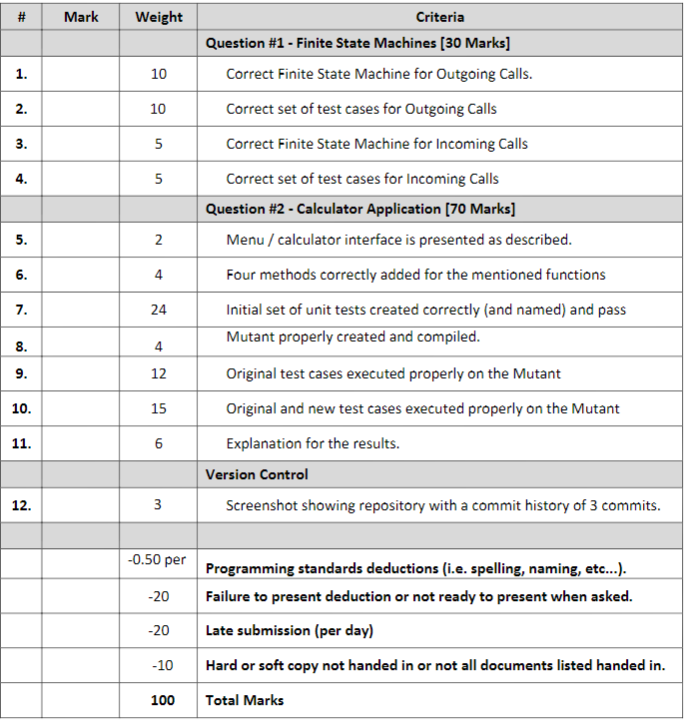
**Assignment #3**

**Tymur Koltunov**

**StudentID 8672727**

# **Rubric**



# **Finite State Machine**

****

# **Test Cases**

|  |  |  |
| --- | --- | --- |
| **TestID** | **Test Steps** | **Expected Results** |
| 1 | In “Idle” state press “voice mail” button | Voice mail menu is opened |
| 2 | In Voice mail menu press “exit” button | “Idle” state initiated |
| 3 | Receive a call | “Ringing(incoming)” state initiated |
| 4 | In “Ringing(incoming)” state press volume up or down button | Dial tone changed(up or down responsively) |
| 5 | In “Ringing(incoming)” state press “answer” button | System goes through “Answering” state to “Talking” state |
| 6 | In “Talking” state press “Hang Up” button | System goes through “Hanging up” state to “Disconnecting” state to “Idle” state |
| 7 | In “Talking” state your opponent presses “Hang Up” button | System goes through “Hanging up” state to “Disconnecting” state to “Idle” state |
| 8 | In “Ringing(incoming)” state press “Hang Up” button | System goes through “Hanging up” state to “Disconnecting” state to “Idle” state |
| 9 | In “Idle” state long press “speed dial” button then press “cancel” button | “Incomplete Dial” message, “Idle” state initiated |
| 10 | In “Idle” state initiate a call with voice command and press “cancel” button | “Incomplete Dial” message, “Idle” state initiated |
| 11 | In “Idle” state input number and press “call” button then press “cancel” button | “Incomplete Dial” message, “Idle” state initiated |
| 12 | In “Idle” state long press “speed dial” button on invalid number | “Invalid number” message, “Idle” state initiated |
| 13 | In “Idle” state initiate a call with voice command on invalid number | “Invalid number” message, “Idle” state initiated |
| 14 | In “Idle” state input invalid number and press “call” button | “Invalid number” message, “Idle” state initiated |
| 15 | In “Idle” state long press “speed dial” button on valid “Long distance” number | “Long Distance” message is showed then system goes through “Connecting” state to “Ringing(outgoing) state |
| 16 | In “Idle” state initiate a call with voice command on valid “Long distance” number | “Long Distance” message is showed then system goes through “Connecting” state to “Ringing(outgoing) state |
| 17 | In “Idle” state input valid “Long distance” number and press “call” button | “Long Distance” message is showed then system goes through “Connecting” state to “Ringing(outgoing) state |
| 18 | In “Idle” state long press “speed dial” button on valid number | System goes through “Connecting” state to “Ringing(outgoing) state |
| 19 | In “Idle” state initiate a call with voice command on valid number | System goes through “Connecting” state to “Ringing(outgoing) state |
| 20 | In “Idle” state input valid number and press “call” button | System goes through “Connecting” state to “Ringing(outgoing) state |
| 21 | In “Ringing(outgoing) state press volume button | Dial tone changed(up or down responsively) |
| 22 | In “Ringing(outgoing)” state if opponent does nothing for amount of time specified in requirements | System goes to “Timing out” state |
| 23 | In “Timing Out” state press “Redial” button | System goes to “Connecting” state |
| 24 | In “Ringing(outgoing)” state if opponent already talking to somebody | System goes through “Busy” state to “Dropped call” state |
| 25 | In “Dropped call” state press “Redial” button | System goes to “Connecting” state |
| 26 | In “Timing Out” state press “Cancel” button | System goes through “Disconnecting” state to “Idle” state |
| 27 | In “Dropped call” state press “Cancel” button | System goes through “Disconnecting” state to “Idle” state |
| 28 | In “Ringing(outgoing) state press “Hang Up” button | System goes through “Hanging Up” state to “Disconnecting” to “Idle” |
| 29 | In “Ringing(outgoing) state your opponent presses “Answer” button | System goes through “Answering” state to “Talking” state |
| 30 | In “Ringing(outgoing)” state your opponent presses “Hang Up” button | System goes through “Busy” state to “Dropped call” state |

# **C# source code**

## **Calc.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AwesomeCalculator

{

public class Calc

{

private double firstNumber;

private double secondNumber;

public Calc()

{

firstNumber = 1;

secondNumber = 1;

}

public Calc(double firstNumber, double secondNumber)

{

this.firstNumber = firstNumber;

this.secondNumber = secondNumber;

}

public double GetFirstNumber()

{

return firstNumber;

}

public double SetFirstNumber(double firstNumber)

{

this.firstNumber = firstNumber;

return this.firstNumber;

}

public double GetSecondNumber()

{

return secondNumber;

}

public double SetSecondNumber(double secondNumber)

{

this.secondNumber = secondNumber;

return this.secondNumber;

}

public double GetAddition()

{

return Math.Round((firstNumber + secondNumber), 2);

}

public double GetSubtraction()

{

return Math.Round((firstNumber - secondNumber), 2);

}

public double GetMultiplication()

{

return Math.Round((firstNumber \* secondNumber), 2);

}

public double GetDivision()

{

try

{

return Math.Round((firstNumber / secondNumber), 2);

}

catch (DivideByZeroException)

{

return (0);

}

}

}

}

## **Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AwesomeCalculator

{

class Program

{

public static int ValidateMenuSelection()

{

string userInput = "";

bool validMenuSelect = false;

while (validMenuSelect == false)

{

Console.WriteLine("1 = Get First Number");

Console.WriteLine("2 = Change First Number");

Console.WriteLine("3 = Get Second Number");

