Tutorial – Unit Testing

Ver. 1.1, 20th March, 2007
Ver. 2.0, 20th October, 2008
Ver. 2.1, 16th October, 2009
Ver. 3.0, 10th October, 2010
Ver. 3.1, 21th October, 2013
Ver. 4.0, 16th October, 2021

Contents

About This Document

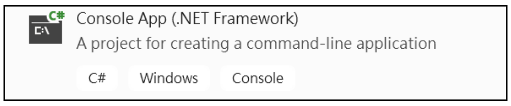
This document describes how to use Visual Studio 2019 to conduct unit testing. A unit testing framework is installed and integrated with Visual Studio. In addition, the unit testing framework offers testers a very useful capability to access/test non-public (private or protected) methods and properties. Another useful feature is the code coverage tool that can be used to analyze the statement coverage of test cases. In this tutorial, the unit testing framework and code coverage tool are used to test an **account** class that stores an amount of money in a bank. The account class keeps a record of all deposits/withdraws (so that the deposits/withdraws are traceable) and can do three jobs:

- ✓ Withdraw – to remove money from the account
- ✓ Deposit – to put money in the account
- ✓ Balance – calculate the money available in the account

Test Bank Account

Step 1 Create a project for production code

Create a C# Console App (.NET Framework) project, named Bank. Note: there are several different Console Application projects in Visual Studio; choosing the wrong ones will not work for this tutorial.


Console App (.NET Framework)
A project for creating a command-line application

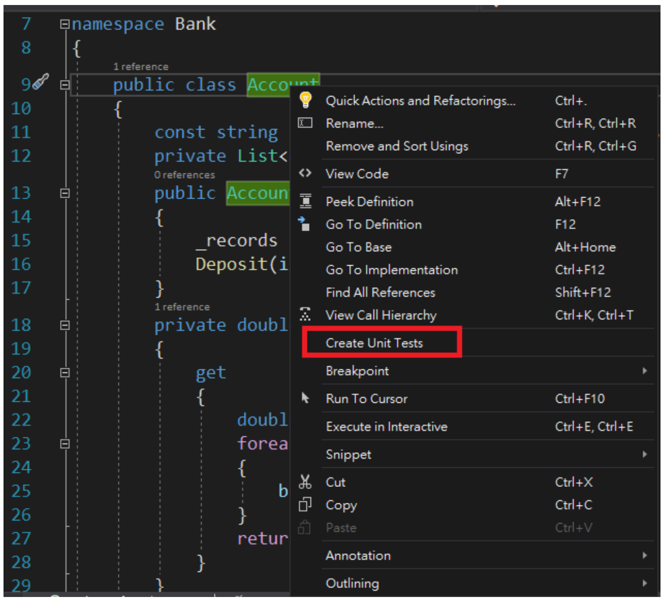
Step 2 Add a class: Account

Add a class, named Account, into the Bank project, and fill the class with the following code. *Note: don't modify any line; just copy and paste the code.*

```
public class Account {
     const string ERROR_MSG = "Sorry, you don't have so much money!!";
     private List<double> _records;
     public Account(uint initAmount) {
          _records = new List<double>();
          Deposit(initAmount);
     }
     // Note: we make Balance private to demonstrate the testing of private members.
     private double Balance {
          get {
               double balance = 0;
               foreach (double record in _records) {
                    balance += record;
               }
               return balance;
          }
     }
     public void Deposit(uint amount) {
          _records.Add(Decimal.ToDouble(amount));
     }
     public void Withdraw(uint amount) {
          if (amount > Balance)
               Console.WriteLine(ERROR_MSG);
          _records.Add(Decimal.ToDouble(-amount));
     }
}
```

Step 3 Create a project for test code

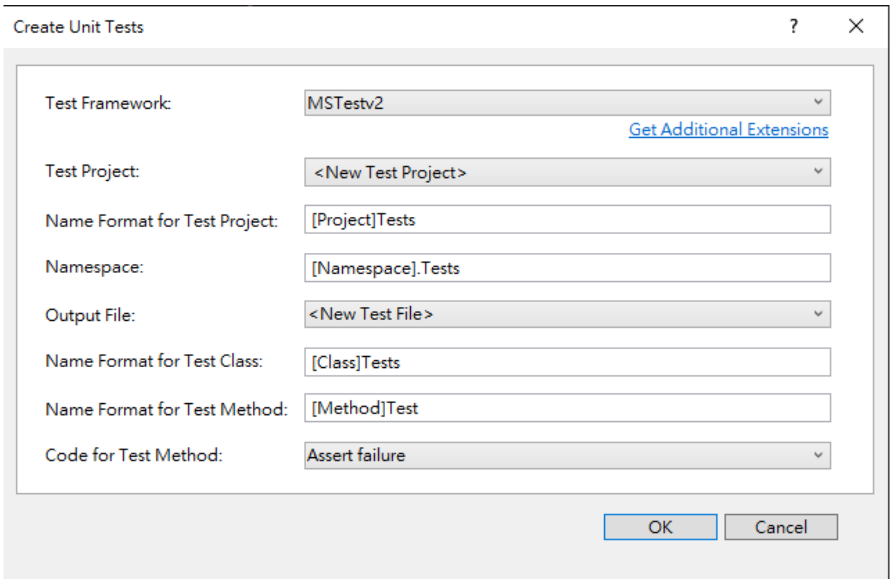
Right click on the Account class in the code editor (Figure 1) and choose the option: "Create Unit Tests" in the popup context menu. In the dialog, shown in Figure 2, accept all default values and click the button OK. A new project called BankTests will be created (see Fig. 3).



```
namespace Bank
{
    1 reference
    public class Account
    {
        const string
        private List<
        0 references
        public Accoun
        {
            _records
            Deposit(i
        }
        1 reference
        private doubl
        {
            get
            {
                doubl
                forea
                {
                    b
                }
                retur
            }
        }
```

Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
View Code F7
Peek Definition Alt+F12
Go To Definition F12
Go To Base Alt+Home
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Create Unit Tests
Breakpoint
Run To Cursor Ctrl+F10
Execute in Interactive Ctrl+E, Ctrl+E
Snippet
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Annotation
Outlining

Figure 1 Create a unit test for the Account class



Create Unit Tests

Test Framework:	MSTestv2
Test Project:	<New Test Project>
Name Format for Test Project:	[Project]Tests
Namespace:	[Namespace].Tests
Output File:	<New Test File>
Name Format for Test Class:	[Class]Tests
Name Format for Test Method:	[Method]Test
Code for Test Method:	Assert failure

Figure 2 Create Unit Tests Dialog

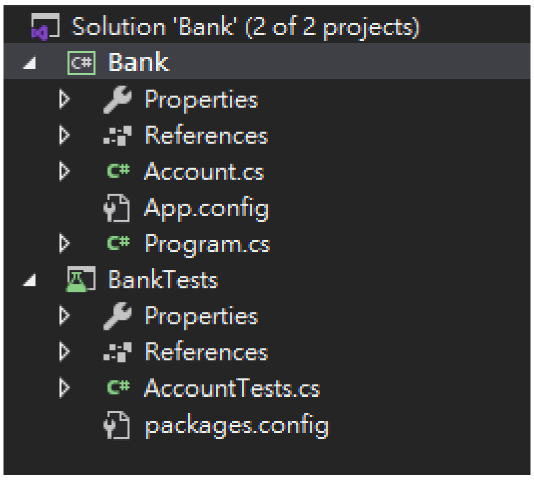
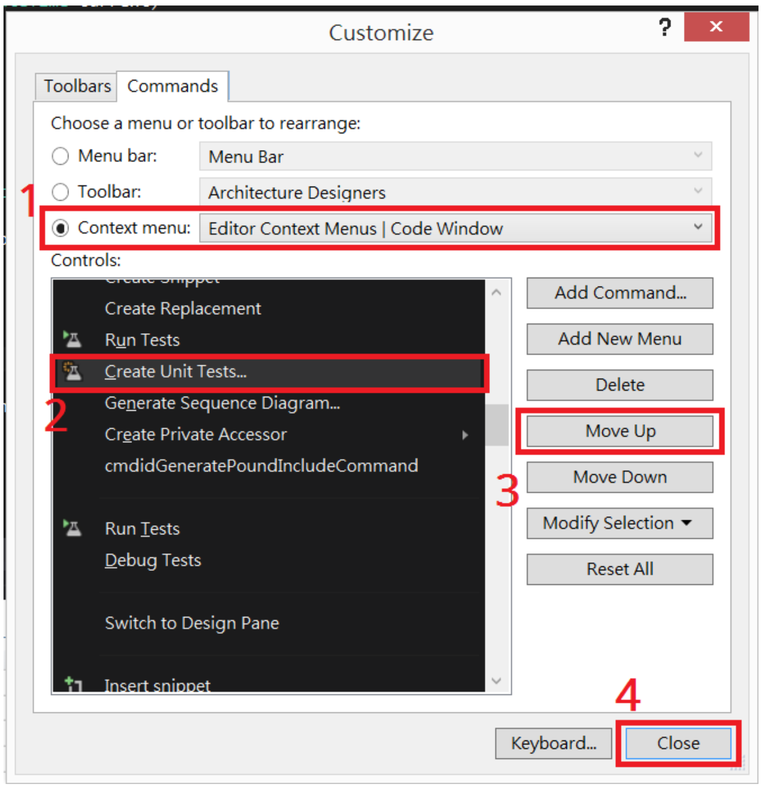


Figure 3 Two separated projects: Bank and BankTests

In case that the option: "Create Unit Tests" do not appear in the popup context menu, you can use the steps shown in Fig 4. to bring it up. Note that you need to restart Visual Studio in this case.



