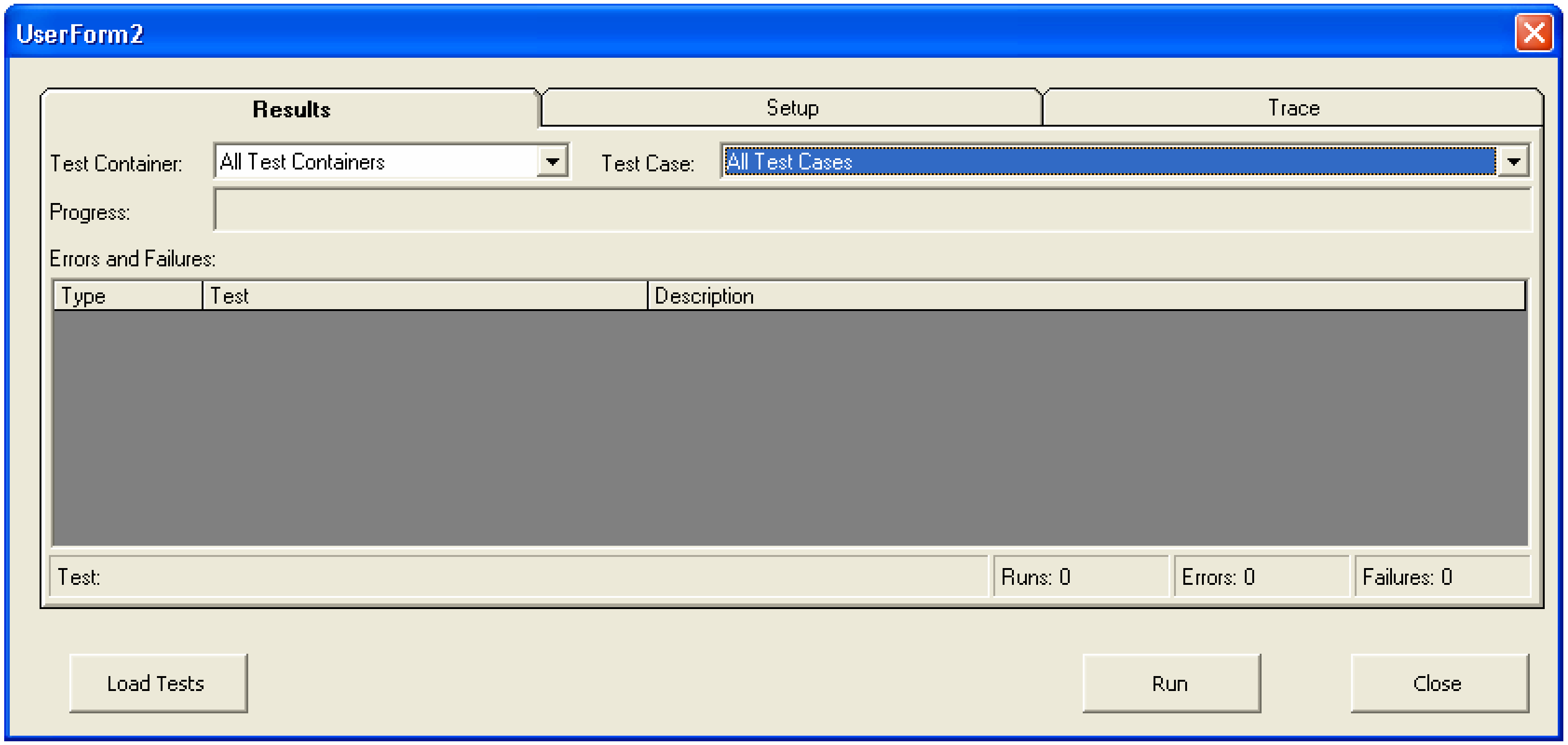
Test Driven Development Demo

Using COMUnit to drive development of Visual Basic macros in a spreadsheet

Framework (a.k.a. Test Harness)

- All TDD test frameworks (xUnit) include a simple tool to run the tests and report the errors.
- The framework typically allows individual tests to be run, or groups of tests to be run.
- The framework typically shows details about failures.
- The framework typically shows a green/red status bar. (red=test failures)

The test execution User Interface



Test Container: All Test Containers

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description

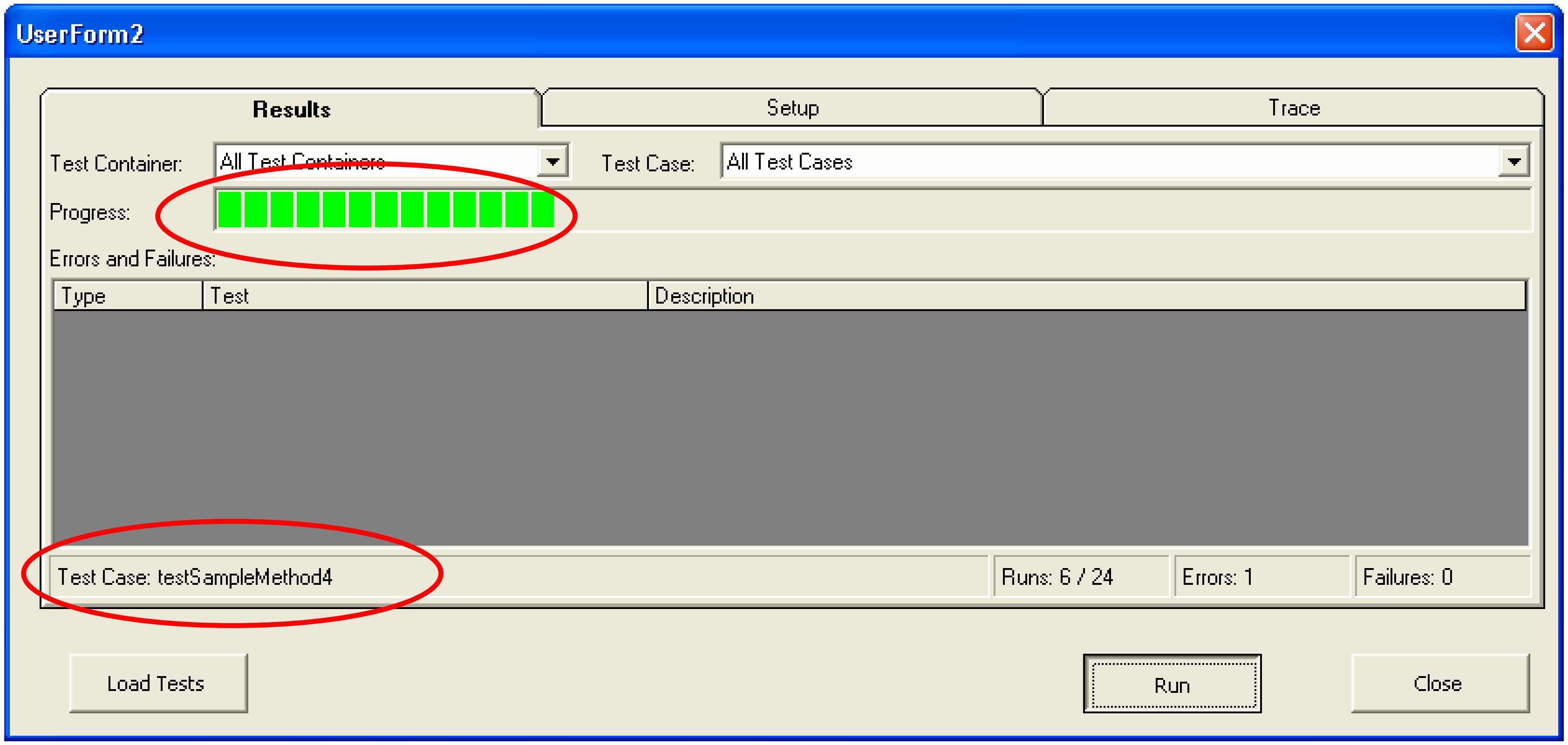
Test:

Runs: 0

Errors: 0

Failures: 0

Elements of the test execution User Interface



Test Container: All Test Containers

Test Case: All Test Cases

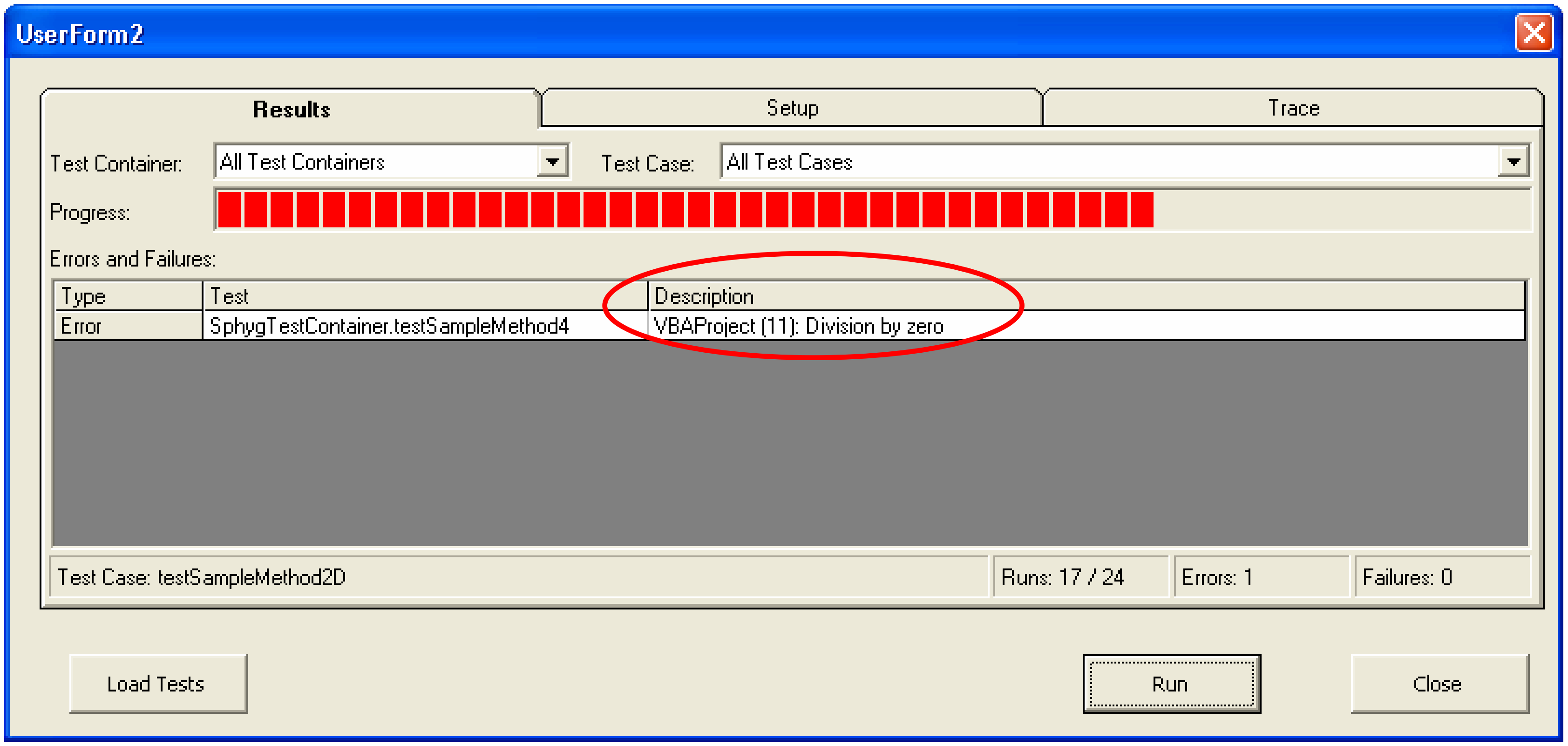
Progress:

Errors and Failures:

Type	Test	Description

Test Case: testSampleMethod4 | Runs: 6 / 24 | Errors: 1 | Failures: 0

Elements of the test execution User Interface



Test Container: All Test Containers

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description
Error	SphygTestContainer.testSampleMethod4	VBAProject (11): Division by zero

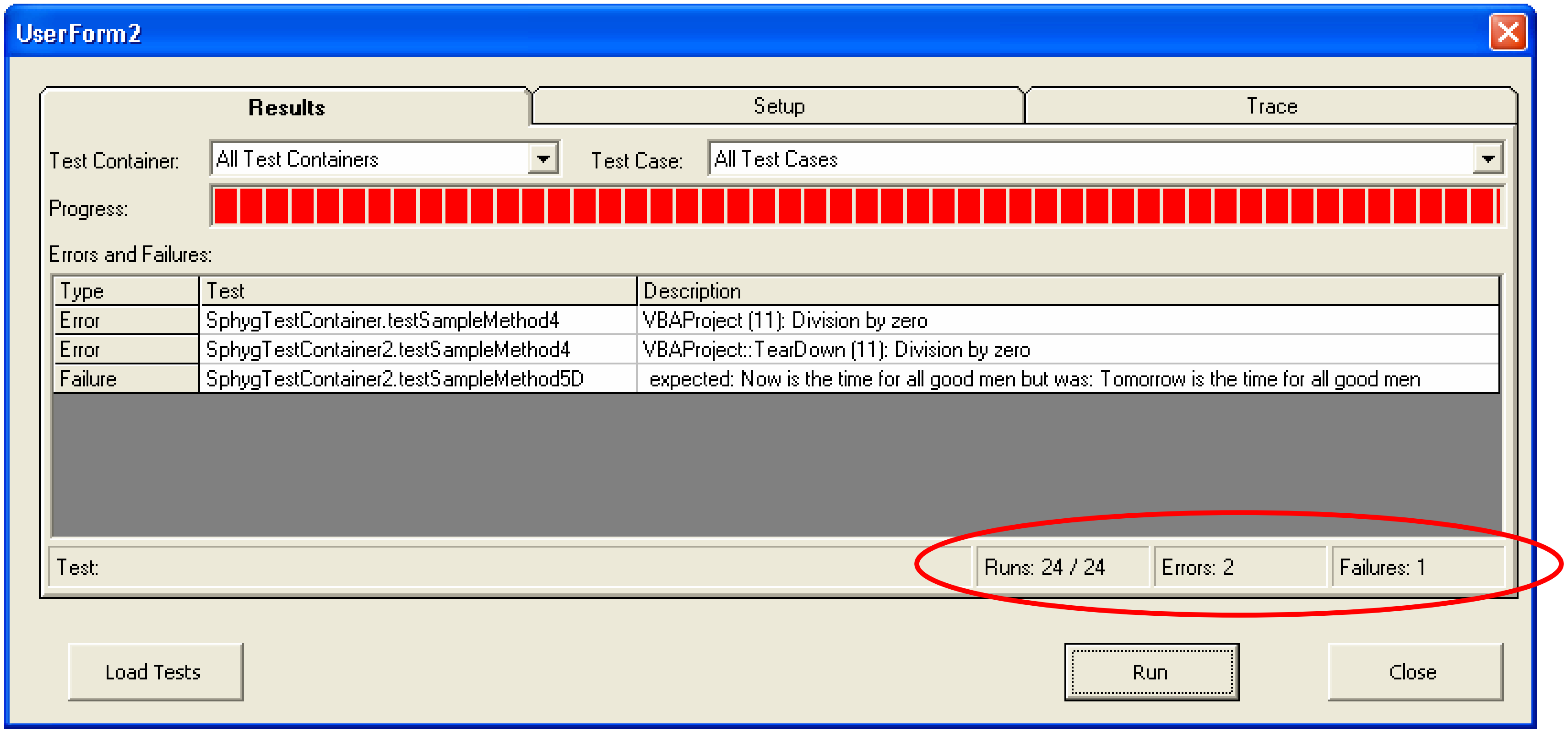
Test Case: testSampleMethod2D

Runs: 17 / 24

Errors: 1

Failures: 0

Elements of the test execution User Interface



Test Container: All Test Containers

Test Case: All Test Cases

Progress:

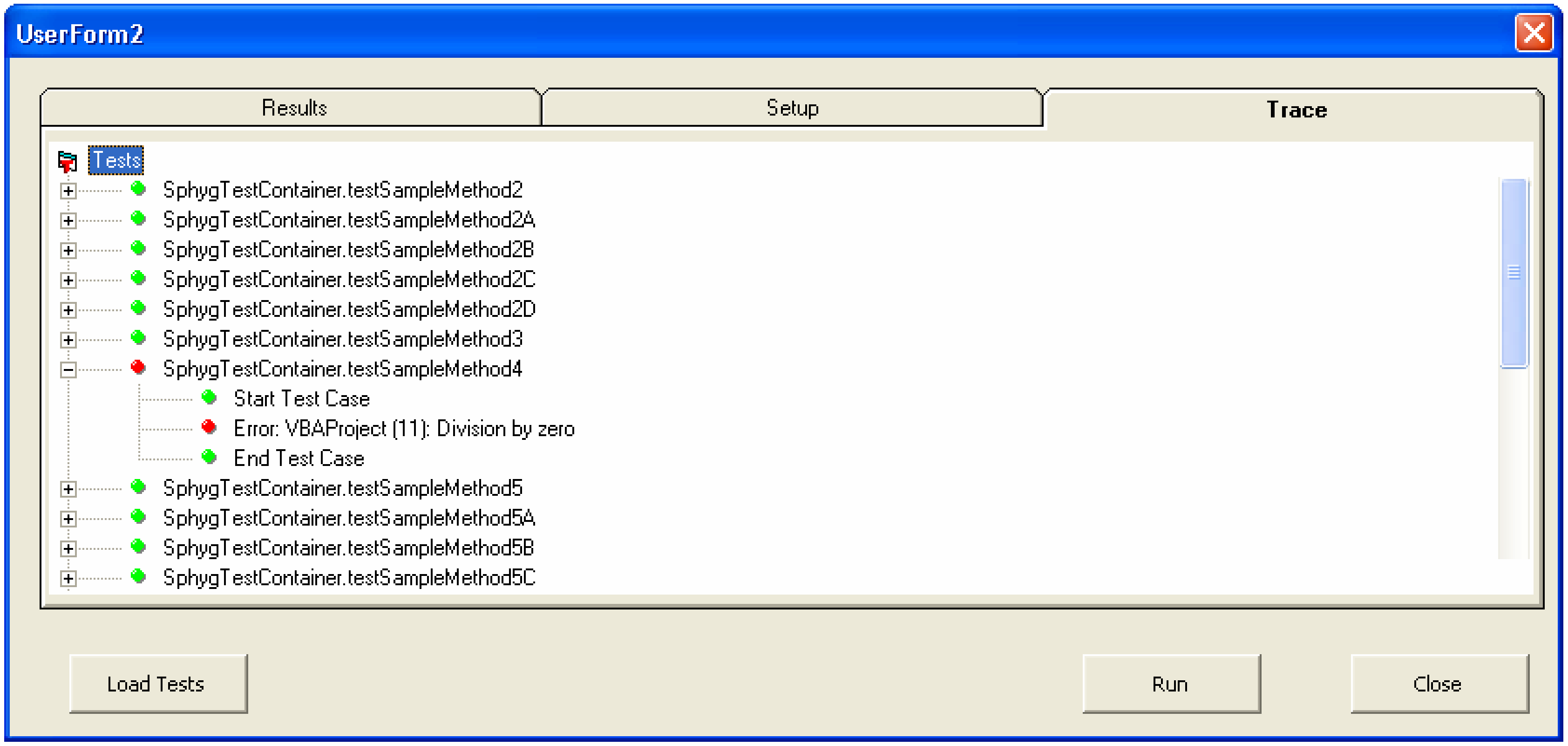
Errors and Failures:

Type	Test	Description
Error	SphygTestContainer.testSampleMethod4	VBAProject (11): Division by zero
Error	SphygTestContainer2.testSampleMethod4	VBAProject::TearDown (11): Division by zero
Failure	SphygTestContainer2.testSampleMethod5D	expected: Now is the time for all good men but was: Tomorrow is the time for all good men

Test:

Runs: 24 / 24 Errors: 2 Failures: 1

Elements of the test execution User Interface



```
Tests
  ● SphygTestContainer.testSampleMethod2
  ● SphygTestContainer.testSampleMethod2A
  ● SphygTestContainer.testSampleMethod2B
  ● SphygTestContainer.testSampleMethod2C
  ● SphygTestContainer.testSampleMethod2D
  ● SphygTestContainer.testSampleMethod3
  ● SphygTestContainer.testSampleMethod4
      ● Start Test Case
      ● Error: VBAProject (11): Division by zero
      ● End Test Case
  ● SphygTestContainer.testSampleMethod5
  ● SphygTestContainer.testSampleMethod5A
  ● SphygTestContainer.testSampleMethod5B
  ● SphygTestContainer.testSampleMethod5C
```

The function to be tested

- Demo testing part of a standard deviation function in the original spreadsheet code
- The calculation is done in two steps, according to a standard deviation formula often used in programs that keep “running sums” as they traverse over data:
  - Add up the samples and the sum of samples
  - “Finish” the calculation according to this formula:

$$s = \sqrt{\frac{N\sum_{i=1}^{N} {x_i}^2 - \left(\sum_{i=1}^{N} x_i\right)^2}{N(N-1)}}$$

Getting started

- Load the spreadsheet that contains the test functions provided by COMUnit.
- Remember the TDD cycle:
  – Write a test
  – Add the test name to the list of tests to run
  – Compile (fails on missing function)
  – Write stub for the function to be tested
  – Run tests, test fails
  – Implement function to be tested
  – Run tests again, test passes

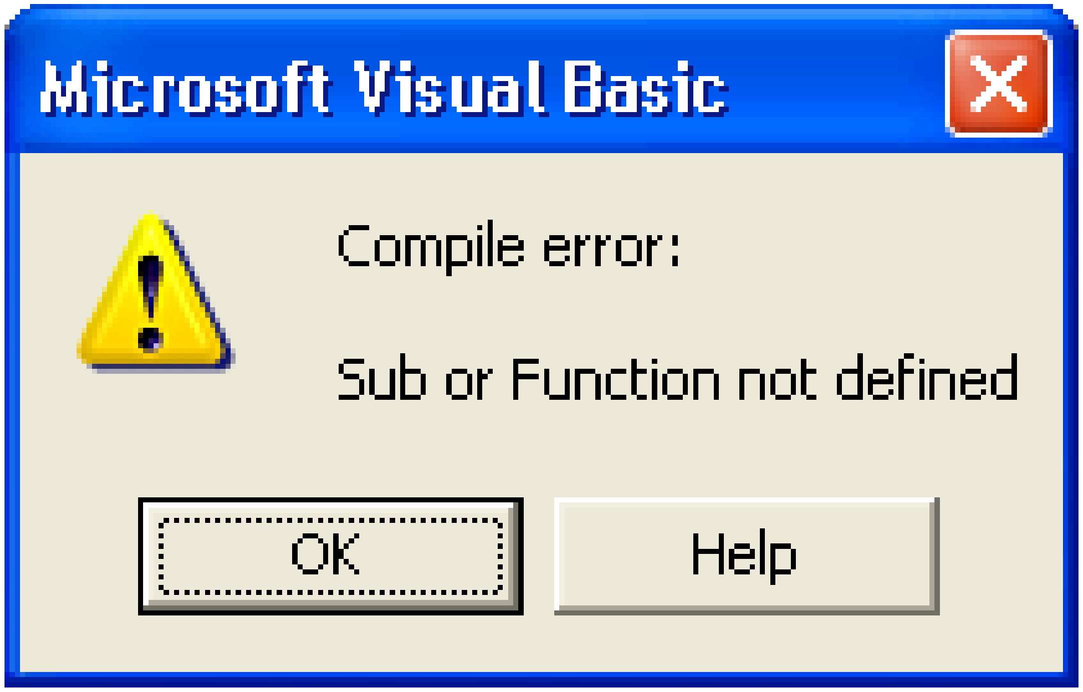
The first Test function

```
Public Sub testFinishStdDevWith3DataPoints(oTestResult As TestResult)
    Dim result As Double
    ' the data points are 7,8,9
    result = FinishStdDev(3, 7 + 8 + 9, 49 + 64 + 81)
    oTestResult.AssertEqualsDouble 1, result, 0.1
End Sub
```

Adding test name to list of tests

```
Public Property Get ITestContainer_TestCaseNames() As Variant()
   ' TODO: add the names of your test methods as a parameter into the Array() function
   ITestContainer_TestCaseNames = Array( _
      "testFinishStdDevWith3DataPoints" _
      )
End Property
```

“Compile” fails



Compile error:

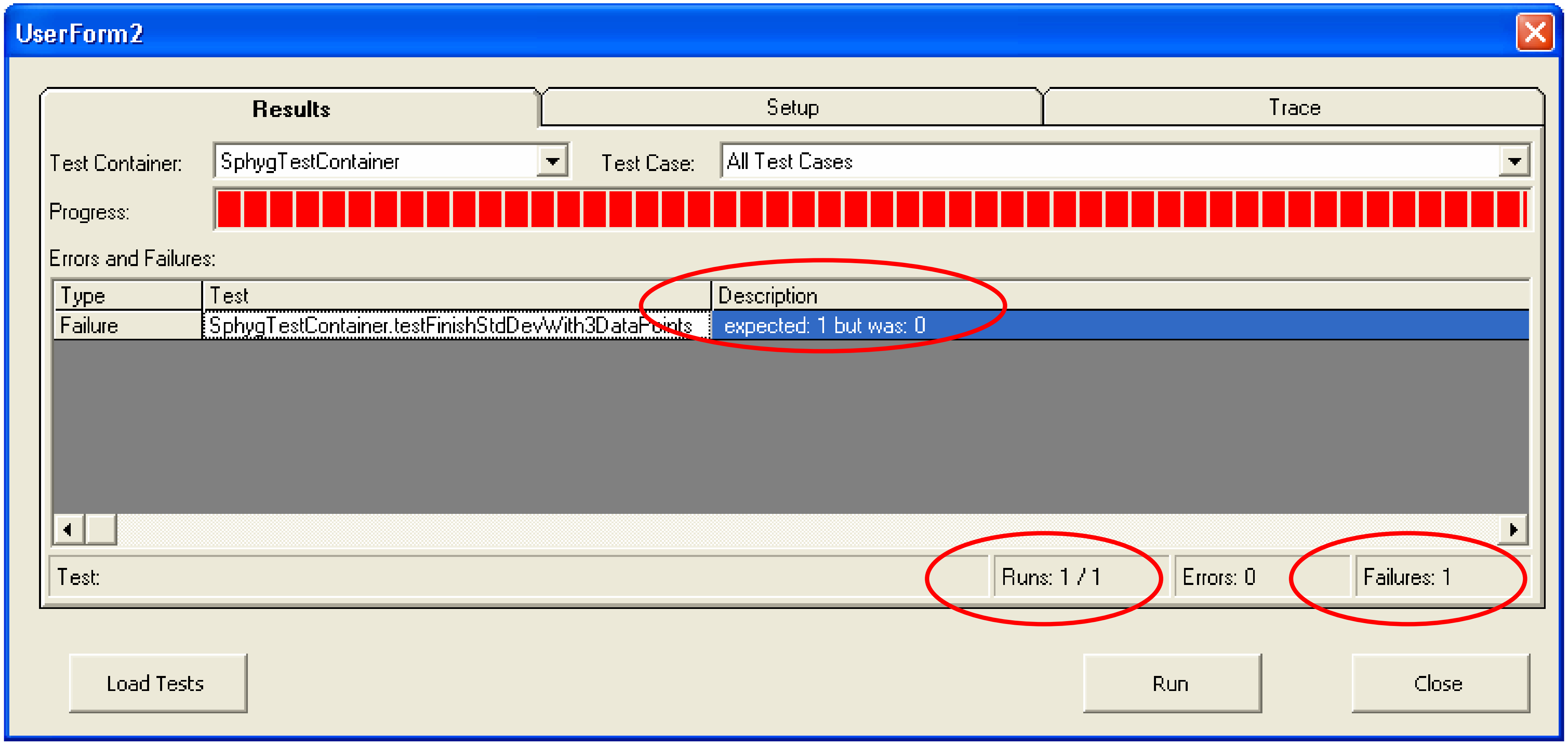
Sub or Function not defined

Write stub function

- Doesn’t actually compute the return value yet

```
Function FinishStdDev(N, sumOfSamples, sumOfSquares)
   FinishStdDev = 0
End Function
```

