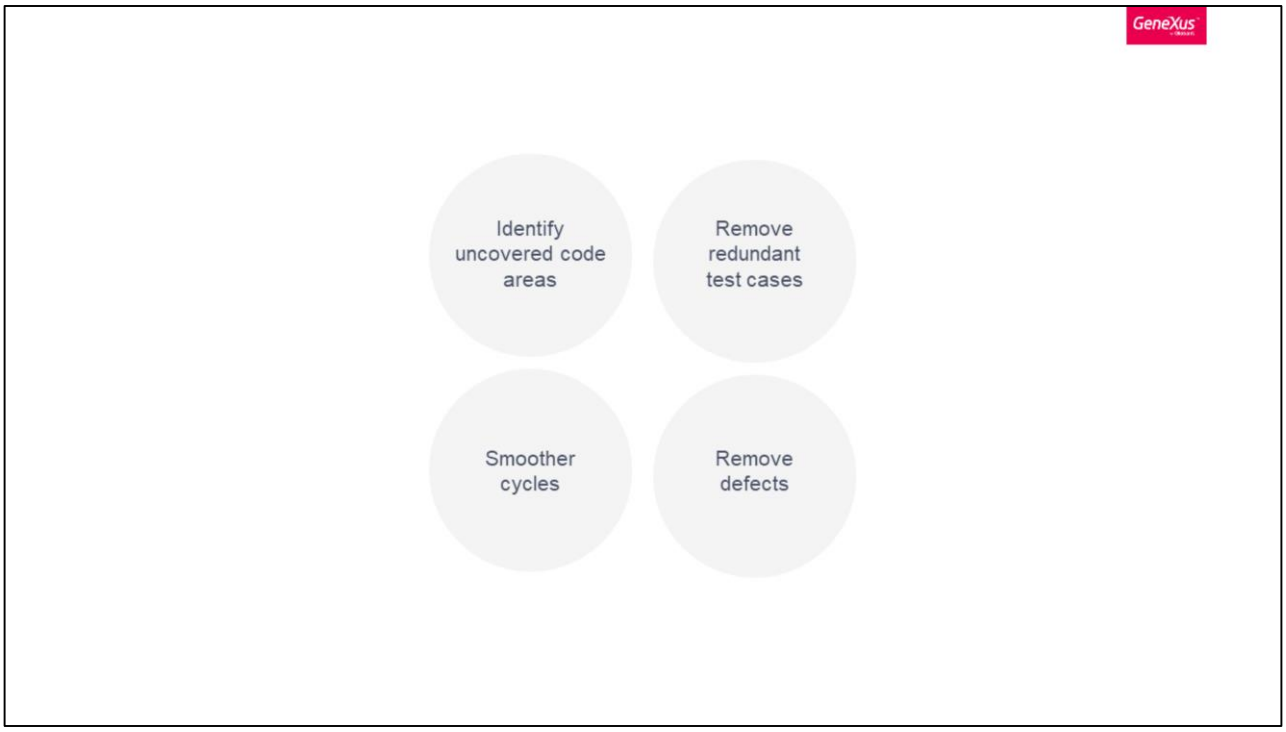
GeneXus™
by Globant

Test Coverage

GeneXus

Benefits of Test Coverage

GeneXus™
by Globant



Test coverage helps to monitor the quality of testing, and assists testers to create tests that cover areas that are missing or not validated yet. With a more structured approach, an aim at 100% requirement coverage and effective testing methods, you will not compromise on quality.

The benefits of this feature are:

(Identify code areas uncovered)
Test coverage helps you unearth areas of a program that have not been covered by a set of test cases. It helps make your application more robust and error-free.

(Removes redundant test cases)
Test coverage is especially useful in identifying and removing test cases that don't make much sense in the current project. Your developers can report these cases to remove them and make the overall code lighter.

(Smoother testing cycles)
You can prevent defect leakage using Test coverage analysis. Test coverage also helps in Regression testing, test case prioritization, test suite augmentation and test suite minimization. All this leads to smoother yet efficient testing cycles.

(Removes defects at early stages)
You can identify gaps in requirements, test cases and defects at early stages of your product development life cycle. It saves you from a lot of headaches later.

