100 XP

Examine the unit testing tools and environment

8 minutes

GitHub Copilot Chat can be used to suggest unit tests based on the code you specify. For example, if you specify a method signature and body, GitHub Copilot Chat can suggest test cases that cover input parameters and expected output values. Once your test project contains a few test cases, you can use GitHub Copilot to generate code completion suggestions for additional test cases. Using code line completions to generate unit tests reduces the repetitive process and helps to ensure that your code is thoroughly tested.

GitHub Copilot Chat uses your code's context and semantics to suggest assertions that ensure the function is working correctly. It also helps you write test cases for edge cases and boundary conditions that might be difficult to identify manually. For example, GitHub Copilot Chat can suggest test cases for error handling, null values, or unexpected input types.

> **Important**
>
> Generated test cases may not cover all possible scenarios. Manual testing and code reviews are necessary to ensure the quality of your code.

GitHub Copilot support for unit testing

Copilot can help with the following testing tasks:

- Set up testing frameworks: get help configuring the right testing framework and VS Code extensions for your project and language.
- Generate test code: create unit tests, integration tests, and end-to-end tests that cover your application code.
- Handle edge cases: generate comprehensive test suites to cover edge cases and error conditions.
- Fix failing tests: receive suggestions for fixing test failures.
- Maintain consistency: personalize Copilot to generate tests that follow your project's coding practices.

Set up your testing framework

To accelerate your testing workflow, Copilot can help set up the testing framework and VS Code extensions for your project. Copilot suggests appropriate testing frameworks based on your project type.

1. Open the Chat view.
2. Enter the `/setupTests` command in the chat input field.
3. Follow GitHub Copilot's guidance to configure your project.

Write unit tests with GitHub Copilot

GitHub Copilot can help you write tests for your application code by generating test code that covers your codebase. This includes unit tests, end-to-end tests, and tests for edge cases.

You can generate unit tests using the following GitHub Copilot tools:

- **Chat view**: Use the Chat view generate unit tests for a project, class, or method using Ask, Edit, or Agent mode.
- **Inline Chat**: Use Inline Chat to generate unit tests for selected classes or methods.
- **Smart actions**: Use the Generate Tests smart action to generate unit tests for selected code without writing a prompt.
- **Code line completions**: Use code line completions to suggest addition unit tests for an existing test case.

Fix failing tests

Copilot integrates with the Test Explorer in Visual Studio Code and can help with fixing failing tests.

1. In the Test Explorer, hover over a failing test.
2. Select the **Fix Test Failure** button (sparkle icon)
3. Review and apply Copilot's suggested fix.

Alternatively, you can:

1. Open the Chat view.
2. Enter the `/fixTestFailure` slash command.
3. Follow Copilot's suggestions to fix the test

> **Tip**
>
> Agent mode monitors the test output when running tests, and automatically attempts to fix and rerun failing tests.

Maintain consistency

If your organization has specific testing requirements, you can customize how Copilot generates tests to ensure they meet your standards. You can personalize how Copilot generates tests by providing custom instructions. For example:

- Specify preferred testing frameworks
- Define naming conventions for tests
- Set code structure preferences
- Request specific test patterns or methodologies

Visual Studio Code support for unit tests

To create and run C# unit tests in Visual Studio Code, you need the following resources:

- The .NET 8.0 SDK or later.
- The C# Dev Kit extension for Visual Studio Code.
- A test framework package added to your project.

C# Dev Kit support for unit tests

The C# Dev Kit extension for Visual Studio Code provides a rich set of features to help you create and manage unit tests for your C# projects. It includes the following features:

- Test Explorer - A tree view to show all the test cases in your workspace.
- Run/Debug test cases - A feature to run and debug test cases.
- View test results - A feature to view the test results.
- Testing commands - Commands to run all tests, run failed tests, and more.
- Testing settings - Settings specific to testing.
- Test framework package

The C# Dev Kit supports the following test frameworks:

- xUnit
- NUnit
- MSTest

The C# Dev Kit enables you to create a test project, add a test framework package, and manage unit tests.

Enabling a test framework

The Command Palette in Visual Studio Code provides the easiest way to enable a test framework for your project. You can open the Command Palette in the following ways:

- Press the **Ctrl** + **Shift** + **P** keys (Windows/Linux) or **Cmd** + **Shift** + **P** (macOS).
- Open the **View** menu, and then select **Command Palette**.
- Open the Solution Explorer view, right-click the solution folder, and then select **New Project**. This option opens the Command Palette with the **.NET:New Project...** command already selected.

The following sections describe how to enable a test framework for your C# project using the Command Palette.

xUnit

Open the Command Palette and select **.NET:New Project...** then select **xUnit Test Project** and provide name and location for the new project. This command creates a new project and directory that uses xUnit as the test library and configures the test runner by adding the following `<PackageReference />` elements to the project file.

