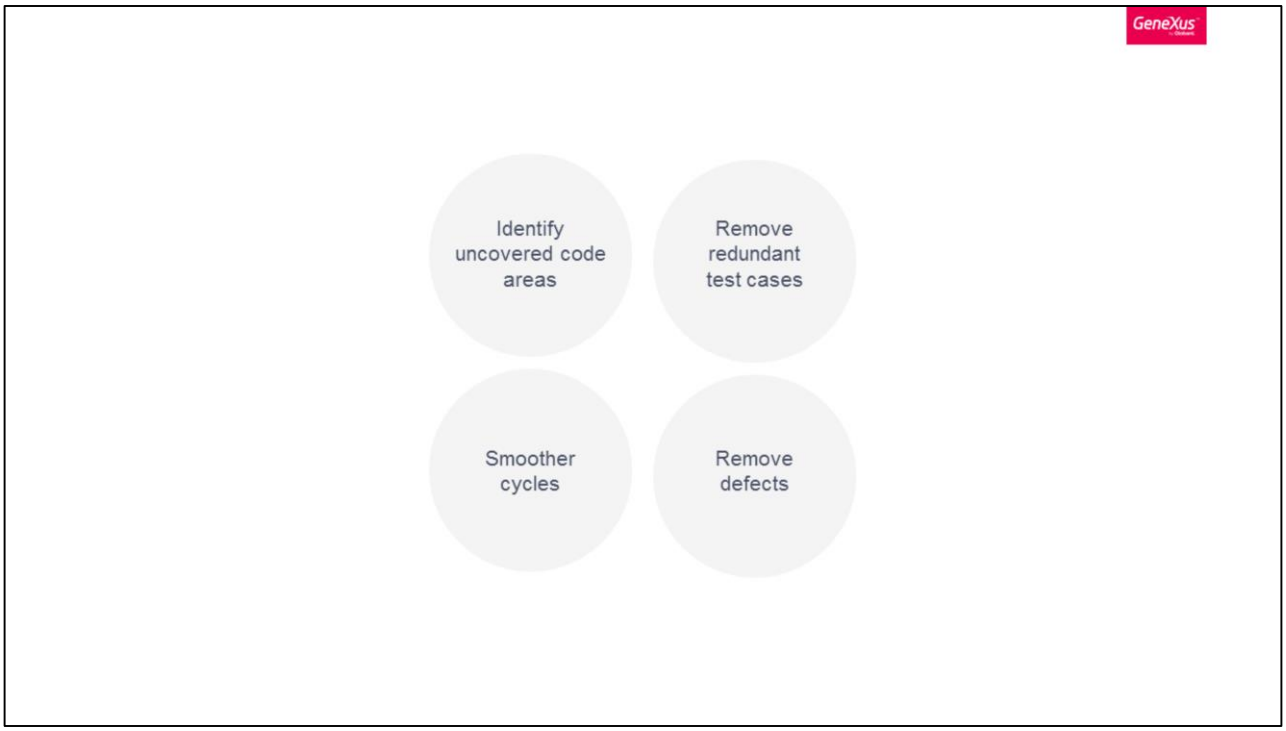

GeneXus™
by Globant


テストカバレッジ
GeneXus

テストカバレッジの有用性

GeneXus by Globant



テストの品質を監視し、テスト担当者が、不足している部分やまだ検証されていない部分をカバーするテストを作成するには、テストカバレッジが役立ちます。
品質を維持するためには、より構造化されたアプローチや、100%の要件カバレッジを目指すこと、そして、効果的なテスト手法が必要です。

この機能のメリットは以下の４つです。

・カバーされていないコード領域の特定
テストカバレッジは、テストケースのセットでカバーされていないコード領域を特定するのに役立ちます。
これにより、テスト漏れをなくし、アプリケーションをより堅牢で、エラーのないものにすることができます。

・冗長なテストケースの削除
テストカバレッジは、現在のプロジェクトにおいてあまり意味のないテストケースを特定し、削除する際に特に有効です。
開発者は、これらのケースを報告して削除し、コード全体を軽くすることができます。

・テストサイクルの円滑化
テストカバレッジ分析により、瑕疵の漏れを防ぐことができます。
また、テストカバレッジはリグレッションテスト、テストケースの優先順位付け、テ