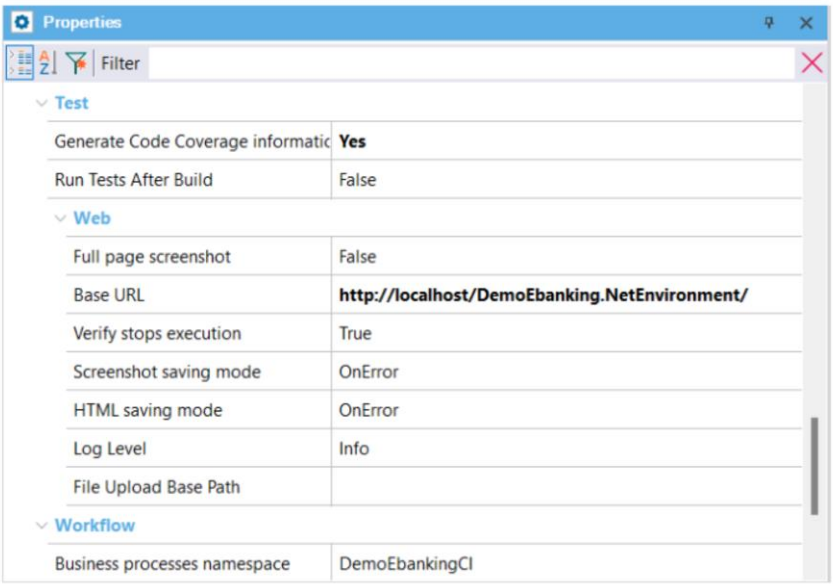
Test Coverage in GeneXus

GeneXus™
by Globant

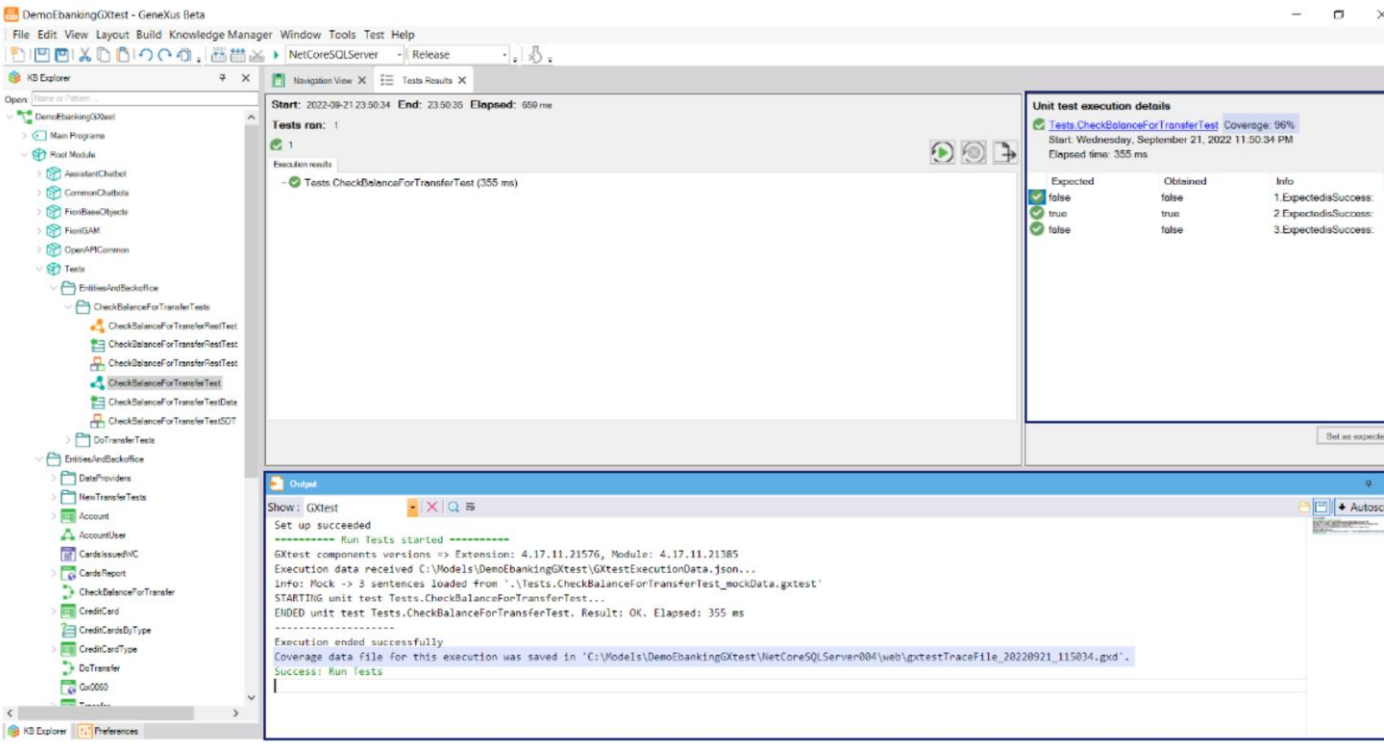
Test coverage is defined as a technique which determines whether our test cases are actually covering the application code and how much code is exercised when we run those test cases.

