Національний технічний університет України «КПІ»

Факультет інформатики та обчислювальної техніки

Кафедра Інформаційних систем та технологій

Лабораторна робота №2

з дисципліни « Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

на тему: « Модульне тестування. Ознайомлення з засобами та практиками модульного тестування»

Виконав:

студент гр. ІС-11

Воробйов Олексій

Викладач:

Бардін В.

2023 рік

**Мета**: навчитися створювати модульні тести для вихідного коду розроблювального програмного забезпечення.

**Завдання:**

1. Додати до проекту власної узагальненої колекції (застосувати виконану лабораторну роботу No1) проект модульних тестів, використовуючи певний фреймворк (Nunit, Xunit, тощо).

2. Розробити модульні тести для функціоналу колекції.

3. Дослідити ступінь покриття модульними тестами вихідного коду колекції, використовуючи, наприклад, засіб AxoCover.

**Варіант 10:**



**Посилання на код GitHub:**

<https://github.com/Limbo2332/Lab1-GenericCollections>  
Код тестів:

public class CollectionAddingTests

{

[TestCase(1)]

[TestCase('h')]

[TestCase("Hello")]

[TestCase(true)]

[TestCase(27.4F)]

public void Check\_CollectionAdd\_Item<T>(T value)

{

// Arrange

var collection = new MyLinkedList<T>();

// Act

collection.Add(value);

// Assert

Assert.Multiple(() =>

{

Assert.That(collection, Is.Not.Null);

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Count, Is.EqualTo(1));

Assert.That(collection.First!.Value, Is.EqualTo(value));

Assert.That(collection.Last!.Value, Is.EqualTo(value));

Assert.That(collection.First, Is.EqualTo(collection.Last));

});

}

[TestCase(1, 2)]

[TestCase('h', '1')]

[TestCase("Hello", "World")]

[TestCase(true, true)]

[TestCase(27.4F, 65.4F)]

public void Check\_CollectionAdd\_TwoItems<T>(T firstValue, T secondValue)

{

// Arrange

var collection = new MyLinkedList<T>();

// Act

collection.Add(firstValue);

collection.Add(secondValue);

// Assert

Assert.Multiple(() =>

{

Assert.That(collection, Is.Not.Null);

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.First!.Next, Is.Not.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Not.Null);

Assert.That(collection.Count, Is.EqualTo(2));

Assert.That(collection.First.Value, Is.EqualTo(firstValue));

Assert.That(collection.First!.Next!.Value, Is.EqualTo(secondValue));

Assert.That(collection.Last.Value, Is.EqualTo(secondValue));

Assert.That(collection.Last!.Previous!.Value, Is.EqualTo(firstValue));

Assert.That(collection.First, Is.Not.EqualTo(collection.Last));

});

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void Check\_CollectionAdd\_ThreeItems<T>(T firstValue, T secondValue, T thirdValue)

{

// Arrange

var collection = new MyLinkedList<T>();

// Act

collection.Add(firstValue);

collection.Add(secondValue);

collection.Add(thirdValue);

// Assert

Assert.Multiple(() =>

{

Assert.That(collection, Is.Not.Null);

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.First!.Next, Is.Not.Null);

Assert.That(collection.Find(1), Is.Not.Null);

Assert.That(collection.Find(1)!.Previous, Is.Not.Null);

Assert.That(collection.Find(1)!.Next, Is.Not.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Not.Null);

Assert.That(collection.Count, Is.EqualTo(3));

Assert.That(collection.First.Value, Is.EqualTo(firstValue));

Assert.That(collection.First!.Next!.Value, Is.EqualTo(secondValue));

Assert.That(collection.Find(1)!.Value, Is.EqualTo(secondValue));

Assert.That(collection.Find(1)!.Next, Is.EqualTo(collection.Last));

Assert.That(collection.Find(1)!.Next!.Value, Is.EqualTo(thirdValue));

Assert.That(collection.Find(1)!.Previous, Is.EqualTo(collection.First));

