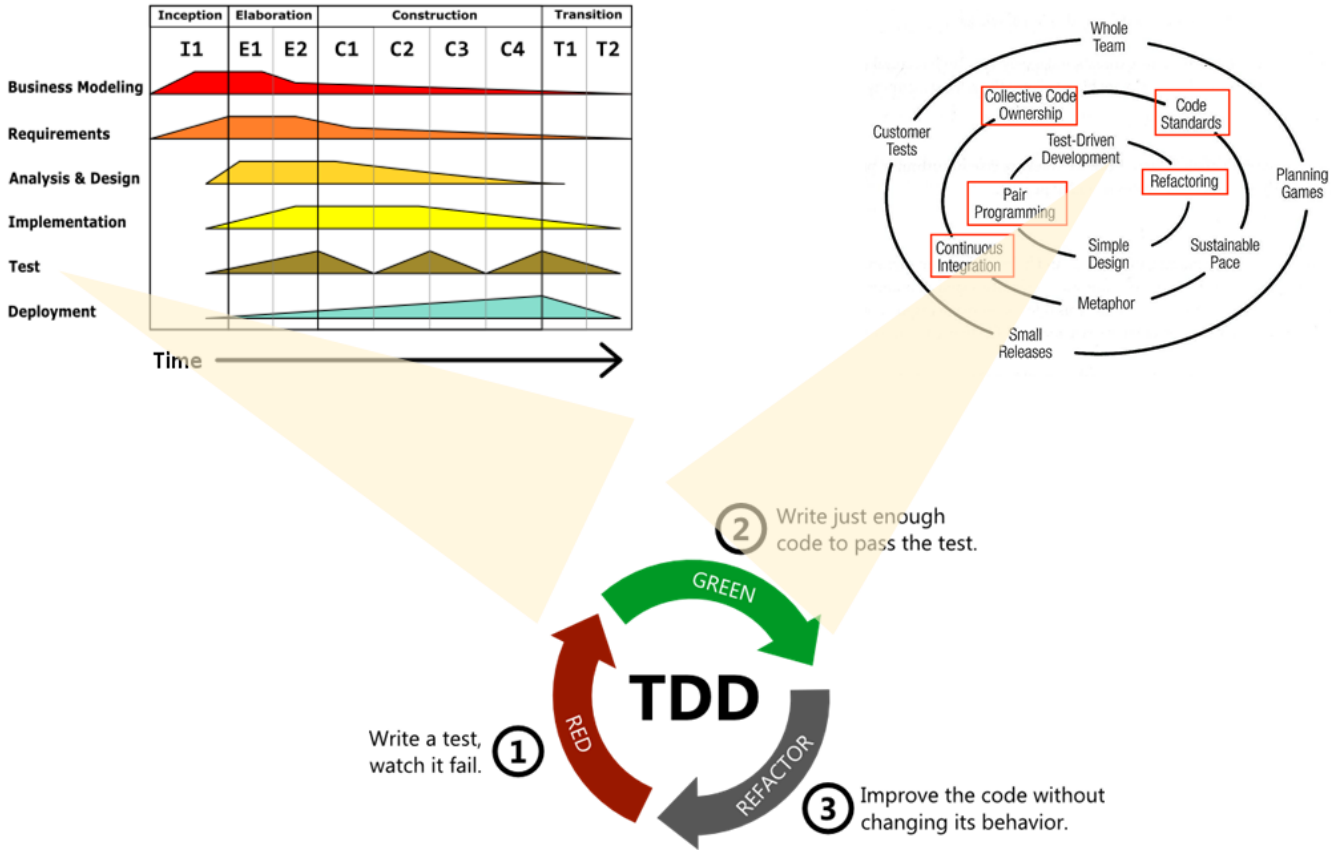
Article
Yuri Marx · Apr 6 10m read

Open Exchange

Unit Tests: Quality of your ObjectScript Code



In the main software development methodologies there is always a chapter dedicated to testing. It is a mandatory approach to achieving quality in deliveries on an ongoing basis.