- `Microsoft.NET.Test.Sdk`
- `xUnit`
- `xunit.runner.visualstudio`
- `coverlet.collector`

From the Terminal, you can run the following command:

.NET CLI

```
   dotnet add [location of your test csproj file] reference [location of
the csproj file for project to be tested]
```

NUnit

Open the Command Palette and select **.NET:New Project...** then select **NUnit3 Test Project** and provide name and location for the new project. This command creates a new project and

directory that uses NUnit as the test library and configures the test runner by adding the following `<PackageReference />` elements to the project file.

- `Microsoft.NET.Test.Sdk`
- `NUnit`
- `NUnit3TestAdapter`

From the Terminal, run the following command:

.NET CLI

```
    dotnet add [location of your test csproj file] reference [location of
the csproj file for project to be tested]
```

MSTest

Open the Command Palette and select **.NET:New Project...** then select **MSTest Test Project** and provide name and location for the new project. This command creates a new project and directory that uses MSTest as the test library and configures the test runner by adding the following `<PackageReference />` elements to the project file.

- `Microsoft.NET.Test.Sdk`
- `MSTest.TestAdapter`
- `MSTest.TestFramework`
- `coverlet.collector`

From the Terminal, run the following command:

.NET CLI

```
    dotnet add [location of your test csproj file] reference [location of
the csproj file for project to be tested]
```

Develop unit tests in Visual Studio Code using GitHub Copilot Chat

The combination of Visual Studio Code and GitHub Copilot Chat provides a powerful environment for creating and running unit tests for your C# projects.

The Unit testing process can be broken down into three stages:

- Use Visual Studio Code to create a test project for your unit tests.

- Use GitHub Copilot Chat to develop unit test cases for your C# project.
- Use Visual Studio Code and the C# Dev Kit to run and manage your unit tests.

Create a test project

You need to create a test project that will be used to hold your unit tests. You can use Visual Studio Code to complete the following tasks:

1. Use the Command Palette to create a test project that uses a specified test framework.
2. Use the integrated Terminal to add a reference to the project you're testing.

This process creates a new project and configures the test runner for the selected framework.

Generate unit test cases using GitHub Copilot Chat

GitHub Copilot Chat can be used to help you write unit test cases for your test framework. Copilot Chat recognizes your test framework and coding style and generates matching code snippets. You can use Copilot Chat to complete the following tasks:

- Write unit test cases for your test framework based on the code open in the editor or the code snippet you highlight in the editor. Copilot identifies your test framework and coding style and generates matching code snippets.
- Identify and write test cases for edge cases and boundary conditions that might be difficult to identify manually. For instance, Copilot can suggest test cases for error handling, null values, or unexpected input types.
- Suggest assertions that ensure the function is working correctly, based on the code's context and semantics. For example, generate assertions to ensure that function input parameters are valid.

Consider the following scenarios when asking Copilot Chat to generate unit test cases:

- If a single method is visible in full in the editor, you could ask Copilot Chat to generate a unit test for the method by typing `Write a unit test for the method in the #editor.`
- If there are multiple methods visible or the intended method extends beyond what's visible in the editor, select the code you want to generate a unit test for, then ask Copilot: `#selection write a unit test for this code.`

Run and manage unit tests in Visual Studio Code

Visual Studio Code and the C# Dev Kit provide a rich set of features to help you run and manage unit tests for your C# projects. You can run/debug test cases, view test results, and manage test cases using the Test Explorer.

- Run/Debug test cases: C# Dev Kit generates shortcuts (the green play button) on the left side of the class and method definition. To run the target test cases, select the green play button. You can also right-click on it to see more options.
- Test Explorer: The Test Explorer is a tree view to show all the test cases in your workspace. You can select the beaker button on the left-side Activity bar of Visual Studio Code to open it. You can also run/debug your test cases and view their test results from there.
- View test results: After running/debugging the test cases, the state of the related test items is updated in both editor decorations and the Test Explorer. You can select the links in the stack trace to navigate to the source location.
- Visual Studio Code testing commands: There are testing commands (for example, Run All Tests) that can be found by searching for Test: in the Command Palette.
- Visual Studio Code testing settings: There are Visual Studio Code settings specific to testing that can be found by searching for Testing in the Settings editor.

Summary

GitHub Copilot Chat can be used to suggest unit tests based on the code you specify. For example, if you specify a method signature and body, GitHub Copilot Chat can suggest test cases that cover input parameters and expected output values. Once your test project contains a few test cases, you can use GitHub Copilot to generate code completion suggestions for additional test cases. Using code line completions to generate unit tests reduces the repetitive process and helps to ensure that your code is thoroughly tested.

GitHub Copilot Chat uses your code's context and semantics to suggest assertions that ensure the function is working correctly. It also helps you write test cases for edge cases and boundary conditions that might be difficult to identify manually. For example, GitHub Copilot Chat can suggest test cases for error handling, null values, or unexpected input types.

Next unit: Create unit tests using the Generate Tests smart action

Previous Next