ストスイートの拡張、テストスイートの最小化にも役立ちます。
これらにより、テストサイクルをより円滑かつ効率的にすることができます。

・初期段階での不具合の発見
製品開発ライフサイクルの初期段階で、要件、テストケース、瑕疵などの欠落を特定することができます。
これにより、サイクルの後半で発生しうる多くの問題から解放されます。

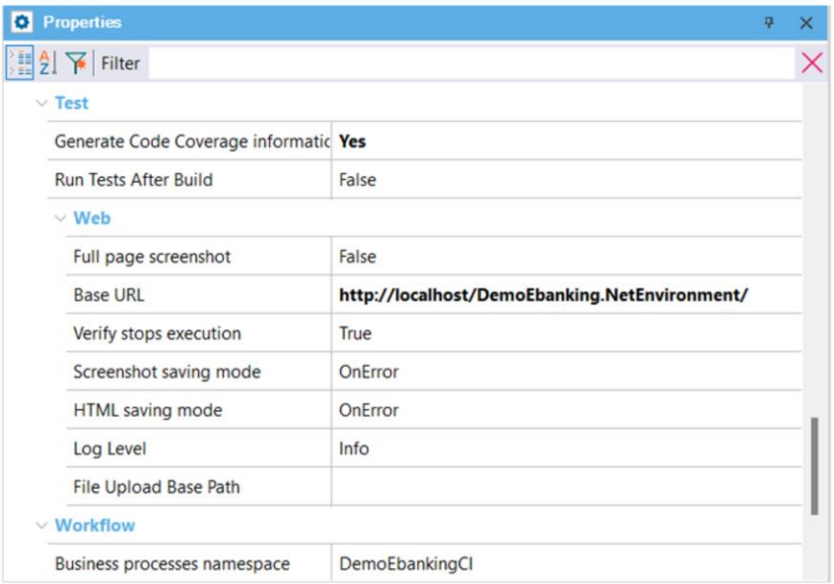
GeneXusでのテストカバレッジ機能

GeneXus by Globant

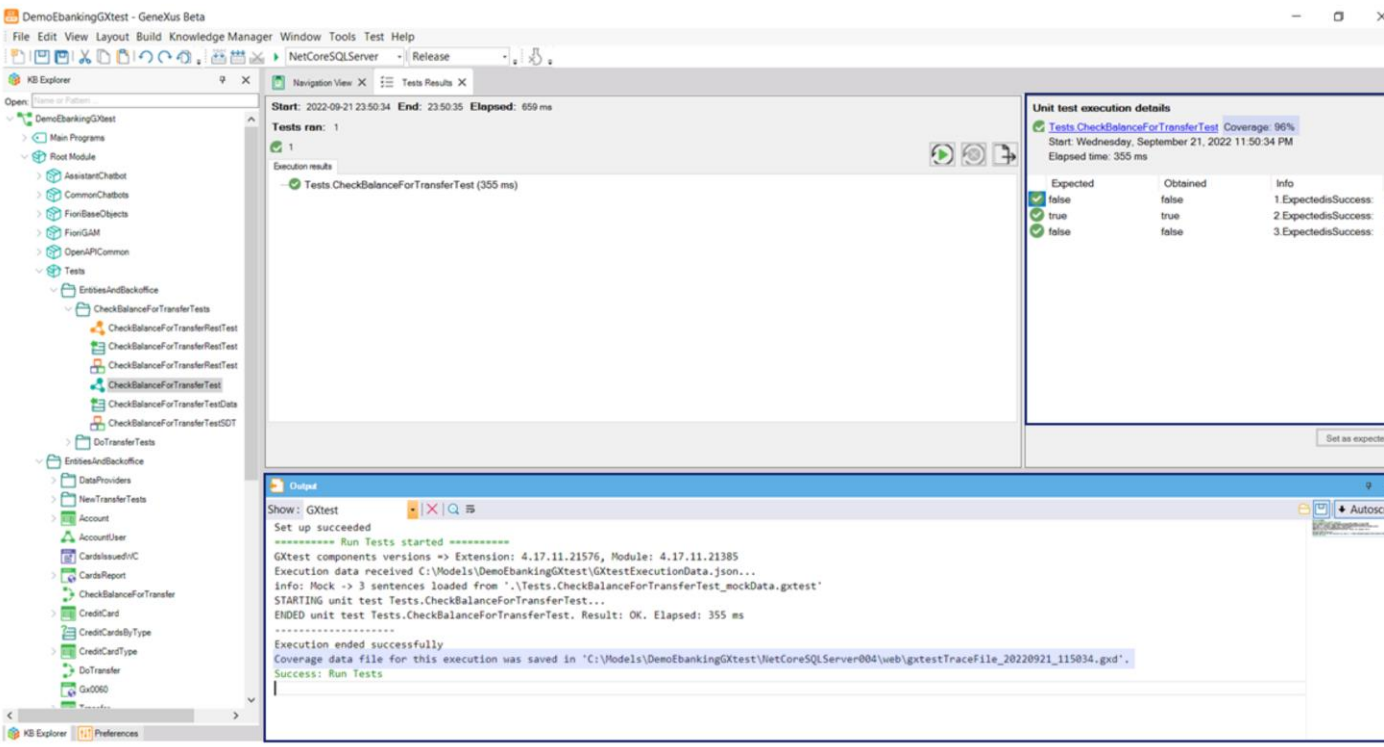
テストカバレッジとは、テストケースが実際にアプリケーションコードをカバーしているか、また、
テストケースを実行したときにどれだけのコードが実行されるかを判断する手法のことを指します。

