

Mutation Testing

PATRICK DRECHSLER



Code Coverage 100%

The image shows a Visual Studio IDE with two code files open: `Rules.cs` and `RulesTests.cs`. The `Rules.cs` file contains the following code:

```
1 namespace DemoLib;
2
3 public class Rules
4 {
5     public bool IsValid(string s)
6     {
7         if (string.IsNullOrEmpty(s))
8         {
9             return false;
10        }
11        if (s.Length > 3)
12        {
13            return false;
14        }
15        else
16        {
17            return true;
18        }
19    }
20
21    public bool AnotherValidation(int a, int b)
22    {
23        if (a + b == 10) return true;
24        if (a - b == 10) return true;
25        if (a * b == 10) return true;
26        if (a / b == 10) return true;
27        if (a % b == 1) return true;
28        return false;
29    }
30
31    public int Looping(int a)
32    {
33        var result = 0;
34        for (int i = 0; i < a; i++)
35        {
36            result++;
37        }
38        return result;
39    }
40
41    public bool Logical(bool b1, bool b2)
42    {
43        if (b1 && b2) return true;
44        if (b1 || b2) return true;
45        return false;
46    }
47
48    public int GetFirst(IEnumerable<int> numbers) => numbers.First();
49
50 }
```

The `RulesTests.cs` file contains the following code:

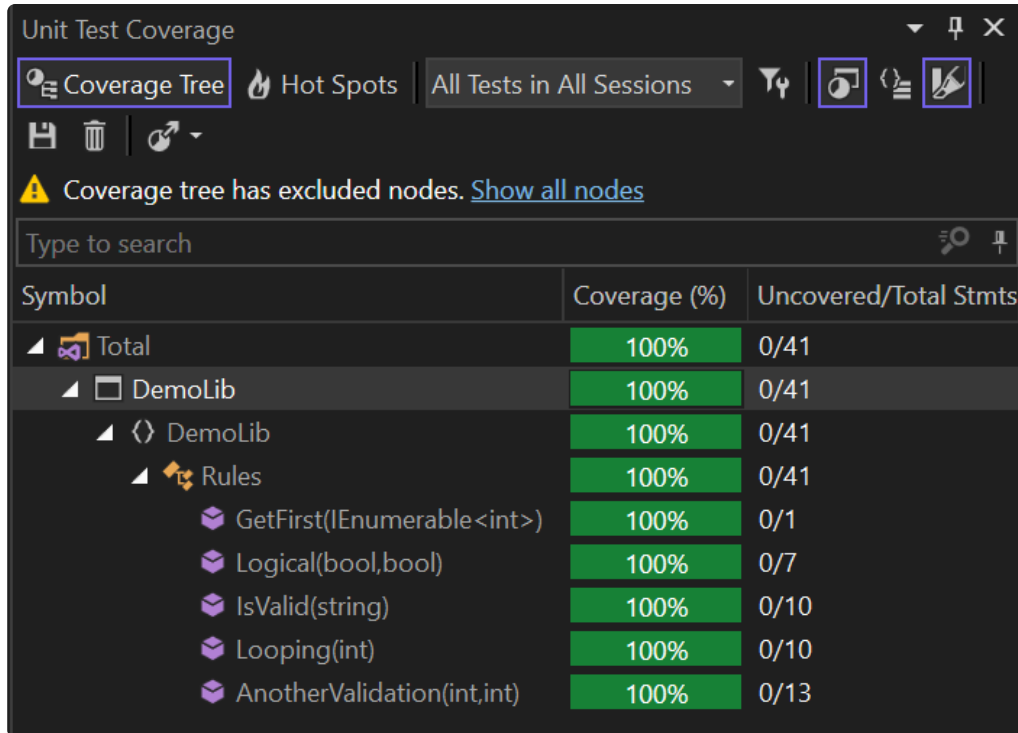
```
13 public RulesTests() => _sut = new Rules();
14
15 // Include each 'InlineData' line one by one, run 'dotnet stryker' and see how it changes the outcome.
16 [Theory]
17 [InlineData("a", true)] // -> 3 survive
18 [InlineData("", false)] // -> 2 survive
19 [InlineData(null, false)] // -> 2 survive
20 [InlineData("123", true)] // -> 1 survive
21 [InlineData("12345", false)] // -> 0 survive
22
23 public void Validation_works(string input, bool expected) =>
24     _sut.IsValid(input).Should().Be(expected);
25
26 [Theory]
27 [InlineData(5, 5, true)] // addition
28 [InlineData(15, 5, true)] // subtraction
29 [InlineData(2, 5, true)] // multiplication
30 [InlineData(20, 2, true)] // division
31 [InlineData(5, 4, true)] // modulo
32 [InlineData(1, 1, false)] // other combinations
33
34 public void AnotherValidation_works(int a, int b, bool expected) =>
35     _sut.AnotherValidation(a, b).Should().Be(expected);
36
37 [Theory]
38 [InlineData(3, 3)]
39
40 [Theory]
41 [InlineData(true, true, true)]
42 [InlineData(true, false, true)]
43 [InlineData(false, true, true)]
44 [InlineData(false, false, false)]
45
46 public void Logical_works(bool a, bool b, bool expected) =>
47     _sut.Logical(a, b).Should().Be(expected);
48
49 // https://github.com/stryker-mutator/stryker-handbook/blob/master/mutator-types.md#method-expression
50 [Theory]
51 [MemberData(nameof(GetFirstData), parameters: 1)] // list with length 1: First() = Last()
52 [MemberData(nameof(GetFirstData), parameters: 2)] // list with 2 different entries
53
54 public void GetFirst_works(IEnumerable<int> collection, int expected) =>
55     _sut.GetFirst(collection).Should().Be(expected);
56
57 2 references | Patrick Drechsler, 3 days ago | 1 author, 2 changes
58 public static IEnumerable<object[]> GetFirstData(int numberOfTests)
59 {
60     var allData = new List<object[]>()
61     {
62         new object[] { new List<int> { 1 }, 1 },
63         new object[] { new List<int> { 1, 2 }, 1 },
64     };
65     return allData.Take(numberOfTests);
66 }
```