Console.WriteLine("4 = Change Second Number");

Console.WriteLine("5 = Perform Addition");

Console.WriteLine("6 = Perform Subtraction");

Console.WriteLine("7 = Perform Multiplication");

Console.WriteLine("8 = Perform Division");

Console.WriteLine("9 = Exit\n");

Console.WriteLine("Please select an option, by entering a number:\n");

userInput = Console.ReadLine();

if (userInput != "1" &&

userInput != "2" &&

userInput != "3" &&

userInput != "4" &&

userInput != "5" &&

userInput != "6" &&

userInput != "7" &&

userInput != "8" &&

userInput != "9")

{

Console.WriteLine("That's not a valid menu option, please try again:\n");

}

else

{

validMenuSelect = true;

}

}

Console.WriteLine();

return int.Parse(userInput);

}

public static double ValidateUserInput(string chosenNumber)

{

double aNumber = 1;

bool isValid = false;

while (isValid == false)

{

Console.WriteLine($"Please enter the {chosenNumber}:");

string userInput = Console.ReadLine();

Console.WriteLine();

bool result = double.TryParse(userInput, out aNumber);

if (result == false)

{

Console.WriteLine("That's not a valid input please, please try again.\n");

}

else

{

isValid = true;

Console.WriteLine($"Your {chosenNumber} has been changed to: {aNumber}.\n");

}

}

return aNumber;

}

static void Main(string[] args)

{

Calc c = new Calc();

bool validCalcSelect = false;

string calcSelection;

int selection;

while (validCalcSelect == false)

{

Console.WriteLine("1 = Use random numbers between 0 and 501 for your calculation\n");

Console.WriteLine("2 = Provide your own numbers\n");

Console.WriteLine("Choose a menu item to begin:");

calcSelection = Console.ReadLine();

Console.WriteLine();

if (calcSelection != "1" && calcSelection != "2")

{

Console.WriteLine("That's not a valid selection, please try again.\n");

}

else if (int.Parse(calcSelection) == 1)

{

validCalcSelect = true;

Random random = new Random();

double firstNumber;

double secondNumber;

firstNumber = Math.Round((random.NextDouble() \* 500), 2);

secondNumber = Math.Round((random.NextDouble() \* 500), 2);

Console.WriteLine($"Your random numbers are {firstNumber} and {secondNumber}.\n");

Calc customCalc = new Calc(firstNumber, secondNumber);

c = customCalc;

}

else if (int.Parse(calcSelection) == 2)

{

validCalcSelect = true;

double firstNumber;

double secondNumber;

firstNumber = ValidateUserInput("firstNumber");

secondNumber = ValidateUserInput("secondNumber");

Console.WriteLine($"Your custom numbers are {firstNumber} and {secondNumber}.\n");

Calc customCalc = new Calc(firstNumber, secondNumber);

c = customCalc;

}

}

selection = ValidateMenuSelection();

while (selection != 9)

{

double result;

switch (selection)

{

case 1:

Console.WriteLine($"First Number is: {c.GetFirstNumber()}\n");

break;

case 2:

result = ValidateUserInput("firstNumber");

c.SetFirstNumber(result);

break;

case 3:

Console.WriteLine($"Second Number is: {c.GetSecondNumber()}\n");

break;

case 4:

result = ValidateUserInput("secondNumber");

c.SetSecondNumber(result);

break;

case 5:

Console.WriteLine($"The result of {c.GetFirstNumber()} + {c.GetSecondNumber()} is: {c.GetAddition()}\n");

break;

case 6:

Console.WriteLine($"The result of {c.GetFirstNumber()} - {c.GetSecondNumber()} is: {c.GetSubtraction()}\n");

break;

case 7:

Console.WriteLine($"The result of {c.GetFirstNumber()} \* {c.GetSecondNumber()} is: {c.GetMultiplication()}\n");

break;

case 8:

Console.WriteLine($"The result of {c.GetFirstNumber()} / {c.GetSecondNumber()} is: {c.GetDivision()}\n");

break;

default:

break;

}

selection = ValidateMenuSelection();

}

}

}

}

## **CalcTests.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using AwesomeCalculator;

namespace UnitTests

{

[TestFixture]

public class CalcTests

{

[Test]

public void Addition\_input12p23And54p65\_expected66p88()

{

//Arrange

double first = 12.23;

double second = 54.65;

double expected = 66.88;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Addition\_input21p56And0p00\_expected21p56()

{

//Arrange

double first = 21.56;

double second = 0.00;

double expected = 21.56;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Addition\_inputNeg56p44And56p44\_expected0p00()

{

//Arrange

double first = -56.44;

double second = 56.44;

double expected = 0.00;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Subtraction\_input77p33And22p12\_expected55p21()

{

//Arrange

double first = 77.33;

double second = 22.12;

double expected = 55.21;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Subtraction\_input10p22And0p00\_expected10p22()

{

//Arrange

double first = 10.22;

double second = 0.00;

double expected = 10.22;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Subtraction\_input77p33And80p00\_expectedNeg2p67()

{

//Arrange

double first = 77.33;

double second = 80.00;

double expected = -2.67;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Multiplication\_input77p33And0p00\_expected0p00()

{

//Arrange

double first = 77.33;

double second = 0.00;

double expected = 0.00;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Multiplication\_input77p33AndNeg1p00\_expectedNeg77p33()

{

//Arrange

double first = 77.33;

double second = -1.00;

double expected = -77.33;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Multiplication\_input77p33And12p44\_expected961p99()

{

//Arrange

double first = 77.33;

double second = 12.44;

double expected = 961.99;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Division\_input77p33AndNeg1p00\_expectedNeg77p33()

{

//Arrange

double first = 77.33;

double second = -1;

double expected = -77.33;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Division\_input0p00And44p98\_expected0()

{

//Arrange

double first = 0.00;

double second = 44.98;

double expected = 0;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Division\_input77p33And0p00\_expectedInfinity()

{

//Arrange

double first = 77.33;

double second = 0;

double actual = 0;

double expected = Double.PositiveInfinity;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

}

}

## **Unit tests for subtraction mutant**

[TestFixture]

public class Mut\_tests

{

[Test]

public void Subtraction\_Input34p54And12p88\_Expected21p66()

{

//Arrange

double first = 34.54;

double second = 12.88;

double expected = 21.66;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Subtraction\_Input10p00And10p00\_Expected0()

{

//Arrange

double first = 10.00;

double second = 10.00;

double expected = 0;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Subtraction\_InputNeg2p22And3p33\_ExpectedNeg5p55()

{

//Arrange

double first = -2.22;

double second = 3.33;

double expected = -5.55;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Subtraction\_Input5p00AndNeg5p00\_Expected10p00()

{

//Arrange

double first = 5.00;

double second = -5.00;

double expected = 10.00;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetSubtraction();

//Assert

Assert.AreEqual(expected, actual);

}

}

## **Unit tests for Addition mutant**

[TestFixture]

public class Mut\_Tests

{

[Test]

public void Addition\_input12p23And54p65\_expected66p88()

