Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №3

Тестирование программных блоков
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 _____ Е.В. Железкин
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021

1 Цель работы

На конкретных примерах ознакомиться с базовыми методами блочного тестирования программного обеспечения.

2 Задача работы

Продемонстрировать понимание и применение на практике ключевых понятий, рассмотренных в этой работе.

Вариант 5

Битовый вектор (реализовать в виде массива, хранящего значения true-false), при этом должны быть реализованы и протестированы следующие побитовые (поэлементные) логические операции:

- Сравнение
- И
- ИЛИ
- НЕ
- Импликация

2.1 Инструкция по запуску

Необходимо установить .Net SDK:

- Страница загрузки для Windows: https://dotnet.microsoft.com/download/dotnet/5.0
- Для Linux:

```
$ sudo apt-get update; \
$ sudo apt-get install -y apt-transport-https && \
$ sudo apt-get update && \
```

$ sudo apt-get install -y dotnet-sdk-5.0

Далее на любой из двух систем выполнить в папке проекта:

$ dotnet test

$ dotnet test -v n (для более подробного вывода)

3 Ход работы

Был проведён анализ тестов из практической работы №2, результат представлен на рисунке 1. В тестах не проверяется возможность выбросов исключений (Рисунок 2). Так как не весь код покрыт тестами, то для начала исправим это. Результат представлен на рисунке №3.