Customize

Choose a menu or toolbar to rearrange:

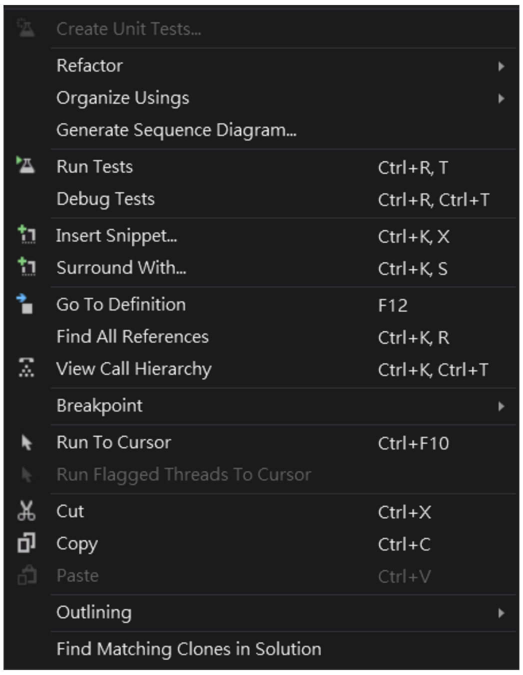
Menu bar: Menu Bar

Toolbar: Architecture Designers

[✓ Context menu: Editor Context Menus | Code Window]

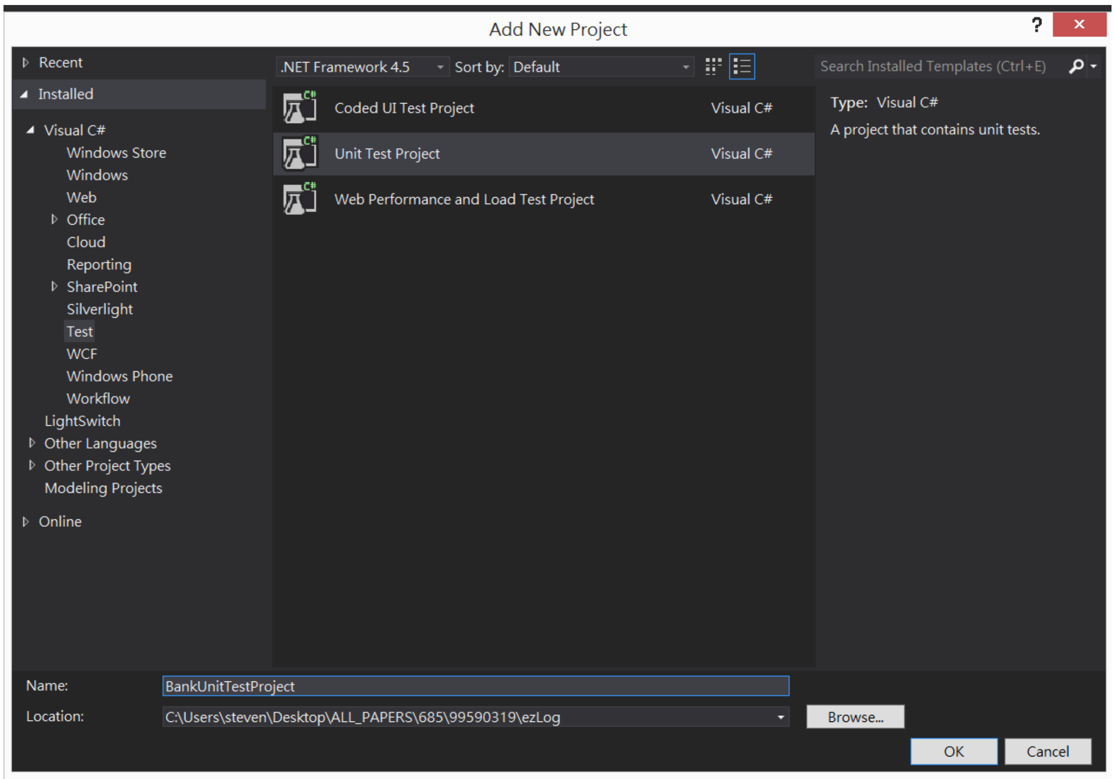
Figure 4 TOOLS > customize... > Commands

In case that the option is disabled, you can add a Unit Test Project in your solution and add a test case in that project.



Create Unit Tests...
Refactor
Organize Usings
Generate Sequence Diagram...
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, X
Surround With... Ctrl+K, S
Go To Definition F12
Find All References Ctrl+K, R
View Call Hierarchy Ctrl+K, Ctrl+T
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining
Find Matching Clones in Solution

Figure 5 Create Unit Tests is **disabled**



Type: Visual C#

A project that contains unit tests.

