# Unit

Unit тест – блок кода (обычно метод), который вызывает тестируемый блок кода и проверяет его правильность работы. Если результат юнит-теста не совпадает с ожидаемым результатом, тест считается не пройденным.

Модульное тестирование (Unit testing) – тестирование каждой атомарной функции приложения отдельно, с использованием объектов искусственно смоделированной среды.

Интеграционное тестирование – вид тестирования, при котором на соответствие требований проверяется интеграция модулей, их взаимодействие между собой, а также интеграция подсистем в одну общую систему.

Системное тестирование – это тестирование программного обеспечения выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям, как функциональным, так и не функциональным

Верификация (verification) - это процесс оценки системы или её компонентов с целью определения того, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа.

Валидация (validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

Делаем ли мы продукт правильно

**Делаем ли мы правильный продукт**

Реализована ли вся функциональность

**Правильно ли реализована функциональность**

Производиться разработчиками Производиться тестировщиками

**Инспектирование кода, сравнение требований**

Выполнение программы

**Объективная оценка реализованных функций**

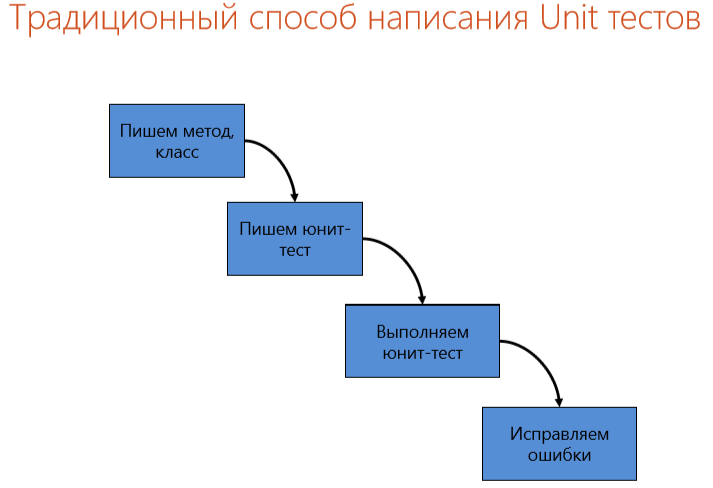
Субъективная оценка приложения

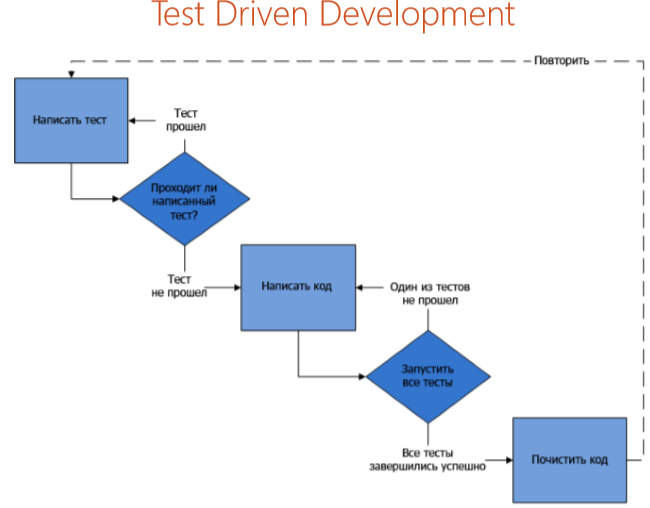
Unit тест должен быть:

• Автоматизированным и повторяемым; • Простым в реализации; • После написания он должен остаться для последующего использования; • Кто угодно в команде должен иметь возможность запустить Unit тест; • Должен запускаться одним нажатием кнопки; • Должен выполнятся быстро.

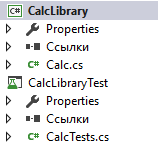
Проекты: <PROJECT\_NAME>.Core <PROJECT\_NAME>.BI <PROJECT\_NAME>.Web

Проекты с тестами: <PROJECT\_NAME>.Core.Tests <PROJECT\_NAME>.BI.Tests <PROJECT\_NAME>.Web.Tests





## Simple test

}

// Класс, которй необходимо тестировать

public class Calc

{

public static double Sum(double x, double y)

{

return x + y;

}

namespace CalcLibraryTest

{

[TestClass]

public class CalcTests

{

[TestMethod]

public void Sum\_10plus20\_30returned()

{

// arrange

double x = 10;

double y = 20;

double expected = 30;

// act

double actual = Calc.Sum(x, y);