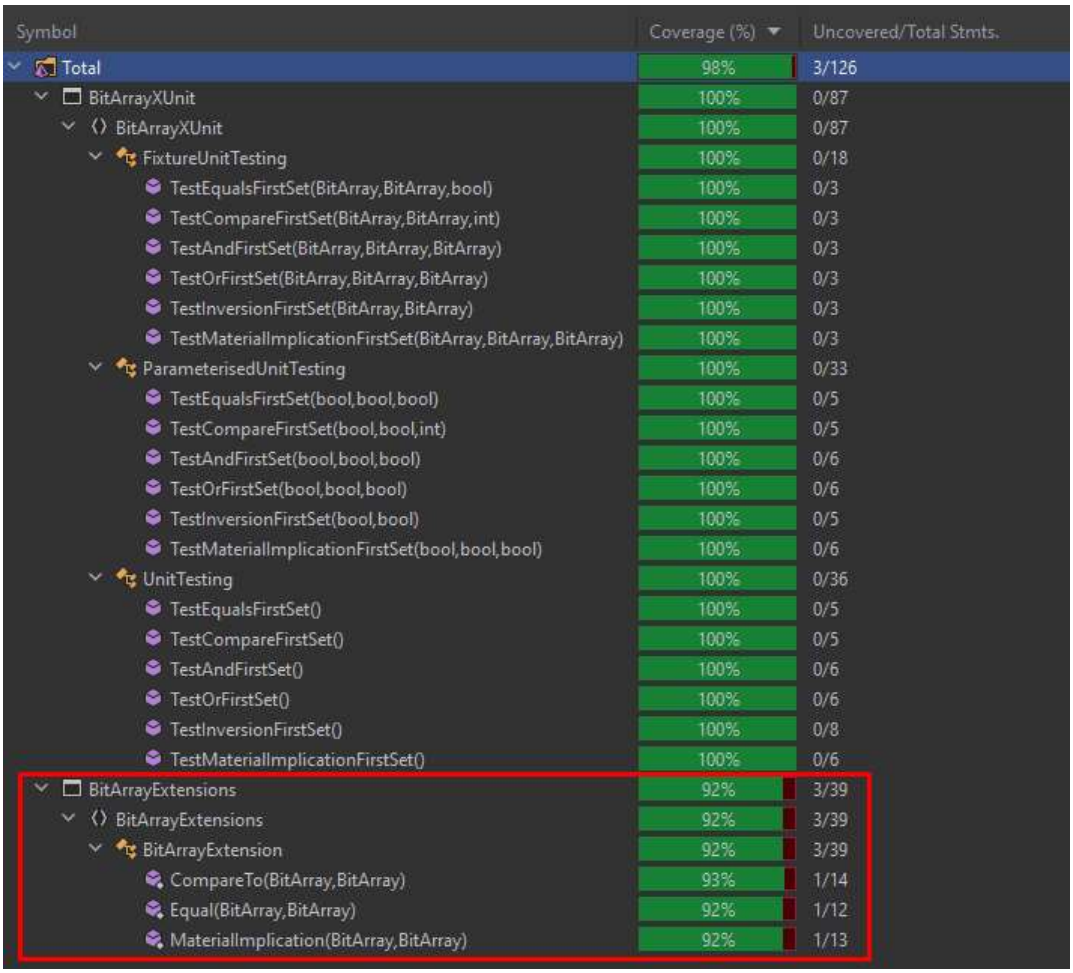


Рисунок 1 – Статистика покрытия кода из ПР2 тестами