Name: BankUnitTestProject

Location: C:\Users\steven\Desktop\ALL_PAPERS\685\99590319\ezLog

Figure 6 Create a Unit Test Project

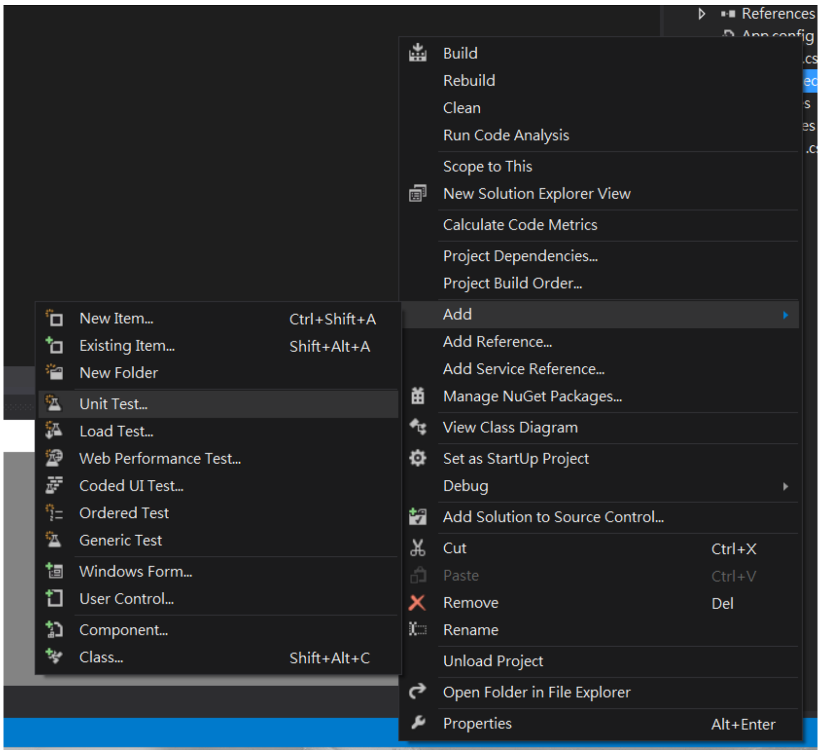


Figure 7 Create a Unit Test

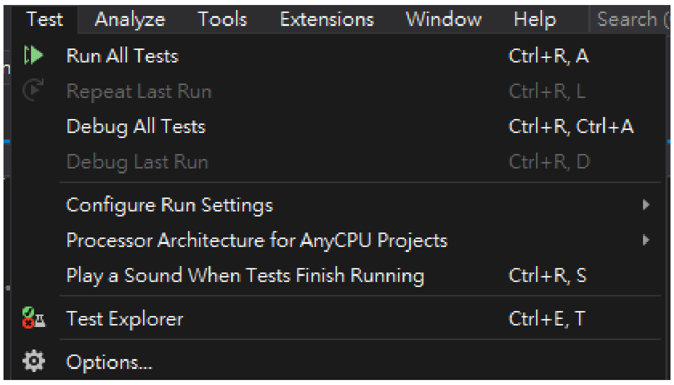
Step 4 Add test cases

Step 3 creates a test project and an AccountTests class. However, the test code generated by Visual Studio is useless in this tutorial. *Copy the following code to replace the automatically generated AccountTests class.* In the following code, the `PrivateObject` class represents the live non-public internal object in the system and can be used to access non-public members and invoke non-public methods (by using the reflection mechanism).

```
    [TestClass()]
    public class AccountTests
    {
        const uint INIT_AMOUNT = 100;
        const uint DEPOSITED = 200;
        const uint BIG_WITHDRAW = 300;
        PrivateObject _accountPrivate;
        Account _account;

        [TestInitialize()]
        public void Initialize()
        {
            _account = new Account(INIT_AMOUNT);
            _accountPrivate = new PrivateObject(_account);
        }

        [TestMethod()]
        public void DepositTest()
        {
            Assert.AreEqual((double)INIT_AMOUNT,
_accountPrivate.GetFieldOrProperty("Balance"));
                _account.Deposit(DEPOSITED);
                Assert.AreEqual((double)(INIT_AMOUNT + DEPOSITED),
_accountPrivate.GetFieldOrProperty("Balance"));
            }

            [TestMethod()]
            public void WithdrawTest()
            {
                Assert.AreEqual((double)INIT_AMOUNT,
_accountPrivate.GetFieldOrProperty("Balance"));
                _account.Withdraw(BIG_WITHDRAW);
                Assert.AreEqual((double)INIT_AMOUNT,
_accountPrivate.GetFieldOrProperty("Balance"));
                _account.Deposit(DEPOSITED);
                Assert.AreEqual((double)(INIT_AMOUNT + DEPOSITED),
_accountPrivate.GetFieldOrProperty("Balance"));
                _account.Withdraw(BIG_WITHDRAW);
                double expected = (double)(INIT_AMOUNT + DEPOSITED) - BIG_WITHDRAW;
                Assert.AreEqual(expected, _accountPrivate.GetFieldOrProperty("Balance"));
            }

            [TestMethod()]
            public void BalanceTest()
            {
                Assert.AreEqual((double)INIT_AMOUNT,
_accountPrivate.GetFieldOrProperty("Balance"));
            }
        }
```