// assert

Assert.AreEqual(expected, actual);

}

}

}

## PasswordStrengthChecker

public static class PasswordStrengthCheker

{

// Возвращает значение определяющее сложность пароля пользователя.

public static int GetPasswordStrength(string password)

{

if (string.IsNullOrEmpty(password))

{

return 0;

}

int result = 0;

// +1 балл за длину.

if (Math.Max(password.Length, 7) > 7)

{

result++;

}

//+1 балл за наличие символа в нижнем регистре

if (Regex.Match(password, "[a-z]").Success)

{

result++;

}

//+1 балл за наличие символа в верхнем регистре

if (Regex.Match(password, "[A-Z]").Success)

{

result++;

}

// +1 балл за наличие числа.

if (Regex.Match(password, "[0-9]").Success)

{

result++;

}

// +1 балл за наличие специального символа.

if (Regex.Match(password,

"[\\!\\@\\#\\$\\%\\^\\&\\\*\\(\\)\\{\\}\\[\\]\\:\\'\\;\\\"\\/\\?\\.\\>\\,\\<\\~\\`\\-\\\\\_\\=\\+\\|]").Success)

{

result++;

}

return result;

}

}

namespace UserRegistration.Tests

{

[TestClass()]

public class PasswordStrengthChekerTests

{

[TestMethod]

public void GetPasswordStrength\_AllCahrs\_5Points()

{

// arrange

string password = "P2ssw0rd#";

int expected = 5;

// act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GetPasswordStrength\_UpperCase\_3Points()

{

// Arrange

string password = "Password";

int expected = 3; // верхний регистр 1, за длину строки 1, за нижний регистр 1

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GetPasswordStrength\_ConteinsNumber\_0\_4Points()

{

// Arrange

string password = "Passw0rd";

// верхний регистр 1, за длину строки 1, за нижний регистр 1

// число 1

int expected = 4;

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GetPasswordStrength\_ConteinsNumber\_1\_4Points()

{

// Arrange

string password = "Passw1rd";

// верхний регистр 1, за длину строки 1, за нижний регистр 1

// число 1

int expected = 4;

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

// Tests for special chars

[TestMethod]

public void GetPasswordStrength\_ContainsSpecialChar\_at\_5Points()

{

// Arrange

string password = "Passw0rd@";

// верхний регистр 1, за длину строки 1, за нижний регистр 1

// число 1, специальный символ 1

int expected = 5;

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GetPasswordStrength\_ContainsSpecialChar\_Hash\_5Points()

{

// Arrange

string password = "Passw0rd#";

// верхний регистр 1, за длину строки 1, за нижний регистр 1

// число 1, специальный символ 1

int expected = 5;

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GetPasswordStrength\_ContainsSpecialChar\_Excl\_5Points()

{

// Arrange

string password = "Passw0rd!";

// верхний регистр 1, за длину строки 1, за нижний регистр 1

// число 1, специальный символ 1

int expected = 5;

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GetPasswordStrength\_ContainsSpecialChar\_Doll\_5Points()

{

// Arrange

string password = "Passw0rd$";

// верхний регистр 1, за длину строки 1, за нижний регистр 1

// число 1, специальный символ 1

int expected = 5;

// Act

int actual = PasswordStrengthCheker.GetPasswordStrength(password);

// Assert

Assert.AreEqual(expected, actual);

}

}

}

## ShoppingCart Initializers

namespace ShoppingCartLibrary

{

[Serializable]

public class ShoppingCart : IDisposable

{

public List<Item> Items = new List<Item>();

public int Count

{

get { return Items.Count; }

}

public void Add(Item item)

{

Items.Add(item);

}

public void Dispose()

{

// cleanup code

}

public void Remove(int index)

{

Items.RemoveAt(index);

}

}

[Serializable]

public class Item

{

public string Name { get; set; }

public int Quantity { get; set; }

}

}

namespace ShopingCartLibrary.Tests

{

[TestClass]

public class TestInitAndCleanUp

{

private ShoppingCart cart;

private Item item;

// Запускается перед стартом каждого тестирующего метода

[TestInitialize]

public void TestInitialize()

{

Debug.WriteLine("Test Initialize");

item = new Item();

item.Name = "Unit Test Video Lessons";

item.Quantity = 10;

cart = new ShoppingCart();

cart.Add(item);

}

// Запускается после завершения каждого тестирующего метода

[TestCleanup]

public void TestCleanUp()

{

Debug.WriteLine("Test CleanUp");

cart.Dispose();

}

[TestMethod]

public void ShopingCart\_CheckOut\_ContainsItem()