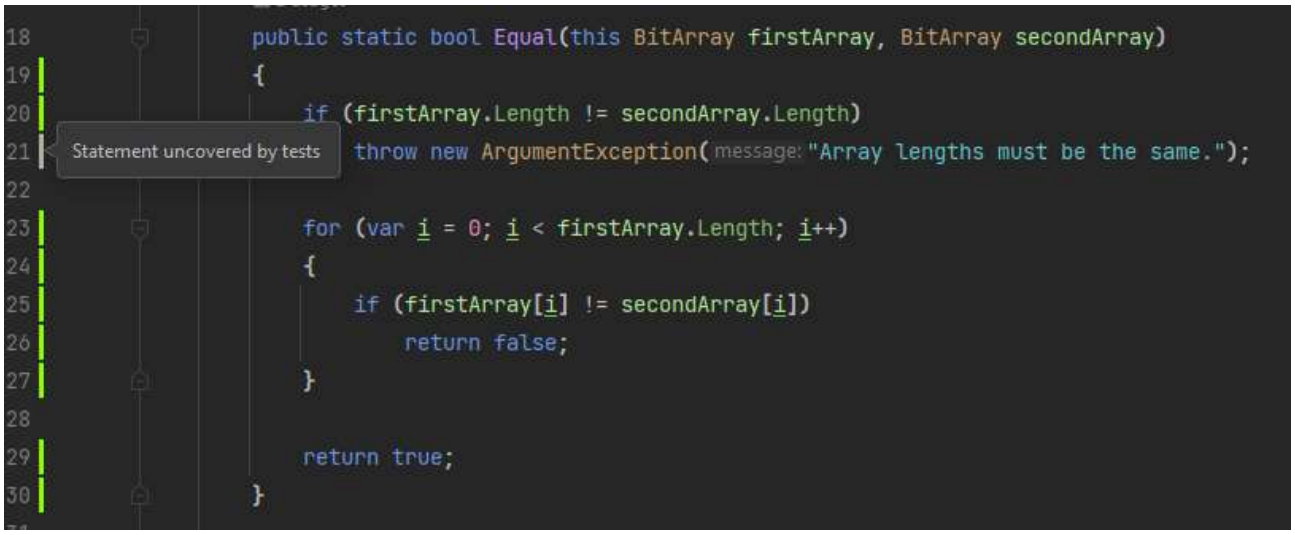


Рисунок 2 – Тестирование с помощью unit-тестов