Assert.That(collection.Find(1)!.Previous!.Value, Is.EqualTo(firstValue));

Assert.That(collection.Last.Value, Is.EqualTo(thirdValue));

Assert.That(collection.Last!.Previous!.Value, Is.EqualTo(secondValue));

});

}

[TestCase(1, 2)]

[TestCase('h', '1')]

[TestCase("Hello", "World")]

[TestCase(true, true)]

[TestCase(27.4F, 65.4F)]

public void Check\_CollectionAdd\_TwoItemsToStart<T>(T firstValue, T secondValue)

{

// Arrange

var collection = new MyLinkedList<T>();

// Act

collection.AddFirst(firstValue);

collection.AddFirst(secondValue);

// Assert

Assert.Multiple(() =>

{

Assert.That(collection, Is.Not.Null);

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.First!.Next, Is.Not.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Not.Null);

Assert.That(collection.Count, Is.EqualTo(2));

Assert.That(collection.First.Value, Is.EqualTo(secondValue));

Assert.That(collection.First!.Next!.Value, Is.EqualTo(firstValue));

Assert.That(collection.Last.Value, Is.EqualTo(firstValue));

Assert.That(collection.Last!.Previous!.Value, Is.EqualTo(secondValue));

Assert.That(collection.First, Is.Not.EqualTo(collection.Last));

});

}

}

public class CollectionClearingTests

{

[Test]

public void Check\_ClearCollection\_WhenNoItems()

{

//Arrange

var collection = new MyLinkedList<int>();

//Act

collection.Clear();

//Assert

Assert.Multiple(() =>

{

Assert.That(collection, Has.Count.EqualTo(0));

Assert.That(collection.First, Is.Null);

Assert.That(collection.Last, Is.Null);

});

}

[TestCase(1)]

[TestCase('h')]

[TestCase("Hello")]

[TestCase(true)]

[TestCase(27.4F)]

public void Check\_ClearCollection\_WhenItems<T>(T value)

{

//Arrange

var collection = new MyLinkedList<T>() { value };

//Act

collection.Clear();

//Assert

Assert.Multiple(() =>

{

Assert.That(collection, Has.Count.EqualTo(0));

Assert.That(collection.First, Is.Null);

Assert.That(collection.Last, Is.Null);

});

}

}

public class CollectionCopyingTests

{

private static readonly object[][] WrongArrayAndIndexData =

{

new object[]

{

new int[3], 4

},

new object[]

{

new int[2], 3

},

new object[]

{

new int[5], -3

},

};

[Test]

public void CheckCopying\_WhenNullArray()

{

//Arrange

var collection = new MyLinkedList<int>();

//Act

int[]? array = null;

//Assert

Assert.Throws<ArgumentNullException>(() => collection.CopyTo(array, 1));

}

[TestCaseSource(nameof(WrongArrayAndIndexData))]

public void CheckCopying\_WhenWrongIndexOutOfArray(int[] array, int arrayIndex)

{

//Arrange

var collection = new MyLinkedList<int>();

//Assert

Assert.Throws<ArgumentOutOfRangeException>(() => collection.CopyTo(array, arrayIndex));

}

[Test]

public void CheckCopying\_WhenNoSpaceInArray()

{

//Arrange

var collection = new MyLinkedList<int>() { 1, 2, 3, 4 };

//Act

int[]? array = new int[2];

//Assert

Assert.Throws<ArgumentException>(() => collection.CopyTo(array, 1));

}

[Test]

public void CheckCopying\_WhenArray()

{

//Arrange

var collection = new MyLinkedList<int>() { 1, 2, 3, 4, 5 };

int[] arrayFromZeroIndex = new int[5];

int[] arrayFromFirstIndex = new int[6];

int[] arrayFromSecondIndex = new int[7];

int[] arrayFromThirdIndex = new int[8];

int[] arrayFromFourthIndex = new int[9];