{

CollectionAssert.Contains(cart.Items, item);

}

[TestMethod]

public void ShopingCart\_RemoveItem\_Empty()

{

int expected = 0;

cart.Remove(0);

Assert.AreEqual(expected, cart.Count);

}

}

}

namespace ShopingCartLibrary.Tests

{

[TestClass]

public class ClassInitAndCleanup

{

private static ShoppingCart cart;

// Запускается перед стартом первого тестирующего метода (один раз для класса)

// Метод должен быть открытым, статическим и принимать параметр типа TestContext

[ClassInitialize]

public static void ClassInitialize(TestContext context)

{

Debug.WriteLine("Class Initialize");

Item item = new Item();

item.Name = "Unit Test Video Lessons";

item.Quantity = 10;

cart = new ShoppingCart();

cart.Add(item);

}

// Запускается после завершения последнего тестирующего метода (один раз для класса)

// Метод должен быть открытым статическим и возвращать void

[ClassCleanup]

public static void TestCleanUp()

{

Debug.WriteLine("Class CleanUp");

cart.Dispose();

}

[TestMethod]

public void ShopingCart\_AddToCart()

{

int expected = cart.Items.Count + 1;

cart.Add(new Item() { Name = "Test", Quantity = 1 });

Assert.AreEqual(expected, cart.Count);

}

[TestMethod]

public void ShopingCart\_RemoveFromCart()

{

int expected = cart.Items.Count - 1;

cart.Remove(0);

Assert.AreEqual(expected, cart.Count);

}

}

}

## Assert methods

[TestClass]

public class MyClassTests

{

[TestMethod]

public void IsSqrtTest()

{

// arrange

const double input = 4;

const double expected = 2;

// act

double actual = MyClass.GetSqrt(input);

// assert

Assert.AreEqual(expected, actual, "Sqrt of {0} should have been {1}!", input, expected);

}

[TestMethod]

public void DeltaTest()

{

const double expected = 3.1;

const double delta = 0.07;

// 3.1622776601683795

// 0.062..

double actual = MyClass.GetSqrt(10);

// Проверка значений на равенство с учетом погрешности delta

Assert.AreEqual(expected, actual, delta, "fail message!!!");

}

[TestMethod]

public void StringAreEqualTest()

{

// arrange

const string input = "HELLO";

const string expected = "hello";

// act and assert

// третий параметр - игнорирование регистра

Assert.AreEqual(expected, input, true);

}

[TestMethod]

public void StringSameTest()

{

string a = "Hello";

string b = "Hello";

// проверка равенства ссылок

Assert.AreSame(a, b);

}

[TestMethod]

public void IntegersSameTest()

{

int i = 10;

int j = 10;

// проверка равенства ссылок

Assert.AreSame(i, j);

}

}

## Collection assert methods

[TestClass]

public class CollectionAssertMethods

{

static List<string> employees;

[ClassInitialize]

public static void InitializeCurrentTest(TestContext testContext)

{

employees = new List<string>();

employees.Add("Ivan");

employees.Add("Sergey");

employees.Add("Anton");

employees.Add("Roman");

}

[TestMethod]

public void AllItemsAreNotNull()

{

// Проверка, что все элементы коллекции созданы

CollectionAssert.AllItemsAreNotNull(employees, "Not null failed");

}

[TestMethod]

public void AllItemsAreUnique()

{

// Проверка значений коллекции на уникальность

CollectionAssert.AllItemsAreUnique(employees, "Uniqueness failed");

}

[TestMethod]

public void AreEqual()

{

List<string> employeesTest = new List<string>();

employeesTest.Add("Sergey");

employeesTest.Add("Ivan");

employeesTest.Add("Anton");

employeesTest.Add("Roman");

// Проверка каждого элемента на равенство, в данном примере первый элемент из коллекции emploees

// не совпадает с первым элементом из коллекции emploeesTest. Тест не пройдет.

CollectionAssert.AreEqual(employees, employeesTest);

}

[TestMethod]

public void AreEquivalent()

{

List<string> employeesTest = new List<string>();

employeesTest.Add("Sergey");

employeesTest.Add("Ivan");

employeesTest.Add("Anton");

employeesTest.Add("Roman");

// Проверка коллекций на наличие одинаковых элементов, порядок которых не важен.

CollectionAssert.AreEquivalent(employees, employeesTest);

}

[TestMethod]

public void employees\_Subset()

{

List<string> employees\_Subset = new List<string>();

employees\_Subset.Add(employees[2]);

employees\_Subset.Add("Alexander"); // Если убрать комментарий - тест не пройдет

// Проверка того, что элементы employees\_Subset входят в коллекцию employees.