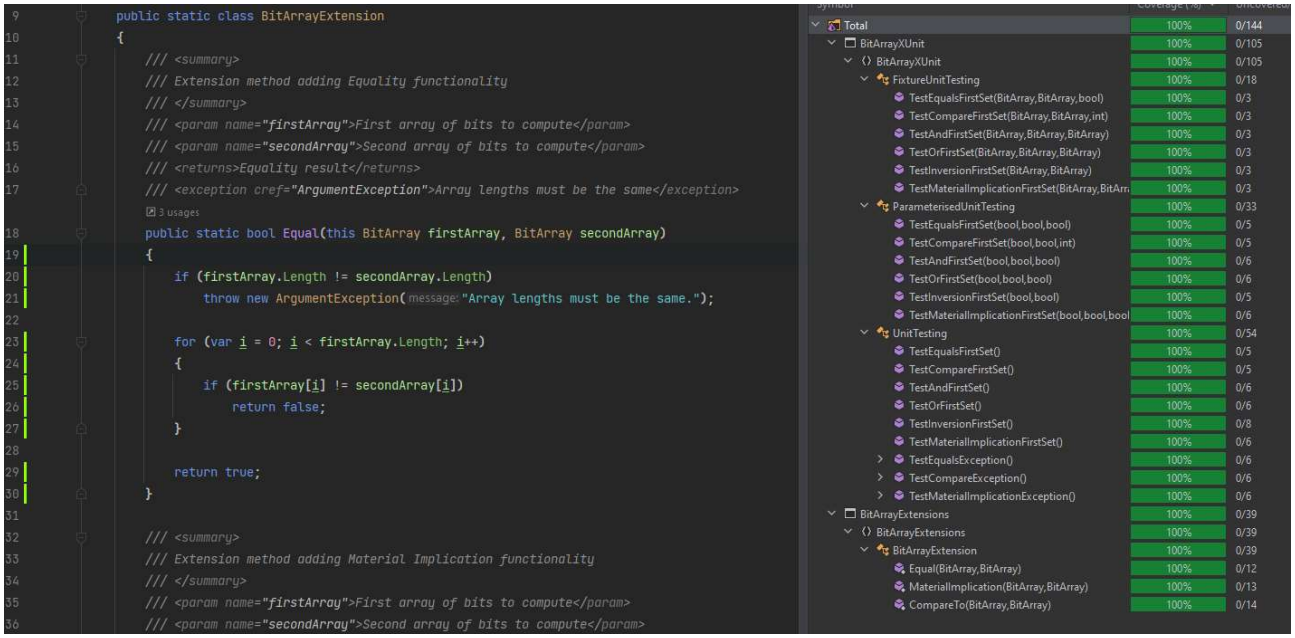
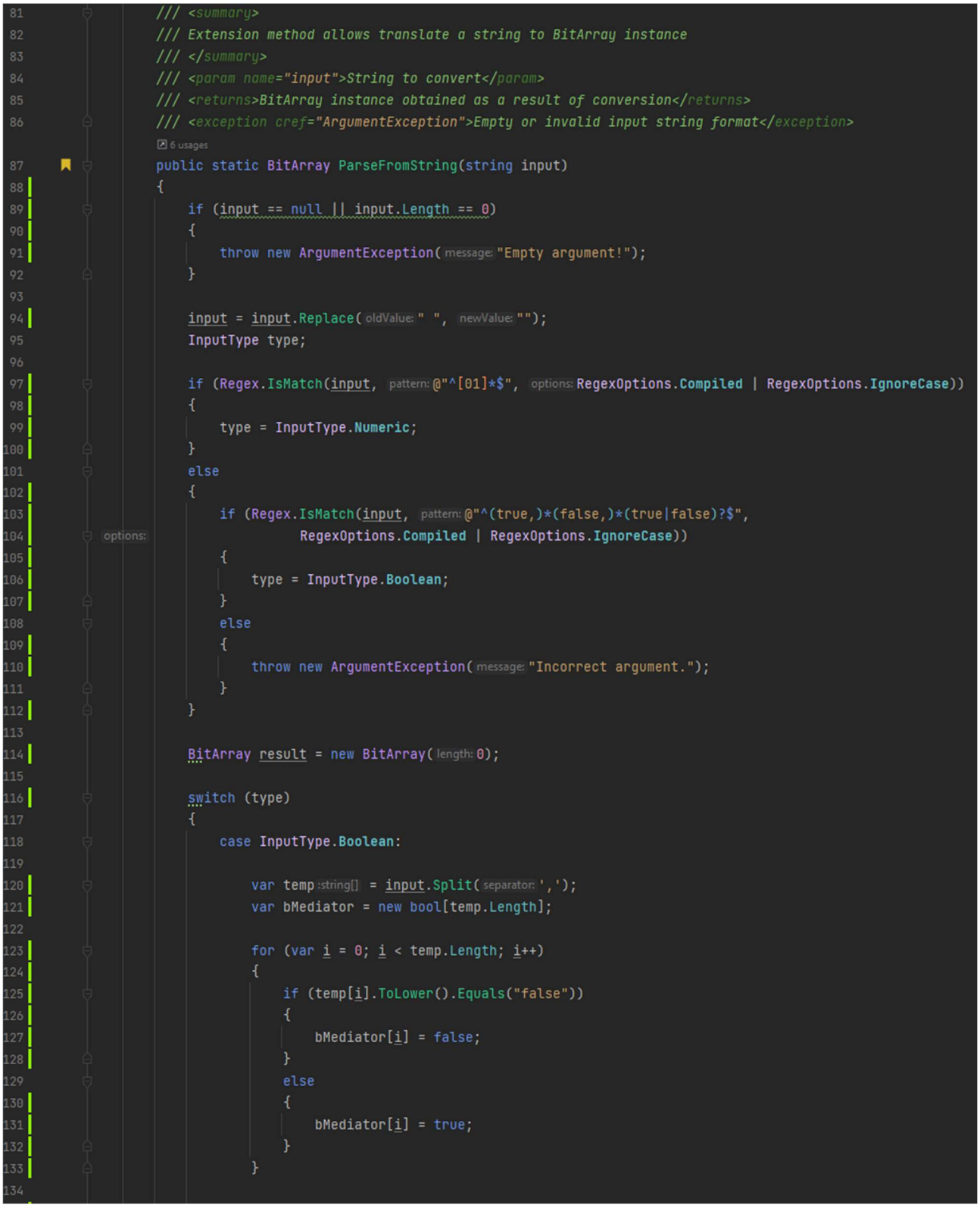