Run tests, test fails



UserForm2

Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description
Failure	SphygTestContainer.testFinishStdDevWith3DataPoints	expected: 1 but was: 0

Test:

Runs: 1 / 1

Errors: 0

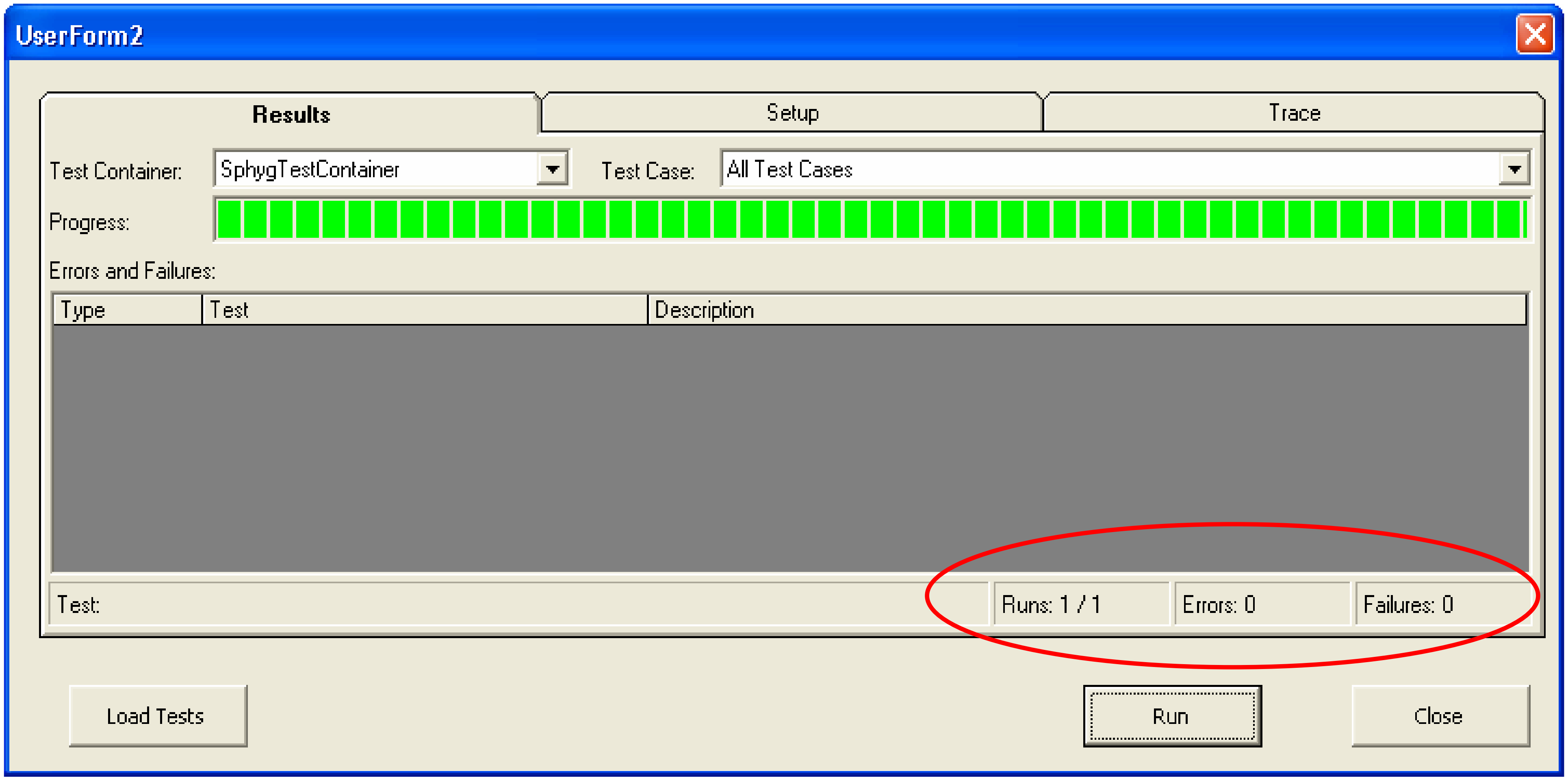
Failures: 1

Implement function to be tested

- Here’s the function from the original Sphygmochron spreadsheet. I’m leaving a few lines commented out because we don’t need them yet. (I’m “implementing” the smallest possible bit of the code to make the test pass.)

```
Function FinishStdDev(N, sumOfSamples, sumOfSquares)
    temp = ((N * sumOfSquares) - (sumOfSamples ^ 2)) / (N * (N - 1))
    'If temp < 0 Then
     '   temp = temp * -1
    'End If
    FinishStdDev = Sqr(temp)
End Function
```

Run tests, test now passes



UserForm2

Results | Setup | Trace

Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description

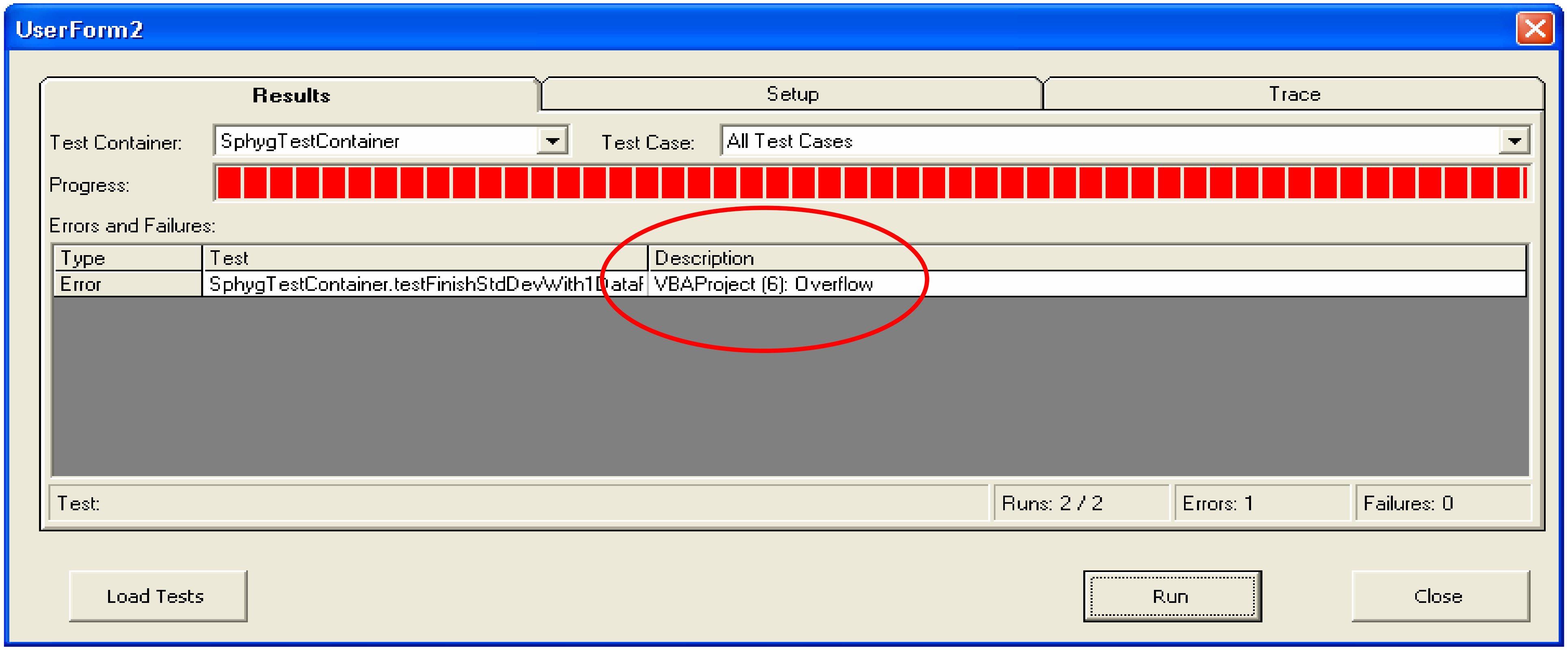
Test:

Runs: 1 / 1 | Errors: 0 | Failures: 0

Write another test

```
Public Sub testFinishStdDevWith1DataPoint(oTestResult As
TestResult)
    Dim result as Double
    ' the data point is 7
    result = FinishStdDev(1, 7, 49)
    oTestResult.AssertEqualsDouble 0, result, 0.0
End Sub
```

Run new test, it fails



Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description
Error	SphygTestContainer.testFinishStdDevWith1Dataf	VBAProject (6): Overflow

Test:

Runs: 2 / 2

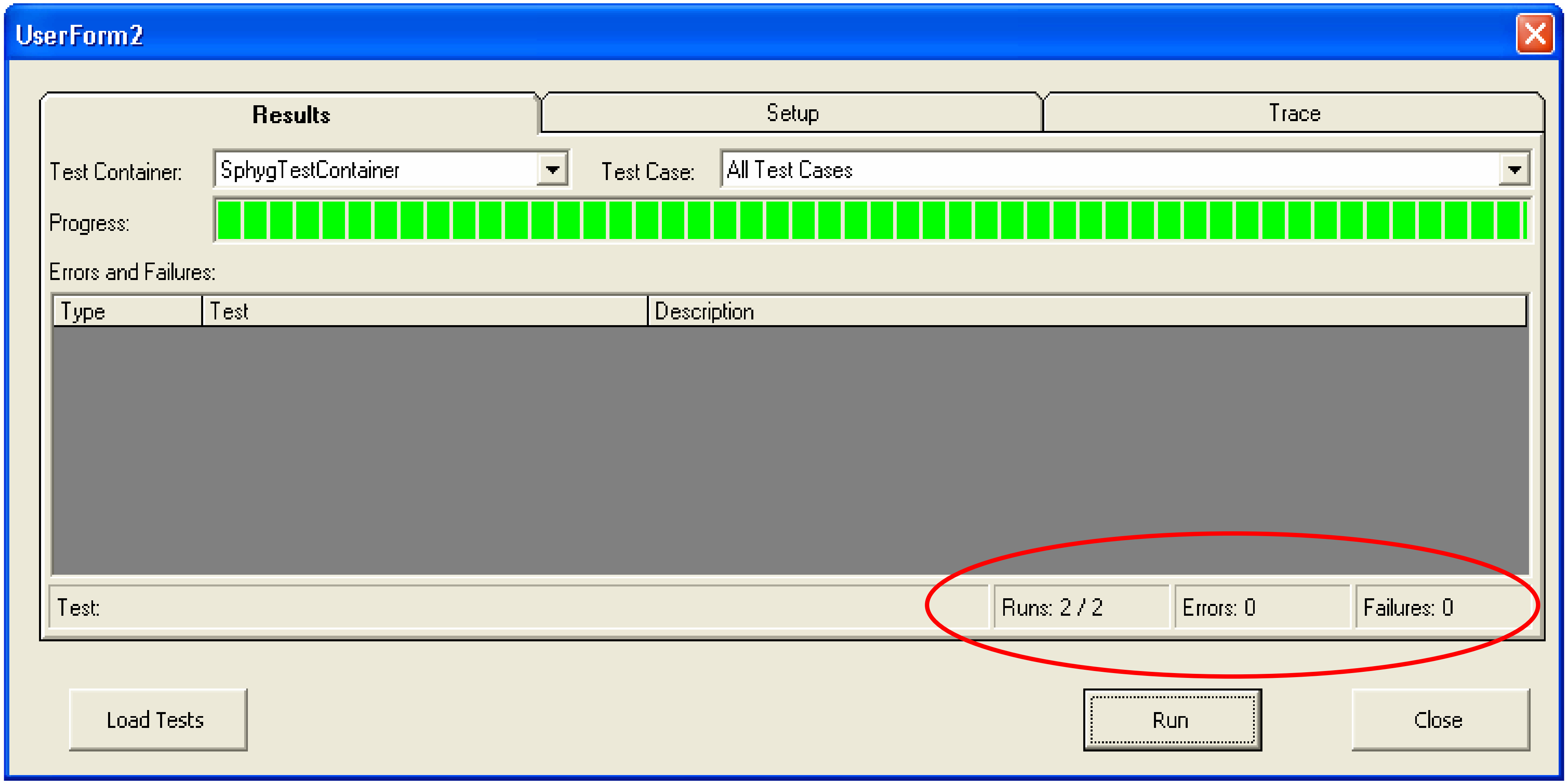
Errors: 1

Failures: 0

Write code that makes the test run

```
Function FinishStdDev(N, sumOfSamples, sumOfSquares)
   If (N = 1) Then
      FinishStdDev = 0
   Else
      temp = ((N * sumOfSquares) - (sumOfSamples ^ 2)) / (N * (N - 1))
      'If temp < 0 Then
       '   temp = temp * -1
      'End If
      FinishStdDev = Sqr(temp)
   End If
End Function
```

Run (all) tests, new test passes



UserForm2

Results | Setup | Trace

Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description

Test:

Runs: 2 / 2 | Errors: 0 | Failures: 0

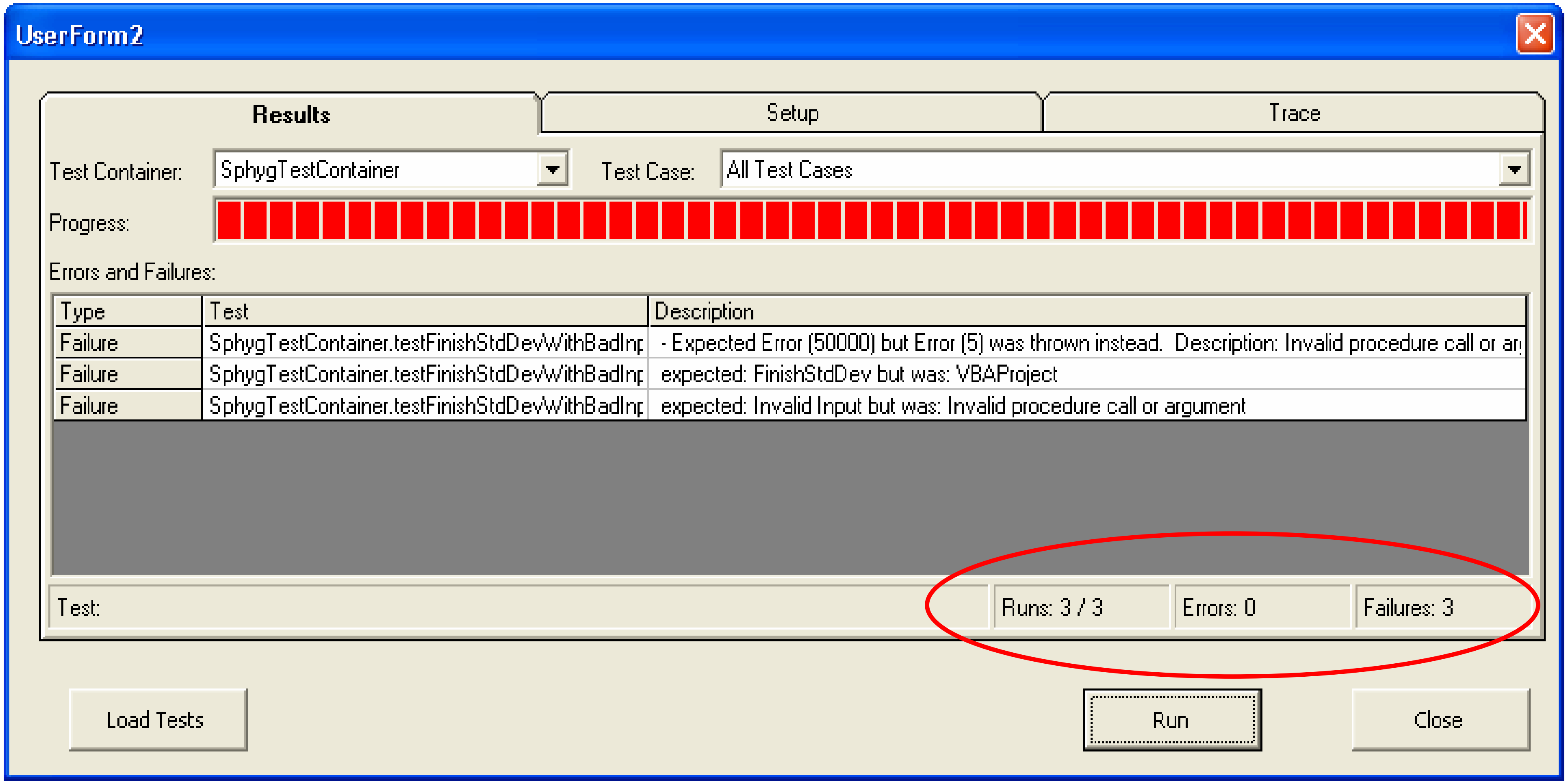
Example error checking test

```
Public Sub testFinishStdDevWithBadInputNegativeIntermediateResult(oTestResult As
TestResult)
   Dim result As Double
   On Error GoTo ErrorCheck

   ' the data points are 1,2,3, the sum of the squares should have been 14, not 5
   result = FinishStdDev(3, 6, 5)
   oTestResult.AddFailure ("Expected 'invalid input' error, but didn't get it")
   Exit Sub

ErrorCheck:
   oTestResult.AssertEqualsError Err, 50000
   oTestResult.AssertEqualsString "FinishStdDev", Err.Source
   oTestResult.AssertEqualsString "Invalid Input", Err.Description
End Sub
```