Step 5 Run unit tests

Click Test → Run All Tests in Solution (Test->Run All Tests; Fig. 8). Visual Studio finds all test methods in the solution and then executes all of them. The result is shown in the bottom. ***Note that the keywords surrounded with brackets ([TestMethod()]) are annotations used to identify test methods. Don't remove them or Visual Studio will not be able to find any test methods to execute.***

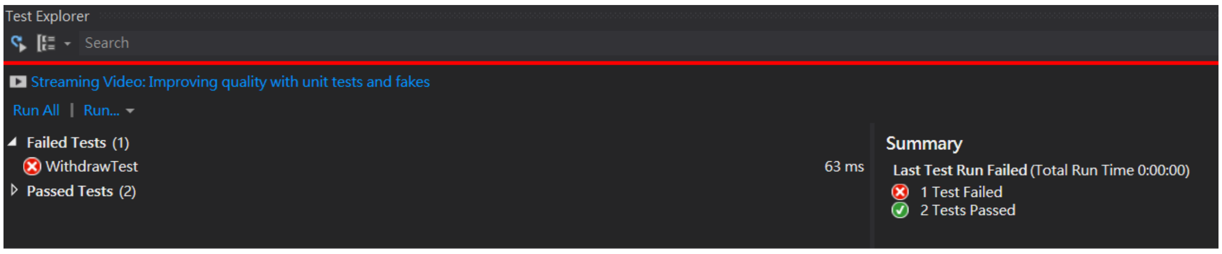


Test
Analyze
Tools
Extensions
Window
Help
Search
Run All Tests Ctrl+R, A
Repeat Last Run Ctrl+R, L
Debug All Tests Ctrl+R, Ctrl+A
Debug Last Run Ctrl+R, D
Configure Run Settings
Processor Architecture for AnyCPU Projects
Play a Sound When Tests Finish Running Ctrl+R, S
Test Explorer Ctrl+E, T
Options...

Figure 8 Run all tests in solution

Step 6 Debugging

After running the test cases, you will find that one of the test cases failed (Fig. 9). Please find and fix the bug in the Account class so that all test cases passes (see Fig. 10).


Test Explorer

Search

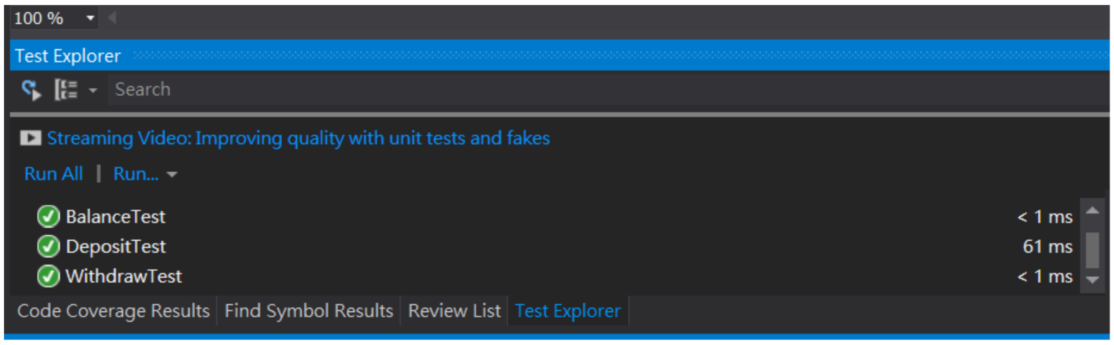
Streaming Video: Improving quality with unit tests and fakes

Run All | Run...

Failed Tests (1)
WithdrawTest 63 ms
Passed Tests (2)

Summary
Last Test Run Failed (Total Run Time 0:00:00)
1 Test Failed
2 Tests Passed

Figure 9 Hmm...A test failed


100 %

Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run...

```
BalanceTest      < 1 ms
DepositTest       61 ms
WithdrawTest     < 1 ms
```

Code Coverage Results | Find Symbol Results | Review List | Test Explorer

Figure 10 All Pass

Step 7 Test Coverage

Since Visual Studio 2019 (community) doesn't support code coverage analysis, you need to install the extension Fine Code Coverage (FCC) to make VS2019 (community) support code coverage analysis. FCC is a free code coverage analysis tool designed for Visual Studio Community Edition (and other editions). Follow the steps shown in Fig. 11, 12, and 13 to install the extension.