Рисунок 3 – Тестирование с помощью unit-тестов

Далее, по примеру в задании, был написан метод, конвертирующий строку в объект класса BitArray:



```
/// <summary>
/// Extension method allows translate a string to BitArray instance
/// </summary>
/// <param name="input">String to convert</param>
/// <returns>BitArray instance obtained as a result of conversion</returns>
/// <exception cref="ArgumentException">Empty or invalid input string format</exception>
6 usages
public static BitArray ParseFromString(string input)
{
    if (input == null || input.Length == 0)
    {
        throw new ArgumentException(message: "Empty argument!");
    }

    input = input.Replace(oldValue: " ", newValue: "");
    InputType type;

    if (Regex.IsMatch(input, pattern: @"^[01]*$", options: RegexOptions.Compiled | RegexOptions.IgnoreCase))
    {
        type = InputType.Numeric;
    }
    else
    {
        if (Regex.IsMatch(input, pattern: @"^(true,)*(false,)*(true|false)?$",
options:                  RegexOptions.Compiled | RegexOptions.IgnoreCase))
        {
            type = InputType.Boolean;
        }
        else
        {
            throw new ArgumentException(message: "Incorrect argument.");
        }
    }

    BitArray result = new BitArray(length: 0);

    switch (type)
    {
        case InputType.Boolean:

            var temp :string[] = input.Split(separator: ',');
            var bMediator = new bool[temp.Length];

            for (var i = 0; i < temp.Length; i++)
            {
                if (temp[i].ToLower().Equals("false"))
                {
                    bMediator[i] = false;
                }
                else
                {
                    bMediator[i] = true;
                }
            }
```

