Classical vs Mockist Testing

```
public class Account
{
    public string Id { get; }
    public decimal Balance { get; private set; }

    public Account(string id, decimal initialBalance)
    {
        Id = id;
        Balance = initialBalance;
    }

    public virtual void Credit(decimal amount)
    {
        if (amount <= 0) throw new ArgumentException("Invalid credit");
        Balance += amount;
    }

    public virtual void Debit(decimal amount)
    {
        if (amount <= 0 || amount > Balance)
            throw new InvalidOperationException("Invalid debit");
        Balance -= amount;
    }
}
```

```
public class MoneyTransferService
{
    public void Transfer(Account fromAcc, Account
toAcc, decimal amount)
    {
        fromAcc.Debit(amount);
        toAcc.Credit(amount);
    }
}
```

```
public class ClassicalMoneyTransferTests
{
    [Fact]
    public void Transfer_ShouldUpdateBalances()
    {
        var fromAcc = new Account("A1", 1000);
        var toAcc = new Account("A2", 500);
        var service = new MoneyTransferService();

        service.Transfer(fromAcc, toAcc, 200);

        fromAcc.Balance.Should().Be(800);
        toAcc.Balance.Should().Be(700);
    }
}
```

Classical Testing

```
public class MockistMoneyTransferTests
{
    [Fact]
    public void Transfer_ShouldCallDebitAndCredit()
    {
        var mockFromAcc = new Mock<Account>("A1", 1000);
        var mockToAcc = new Mock<Account>("A2", 500);

        var service = new MoneyTransferService();

        service.Transfer(mockFromAcc.Object, mockToAcc.Object, 200);

        mockFromAcc.Verify(a => a.Debit(200), Times.Once);
        mockToAcc.Verify(a => a.Credit(200), Times.Once);
    }
}
```

Mockist Testing

Classical Testing Philosophy

Test the outcome of operations

Use real objects whenever practical

Mock only external dependencies (databases, web services, file systems, queues)

Verify the final state rather than how you got there

Mockist Testing Philosophy

Test the interactions between objects

Mock all dependencies to isolate the unit under test

Focus on how objects collaborate

Verify that methods are called with correct parameters

Classical Testing - Pros vs Cons

Refactoring-friendly: Tests don't break when you change implementation details

Realistic: Tests exercise real code paths

Unmanageable: When external dependencies are involved

Cascading failures: One broken class breaks many tests

Mockist Testing - Pros vs Cons

Smooth Execution: When external dependencies are involved

Precise isolation: Failures point to exact component

Coupling: Tests become coupled to implementation details

Mock complexity: Heavy setup and verification logic

Few Other Points

In a codebase, you generally can't have only Classical testing because you will have external dependencies which will require mockist testing.

However, in a codebase you can have only Mockist testing if you stick strictly with Mockist testing.

In Classical testing, some classes / utils are tested transitively through higher level classes instead of being tested explicitly unless they have complex logic.

When To Use What

Use Classical testing to verify system behavior and Mockist testing to verify interactions with external collaborators — mixing both thoughtfully in your tests.

Use Classical Testing For	Use Mockist Testing For
Domain Logic & Business rules	When Dependencies are external - databases, web services, file systems, queues
Mathematical / Algorithmic Code	
Value Objects & Data Structures	
When Dependencies are not external	