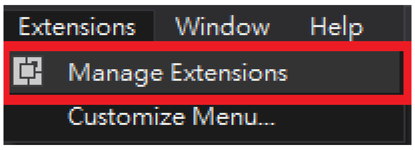
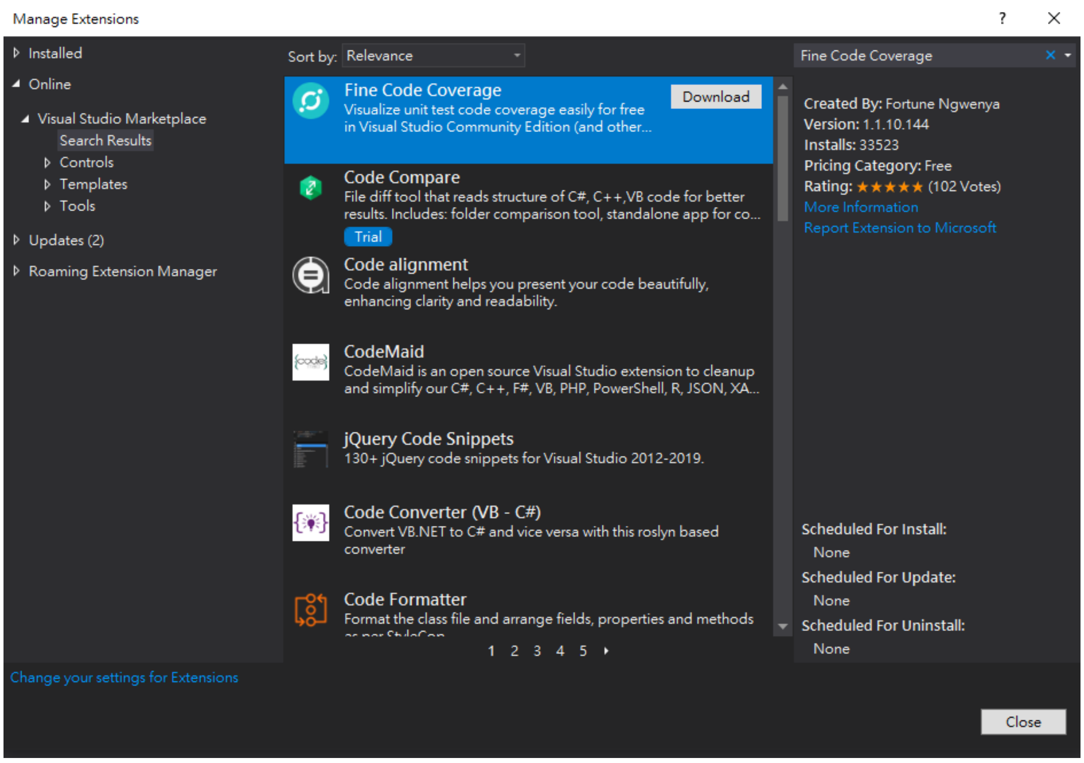


Figure 11 Manage Extensions



Fine Code Coverage
Visualize unit test code coverage easily for free in Visual Studio Community Edition (and other...

Code Compare
File diff tool that reads structure of C#, C++,VB code for better results. Includes: folder comparison tool, standalone app for co...

Code alignment
Code alignment helps you present your code beautifully, enhancing clarity and readability.

CodeMaid
CodeMaid is an open source Visual Studio extension to cleanup and simplify our C#, C++, F#, VB, PHP, PowerShell, R, JSON, XA...

jQuery Code Snippets
130+ jQuery code snippets for Visual Studio 2012-2019.

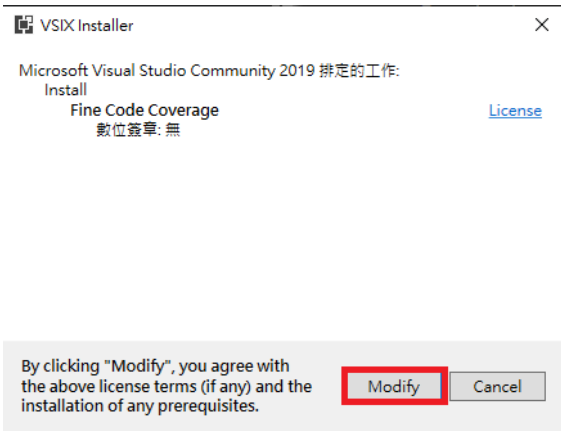
Code Converter (VB - C#)
Convert VB.NET to C# and vice versa with this roslyn based converter