Agile methodologies indicate that 80% of the logic should be verified with unit tests, which are executed early in the development cycle and have a low maintenance cost as can be seen in the pyramid.

By default, in the GeneXus IDE, this feature comes disabled. To use it, the first thing to do is to enable Code Coverage for your environment.



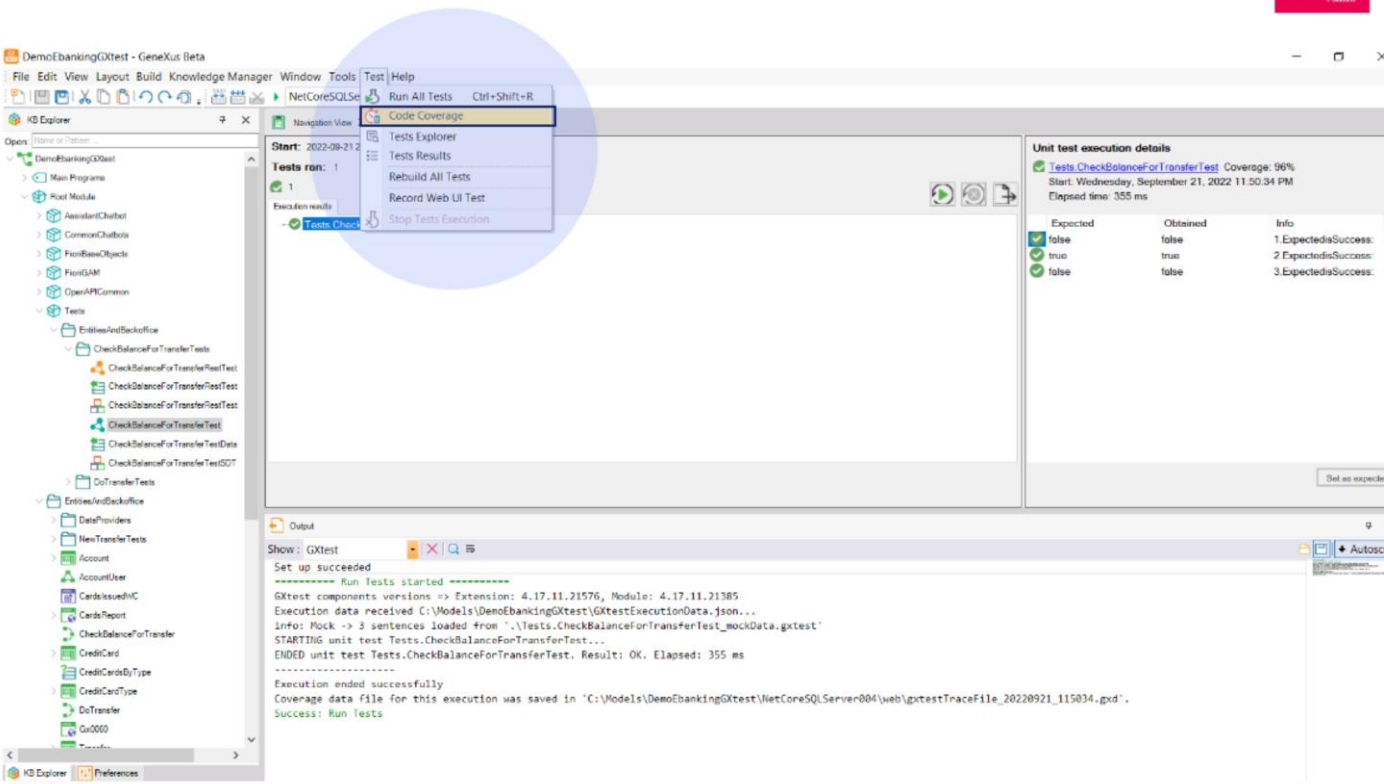
To do this, set the ***Generate Code Coverage information*** property in the application environment to 'Yes' and **rebuild all the KB**.