CollectionAssert.IsSubsetOf(employees\_Subset, employees, "failed!");

}

}

## String assert methods

[TestClass]

public class StringAssertMethods

{

[TestMethod]

public void StringContainsTest()

{

// Проверка на вхождение в строку подстроки

StringAssert.Contains("Assert samples", "sam");

}

[TestMethod]

public void StringMatchesTest()

{

// Проверка с использованием регулярного выражения

StringAssert.Matches("123", new Regex(@"\d{3}"));

}

[TestMethod]

public void StringStartsWithTest()

{

// Проверка того, что строка начинается с определенного слова

StringAssert.StartsWith("Hello world", "Hello");

}

[TestMethod]

public void StringEndsWithTest()

{

// Проверка того, что строка заканчивается определенным словом

StringAssert.EndsWith("Hello world", "world");

}

}

## Expecting Exception

namespace AssertSamples.Tests

{

[TestClass]

public class ExpectingExceptions

{

// ExpectedException - тест будет успешным если в процессе выполнения будет выброшено исключение с типом ArgumentNullException

[ExpectedException(typeof(ArgumentNullException), "Exception was not thrown")]

[TestMethod]

public void MyClass\_SayHello\_Exception()

{

MyClass instance = new MyClass();

instance.SayHello(null);

}

[TestMethod]

public void MyClass\_SayHello\_ReturnDmitriy()

{

// arrange

MyClass instance = new MyClass();

string expected = "Hello Dmitriy";

// act

string actual = instance.SayHello("Dmitriy");

// assert

Assert.AreEqual(expected, actual);

}

}

}

## DataDrivenTest

namespace UserManagerLibrary.Tests

{

// Data Driven Test - тест, который позволяет использовать для запуска данные из внешнего источника.

// Тест запускается по одному разу на каждую запись из источника данных. Это позволяет проверить разные входные данных

// используя один метод.

[TestClass]

public class UserManagerTests

{

public TestContext TestContext { get; set; }

private UserManager manager = new UserManager();

// DataSource - определение источника данных.

// 1 параметр - имя провайдера

// 2 параметр - строка подключения или путь к файлу

// 3 параметр - имя таблицы или элемента в XML

// 4 параметр - как происходит доступ к записям из источника данных

[DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML",

"testData.xml",

"User",

DataAccessMethod.Sequential)]

[TestMethod]

public void UserManager\_Add\_FromXML()

{

string userId = Convert.ToString(TestContext.DataRow["userid"]);

string telephone = Convert.ToString(TestContext.DataRow["telephone"]);

string email = Convert.ToString(TestContext.DataRow["email"]);

bool result = manager.Add(userId, telephone, email);

Assert.IsTrue(result, "Пользователь не может быть создан");

}

}

}

<?xml version="1.0" encoding="utf-8"?>

<UserDetails>

<User userid="Dmitriy" telephone="+38 000 123 23 42" email="dmitriy@itvdn.com"/>

<User userid="Ivan" telephone="+38 000 123 23 42" email="ivan@itvdn.com"/>

<User userid="Alex" telephone="+38 000 123 23 42" email="alex@itvdn.com"/>

<User userid="Oleg" telephone="+38 000 123 23 42" email="oleg@itvdn.com"/>

<User userid="Petro" telephone="+38 000 123 23 42" email="p@cbsarea.com"/>

<User userid="Roman" telephone="+38 000 123 23 42" email="roman@cbsarea.com"/>

<User userid="Nikolay" telephone="+38 000 123 23 42" email="nikolay@test.com"/>

</UserDetails>

## TestContext

[TestClass]

public class MyTestSample

{

// TestContext используется для хранения информации о текущем юнит тесте

// При тестировании веб сервисов хранит URL

// При тестировании ASP.NET приложений - предоставляет доступ к ASP странице

// При использовании Data Driven тестов предоставляет доступ к источнику данных

public TestContext TestContext { get; set; }

[TestMethod]

public void TestMethod1()

{

TestContext.WriteLine("TestRunDirectory {0}", TestContext.TestRunDirectory);

TestContext.WriteLine("TestName {0}", TestContext.TestName);

TestContext.WriteLine("CurrentTestOutcome {0}", TestContext.CurrentTestOutcome);

}

[TestCleanup]

public void CleanUp()

{

TestContext.WriteLine("TestName (CleanUp) {0}", TestContext.TestName);

TestContext.WriteLine("CurrentTestOutcome (Cleanup) {0}", TestContext.CurrentTestOutcome);

}

}

# Stub

Тесты состояния – проверяют состояние объекта после вызова метода (проверка того, что вызванный метод отработал правильно).