{

//Arrange

double first = 12.23;

double second = 54.65;

double expected = 66.88;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Addition\_input21p56And1p00\_expected22p56()

{

//Arrange

double first = 21.56;

double second = 1.00;

double expected = 22.56;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Addition\_input56p44And56p44\_expected112p88()

{

//Arrange

double first = 56.44;

double second = 56.44;

double expected = 112.88;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Addition\_input56p44AndNeg56p44\_expected0p00()

{

//Arrange

double first = 56.44;

double second = -56.44;

double expected = 0;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetAddition();

//Assert

Assert.AreEqual(expected, actual);

}

}

## **Unit tests for Division mutant**

[TestFixture]

public class Mut\_Tests

{

[Test]

public void Division\_input77p33AndNeg2p00\_expectedNeg38p66()

{

//Arrange

double first = 77.33;

double second = -2;

double expected = -38.66;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Division\_input10p00And44p98\_expected0p22()

{

//Arrange

double first = 10.00;

double second = 44.98;

double expected = 0.22;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Division\_input77p33And77p33\_expected1p00()

{

//Arrange

double first = 77.33;

double second = 77.33;

double actual = 0;

double expected = 1;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Division\_input1p33And1p34\_expected2p67()

{

//Arrange

double first = 1.33;

double second = 1.34;

double actual = 0;

double expected = 0.99;

Calc c = new Calc(first, second);

//Act

actual = c.GetDivision();

//Assert

Assert.AreEqual(expected, actual);

}

}

## **Unit tests for Multiplication mutant**

[TestFixture]

public class Mut\_Tests

{

[Test]

public void Multiplication\_input77p33And12p00\_expected927p96()

{

//Arrange

double first = 77.33;

double second = 12.00;

double expected = 927.96;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Multiplication\_input77p33And0p50\_expected77p33()

{

//Arrange

double first = 77.33;

double second = 0.50;

double expected = 38.66;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Multiplication\_input70p20AndNeg2p00\_expectedNeg140p40()

{

//Arrange

double first = 70.20;

double second = -2.00;

double expected = -140.40;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

//Assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Multiplication\_input50p22AndNeg50p22\_expectedNeg2522p05()

{

//Arrange

double first = 50.22;

double second = -50.22;

double expected = -2522.05;

double actual = 0;

Calc c = new Calc(first, second);

//Act

actual = c.GetMultiplication();

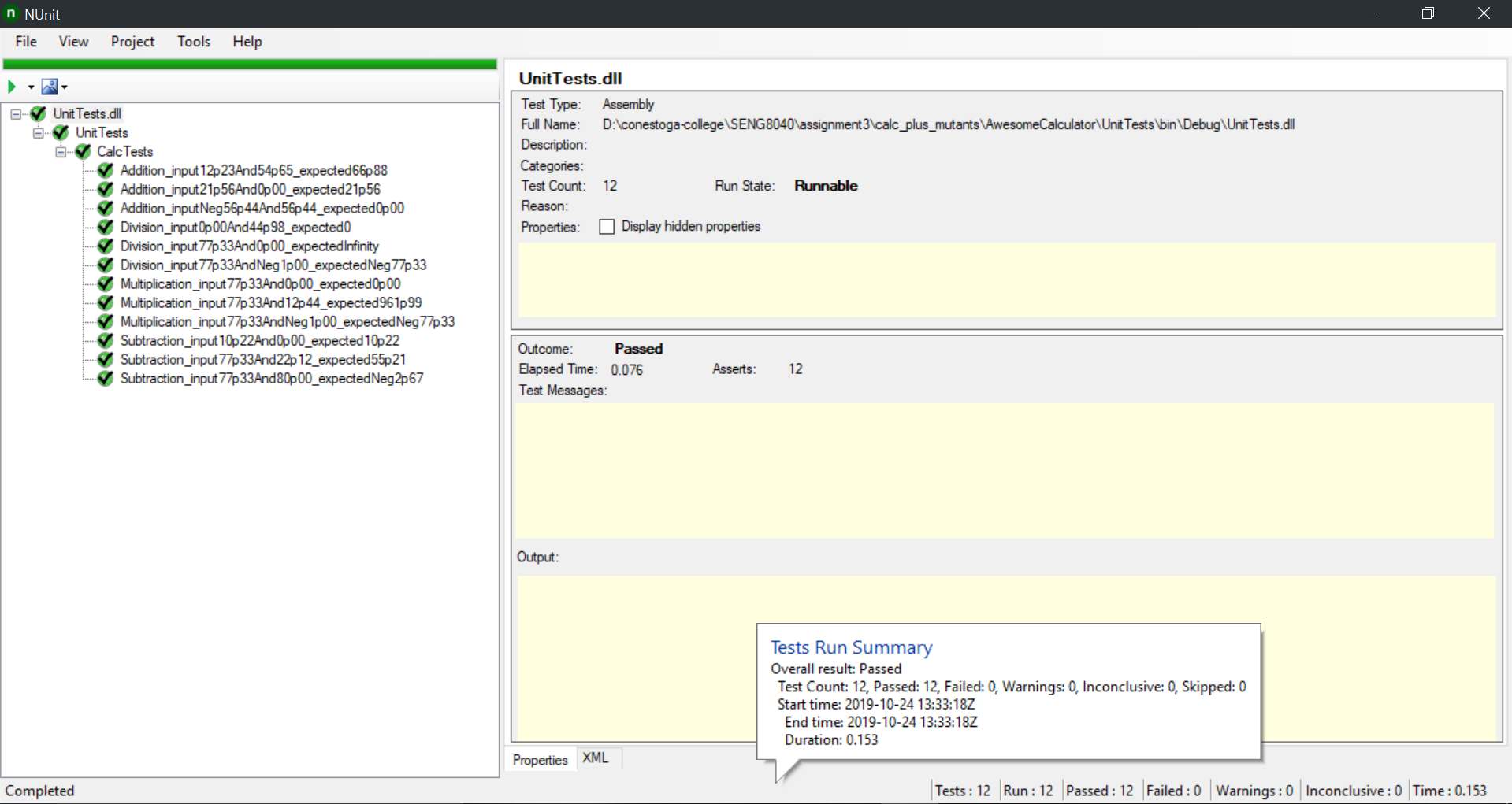
//Assert

Assert.AreEqual(expected, actual);