//Act

collection.CopyTo(arrayFromZeroIndex, 0);

collection.CopyTo(arrayFromFirstIndex, 1);

collection.CopyTo(arrayFromSecondIndex, 2);

collection.CopyTo(arrayFromThirdIndex, 3);

collection.CopyTo(arrayFromFourthIndex, 4);

//Assert

Assert.Multiple(() =>

{

Assert.That(new int[5] { 1, 2, 3, 4, 5 }, Is.EqualTo(arrayFromZeroIndex));

Assert.That(new int[6] { 0, 1, 2, 3, 4, 5 }, Is.EqualTo(arrayFromFirstIndex));

Assert.That(new int[7] { 0, 0, 1, 2, 3, 4, 5 }, Is.EqualTo(arrayFromSecondIndex));

Assert.That(new int[8] { 0, 0, 0, 1, 2, 3, 4, 5 }, Is.EqualTo(arrayFromThirdIndex));

Assert.That(new int[9] { 0, 0, 0, 0, 1, 2, 3, 4, 5 }, Is.EqualTo(arrayFromFourthIndex));

});

}

}

public class CollectionCreationTests

{

private static readonly List<IEnumerable<object>> itemsCollection = new List<IEnumerable<object>>

{

new List<object> {1, 2, 3, 4},

new List<object> {true, false, false, true, false},

new List<object> {"", "Hello", "Boom", "World", "1", "3"},

new List<object> { new List<object> { } },

new List<object> {6.7M, 7.8M},

};

[Test]

public void Check\_CollectionCreationIfNoItems()

{

// Assert

Assert.Throws<ArgumentNullException>(() => new MyLinkedList<object>(Enumerable.Empty<object>()));

}

[Test]

public void Check\_CollectionCreationIfNullCollection()

{

// Assert

Assert.Throws<ArgumentNullException>(() => new MyLinkedList<object>(null));

}

[TestCaseSource(nameof(itemsCollection))]

public void Check\_CollectionCreationIfItems<T>(IEnumerable<T> items)

{

// Act

var collection = new MyLinkedList<T>(items);

// Assert

Assert.Multiple(() =>

{

Assert.That(collection, Is.Not.Null);

Assert.That(collection, Has.Count.EqualTo(items.Count()));

});

}

[Test]

public void Check\_CollectionIsReadOnly()

{

//Arrange

var collection = new MyLinkedList<int>();

//Assert

Assert.That(collection.IsReadOnly, Is.False);

}

[Test]

public void GetEnumerator\_ShouldReturnElementsInCorrectOrder()

{

// Arrange

var myEnumerable = new MyLinkedList<int>() { 1, 2, 3 };

// Act

var enumerator = myEnumerable.GetEnumerator();

// Assert

foreach (var element in myEnumerable)

{

Assert.Multiple(() =>

{

Assert.That(enumerator.MoveNext(), Is.True);

Assert.That(element, Is.EqualTo(enumerator.Current));

});

}

Assert.That(enumerator.MoveNext(), Is.False);

}

}

public class CollectionFindingTests

{

[Test]

public void FindItem\_WhenNoItems()

{

//Arrange

var collection = new MyLinkedList<bool>();

//Act

var item = collection.Find(true);

//Assert

Assert.That(item, Is.Null);

}

[Test]

public void FindItem\_NotInItems()

{

//Arrange

var collection = new MyLinkedList<bool> { false, false, false };

//Act

var item = collection.Find(true);

//Assert

Assert.That(item, Is.Null);

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void FindItem\_InItems<T>(T firstValue, T secondValue, T thirdValue)

{

//Arrange

var collection = new MyLinkedList<T> { firstValue, secondValue, thirdValue };

//Act

var item = collection.Find(value: secondValue);

//Assert

Assert.Multiple(() =>

{

Assert.That(item, Is.Not.Null);

Assert.That(item!.Value, Is.EqualTo(secondValue));

});

}

[TestCase(-1)]

[TestCase(int.MaxValue)]

[TestCase(4)]

public void FindItem\_WhenWrongIndex(int index)