Рисунок 4 – Часть метода ParseFromString

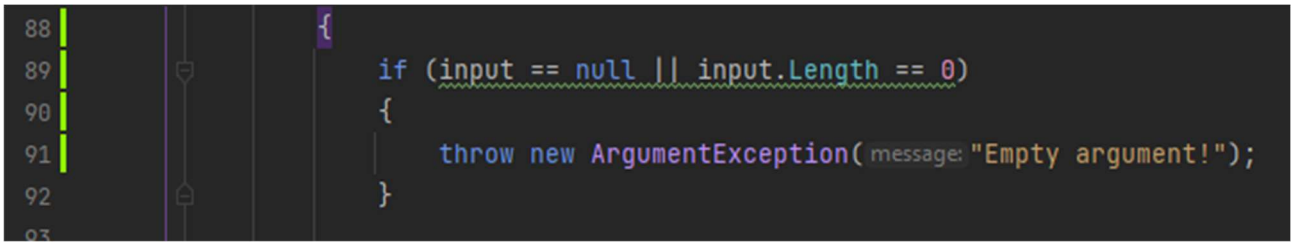


Рисунок 5 – Составной условный оператор

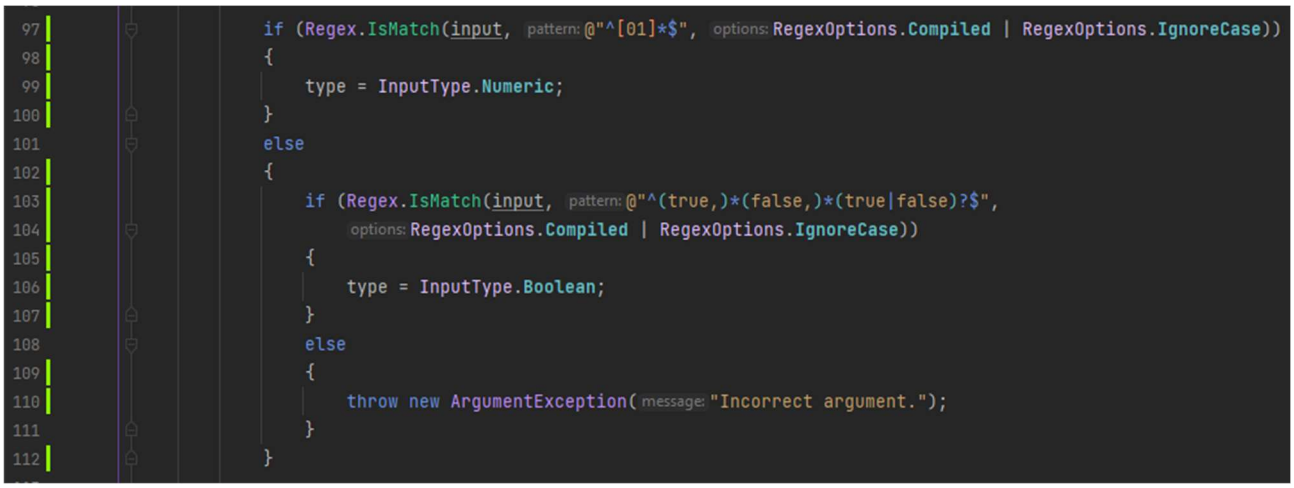


Рисунок 6 – Условные операторы

```
116         switch (type)
117         {
118             case InputType.Boolean:
119 
120                 var temp :string[] = input.Split( separator: ',');
121                 var bMediator = new bool[temp.Length];
122 
123                 for (var i = 0; i < temp.Length; i++)
124                 {
125                     if (temp[i].ToLower().Equals("false"))
126                     {
127                         bMediator[i] = false;
128                     }
129                     else
130                     {
131                         bMediator[i] = true;
132                     }
133                 }
134 
135                 result = new BitArray(bMediator);
136 
137                 break;
138 
139             case InputType.Numeric:
140 
141                 var nMediator = new bool[input.Length];
142 
143                 for (var i = 0; i < input.Length; i++)
144                 {
145                     if (input[i] == '0')
146                     {
147                         nMediator[i] = false;
148                     }
149                     else
150                     {
151                         nMediator[i] = true;
152                     }
153                 }
154 
155                 result = new BitArray(nMediator);
156 
157                 break;
158         }
```