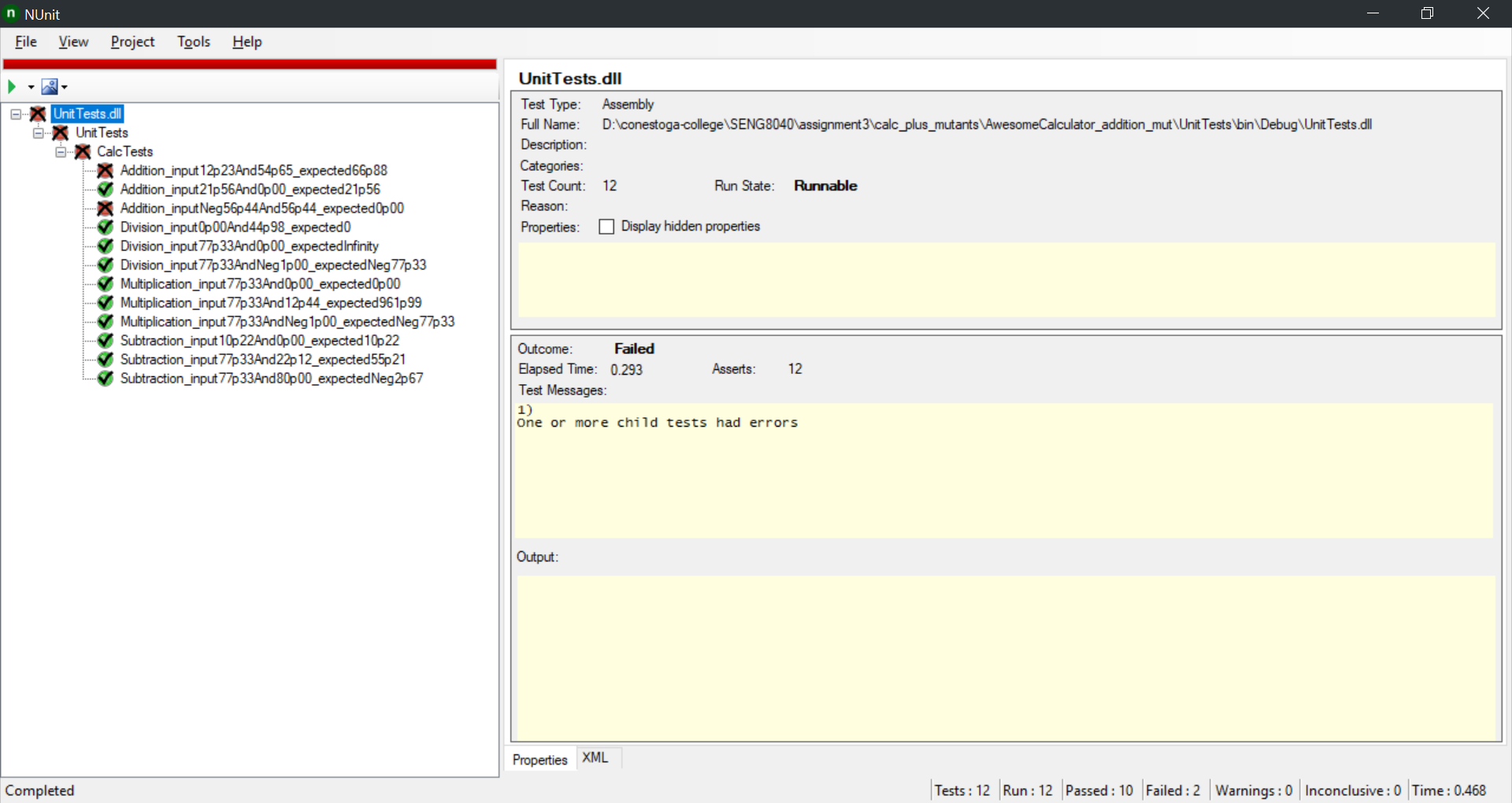
}

}

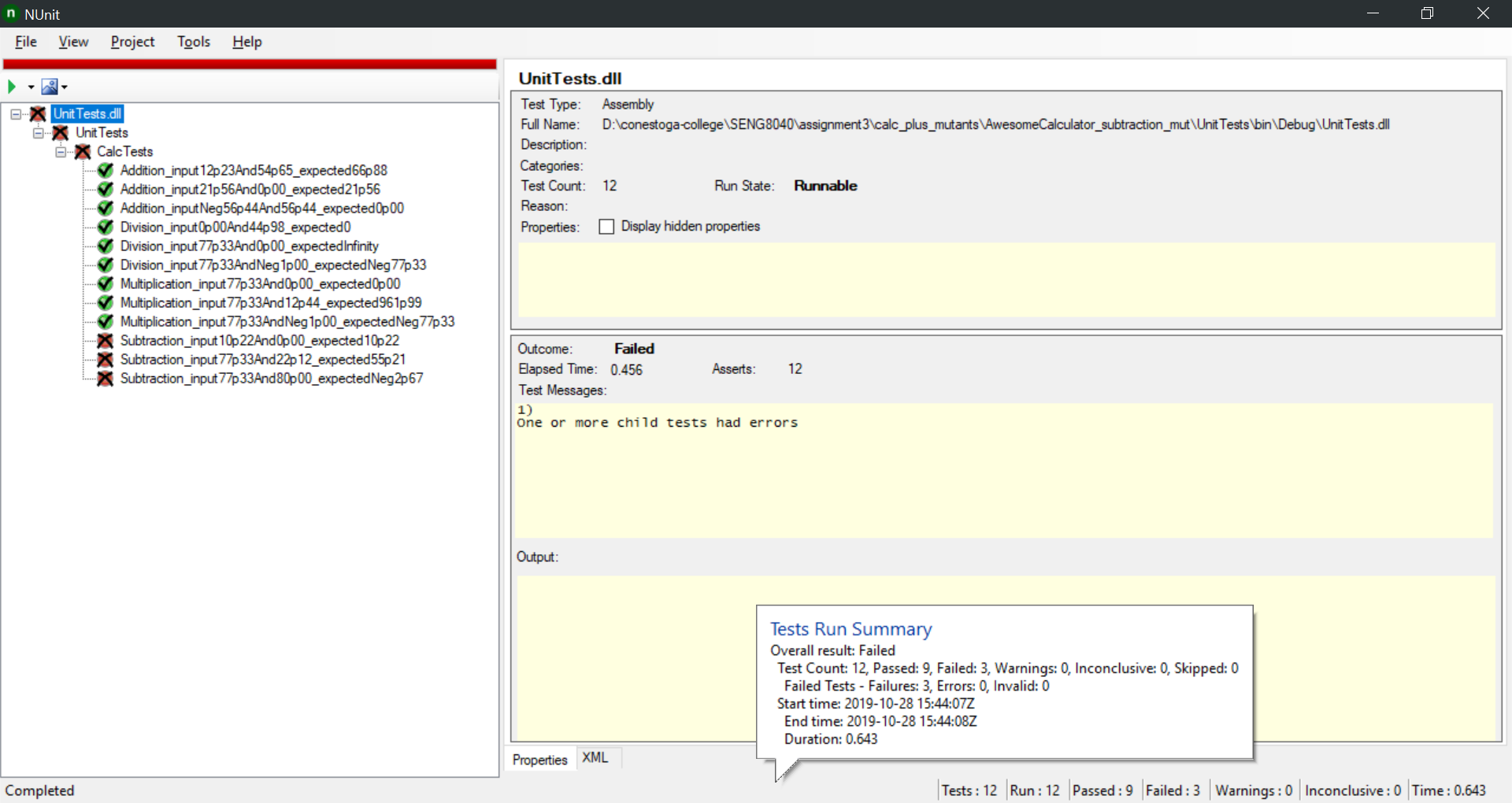
**Unit tests for main program**



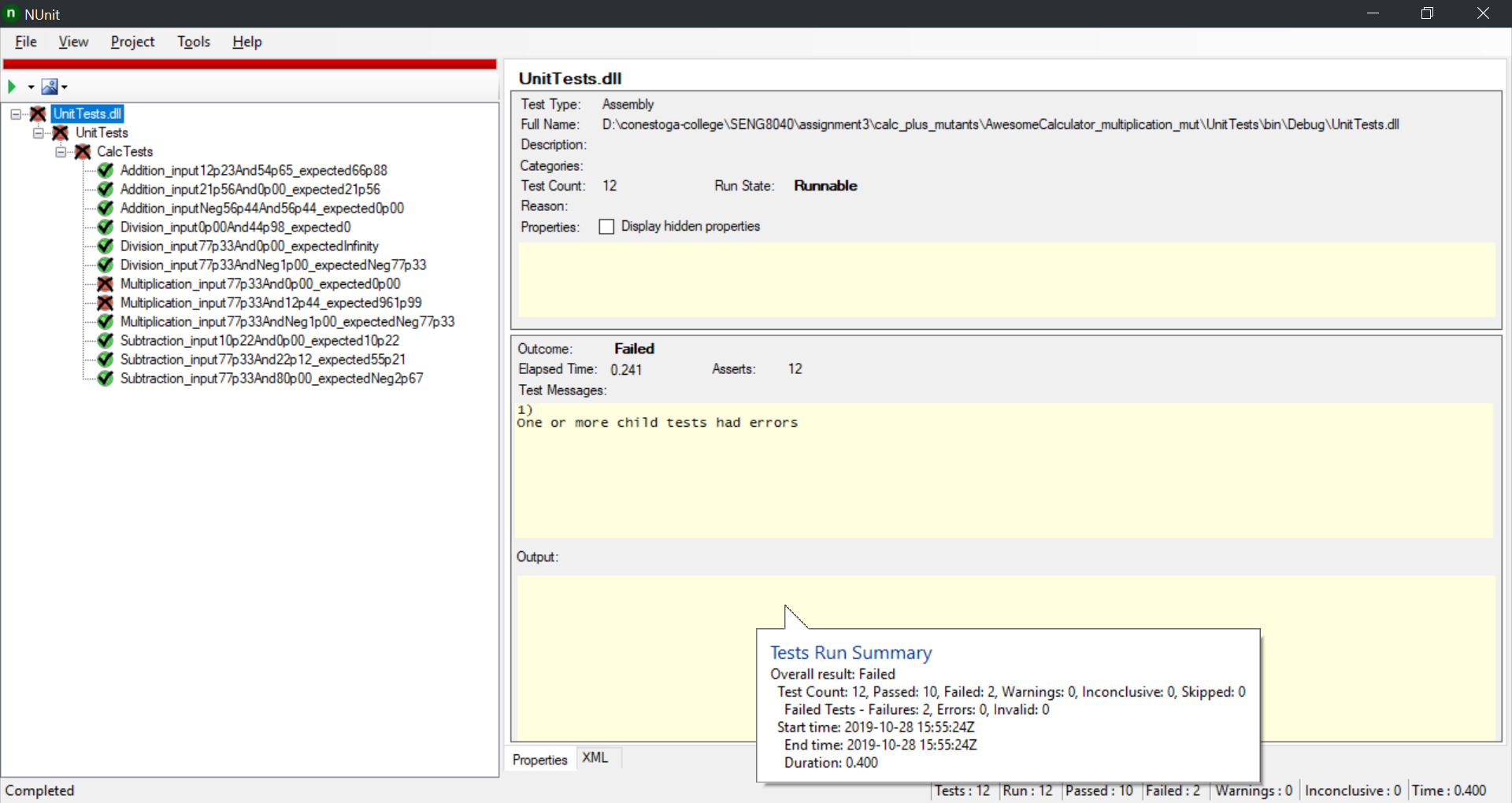