アジャイル手法では、ロジックの80%はユニットテストで検証されるべきとされています。
ユニットテストは開発サイクルの初期に実行され、先述のピラミッドにもあった通りメンテナンスコストが低くなっています。

GeneXus IDEでは、テストカバレッジ機能は、「無効」が既定値となります。
そのため、まず開発者は、コードカバレッジを有効にする必要があります。



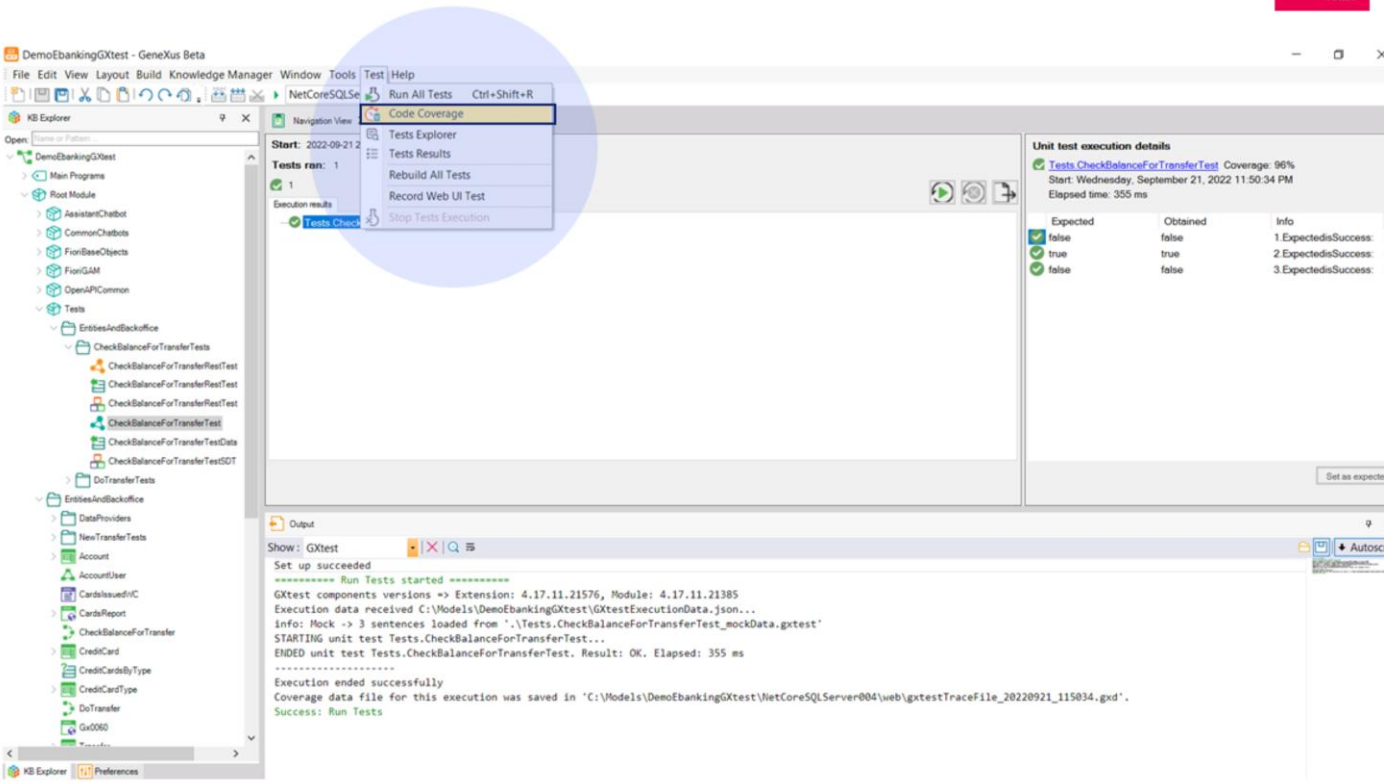
カバレッジ機能を有効化するには、KBのEnvironmentノードにある“Generate Code Coverage information“プロパティを「Yes」に設定し、「全てリビルド」を実行してください。