Тесты взаимодействия– проверяют правильность взаимодействия тестируемого объекта с окружением (внешними зависимостями).

Dummy object – передается в тестируемый класс в качестве параметра. Не имеет поведения и с ним ничего не происходит.

Test stub – Используется для получения данных от внешних зависимостей, подменяя ее.

Test spy – используется для тестов взаимодействия. Основная задача – запись данных, которые поступают от тестируемого объекта.

Mock object – похож на test spy но не записывает данные от тестируемого объекта, а самостоятельно их проверяет.

Fake object - замена тяжеловесной внешней зависимости легковесной реализацией. Эмулятор БД в памяти, фальшивый веб сервис.

Dependency Injection – паттерн описывающий технику внедрения внешней зависимости в программный компонент.

Способы внедрения зависимостей: • через конструктор. • через свойство. • через интерфейсы.

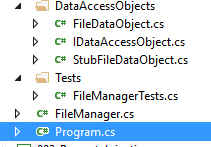
Способы создания экземпляров зависимостей: Через контейнеры (Unity, Ninject, Spring .NET)

## ConstructorInjection

Dependency Injection – паттерн описывающий технику внедрения внешней зависимости в программный компонент.

Способы внедрения зависимостей: • через конструктор. • через свойство. • через интерфейсы.

Способы создания экземпляров зависимостей: Через контейнеры (Unity, Ninject, Spring .NET)



namespace \_002\_ConstructorInjection

{

// В данном примере мы уменьшаем связанность выделяя интерфейс IDataAccessObject для объектов работающих с файловой системой.

// В данном примере происходит внедрение зависимости через конструктор.

public class FileManager

{

private IDataAccessObject dataAccessObject;

public FileManager()

{

dataAccessObject = new FileDataObject();

}

public FileManager(IDataAccessObject dataAccessObject)

{

this.dataAccessObject = dataAccessObject;

}

public bool FindLogFile(string fileName)

{

List<string> files = dataAccessObject.GetFiles();

foreach (var file in files)

{

if (file.Contains(fileName))

{

return true;

}

}

return false;

}

}

}

namespace \_002\_ConstructorInjection

{

class Program

{

static void Main()

{

string fileName = "file2.log";

FileManager mgr = new FileManager();

if (mgr.FindLogFile(fileName))

{

Console.WriteLine("File {0} is found!", fileName);

}

else

{

Console.WriteLine("File {0} is not found!", fileName);

}

Console.ReadKey();

}

}

}

namespace \_002\_ConstructorInjection

{

public interface IDataAccessObject

{

List<string> GetFiles();

}

}

namespace \_002\_ConstructorInjection

{

public class FileDataObject : IDataAccessObject

{

public List<string> GetFiles()

{

string path = Directory.GetCurrentDirectory();

List<string> list = new List<string>();

DirectoryInfo d = new DirectoryInfo(path);

FileInfo[] files = d.GetFiles();

foreach (var file in files)

{

list.Add(file.Name);

}

return list;

}

}

}

namespace \_002\_ConstructorInjection

{

public class StubFileDataObject : IDataAccessObject

{

public List<string> GetFiles()

{

List<string> list = new List<string>();

list.Add("file1.txt");

list.Add("file2.log");

list.Add("file3.exe");

list.Add("main.log");

return list;

}

}

}

namespace \_002\_ConstructorInjection.Tests

{

[TestClass]

public class FileManagerTests

{

[TestMethod]

public void FileManager\_FindFileLogByName\_2()

{

IDataAccessObject accessObject = new StubFileDataObject();

FileManager fileManager = new FileManager(accessObject); // Dependency Injection

string fileName = "main.log";

bool exists = fileManager.FindLogFile(fileName);

Assert.IsTrue(exists, "File {0} was not found.", fileName);

}

}

}

## Interface injection

class FileManager

{

// Внедрение зависимости через интерфейс

public bool FindLogFile(string fileName, IDataAccessObject dataAccessObject)

{

if (dataAccessObject == null)

{

throw new ArgumentNullException("dataAccessObject", "Parameter dataAcessObject cannot be null");

}

List<string> files = dataAccessObject.GetFiles();

foreach (var file in files)

{

if (file.Contains(fileName))

{

return true;

}

}

return false;

}

}

}

class Program

{

static void Main()

{

string fileName = "file2.log";

FileManager mgr = new FileManager();

// Внедерние зависимости.