{

//Arrange

var collection = new MyLinkedList<int> { 1, 2, 3 };

//Assert

Assert.Throws<ArgumentOutOfRangeException>(() => collection.Find(index: index));

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void FindItem\_ByIndex\_WhenInCollection<T>(T firstValue, T secondValue, T thirdValue)

{

//Arrange

var collection = new MyLinkedList<T> { firstValue, secondValue, thirdValue };

//Act

var item = collection.Find(index: 1);

//Assert

Assert.Multiple(() =>

{

Assert.That(item, Is.Not.Null);

Assert.That(item!.Value, Is.EqualTo(secondValue));

});

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void CheckIf\_ItemInCollection<T>(T firstValue, T secondValue, T thirdValue)

{

//Arrange

var collection = new MyLinkedList<T> { firstValue, secondValue, thirdValue };

//Act

var firstResult = collection.Contains(firstValue);

var secondResult = collection.Contains(secondValue);

var thirdResult = collection.Contains(thirdValue);

//Assert

Assert.Multiple(() =>

{

Assert.That(firstResult, Is.True);

Assert.That(secondResult, Is.True);

Assert.That(thirdResult, Is.True);

});

}

[Test]

public void CheckIf\_ItemNotInCollection()

{

//Arrange

var collection = new MyLinkedList<int>();

//Act

var result = collection.Contains(1);

//Assert

Assert.That(result, Is.False);

}

[Test]

public void Check\_Indexer\_IfNoInCollection()

{

//Arrange

var collection = new MyLinkedList<int>();

//Assert

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = collection[1]);

}

[Test]

public void Check\_Indexer\_IfInCollection()

{

//Arrange

var collection = new MyLinkedList<int>() { 1, 2, 3 };

//Act

var result = collection[1];

//Assert

Assert.That(result, Is.EqualTo(2));

}

}

public class CollectionRemovingTests

{

[Test]

public void Check\_CollectionRemoveItem\_WhenEmptyCollection()

{

//Arrange

var collection = new MyLinkedList<int>();

//Act

var result = collection.Remove(1);

//Assert

Assert.That(result, Is.False);

}

[Test]

public void Check\_CollectionRemoveItem\_WhenNoItemInCollection()

{

//Arrange

var collection = new MyLinkedList<int>() { 1, 2, 3 };

//Act

var result = collection.Remove(0);

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.False);

Assert.That(collection, Has.Count.EqualTo(3));

});

}

[TestCase(1)]

[TestCase('h')]

[TestCase("Hello")]

[TestCase(true)]

[TestCase(27.4F)]

public void Check\_CollectionRemoveItem\_WhenOneItemInCollection<T>(T value)

{

//Arrange

var collection = new MyLinkedList<T>{

value,

};

//Act

var result = collection.Remove(value);

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.True);

Assert.That(collection, Has.Count.EqualTo(0));

Assert.That(collection.First, Is.Null);

Assert.That(collection.Last, Is.Null);

});

}

[TestCase(1, 2)]

[TestCase('h', '1')]

[TestCase("Hello", "World")]

[TestCase(true, true)]

[TestCase(27.4F, 65.4F)]

public void Check\_CollectionRemoveItem\_WhenTwoItemsInCollection<T>(T firstValue, T secondValue)

{

//Arrange

var collection = new MyLinkedList<T>{

firstValue,

secondValue

};

//Act

var result = collection.Remove(secondValue);

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.True);

Assert.That(collection, Has.Count.EqualTo(1));

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Next, Is.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Null);

});

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void Check\_CollectionRemoveMiddleItem\_WhenThreeItemsInCollection<T>(T firstValue, T secondValue, T thirdValue)

{

//Arrange

var collection = new MyLinkedList<T>{

firstValue,

secondValue,

thirdValue

};

//Act

var result = collection.Remove(secondValue);

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.True);

Assert.That(collection, Has.Count.EqualTo(2));

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Next, Is.Not.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Not.Null);

Assert.That(collection.First!.Next, Is.EqualTo(collection.Last));