Code Formatter
Format the class file and arrange fields, properties and methods

Created By: Fortune Ngwenya
Version: 1.1.10.144
Installs: 33523
Pricing Category: Free
Rating: ★★★★★ (102 Votes)
More Information
Report Extension to Microsoft

Scheduled For Install:
None
Scheduled For Update:
None
Scheduled For Uninstall:
None

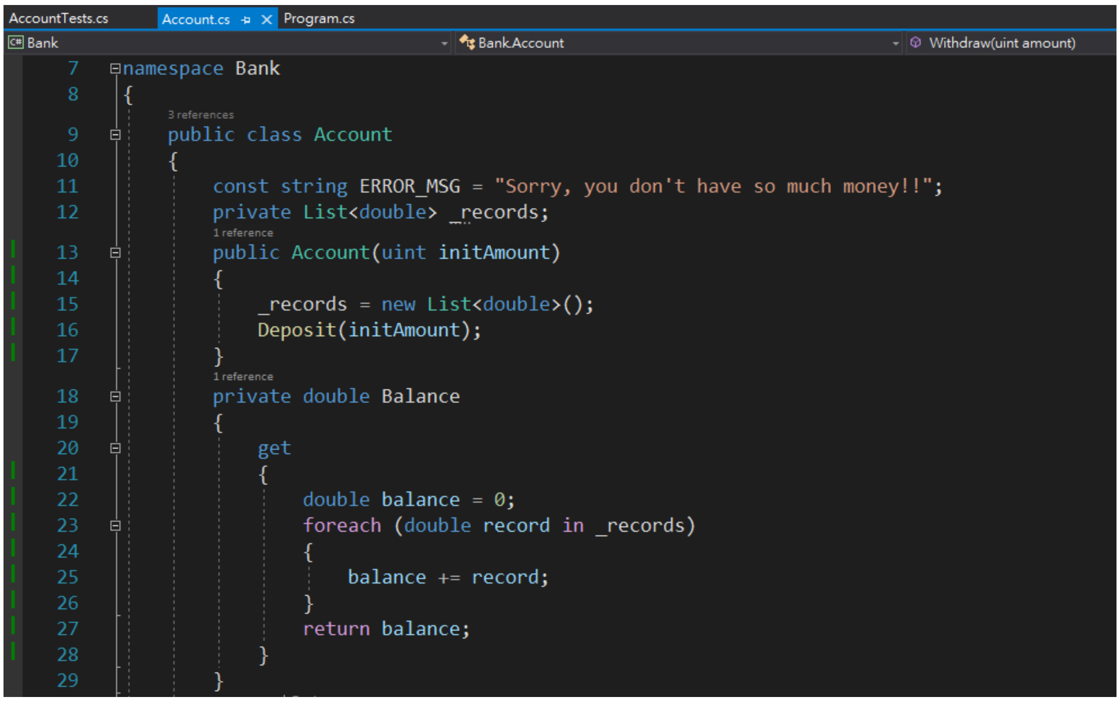
Figure 12 **Download extension** “Fine Code Coverage” and **close Visual Studio**



Microsoft Visual Studio Community 2019 排定的工作:
Install
Fine Code Coverage
數位簽章: 無

By clicking "Modify", you agree with the above license terms (if any) and the installation of any prerequisites.

Figure 13 Modify Visual Studio and open Visual Studio



```
namespace Bank
{
    3 references
    public class Account
    {
        const string ERROR_MSG = "Sorry, you don't have so much money!!";
        private List<double> _records;
        1 reference
        public Account(uint initAmount)
        {
            _records = new List<double>();
            Deposit(initAmount);
        }
        1 reference
        private double Balance
        {
            get
            {
                double balance = 0;
                foreach (double record in _records)
                {
                    balance += record;
                }
                return balance;
            }
        }
```

Figure 14 Run all tests again, and then you will get highlights on the code

Then you can rerun all tests to view the code coverage. Fig. 15 shows how to open the FCC window (View -> Other Windows -> Fine Code Coverage) which shows the code coverage result in the bottom of Visual Studio. Overall, the AccountTests class covered 100% statements of the Account class (Fig. 16).