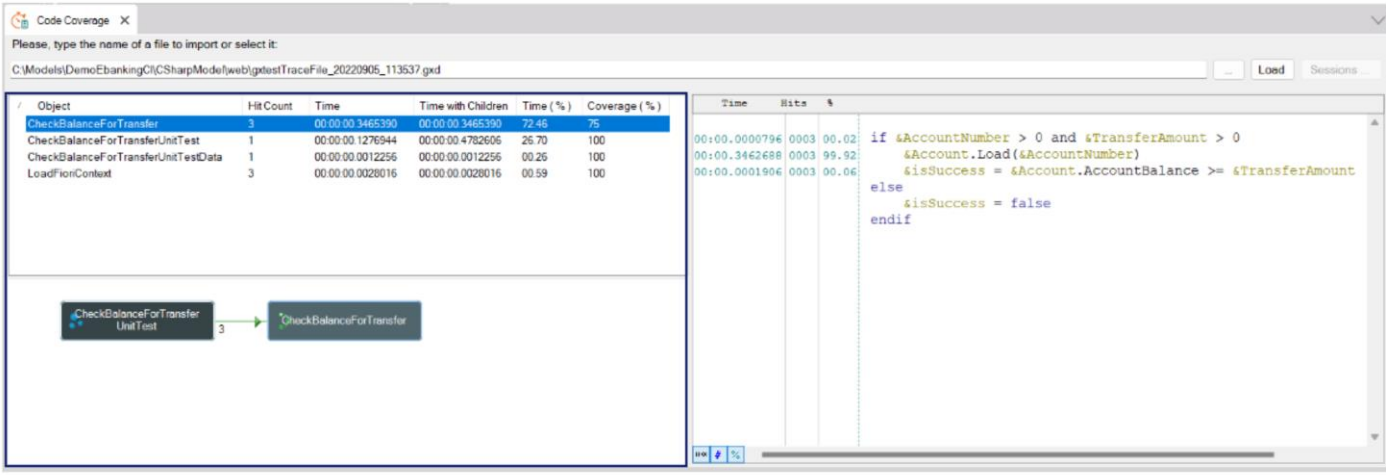
Once the Rebuild All operation finishes, you will be able to visualize the Coverage execution metric after each tests execution.

In the CheckBalanceForTransferUnitTest test execution you can visualize the Coverage percent in the Test Results panel. In the Output GeneXus panel, you will find the path of the Coverage data file for this execution. Coverage information is always related to a particular execution.

The displayed coverage value is the percentage of total lines of all the objects called by the tests aggregated in the execution performed. This means that if you execute 2 tests that each covers half of the lines of an object, the coverage displayed will be 100% as both tests aggregated cover all the lines of the tested object.



To open the coverage file and see the details of the Test Coverage go to *Test -> Code Coverage*



Load the Coverage data execution file selecting "..." button and then clicking *Load* you will see Coverage detail information.