//mgr.FindLogFile(fileName, new StubFileDataObject());

if (mgr.FindLogFile(fileName, new FileDataObject()))

{

Console.WriteLine("File {0} is found!", fileName);

}

else

{

Console.WriteLine("File {0} is not found!", fileName);

}

Console.ReadKey();

}

}

## Factory method

namespace \_009\_LocalAbstractMethod

{

public class FileManager

{

public FileManager()

{

}

// Использование фабричного метода для тестирования.

// В юнит тесте данный метод можно переопределить для того чтобы он возвращал заглушку.

protected virtual IDataAccessObject CreateDataAccessObject()

{

return new FileDataObject();

}

public bool FindLogFile(string fileName)

{

var dataAccessObject = CreateDataAccessObject();

List<string> files = dataAccessObject.GetFiles();

foreach (var file in files)

{

if (file.Contains(fileName))

{

return true;

}

}

return false;

}

}

}

namespace \_008\_LocalFactoryMethod.Tests

{

class FileManagerUnderTest : FileManager

{

protected override IDataAccessObject CreateDataAccessObject()

{

return new StubFileDataObject();

}

}

[TestClass]

public class FileManagerTests

{

[TestMethod]

public void FileManager\_FindFileLogByName\_5()

{

FileManager fileManager = new FileManagerUnderTest();

string fileName = "main.log";

bool exists = fileManager.FindLogFile(fileName);

Assert.IsTrue(exists, "File {0} was not found.", fileName);

}

}

}

## ExctractAndOverride

namespace \_006\_ExtractAndOverride

{

class FileManager

{

public FileManager()

{

}

protected virtual List<string> GetLogFiles()

{

return new FileDataObject().GetFiles();

}

public bool FindLogFile(string fileName)

{

List<string> files = GetLogFiles();

foreach (var file in files)

{

if (file.Contains(fileName))

{

return true;

}

}

return false;

}

}

}

namespace \_006\_ExtractAndOverride.Tests

{

class FileManagerUnderTest : FileManager

{

protected override List<string> GetLogFiles()

{

return new List<string> { "file1.txt", "file2.log", "file3.exe", "main.log" };

}

}

[TestClass]

public class FileManagerTests

{

[TestMethod]

public void FileManager\_FindFileLogByName\_6()

{

FileManager fileManager = new FileManagerUnderTest();

string fileName = "main.log";

bool exists = fileManager.FindLogFile(fileName);

Assert.IsTrue(exists, "File {0} was not found.", fileName);

}

}

}

## FactoryInjection

namespace \_007\_FactoryInjection

{

public class FileManager

{

private IDataAccessObject dataAccessObject;

public FileManager()

{

dataAccessObject = DataAccessFactory.Create();

}

public bool FindLogFile(string fileName)

{

List<string> files = dataAccessObject.GetFiles();

foreach (var file in files)

{

if (file.Contains(fileName))

{

return true;

}

}

return false;

}

}

}

namespace \_007\_FactoryInjection

{

public class DataAccessFactory

{

private static IDataAccessObject dataAcessObject;

internal static IDataAccessObject Create()

{

if (dataAcessObject != null)

{

return dataAcessObject;

}

return new FileDataObject();

}

[Conditional("DEBUG")]

public static void SetDataAccessObject(IDataAccessObject customDataObject)

{

dataAcessObject = customDataObject;

}

}

}

namespace \_007\_FactoryInjection.Tests

{

[TestClass]

public class FileManagerTests

{

[TestMethod]

public void FileManager\_FindFileLogByName\_7()

{

DataAccessFactory.SetDataAccessObject(new StubFileDataObject());

FileManager fileManager = new FileManager();

string fileName = "main.log";

bool exists = fileManager.FindLogFile(fileName);

Assert.IsTrue(exists, "File {0} was not found.", fileName);

}

}

}

## InternalVisibleTo

/ InternalsVisibleTo - указывает, что типы, которые обычно доступны только в текущей сборке будут доступны также в сборке,

// которая определена в параметрах.