**Initial Unit tests for Addition mutant**



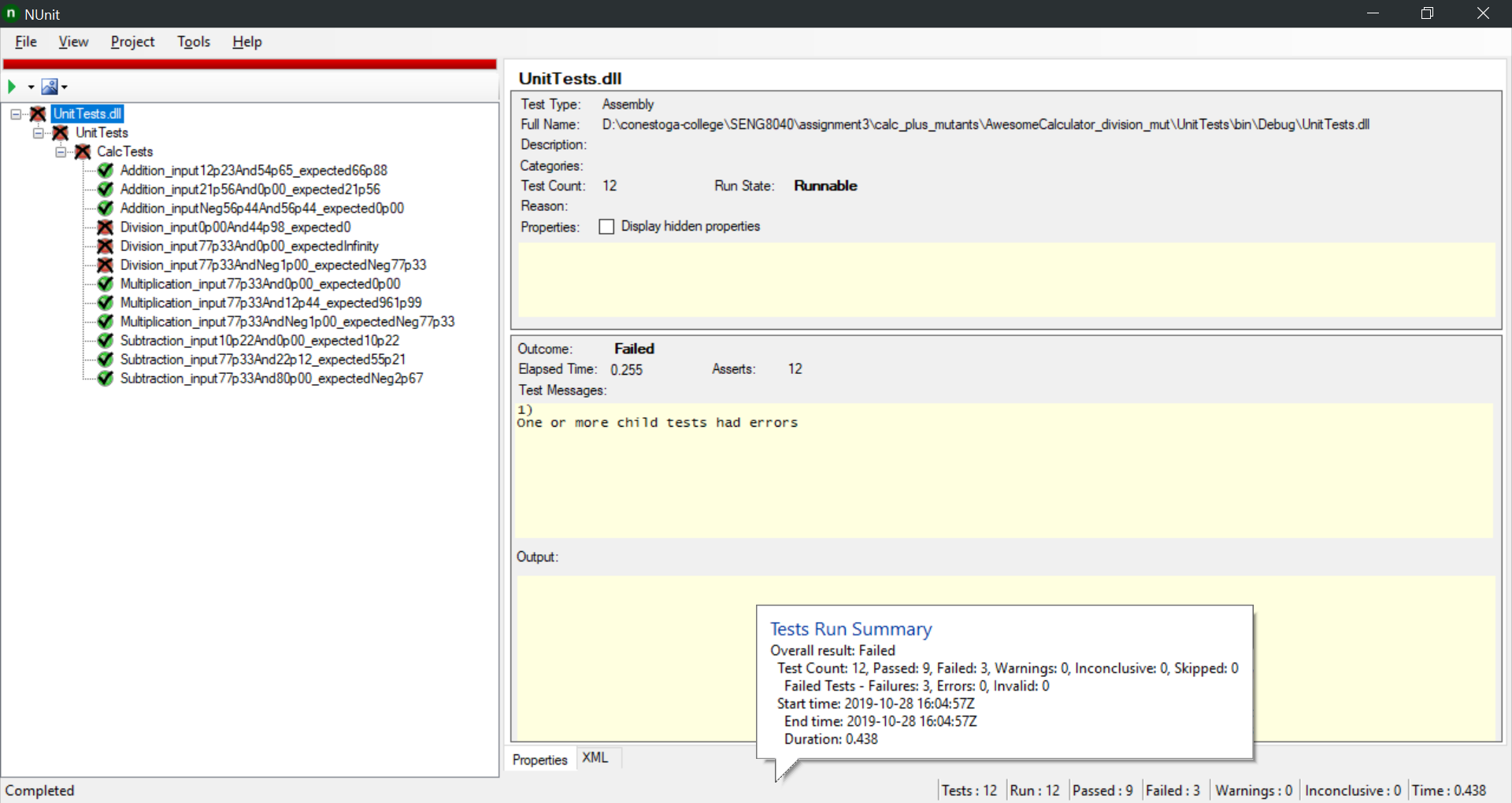
**Initial tests run on Subtraction mutant**