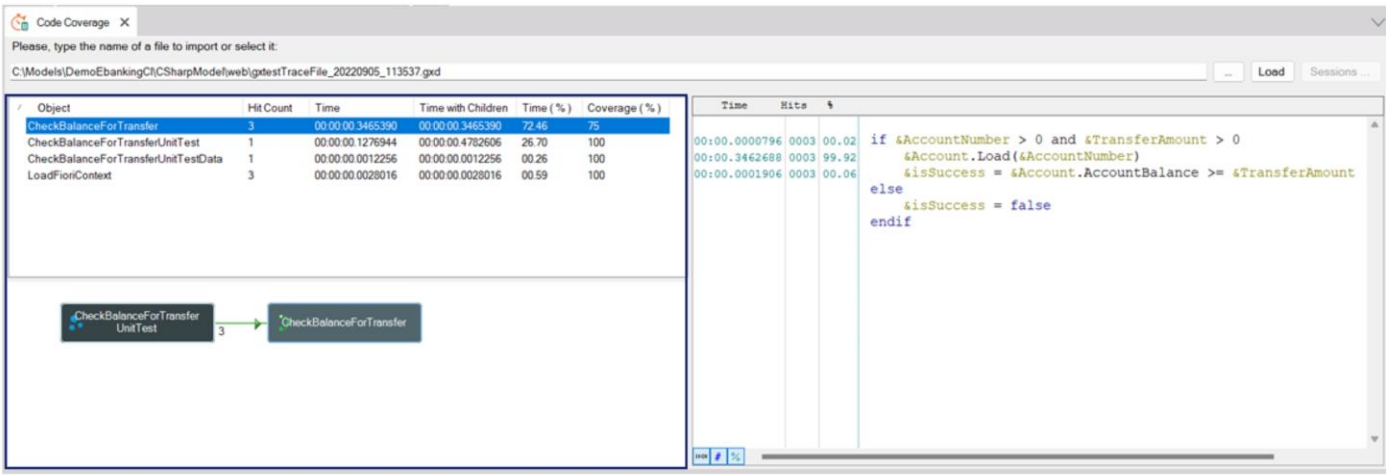
「全てリビルド」が終了すると、各テスト実行後に カバレッジによるテストの実行指標を表示できるようになります。

実際に、スライドで表示しているように CheckBalanceForTransferUnitTestのテスト実行では、テスト結果でカバレッジ割合が表示されています。

また、GeneXusの出力に、今回の実行のカバレッジデータファイルのパスが表示されます。
テスト結果で表示されるカバレッジ割合は、実行されたすべてのテストにおいて、対象となるすべてのオブジェクトの総行数を集約したものに対する値です。
つまり、一つのオブジェクトの半分ずつの行を対象とする二つのテストを実行した場合、両方のテストを合計すると、
テストされたオブジェクトのすべての行をカバーするため、表示されるカバレッジは 100%となります。



カバレッジファイルを開き、テストカバレッジの詳細を見るには、ツールバー → テスト → コードカバレッジをクリックします。



カバレッジデータファイルを"... "ボタンをクリックし、選択後、ロードをクリックすると、カバレッジの詳細情報 が表示されます。
左側のセクションには、実行に関与したすべてのオブジェクトが表示され、右側のセクションには、左側で選択したオブジェクトの実行情報が表示されます。