[assembly: InternalsVisibleTo("FileManagerLibrary.Tests")]

namespace FileManagerLibrary

{

class FileManager

{

private IDataAccessObject dataAccessObject;

public FileManager()

{

dataAccessObject = new FileDataObject();

}

public FileManager(IDataAccessObject dataAccessObject)

{

this.dataAccessObject = dataAccessObject;

}

public bool FindLogFile(string fileName)

{

List<string> files = dataAccessObject.GetFiles();

foreach (var file in files)

{

if (file.Contains(fileName))

{

return true;

}

}

return false;

}

}

}

namespace FileManagerLibrary.Tests

{

[TestClass]

public class FileManagerTests

{

[TestMethod]

public void FileManager\_FindFileLogByName()

{

IDataAccessObject accessObject = new StubFileDataObject();

// Для того, чтобы internal классы и методы были доступны в тестовой сборке, в проекте FileManagerLibrary

// добавлен атрибут [assembly: InternalsVisibleTo("FileManagerLibrary.Tests")]

FileManager fileManager = new FileManager(accessObject);

string fileName = "main.log";

bool exists = fileManager.FindLogFile(fileName);

Assert.IsTrue(exists, "File {0} was not found.", fileName);

}

}

}

# Mock

Mock-объект – это управляемая замена существующих зависимостей в системе. Mock-объекты заменяют реальные объекты системы и позволяют проверить вызовы своих членов тестируемым классом. Mock-объекты отличаются от Stub объектов тем, что они могут быть причиной неуспешного завершения юнит теста.

Isolation Framework – это инструмент автоматизирующий процесс создания заглушек для классов, интерфейсов методов, используемых в тестируемом методе.

• Rhino Mocks • Moq • Typemock Isolator • Microsoft Fakes • NMock2

## MockMoq

namespace LoggingLibrary

{

public class Logger

{

private ISensitiveDataScruber \_sensitiveDataScruber;

private IMessageBodyGenerator \_messageBodyGenerator;

private IMessageHeaderGenerator \_messageHeaderGenerator;

private ILoggingConfiguration \_loggingConfiguration;

public Logger(ILoggingConfiguration loggingConfiguration, IMessageHeaderGenerator messageHeaderGenerator,

IMessageBodyGenerator messgaeBodyGenerator, ISensitiveDataScruber sensitiveDataScruber)

{

\_sensitiveDataScruber = sensitiveDataScruber;

\_messageBodyGenerator = messgaeBodyGenerator;

\_messageHeaderGenerator = messageHeaderGenerator;

\_loggingConfiguration = loggingConfiguration;

}

public void CreateEntry(string message, LogLevel level)

{

\_messageHeaderGenerator.CreateHeader(level);

if (\_loggingConfiguration.LogStackFor(level))

{

Console.Write("Stack: ");

}

string clearMessage = \_sensitiveDataScruber.ClearSensitive(message);

\_messageBodyGenerator.CreateBody(clearMessage);

}

}

}

namespace LoggingLibrary

{

public interface ISensitiveDataScruber

{

string ClearSensitive(string message);

}

}

namespace LoggingLibrary

{

public interface IMessageHeaderGenerator

{

void CreateHeader(LogLevel level);

}

}

namespace LoggingLibrary

{

public interface IMessageBodyGenerator

{

void CreateBody(string message);

}

}

namespace LoggingLibrary

{

public interface ILoggingConfiguration

{

bool LogStackFor(LogLevel level);

}

}

namespace LoggingLibrary

{

public class SensitiveDataScruber : ISensitiveDataScruber

{

// Удаление лишних значений из строки

public string ClearSensitive(string message)

{

Debug.WriteLine("SensitiveDataScruber.ClearSensitive");

return message;

}

}

}

namespace LoggingLibrary

{

public class MessageHeaderGenerator : IMessageHeaderGenerator

{

// Создание заголовка сообщения

public void CreateHeader(LogLevel level)

{

Debug.WriteLine("MessageHeaderGenerator.CreateHeader");

}

}

}

namespace LoggingLibrary

{

public class LoggingConfiguration : ILoggingConfiguration

{

// Определение необходимости логирования стэка вызовов

public bool LogStackFor(LogLevel level)

{

if (level == LogLevel.Error)

{

return true;

}

else

{

return false;

}

}

}

}

namespace LoggingLibrary

{

public class MessageBodyGenerator : IMessageBodyGenerator

{

// Создание тела сообщения

public void CreateBody(string message)

{

Debug.Write("MessageBodyGenerator.CreateBody");

}

}

}

using Moq;

namespace LogginLibrary.Tests

{

// moq - mocking framework для .NET Используется для юнит тестирования для изоляции класса, который тестируется

// от его зависимостей и необходим, для того, чтобы убедиться, что на зависимостях были вызваны нужные методы.