Run tests, error checking test fails



Test Container: SphygTestContainer

Test Case: All Test Cases

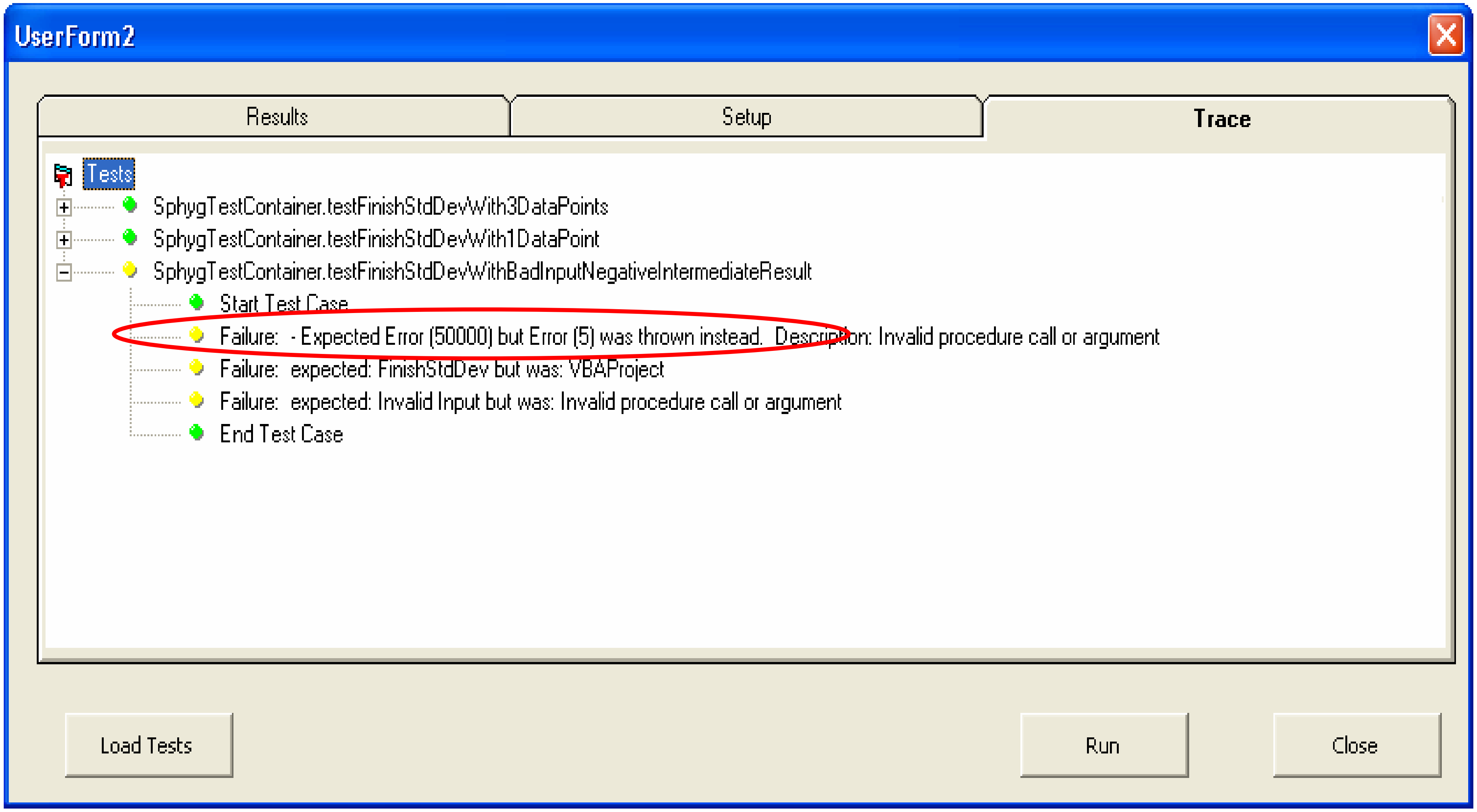
Progress:

Errors and Failures:

Type	Test	Description
Failure	SphygTestContainer.testFinishStdDevWithBadInp	- Expected Error (50000) but Error (5) was thrown instead. Description: Invalid procedure call or ar
Failure	SphygTestContainer.testFinishStdDevWithBadInp	expected: FinishStdDev but was: VBAProject
Failure	SphygTestContainer.testFinishStdDevWithBadInp	expected: Invalid Input but was: Invalid procedure call or argument

Test: | Runs: 3 / 3 | Errors: 0 | Failures: 3

All failures were in the new test



```
Tests
  SphygTestContainer.testFinishStdDevWith3DataPoints
  SphygTestContainer.testFinishStdDevWith1DataPoint
  SphygTestContainer.testFinishStdDevWithBadInputNegativeIntermediateResult
      Start Test Case
      Failure:  - Expected Error (50000) but Error (5) was thrown instead.  Description: Invalid procedure call or argument
      Failure:  expected: FinishStdDev but was: VBAProject
      Failure:  expected: Invalid Input but was: Invalid procedure call or argument
      End Test Case
```

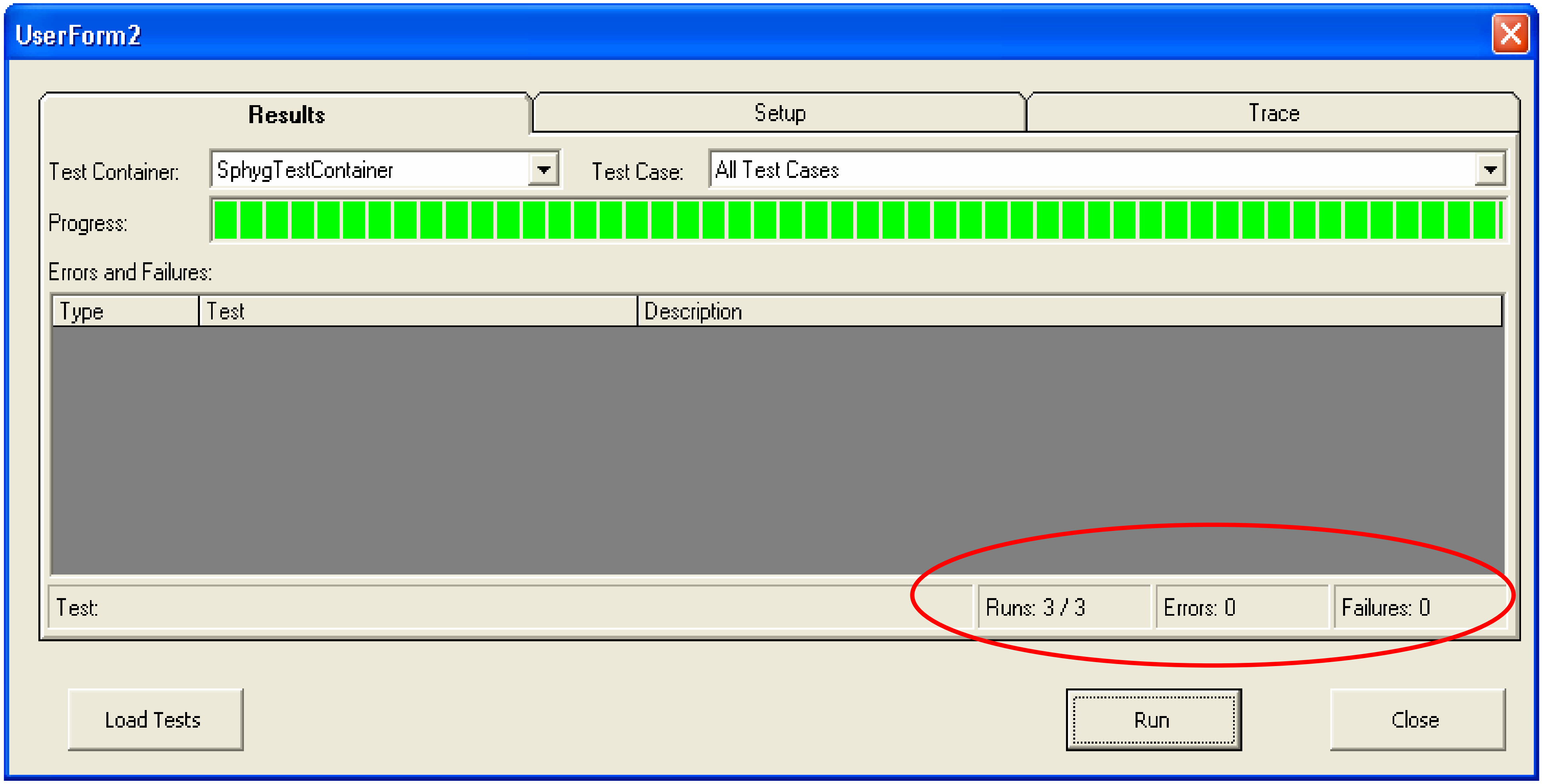
Adding error checking

- The original code in the Sphygmochron had that “If Temp < 0” code to prevent the square-root operation from aborting on negative numbers
- However, a negative number could never occur there, *and* it prevented the original programmer from detecting bad input such as in our current test
  - We have found two errors (or at least weaknesses) in the original code

Write error checking code

```
Function FinishStdDev(N, sumOfSamples, sumOfSquares)
    If (N = 1) Then
        FinishStdDev = 0
    Else
        temp = ((N * sumOfSquares) - (sumOfSamples ^ 2)) / (N * (N - 1))
        ~~'If temp < 0 Then~~
        ~~ '    temp = temp * -1~~
        ~~'End If~~
        If (temp < 0) Then
            Err.Raise 50000, "FinishStdDev", "Invalid Input"
        End If
        FinishStdDev = Sqr(temp)
    End If
End Function
```

Run tests, error checking test passes



UserForm2

Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description

Test:

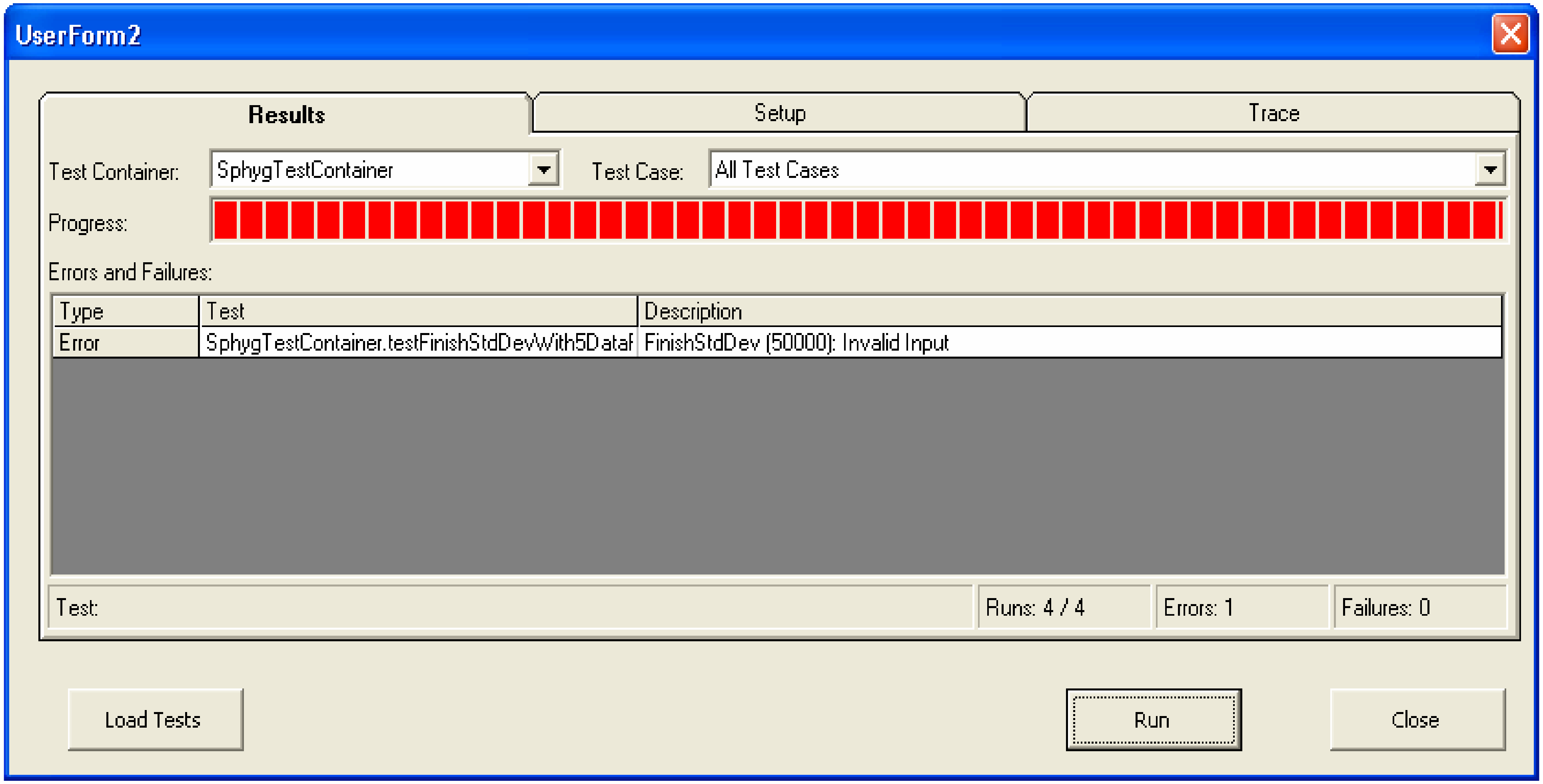
Runs: 3 / 3 | Errors: 0 | Failures: 0

Write a new test

- This test should have worked, but I screwed it up. It was “too hard” to write. The first argument, the number of data points, should have been 5.

```
Public Sub testFinishStdDevWith5DataPoints(oTestResult As TestResult)
    Dim result As Double
    ' the data points are 2, 4, 6, 8, 10
    result = FinishStdDev(3, 30, 2 * 2 + 4 * 4 + 6 * 6 + 8 * 8 + 10 * 10)
    oTestResult.AssertEqualsDouble 3.16227, result, 0.0001
End Sub
```

Run tests, newest test fails



Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description
Error	SphygTestContainer.testFinishStdDevWith5Dataf	FinishStdDev (50000): Invalid Input

Test:

Runs: 4 / 4

Errors: 1

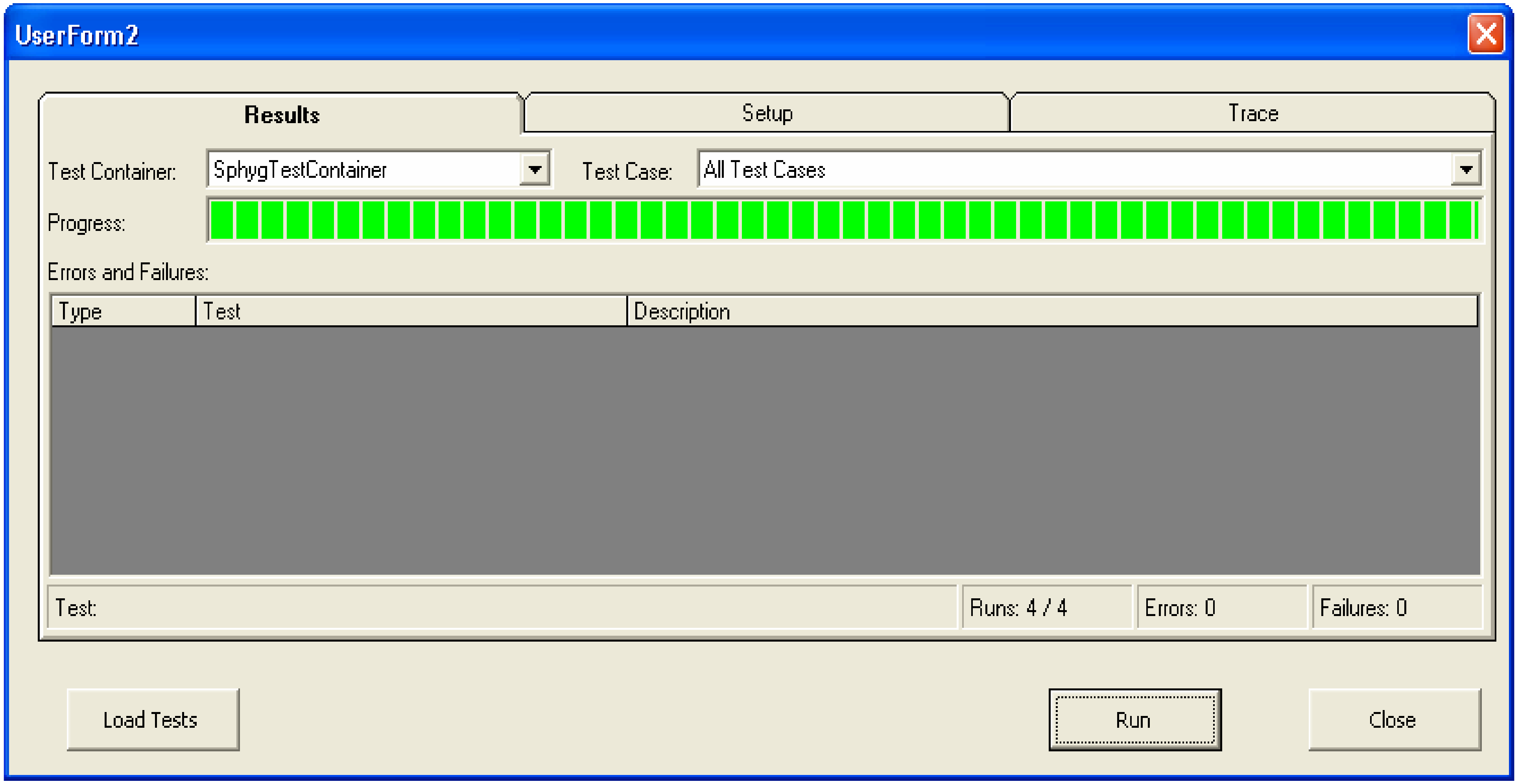
Failures: 0

Fix the new test

- This test should have worked, but I screwed it up. It was “too hard” to write. The first argument, the number of data points, should have been 5.

```
Public Sub testFinishStdDevWith5DataPoints(oTestResult As TestResult)
    Dim result As Double
    ' the data points are 2, 4, 6, 8, 10
    result = FinishStdDev(5, 30, 2 * 2 + 4 * 4 + 6 * 6 + 8 * 8 + 10 * 10)
    oTestResult.AssertEqualsDouble 3.16227, result, 0.0001
End Sub
```

Run tests, now it passes



UserForm2

Type	Test	Description

Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Test:

Runs: 4 / 4

Errors: 0

Failures: 0

Refactor to simplify tests

- Now that all the tests are running, it’s OK to refactor a little bit to make it easier to write and maintain the tests
- I write a utility function to help prepare the arguments to the function being tested