There are two types of test:

1. **White Box Test**: These are tests that examine the quality of the source code and application functionality. In this type of test we have:
    1. **Static analysis**: static analysis solutions are used (there is no functionality execution at the time of testing) of the source code, where naming patterns, indentation, declared and unused variables, coupling index between components, among other defined criteria are evaluated within analysis solutions. Generally, within the development environment, lines of source code with quality problems are pointed out, allowing the developer to act to solve the problem.
    2. **Unit test**: test classes (Test Unit) are created, one for each functionality topic (Maintain Customers, Transaction Collection, etc.), grouped in Test Suites (set of Test Units), one suite for each application module (Registration Module, Sales Module, etc.) or for the application as a whole. Generally, an HTML Report is issued with the results found.
2. **Black Box Test:** tests performed on the application's binary, without access to the application's source code. In this type we have:
    1. **Performance Test**: Data load and execution scripts are defined for the main functionalities to assess whether the application and execution environment will support the amount of transactions and

users expected.
2. **Penetration Test - PenTest**: vulnerability analysis solutions are used for the IP addresses and ports of the application, the products that host the solution (web server, application and database) and the application endpoints (API, Web Services, File Exchange Directories and other interaction points). Usually, a PDF report is issued with the vulnerabilities found.
3. **Functional Testing**: Test Analysts and testers write and execute several test scenarios from the application's visual interfaces, aiming to identify functional flaws (functionality with error from the end-user's point of view). Today there are solutions that also automate these tests from the visual interfaces.

An important objective of the Testing Discipline is to find the sweet spot of test coverage (features covered by tests). In the projects I'm involved in, the goal is to achieve at least 75% of the functionality. The big secret is to focus your tests on the business layer of the application, because that's where the business rules and the implementation of the functional requirements delivered to the end user are. Different visual interfaces and integration points always end up requiring the business layer. In this way, when automating the tests of the functional layer, the coverage of the tests will have a relevant rate.

This article presents how to automate the unit tests of the business layer of your IRIS application, in order to demonstrate how to achieve good test coverage rates and good quality in IRIS applications delivered to the end user.

Download the app to be tested

Go to https://openexchange.intersystems.com/package/global-mindmap and click Github buttion. Follow these steps:

1. Clone the project

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

2. Do docker build inside in the project root folder:

```
$ docker-compose build
```

3. Execute the Docker Container:

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

See the app in action

- Local env: http://localhost:3000
- On YouTube: https://www.youtube.com/watch?v=M6EupvAATro

Now, go to the tests

Business Class to be tested

Business Class Target for testing

The business class has 3 features to be tested:

1. **StoreMindmapNode**: functionality used to record data from a mind map node in the database in the form of globals.
2. **GetMindmap**: functionality used to return all the nodes of the mind map stored in globals, that is, the mind map as a whole.
3. **DeleteMindmapNode**: functionality used to delete a mental map node from the database (deletes the global node that stores the node).

Creating the test suite - A set of Test Cases

Just create a Directory at the root of the src directory, with the name of the test suite, in this example the name created was UnitTests.

Creating Test Suite Test Cases

You must create test classes that inherit from %UnitTest.TestCase. The class name, as a good practice, should be TestClassNameToBeTested + Test. In our example the class is called GlobalMindMapServiceTest. See the content of the class:

Test case class

Notice that for each functionality to be tested, we have a Method with the following naming convention: Test +NameOfMethodToBeTested. It is important that each test method does everything necessary for the test to run. In the Delete test, for example, a record is first created, which will be used to delete and thus test if the deletion is working.

For each Method there must be at least one call to $$$Assert.... This macro is the execution and recording of the test condition itself. In our example, we use $$$AssertEquals in all methods. In this case, if the test condition is equal to the result of executing the business method, the test is marked as successful, otherwise it is marked as unsuccessful. There are several Assert options, see more details at https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls....

It is also possible to use the OnBeforeAllTests Method to create the data and conditions necessary for the tests and use OnAfterAllTests to clean the data and environments to their original state.