・Hit CountまたはHitsは、そのオブジェクトが実行された回数です。

・Timeは経過時間です。

・Time with Childrenは、そのオブジェクトによって呼び出されたオブジェクトの経過時間を加えた経過時間です。

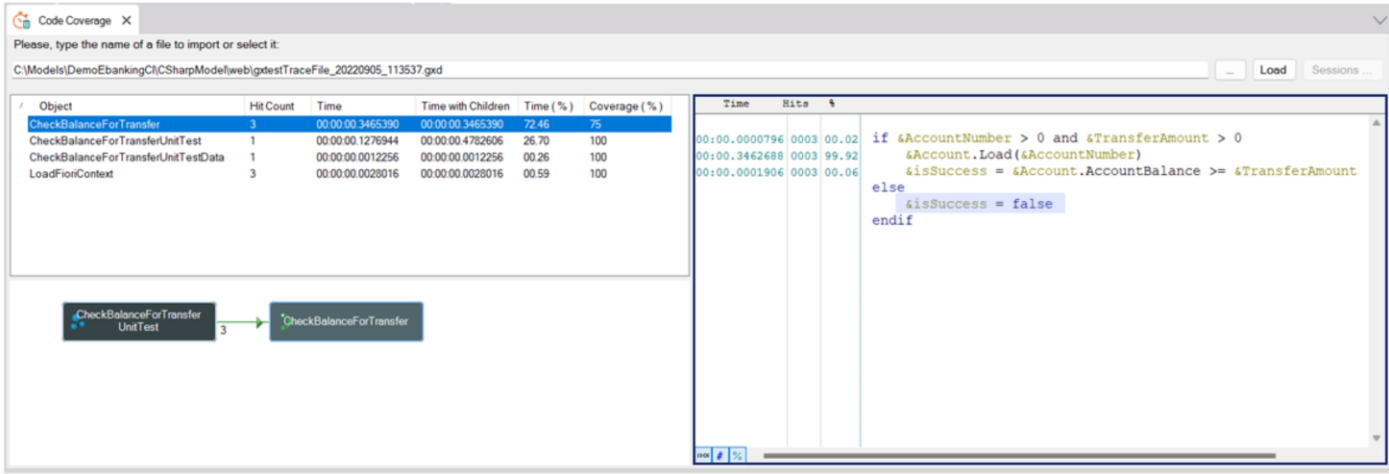
・Time(%)は、テストの実行に掛かった時間のうちの経過時間の割合です。

・Coverage(%) はカバレッジの割合、つまり全行数に対して実行された行数の割合です。

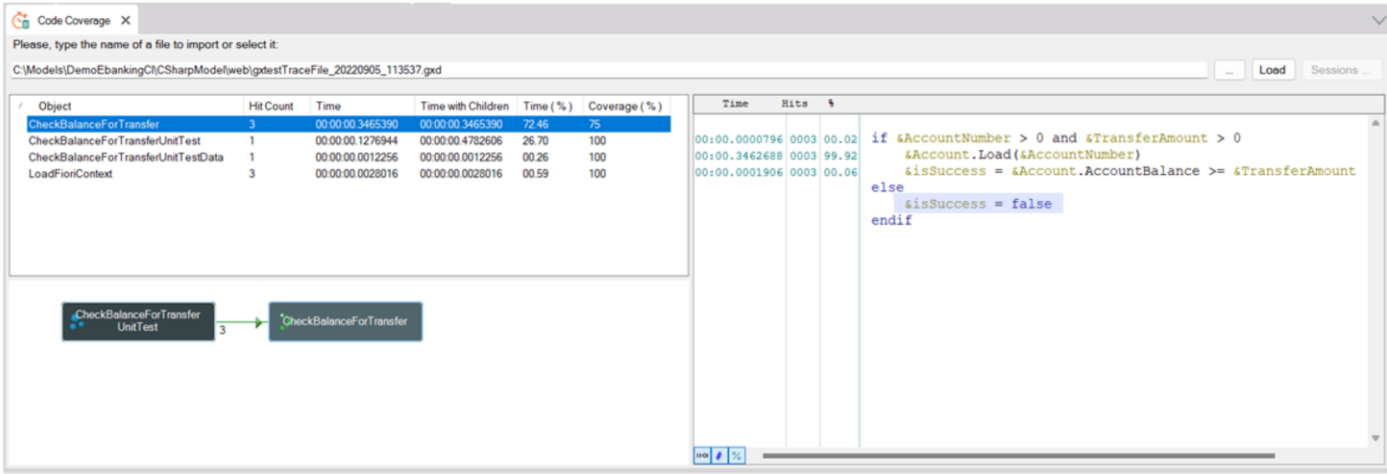
左側のセクションでオブジェクトを選択すると、下にコールツリーを示すグラフが表示されます。

