Best Practices for Naming Unit Tests

Naming unit tests clearly and descriptively is essential for maintaining readable and understandable test code. A good test name should convey what is being tested, under what conditions, and what the expected outcome is. Below are some best practices and examples for naming unit tests.

1. Use the "MethodUnderTest_StateUnderTest_ExpectedBehavior" Convention

This convention ensures that test names clearly express:

- **Method Under Test**: The method or functionality being tested.
- **State Under Test**: The scenario or input being tested.
- **Expected Behavior**: The outcome that is expected.

Example:

```
public void MapToDTO_When_UserModelIsValid_Should_ReturnUserDTO()
```

- **Method Under Test**: `MapToDTO`
- **State Under Test**: `When_UserModelIsValid`
- **Expected Behavior**: `Should_ReturnUserDTO`

2. Include "When" and "Should" in Test Names

- **"When"**: Describes the specific scenario or condition under which the test is executed.
- **"Should"**: Indicates the expected behavior or outcome.

Example:

```
public void GetUserById_When_IdIsValid_Should_ReturnUser()
public void GetUserById_When_IdIsInvalid_Should_ThrowNotFoundException()
```

These test names indicate that the behavior of `GetUserById` is being checked under two different conditions.

3. Be Descriptive, But Keep It Concise

Test names should be descriptive enough to give a clear idea of what's being tested, without being too long.

Example:

```
public void Login_When_ValidCredentialsAreProvided_Should_ReturnSuccess()
```

4. Avoid Technical Jargon or Internal Code Names

Focus on describing the behavior from a user or system perspective rather than internal implementation details.

Instead of:

```
public void CheckUserObjectId_WithValidId_ReturnsCorrectDTO()
```

Prefer:

```
public void GetUserById_When_IdIsValid_Should_ReturnUserDTO()
```

5. Single Responsibility

Each test should focus on one behavior or condition, making it easier to identify what went wrong if a test fails.

Example:

```
public void
CalculateTotalPrice_When_DiscountApplied_Should_ReturnDiscountedPrice()
public void CalculateTotalPrice_When_NoDiscountApplied_Should_ReturnFullPrice()
```

6. Test Name Structure for Different Types of Tests

Integration Tests:

```
public void SaveUser_When_DatabaseIsAvailable_Should_SaveSuccessfully()
public void SaveUser_When_DatabaseIsUnavailable_Should_ThrowDatabaseException()
```

End-to-End Tests:

```
public void RegisterUser_When_ValidInputIsGiven_Should_RegisterSuccessfully()
```

7. Parameterization

For methods with multiple test cases, parameterized tests help keep names concise.

Example using xUnit:

```
[Theory]
[InlineData(5, true)]
[InlineData(0, false)]
public void
IsEligibleForDiscount_When_ValidPurchaseAmount_Should_ReturnExpectedResult(int
purchaseAmount, bool expectedResult)
```

Summary of a Good Test Name:

- **[MethodUnderTest]*[Condition or Input]*[ExpectedOutcome]**
  - This structure communicates **what** is being tested, **when** it is being tested, and **what** the expected outcome is.