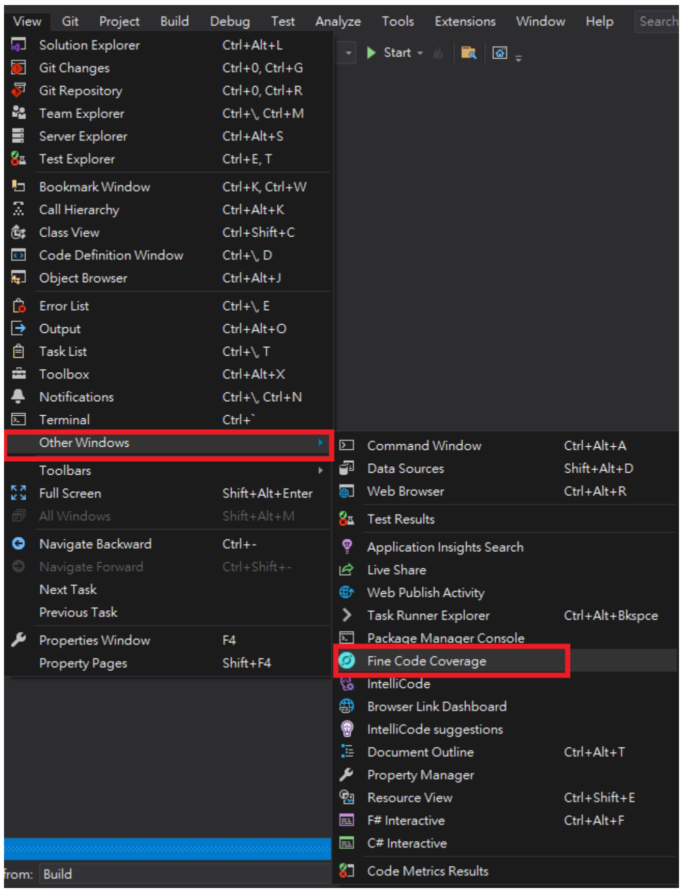
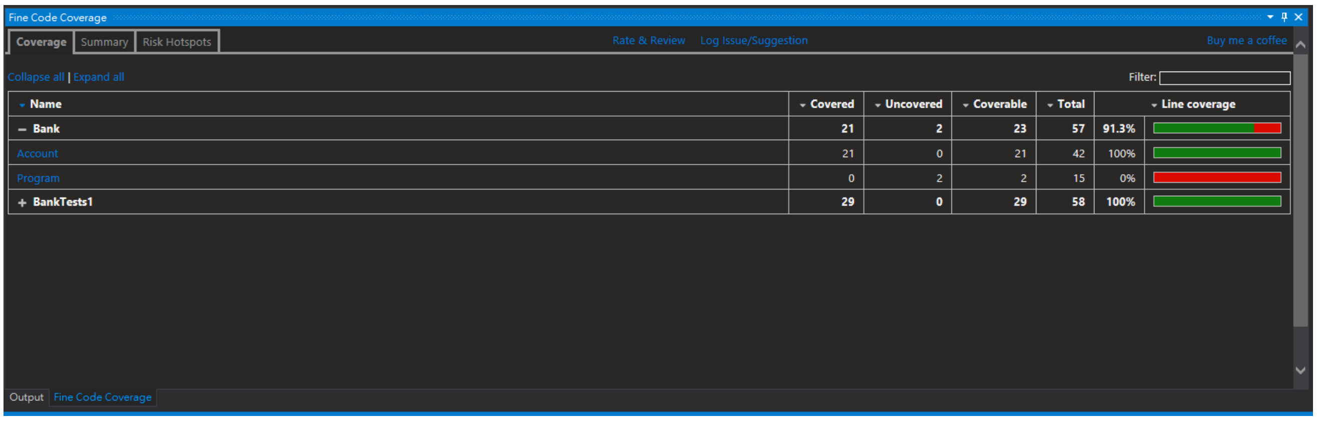

View Git Project Build Debug Test Analyze Tools Extensions Window Help Search
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+R
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Test Explorer Ctrl+E, T
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+\, Ctrl+N
Terminal Ctrl+`
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
All Windows Shift+Alt+M
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Next Task
Previous Task
Properties Window F4
Property Pages Shift+F4
Start
Command Window Ctrl+Alt+A
Data Sources Shift+Alt+D
Web Browser Ctrl+Alt+R
Test Results
Application Insights Search
Live Share
Web Publish Activity
Task Runner Explorer Ctrl+Alt+Bkspce
Package Manager Console
Fine Code Coverage
IntelliCode
Browser Link Dashboard
IntelliCode suggestions
Document Outline Ctrl+Alt+T
Property Manager
Resource View Ctrl+Shift+E
F# Interactive Ctrl+Alt+F
C# Interactive
Code Metrics Results
from: Build

Figure 15 open Fine Code Coverage


Fine Code Coverage
Coverage Summary Risk Hotspots
Rate & Review Log Issue/Suggestion
Buy me a coffee
Collapse all | Expand all
Filter:
Output Fine Code Coverage

Name	Covered	Uncovered	Coverable	Total	Line coverage	
− Bank	21	2	23	57	91.3%	🟩🟥
Account	21	0	21	42	100%	🟩
Program	0	2	2	15	0%	🟥
+ BankTests1	29	0	29	58	100%	🟩

Figure 16 100% covered Account class

IMPORTANT NOTE:

If a unit test method uses a file, you may need to copy the file to [TestProject] -> bin -> debug so that FCC can find the file. Otherwise, FCC could report a very low code coverage.

-- The End --