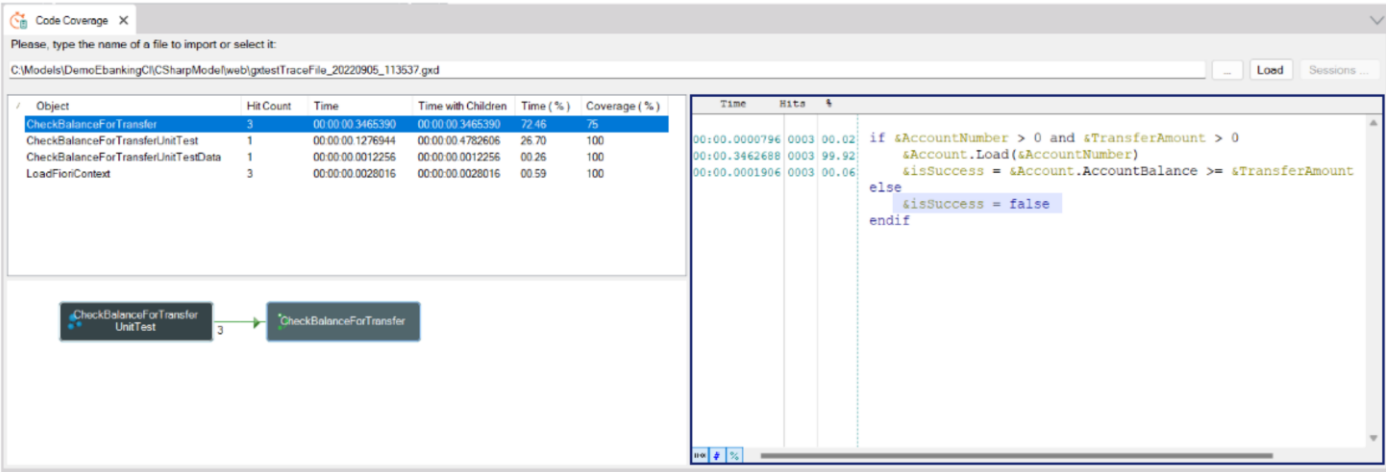
On the left section, you can see every object involved in the execution and its respective execution information when you select them.

The **Hit Count or Hits** is the number of times the object has been executed
The **Time** is the Elapsed Time
The **Time with Children** is also the Elapsed Time, adding the elapsed time of the objects called by it
The **Time (%)** is the Total percentage of the elapsed time related to the rest
The **Coverage (%)** is the Percentage of coverage, that means the lines that were executed over total number of lines.

When an object from this list is selected, below a graph

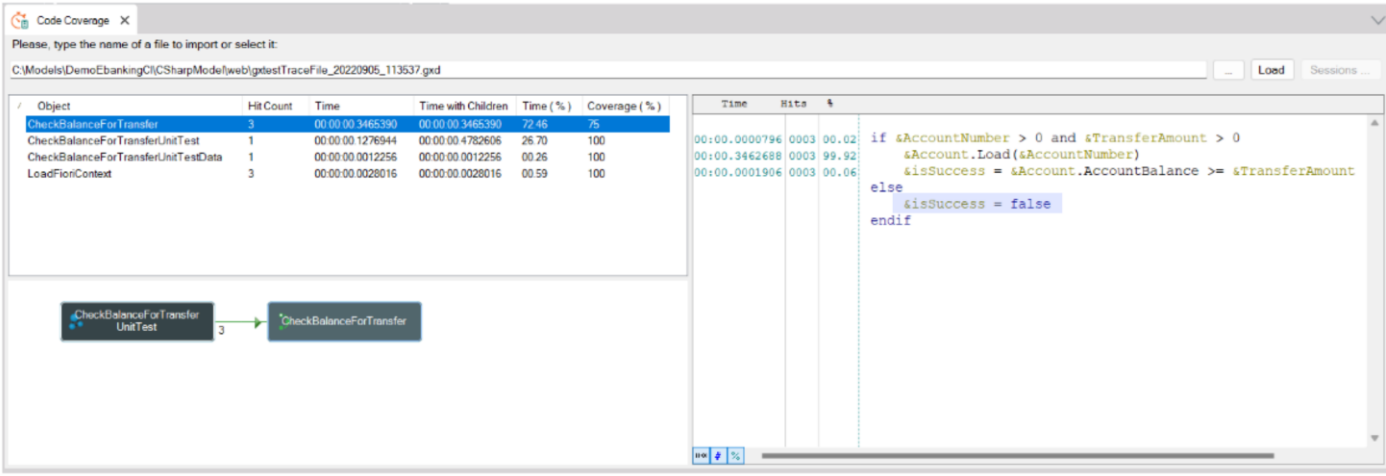
indicating the call tree is shown. For example, in the current screenshot you can see that CheckBalanceForTransfer is called by CheckBalanceForTransferUnitTest.

On the right panel, the line codes with their respective trace information are shown.



Note that it may have lines without information, that means that during the execution these lines weren't executed.

In this example, we can see that the else condition was not covered.



So, let’s understand how to use the Test coverage feature.