Assert.That(collection.Last!.Previous, Is.EqualTo(collection.First));

Assert.That(collection.First.Value, Is.EqualTo(firstValue));

Assert.That(collection.Last.Value, Is.EqualTo(thirdValue));

});

}

[Test]

public void Check\_CollectionRemoveFirstItem\_WhenOneItemInCollection()

{

//Arrange

var collection = new MyLinkedList<int>() { 1 };

//Act

var result = collection.RemoveFirst();

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.True);

Assert.That(collection, Has.Count.EqualTo(0));

Assert.That(collection.First, Is.Null);

Assert.That(collection.Last, Is.Null);

});

}

[TestCase(1, 2)]

[TestCase('h', '1')]

[TestCase("Hello", "World")]

[TestCase(true, true)]

[TestCase(27.4F, 65.4F)]

public void Check\_CollectionRemoveFirstItem\_WhenTwoItemsInCollection<T>(T firstValue, T secondValue)

{

//Arrange

var collection = new MyLinkedList<T>{

firstValue,

secondValue

};

//Act

var result = collection.RemoveFirst();

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.True);

Assert.That(collection, Has.Count.EqualTo(1));

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Next, Is.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Null);

});

}

[Test]

public void Check\_CollectionRemoveFirstItem\_WhenNoItemsInCollection()

{

//Arrange

var collection = new MyLinkedList<int>();

//Act

var result = collection.RemoveFirst();

//Assert

Assert.That(result, Is.False);

}

[Test]

public void Check\_CollectionRemoveLastItem\_WhenNoItemsInCollection()

{

//Arrange

var collection = new MyLinkedList<int>();

//Act

var result = collection.RemoveLast();

//Assert

Assert.That(result, Is.False);

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void Check\_CollectionRemoveItem\_InTheStart<T>(T firstValue, T secondValue, T thirdValue)

{

//Arrange

var collection = new MyLinkedList<T>()

{

firstValue,

secondValue,

thirdValue

};

//Act

var result = collection.RemoveFirst();

//Assert

Assert.Multiple(() =>

{

Assert.That(result, Is.True);

Assert.That(collection, Has.Count.EqualTo(2));

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.First!.Next, Is.Not.Null);

Assert.That(collection.First!.Value, Is.EqualTo(secondValue));

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Not.Null);

Assert.That(collection.Last!.Value, Is.EqualTo(thirdValue));

});

}

[TestCase(1, 2, 3)]

[TestCase('h', '1', 'h')]

[TestCase("Hello", "World", "!")]

[TestCase(true, true, false)]

[TestCase(27.4F, 65.4F, 14.0F)]

public void Check\_CollectionRemoveItems\_InTheStart\_And\_InTheEnd<T>(T firstValue, T secondValue, T thirdValue)

{

//Arrange

var collection = new MyLinkedList<T>()

{

firstValue,

secondValue,

thirdValue

};

//Act

var firstResult = collection.RemoveFirst();

var secondResult = collection.RemoveLast();

//Assert

Assert.Multiple(() =>

{

Assert.That(firstResult, Is.True);

Assert.That(secondResult, Is.True);

Assert.That(collection, Has.Count.EqualTo(1));

Assert.That(collection.First, Is.Not.Null);

Assert.That(collection.First!.Previous, Is.Null);

Assert.That(collection.First!.Next, Is.Null);

Assert.That(collection.First!.Value, Is.EqualTo(secondValue));

Assert.That(collection.Last, Is.Not.Null);

Assert.That(collection.Last!.Next, Is.Null);

Assert.That(collection.Last!.Previous, Is.Null);

Assert.That(collection.Last!.Value, Is.EqualTo(secondValue));

});

}

}

**Висновки:** в ході роботи я навчився створювати модульні тести для вихідного коду розроблювального програмного забезпечення.