**Initial Unit tests for Multiplication mutant**



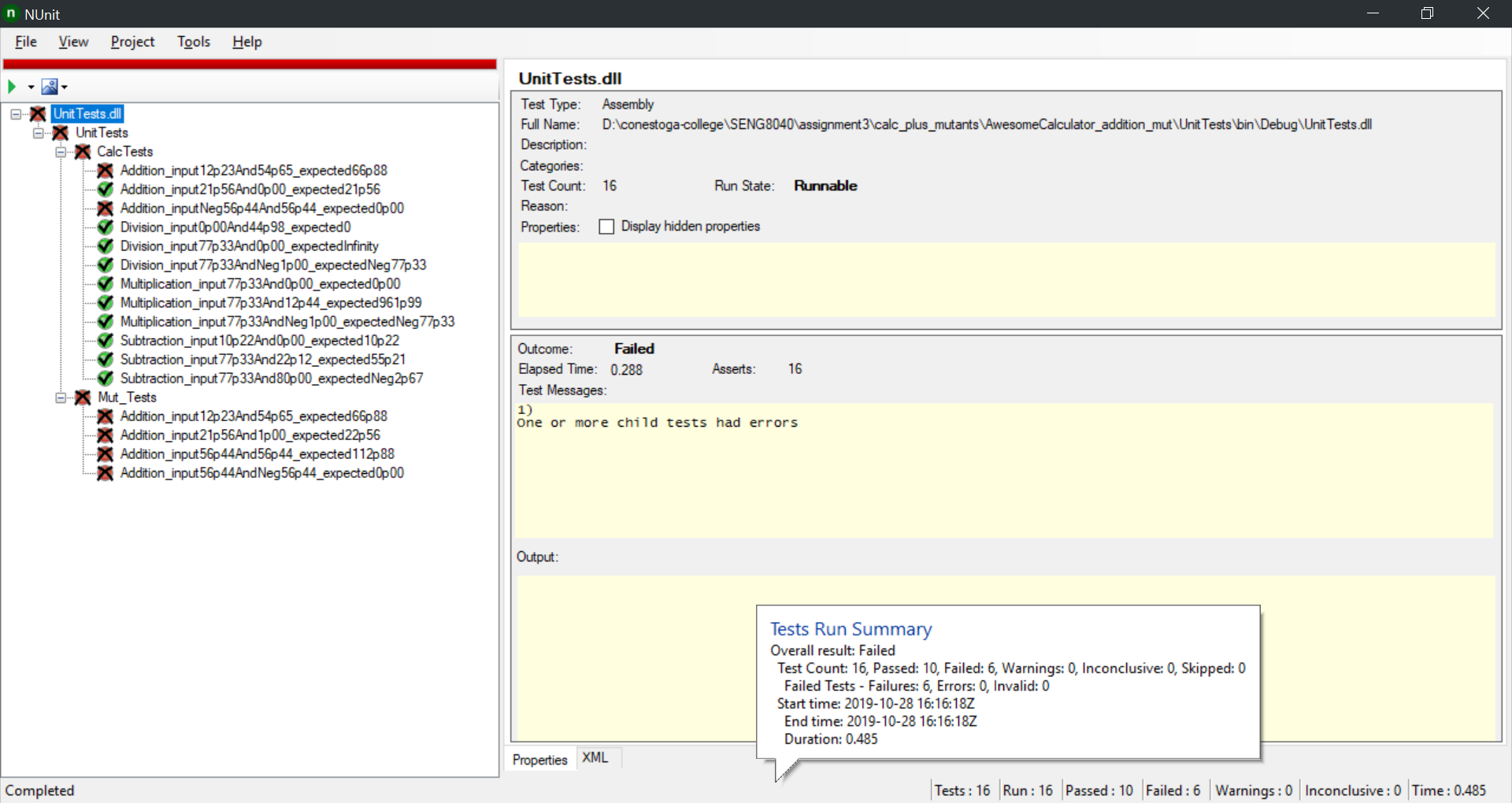
**Initial Unit tests for Division mutant**



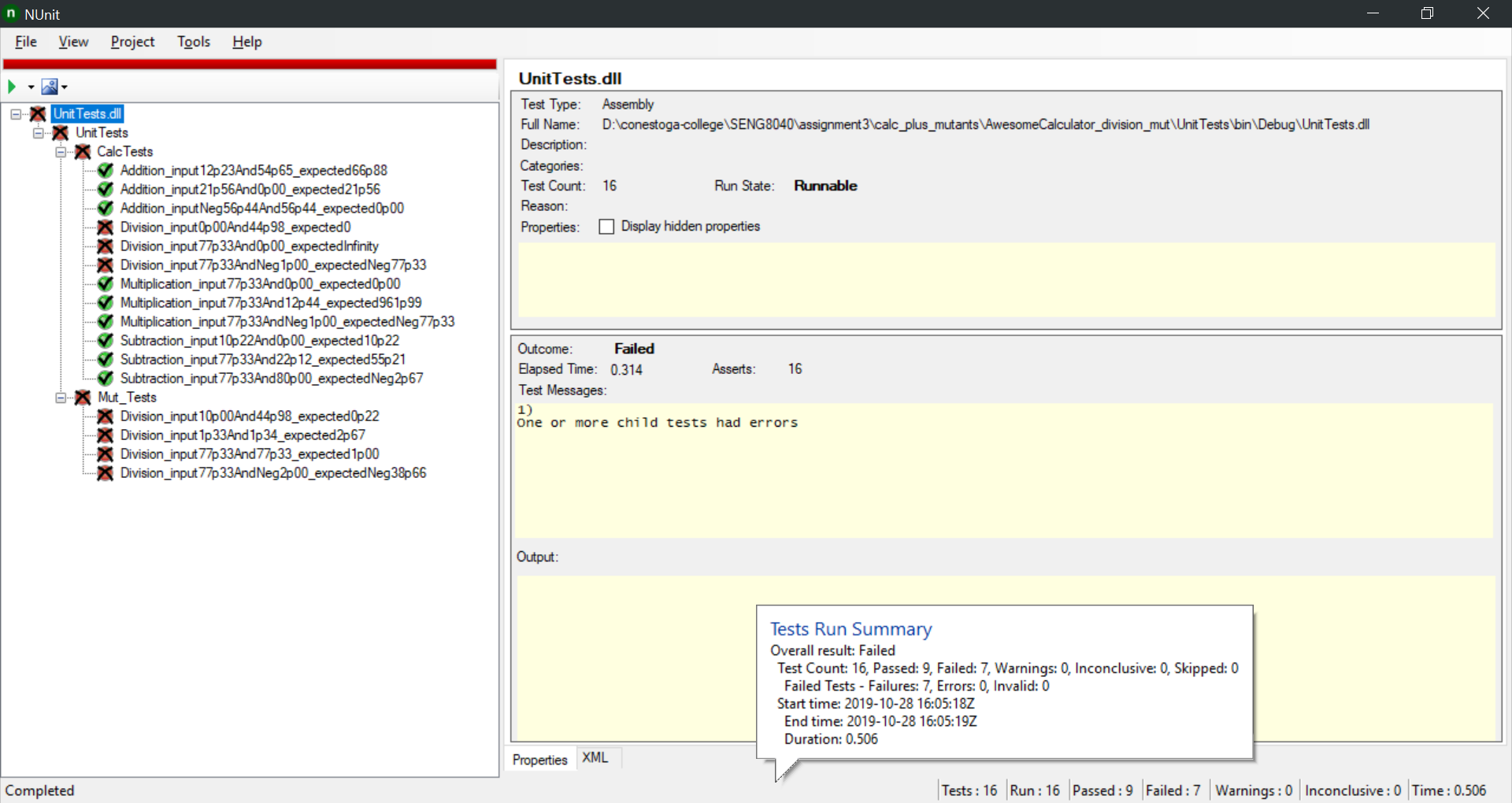
## **Results about initial test suites for mutants**