One of the main variables to monitor is the “Hit count” by which we visualize the number of times the object has been exercised. Also in the right detail, we can see the Hits for each code line of the procedure. With both data, we can gather that the procedure CheckBalanceForTransfer has 3 test cases, and the three cases go into the “if” condition. In consequence, it would be convenient to verify that redundant test cases are not being executed in the "if" condition.

The most important metric is the Coverage percentage column, we can see that 75% of the procedure CheckBalanceForTransfer was exercised. That means that not all code lines were tested, and we should add more test cases to get full test coverage. As we can see in the right detail, the else condition is not exercised by the test.

So, the developer should add a test case for this condition.

CheckBalanceForTransferUnitTestData

Source | Rules | Variables | Help | Documentation

```
CheckBalanceForTransferUnitTestSDT
{
    TestCaseId = '1'
    AccountNumber = 5
    TransferAmount = 200
    ExpectedisSuccess = false
    MsgisSuccess = ''
}

CheckBalanceForTransferUnitTestSDT
{
    TestCaseId = '2'
    AccountNumber =  6
    TransferAmount = 200
    ExpectedisSuccess = true
    MsgisSuccess = ''
}

CheckBalanceForTransferUnitTestSDT
{
    TestCaseId = '3'
    AccountNumber =  8
    TransferAmount = 200
    ExpectedisSuccess = false
    MsgisSuccess = ''
}

CheckBalanceForTransferUnitTestSDT
{
    TestCaseId = '4'
    AccountNumber =  6
    TransferAmount = -200
    ExpectedisSuccess = false
    MsgisSuccess = ''
}
```

CheckBalanceForTransfer

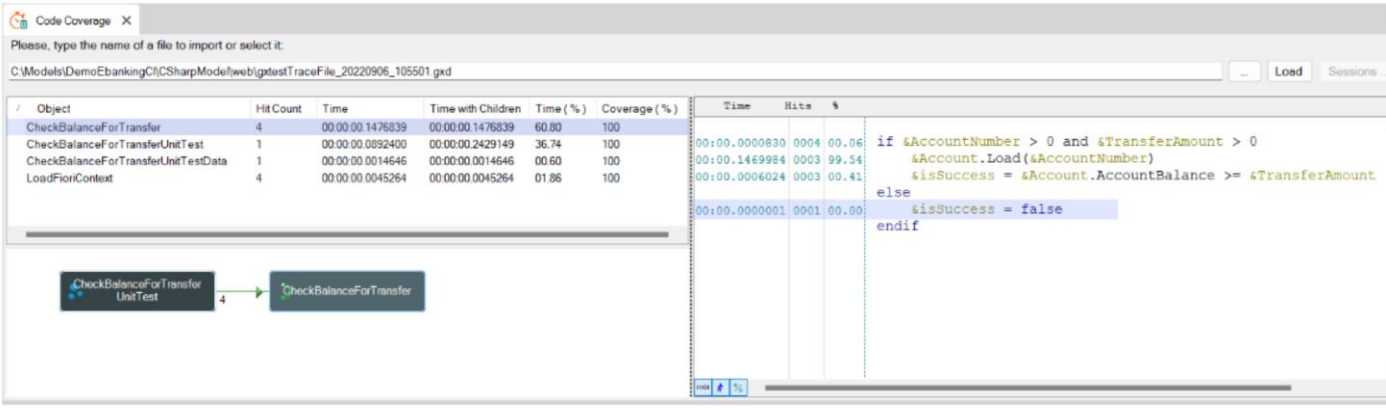
Source | Layout | Rules | Conditions | Variables | Help | Documentation

Subroutines

```
if &AccountNumber > 0 and &TransferAmount > 0
    &Account.Load(&AccountNumber)
    &isSuccess = &Account.AccountBalance >= &TransferAmount
else
    &isSuccess = false
endif
```

So, we go to the test cases defined in the data provider and we will add the test cases for the else condition.

In this example, we will add one more test case with id 4 in which I defined a negative transfer amount to execute the else condition of the proc.



After running the unit test again, we will see the impact in the Coverage metric of the procedure.

We can see in the Coverage (%) column that the procedure has the 100% because all code lines were exercised? in this unit test execution.

GeneXus™
by Globant

training.genexus.com
wiki.genexus.com