例えば、表示されているスクリーンショットでは、CheckBalanceForTransferがCheckBalanceForTransferUnitTestから呼び出されていることが分かります。

右側のセクションには、行コードとそれぞれのトレース情報が表示されています。



このうち、情報のない行があることに注目してください。
これは、テストの実行中にこれらの行が実際には実行されなかったことを意味します。
この例では、else条件がカバーされていないことがわかります。



それでは、テストカバレッジ機能の見方をご説明します。

主な監視対象は、オブジェクトが実行された回数を表示する「Hit数」です。
また、右側のセクションでは、プロシージャの各コード行のヒット数を見ることができます。

この二つのデータから、CheckBalanceForTransferプロシージャには三つのテストケースがあり、
その三つのケースは「if」条件に入っていることが分かります。
そのため、if条件の中で冗長なテストケースが実行されていないことを確認することができます。

最も重要な指標はカバレッジ率の欄で、プロシージャCheckBalanceForTransferの75%が実行されたことがわかります。
つまり、すべてのコードラインがテストされたわけではないので、完全なテストカバレッジを得るためには、もっとテストケ

ースを追加する必要があるということです。

右側のセクションを見ればわかるように、else条件はテストによって実行されていません。
したがって、開発者はこの条件に対するテストケースを追加しなければなりません。

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 = ''
}
```

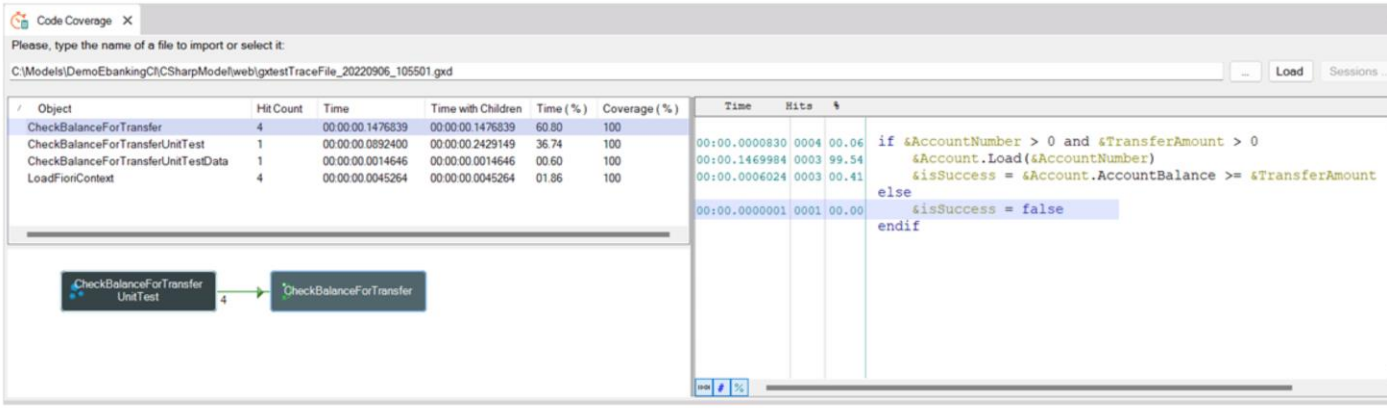
CheckBalanceForTransfer

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
```

そこで、データプロバイダに定義されているテストケースに移動し、else条件のテストケースを追加していきます。
この例では、TestCaseId=4のテストケースを一つ追加します。
このテストケースでは、プロックのelse条件を実行するために、負の送金額を定義しました。



ユニットテストを再度実行すると、プロシージャのカバレッジによるテストの実行指標が変化したことがわかります。
Coverage(%)の列を見ると、このユニットテストの実行ですべてのコード行が実行されたため、プロシージャが100%になっていることがわかります。


GeneXus™
by Globant
training.genexus.com
wiki.genexus.com