As we can see from screenshots, mutation score of initial test suit is 100%. No more than one of three unit tests passes.

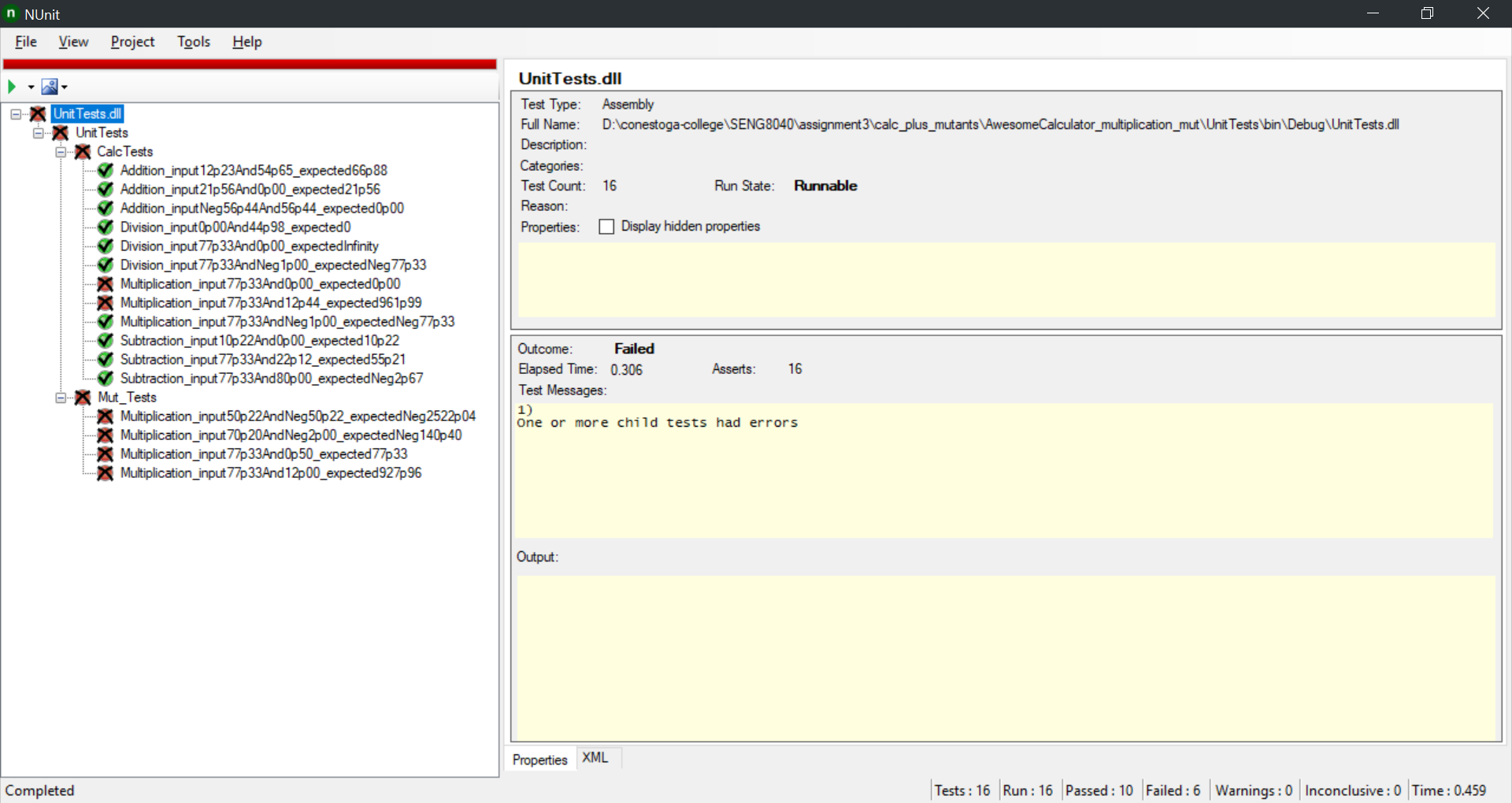
**Additional Unit tests for Addition mutant**