**Питання до роботи:**

*1)* *Що таке модульне тестування? Яка сфера їх застосування, що саме*

*варто покривати модульними тестами?*

Модульне тестування є одним із видів тестування програмного забезпечення, спрямованим на перевірку правильності роботи окремих модулів (частин) програми. Модуль в даному контексті - це найменша одиниця програми, яка може бути протестована ізольовано від інших частин системи. У C# модуль може бути, наприклад, класом, методом або іншою відокремленою частиною програмного коду. Зі сфер застосування: рефакторинг, підтримка коду та забезпечення якості коду.

*2)* *Наведіть вимоги для гарних модульних тестів?*

**Незалежність**: модульні тести не впливають одне на одного, що не викликає каскадні помилки.

**Автоматизація**: автоматизація тестів дозволяє легко виконувати їх навіть при зміні коду.

**Покриття коду:** тести повинні покривати майже всі важливі частини коду.

**Швидкість виконання:** тести повинні виконуватись швидко, особливо під час неперервного тестування.

**Легкість обслуговування:** при зміні функціоналу програми, тести повинні легко змінюватись.

*3)* *Що таке TDD, коли доцільно застосовувати цей підхід?*

TDD (test-driven development) – розробка програмного забезпечення через попереднє написання тестів. Варто застосовувати цей підхід, коли очікуєш, що код буде часто змінюватись.

*4)* *Розкажіть про принцим “Triple A”, які переваги та недоліки він має?*

Принцип "Triple A" (або AAA) - це підхід до написання модульних тестів, який ставить у центр тестування три основні етапи: Arrange, Act, і Assert. Цей підхід допомагає структурувати тестовий код та полегшити зрозуміння тестового сценарію.

Основні етапи "Triple A":

1. Arrange (Підготовка): визначення вхідних даних і стану системи, створення необхідних об'єктів та ресурсів.

2. Act (Дія): виклик методу або функції, яку ви тестуєте, виконання певної дії або послідовності дій.

3. Assert (Підтвердження): перевірка, чи отриманий результат відповідає очікуваному, перевірка того, чи змінні, стани, або ефекти дій відповідають очікуванням.

*5)* *Розкажіть про Mock & Stub, а також їх відмінності.*

Mock та Stub - це об'єкти, які використовуються в тестуванні програмного забезпечення для імітації певної поведінки об'єктів чи системи та забезпечення контролю над умовами тестування. Ці терміни описують два різні підходи до створення фейкових об'єктів.

Mock набагато досконаліші, ніж Stub. У той час як Stub запрограмовані на отримання того самого результату на основі набору конкретних вхідних даних, Mock можна запрограмувати, щоб знати, скільки разів і в якому порядку слід викликати функції під час тестування. Stub не можуть відстежувати ці деталі, що робить Mock ідеальним для більших і складніших тестів.

*6)* *Що таке рефакторинг, яку роль в ньому відіграють модульні тести?*

Рефакторинг - це процес вдосконалення внутрішньої структури коду без зміни зовнішньої його поведінки. Основна ідея полягає в тому, щоб робити код більш зрозумілим, гнучким та легко супроводжуваним. Під час рефакторингу може бути здійснена переструктуризація коду, видалення дублюючого коду, виправлення неправильностей та інші покращення.

Модульні тести допомагають визначити, чи функціональність коду залишається незмінною після проведення рефакторингу. Якщо тести проходять після внесення змін, це означає, що ви не порушили існуючу функціональність коду.

*7)* *Розкрийте поняття піраміда тестування. Які ще підходи до*

*тестування знаєте?*

Згідно з пірамідою тестування, на найнижчому рівні розташовані широкі та швидкі тести, які покривають базовий функціонал (наприклад, модульні тести), а на вищих рівнях - більше специфічні та обширні тести (наприклад, функціональні та інтеграційні тести).

Unit Tests – модулі,

Intergration Tests – взаємодія між компонентами,

Functional Tests – функціональність малих компонентів відносно великих

User Interface Tests – помилки при взаємодії користувачів з програмою

Smoke Testing – перевірка базового функціоналу для того, щоб тестувати детальніше