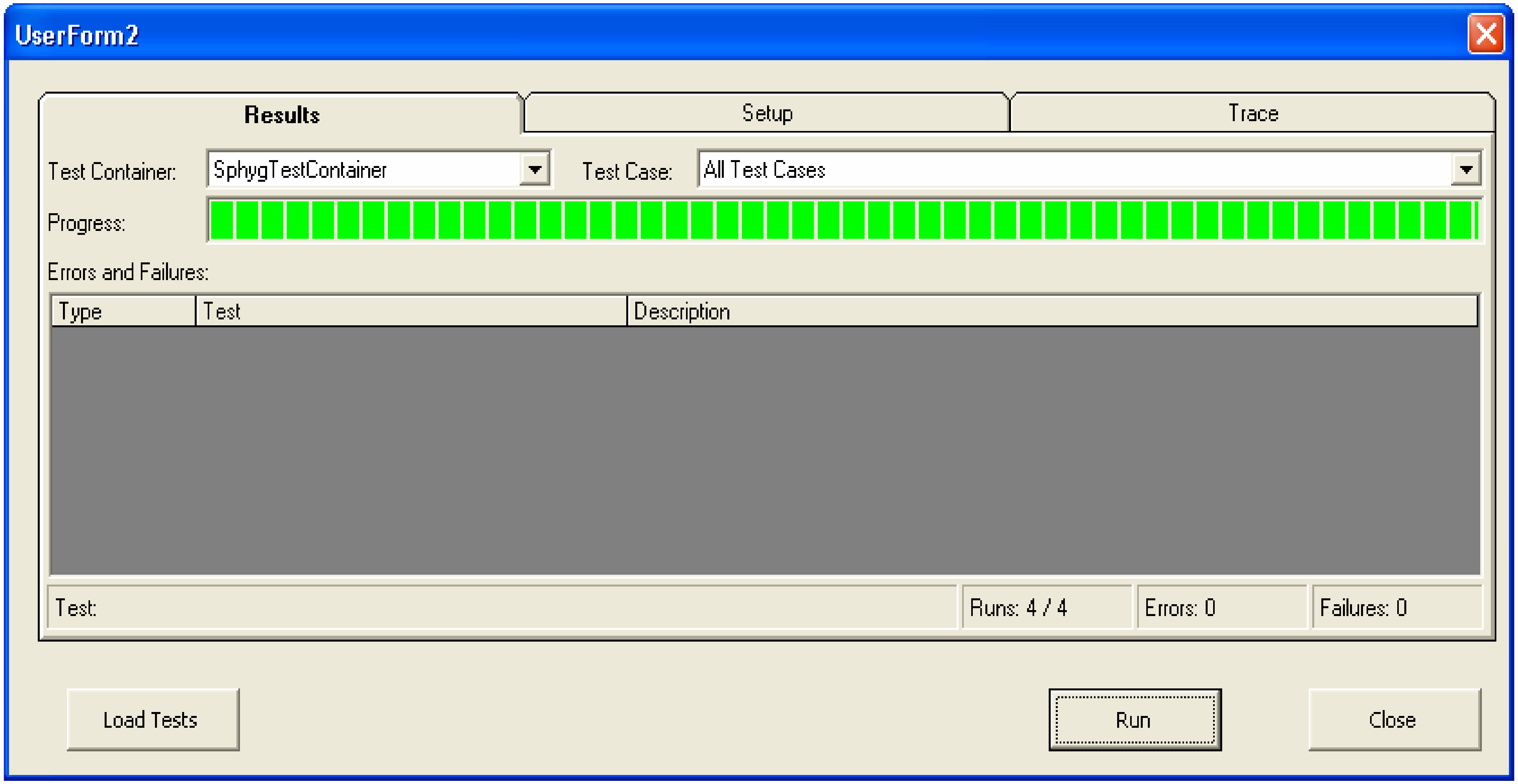
```
Function utilCallFinishStdDev(inputdata As Variant)
   Dim N, sumOfSamples, sumOfSquares
   Dim i
   N = UBound(inputdata) + 1
   sumOfSamples = 0
   sumOfSquares = 0
   For i = 0 To UBound(inputdata)
      sumOfSamples = sumOfSamples + inputdata(i)
      sumOfSquares = sumOfSquares + inputdata(i) * inputdata(i)
   Next i
   utilCallFinishStdDev = FinishStdDev(N, sumOfSamples, sumOfSquares)
End Function
```

Refactor to simplify tests

- I use the utility function in all the tests possible
- I no longer need the comment telling me what the data points are, to count the points, nor compute either of the sums
- This test is simpler and *much* easier to get right

```
Public Sub testFinishStdDevWith5DataPoints(oTestResult As TestResult)
    Dim result As Double
    ~~' the data points are 2, 4, 6, 8, 10~~
    ~~result = FinishStdDev(5, 30, 2 * 2 + 4 * 4 + 6 * 6 + 8 * 8 + 10 * 10)~~
    result = utilCallFinishStdDev(Array(2, 4, 6, 8, 10))
    oTestResult.AssertEqualsDouble 3.16227, result, 0.0001
End Sub
```

Run tests, it still passes



UserForm2

Results | Setup | Trace

Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description

Test:

Runs: 4 / 4

Errors: 0

Failures: 0

Another simplification

- Move the “assertion” down into the utility function

```
Sub utilCallFinishStdDev(inputdata As Variant, expectedResult As Double,
oTestResult As TestResult)
   Dim N, sumOfSamples, sumOfSquares
   Dim i
   Dim result As Double
   N = UBound(inputdata) + 1
   sumOfSamples = 0
   sumOfSquares = 0
   For i = 0 To UBound(inputdata)
      sumOfSamples = sumOfSamples + inputdata(i)
      sumOfSquares = sumOfSquares + inputdata(i) * inputdata(i)
   Next i
   result = FinishStdDev(N, sumOfSamples, sumOfSquares)
   oTestResult.AssertEqualsDouble expectedResult, result, 0.0001
End Sub
```

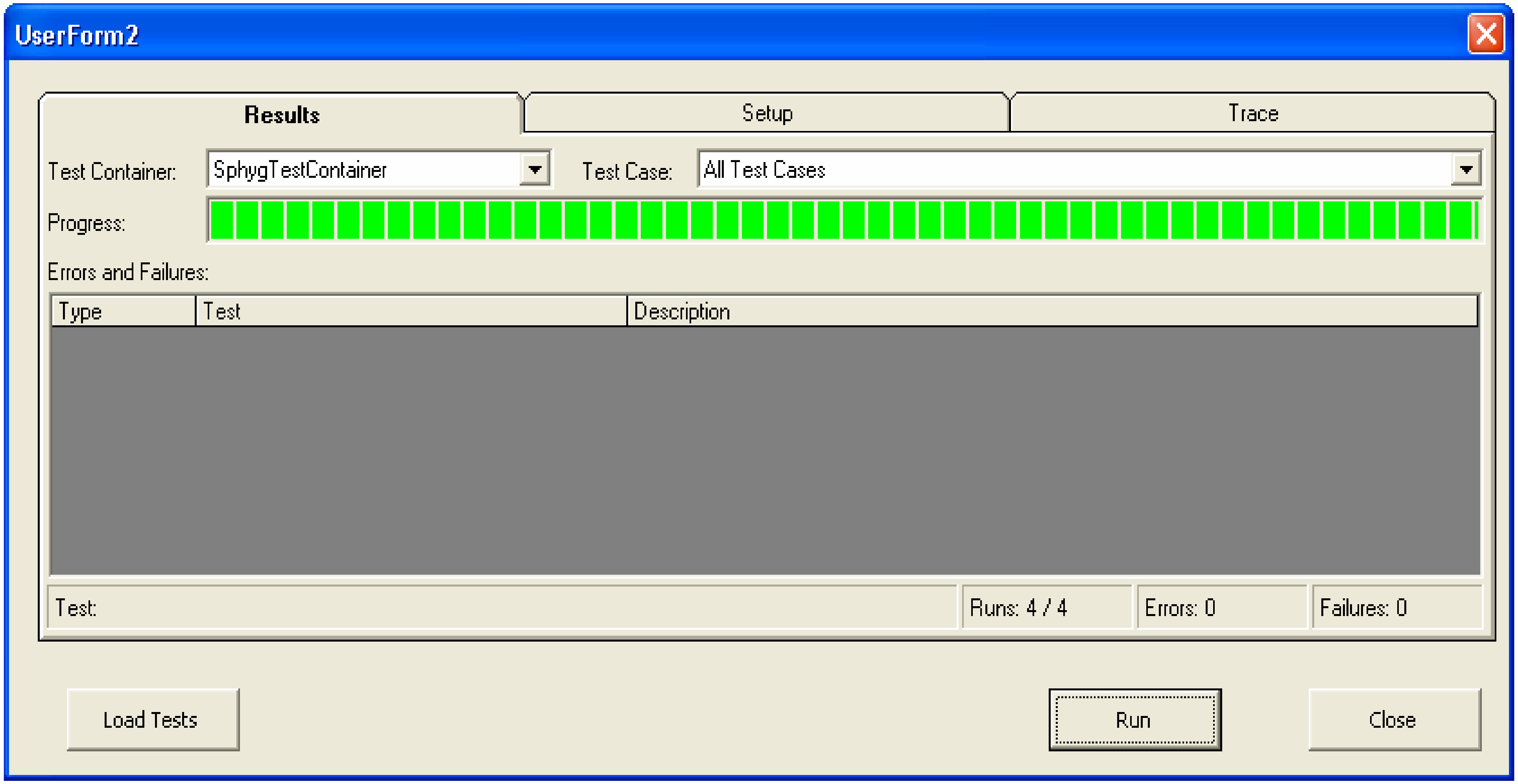
Another simplification

- The test itself gets *very* simple. In fact, *all* of the “positive” tests for the FinishStdDev function become one line long.

```
~~Public Sub testFinishStdDevWith5DataPoints(oTestResult As TestResult)~~
~~    Dim result As Double~~
~~    result = utilCallFinishStdDev(Array(2, 4, 6, 8, 10))~~
~~    oTestResult.AssertEqualsDouble 3.16227, result, 0.0001~~
~~End Sub~~
```

```
Public Sub testFinishStdDevWith5DataPoints(oTestResult As TestResult)
    utilCallFinishStdDev Array(2, 4, 6, 8, 10), 3.16227, oTestResult
End Sub
```

Run tests, it still passes



Test Container: SphygTestContainer

Test Case: All Test Cases

Progress:

Errors and Failures:

Type	Test	Description

Test:

Runs: 4 / 4

Errors: 0

Failures: 0

Wrap-up

- I took 4 lines of code from the original Sphygmochron code and made it a testable function
- I found a divide-by-zero error, an apparently unnecessary and poorly-coded “absolute value” function, and a missed opportunity for detecting bad input data, all in what we *thought* were 4 correct lines of code
- Refactoring tests gave me a very easy (1-line) mechanism to use to write future tests for this function
- If *any* future programmer changes the function, my tests prevent from breaking functionality that I know is important today
- TDD enabled me to do this by encouraging thinking about valuable test cases before implementing the parts of the function