Рисунок 7 – Оператор switch - case

Я смог выделить только два класса эквивалентности битовых векторов: недействительные (пустые) и действительные (все прочие).

```
[Test]
public void Test1()
{
    var actual :BitArray = BitArrayExtension.ParseFromString( input: "110");
    var expected = new BitArray( values: new bool[] { true, true, false });

    Assert.AreEqual(expected, actual);
}

[Test]
public void Test2()
{
    var actual :BitArray = BitArrayExtension.ParseFromString( input: "true,True,false");
    var expected = new BitArray( values: new bool[] { true, true, false });

    Assert.AreEqual(expected, actual);
}
```

Рисунок 8 – Тестирование потенциальных действительных битовых векторов

```
[Test]
public void Test20()
{
    Assert.Catch( code: () => BitArrayExtension.ParseFromString( input: null));
}
```

Рисунок 9 – Тестирование потенциальных недействительных битовых векторов

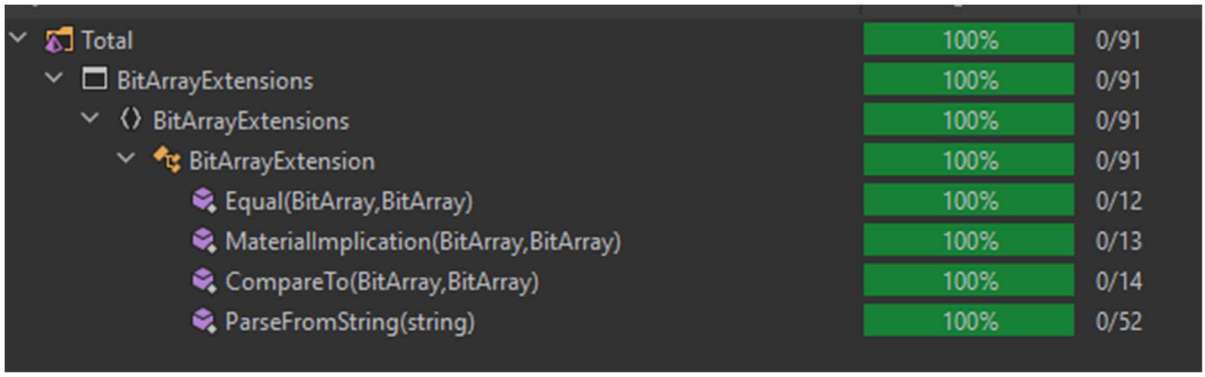


Рисунок 10 – Полное покрытие кода тестами

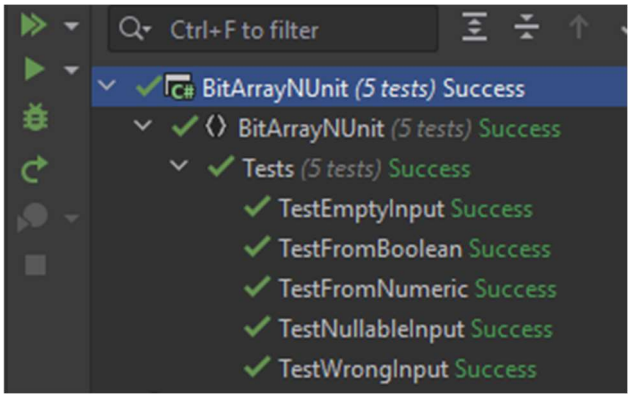


Рисунок 11 – Результаты тестов

Так же был добавлен и покрыт тестами метод «TripleAnd» (Рисунок 12-14)

```
/// <summary>
/// Extension "triple and" method
/// </summary>
/// <param name="first">First BitArray instance for comparison</param>
/// <param name="second">Second BitArray instance for comparison</param>
/// <param name="third">Third BitArray instance for comparison</param>
/// <returns>"Triple and" result</returns>
1 usage  new *
public static bool TripleAnd(BitArray first, BitArray second, BitArray third)
{
    return first.Equal(second) && second.Equal(third);
}
```

Рисунок 12 – Метод «TripleAnd»

Рисунок 13 – Тест для метода

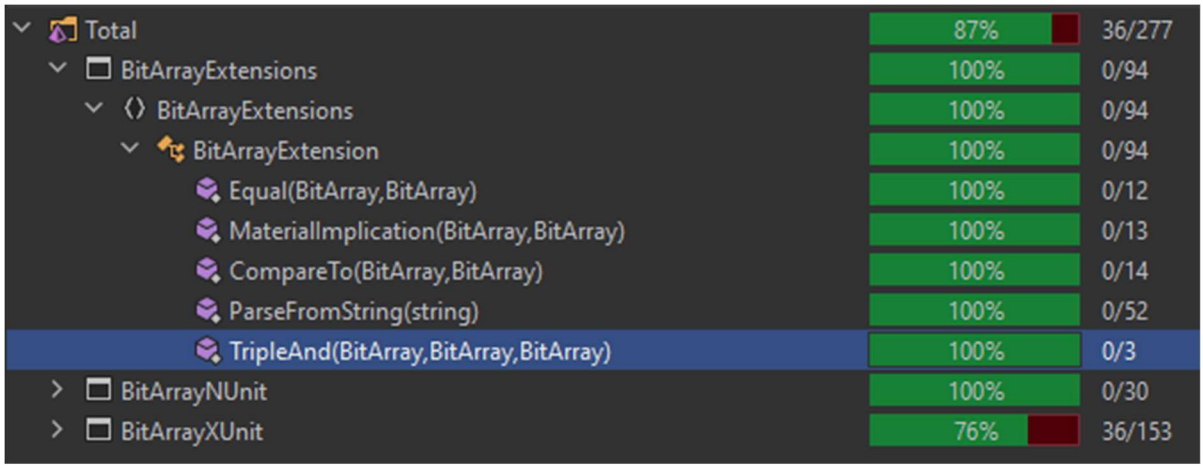


Рисунок 14 – Покрытие кода

Как видно из предыдущего теста, встроенная система покрытия тестов в среду разработки JetBrains Rider не настолько чувствительная(в настройках тоже не возможности это изменить) и не требует тестировать альтернативное поведение метода. Для полноты тестирования последней функции напишем ещё один тест для альтернативного исхода (Рисунок 15):

```
[Test]
new *
public void TestTripleAndTrue()
{
    var first :BitArray = BitArrayExtension.ParseFromString( input: "11011");
    var second :BitArray = BitArrayExtension.ParseFromString( input: "true,True,false,true,true");
    var third = new BitArray( values: new bool[] { true, true, false, true, true });

    Assert.True(BitArrayExtension.TripleAnd(first, second, third));
}

[Test]
new *
public void TestTripleAndFalse()
{
    var first :BitArray = BitArrayExtension.ParseFromString( input: "11011");
    var second :BitArray = BitArrayExtension.ParseFromString( input: "true,True,false,true,true");
    var third = new BitArray( values: new bool[] { true, true, true, true, true });

    Assert.False(BitArrayExtension.TripleAnd(first, second, third));
}
```

Рисунок 15 – Тест на «ложь» функции «TripleAnd»

4 Вывод

В ходе данной лабораторной работы были изучены базовые методы блочного тестирования программного обеспечения на конкретных примерах.