Executing the tests

With the suite and test cases ready, it's time to perform the tests. For this you need to access the Terminal, in the namespace where the classes are located, see the steps for the example in this article:

1. On Terminal, namespace IRISAPP, it is necessary to go to the namespace that contains the business and test classes:

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

2. It is necessary to point to the folder where the Test Suite folder is located (in the example it is the UnitTests folder inside src)

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

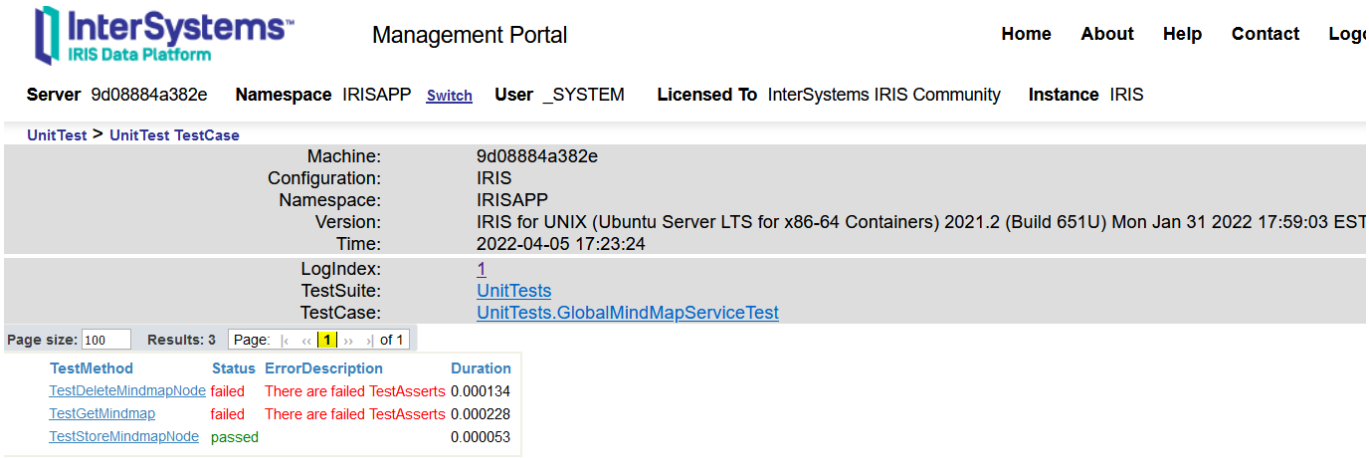
3. Execute the unit tests

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

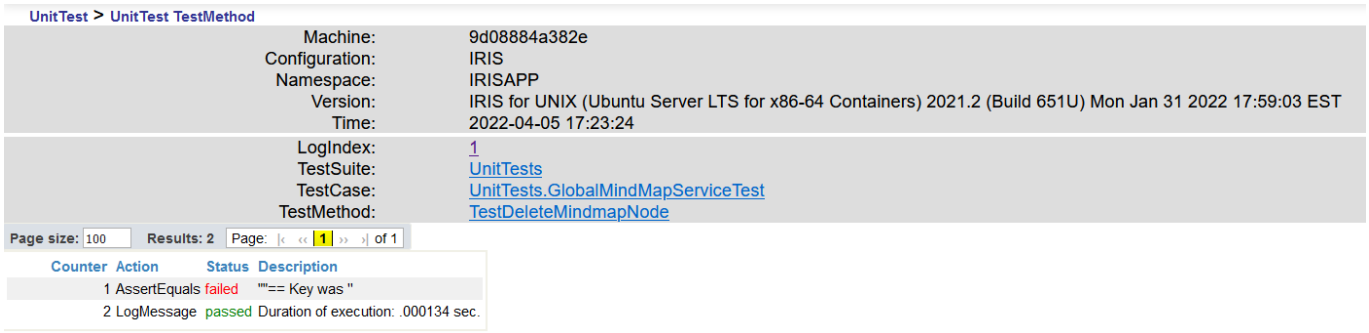
4. See the results in http://localhost:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=1&$NAMESPACE=IRISAPP

Analyzing the Results

The report from the previous step returns in this layout:



It is possible to know what passed (green) and what failed (red). By clicking on red, it is possible to know where the error occurred:



Now just act on the fixes.

If you liked the sample application, vote for it in the Global Contest.

#Testing #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL: https://community.intersystems.com/post/unit-tests-quality-your-objectscript-code