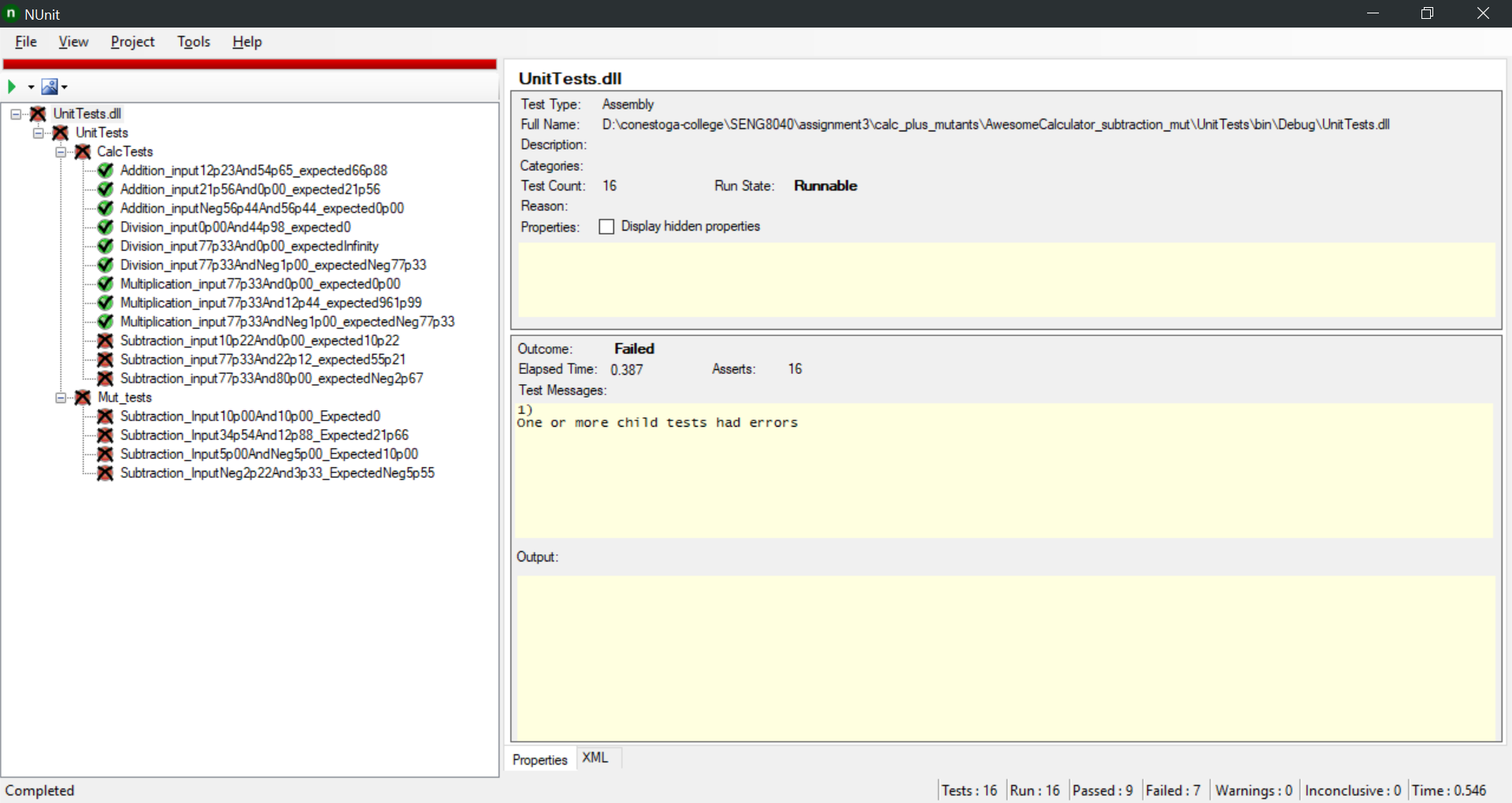
**Additional Unit tests for Division mutant**



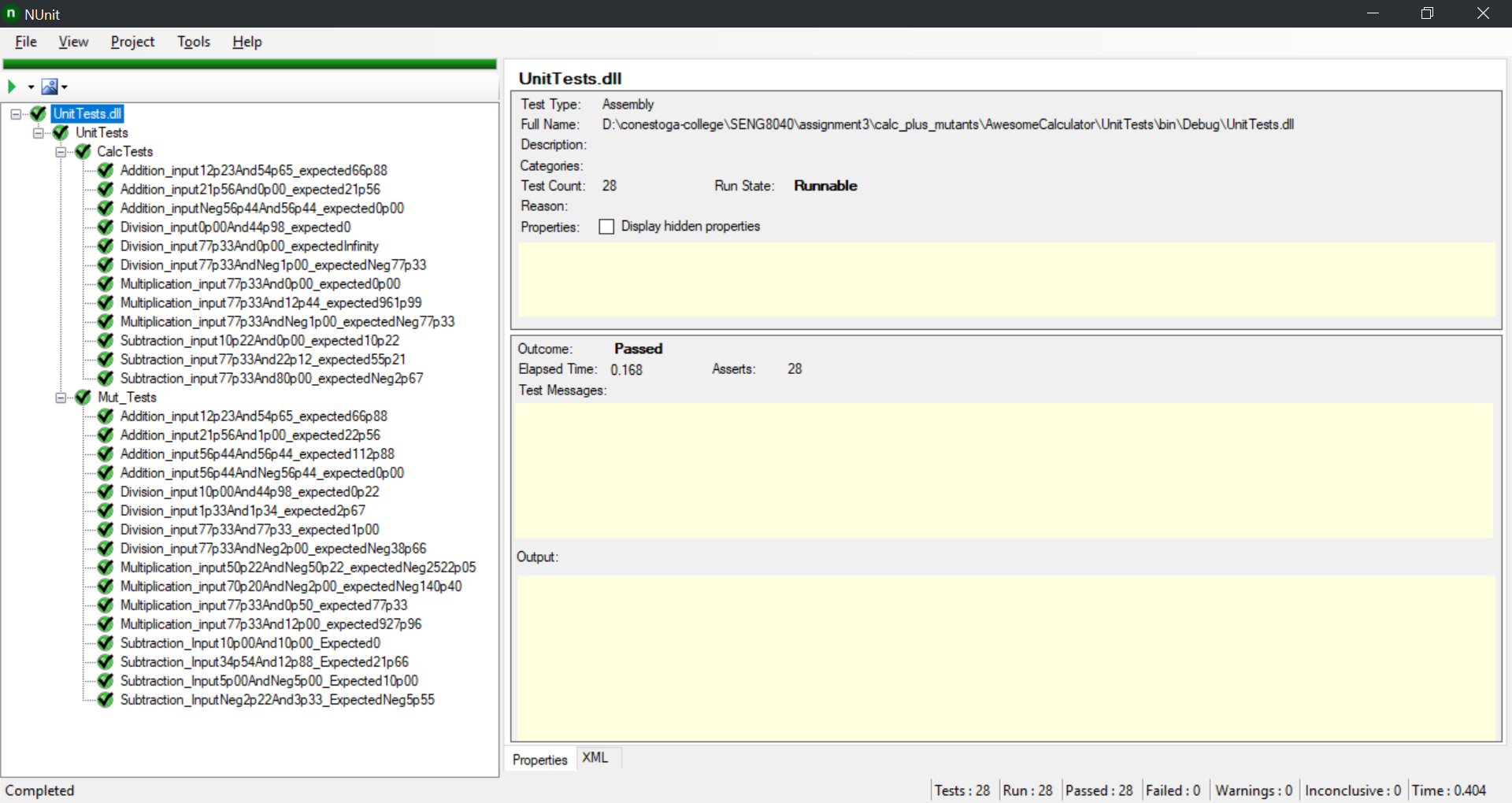
**Additional Unit tests for Multiplication mutant**



**Additional Unit tests for Subtraction mutant**



**Mutant tests run on main program**



## **Results of additional unit testing for mutants**

After adding 4 unit tests for each mutant the mutation score is still 100% but we improved the quality of test suite by expanding it and making it more robust to other changes to the code. We assuring that these methods will work as supposed to.

## **GIT**