// https://en.wikipedia.org/wiki/Mock\_object

// Для того чтобы установить библиотеку moq воспользуйтесь командой Install-Package moq

[TestClass]

public class LoggignTests

{

Logger \_logger;

Mock<ILoggingConfiguration> \_mockLoggingConfig;

Mock<IMessageBodyGenerator> \_mockBodyGenerator;

Mock<IMessageHeaderGenerator> \_mockHeaderGenerator;

Mock<ISensitiveDataScruber> \_mockSensitiveDataScruber;

[TestInitialize]

public void Initialize()

{

// создание mock объектов

\_mockLoggingConfig = new Mock<ILoggingConfiguration>();

\_mockBodyGenerator = new Mock<IMessageBodyGenerator>();

\_mockHeaderGenerator = new Mock<IMessageHeaderGenerator>();

\_mockSensitiveDataScruber = new Mock<ISensitiveDataScruber>();

// \_mockLoggingConfig.Object - объект заглушка с интерфейсом зависимости

\_logger = new Logger(\_mockLoggingConfig.Object,

\_mockHeaderGenerator.Object,

\_mockBodyGenerator.Object,

\_mockSensitiveDataScruber.Object);

}

[TestMethod]

public void Logger\_CreateEntry\_SensitiveDataShouldBeScrubed()

{

// arrange

// Setup - устанавливаем параметры, определяющие корректность работы тестируемого кода.

// .Setup(x => x.ClearSensitive(It.IsAny<string>())) - ожидаем вызов метода ClearSensitive с любым строковым значением в качестве параметра.

\_mockSensitiveDataScruber.Setup(x => x.ClearSensitive(It.IsAny<string>()));

// act

\_logger.CreateEntry("Test log", LogLevel.Error);

// assert

// VerifyAll - проверяем, правильно ли взаимодействовал тестируемый метод с mock объектом

\_mockSensitiveDataScruber.VerifyAll();

}

[TestMethod]

public void Logger\_CreateEntry\_HeaderCreated()

{

\_mockHeaderGenerator.Setup(x => x.CreateHeader(It.IsAny<LogLevel>()));

\_logger.CreateEntry("Test log", LogLevel.Error);

\_mockHeaderGenerator.VerifyAll();

}

[TestMethod]

public void Logger\_CreateEntry\_BodyGeneratedCreated()

{

\_mockBodyGenerator.Setup(x => x.CreateBody(It.IsAny<string>()));

\_logger.CreateEntry("Test log", LogLevel.Error);

\_mockBodyGenerator.VerifyAll();

}

}

}

## Mock

namespace CustomerServiceLibrary

{

public interface ICustomerRepository

{

void Save(Customer customer);

}

public class Customer

{

public Customer(string fn, string ln)

{

}

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

}

public class CustomerDTO

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

public class CustomerService

{

private ICustomerRepository \_repositry;

public CustomerService(ICustomerRepository repository)

{

\_repositry = repository;

}

public void Create(IEnumerable<CustomerDTO> customers)

{

foreach (var customer in customers)

{

\_repositry.Save(new Customer(customer.FirstName, customer.LastName));

}

}

}

}

namespace UserServicesLibrary.Tests

{

[TestClass]

public class CustomerServiceTests

{

[TestMethod]

public void CreateMethod\_Save\_WasCalled()

{

// arrange

List<CustomerDTO> list = new List<CustomerDTO>() {

new CustomerDTO() { FirstName ="Ivan", LastName="Ivanov" },

new CustomerDTO() { FirstName ="Petr", LastName="Petrov" },

new CustomerDTO() { FirstName ="Fedor", LastName="Fedorov" }

};

Mock<ICustomerRepository> mock = new Mock<ICustomerRepository>();

ICustomerRepository repository = mock.Object;

CustomerService service = new CustomerService(repository);

// act

service.Create(list);

// assert

mock.Verify(x => x.Save(It.IsAny<Customer>()));

}

[TestMethod]

public void CreateMethod\_Save\_WassCalled\_ThreeTimes()

{

// arrange

List<CustomerDTO> list = new List<CustomerDTO>() {

new CustomerDTO() { FirstName ="Ivan", LastName="Ivanov" },

new CustomerDTO() { FirstName ="Petr", LastName="Petrov" },

new CustomerDTO() { FirstName ="Fedor", LastName="Fedorov" }

};

Mock<ICustomerRepository> mock = new Mock<ICustomerRepository>();

ICustomerRepository repository = mock.Object;

CustomerService service = new CustomerService(repository);

// act

service.Create(list);

// assert

mock.Verify(x => x.Save(It.IsAny<Customer>()), Times.Exactly(3));

}

}

}