A red box highlights the text "100% code coverage" in the center of the image. A red arrow points from this box to the "Unit Test Coverage" window on the right side of the image. The "Unit Test Coverage" window shows a table with the following data:

Symbol	Coverage (%)	Uncovered
Total	100%	0/41
DemoLib	100%	0/41
Rules	100%	0/41
IsValid	100%	0/1
AnotherValidation	100%	0/7
Logical	100%	0/10
Looping	100%	0/10
Another	100%	0/13

Code Coverage 100%

what more do you want?



The screenshot shows the 'Unit Test Coverage' window. At the top, there are tabs for 'Coverage Tree' (selected), 'Hot Spots', and a dropdown menu set to 'All Tests in All Sessions'. Below the tabs are icons for saving, deleting, and refreshing. A warning message states: 'Coverage tree has excluded nodes. [Show all nodes](#)'. Below this is a search bar with the placeholder text 'Type to search'. The main area displays a table with three columns: 'Symbol', 'Coverage (%)', and 'Uncovered/Total Stmts'.

Symbol	Coverage (%)	Uncovered/Total Stmts
▲ Total	100%	0/41
▲ DemoLib	100%	0/41
▲ DemoLib	100%	0/41
▲ Rules	100%	0/41
GetFirst(IEnumerable<int>)	100%	0/1
Logical(bool,bool)	100%	0/7
IsValid(string)	100%	0/10
Looping(int)	100%	0/10
AnotherValidation(int,int)	100%	0/13



Example

```
public class Rules
{
    public bool IsValid(string s)
    {
        if (string.IsNullOrEmpty(s))
        {
            return false;
        }

        if (s.Length > 3)
        {
            return false;
        }

        return true;
    }
}
```

```
public class RulesTests
{
    private readonly Rules _sut;

    public RulesTests() => _sut = new Rules();

    [Theory]
    [InlineData("", false)]
    [InlineData("a", true)]
    [InlineData("12345", true)]
    public void Validation_works(
        string input, bool expected) =>
        _sut.IsValid(input).Should().Be(expected);
}
```

- 100% coverage...
- but, are we covering all corner cases?

Let's create some mutants!

Let's change

```
if (s.Length > 3)
```

to

```
if (s.Length < 3) // <- this is a "MUTANT"
```

```
if (s.Length >= 3) // <- this is another "MUTANT"
```

```
if (s.Length <= 3) // <- ...and another "MUTANT"
```

Do we still have the same code coverage?





Concept

- Production code is modified (by the mutation testing framework)
- Test suite is run

Did any mutants survive?

- If all mutants die, the test suite is fine 🍌
- But if some mutants survive, the tests are not covering all cases 👁👁
 - 🙌 take a closer look

Many mutation frameworks generate an interactive html report





Example mutations

<https://stryker-mutator.io/docs/stryker-net/mutations/>

Arithmetic Operators
(*arithmetic*)

Equality Operators (*equality*)

Logical Operators (*logical*)

Boolean Literals (*boolean*)

Assignment Statements
(*assignment*)

Collection initialization
(*initializer*)

Removal mutators (*statement*,
block)

Unary Operators (*unary*)

Update Operators (*update*)

Checked Statements (*checked*)

Linq Methods (*linq*)

String Literals and Constants
(*string*)

Bitwise Operators (*bitwise*)

Regular Expressions (*regex*)



Watch the video "How to test your tests in .NET" from *Nick Chapsas*



Stryker Mutator



Stryker.NET



Mutations

On this page



Mutations

Stryker supports a variety of mutators, which are listed below. In parentheses the names of correspondent mutations are specified, which you might need for the `exclude-mutations` section of the configuration.

Do you have a suggestion for a (new) mutator? Feel free to create an [issue](#)!

Arithmetic Operators (*arithmetic*)



Live Demo





Other languages

Frameworks are available for many languages:

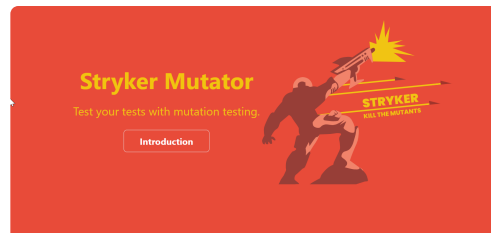
- 🐙 Java (using PIT)
- Scala (using Stryker4s)
- 🐙 C# (using Stryker.NET)
- 🐙 Javascript/Typescript (using StrykerJS)
- 🐙 Python (using Cosmic Ray or mutmut)
- Haskell (using MuCheck)
- ...

🐙: Example project in the repo



Real world mutation testing

PIT is a state of the art **mutation testing** system, providing **gold standard test coverage** for Java and the jvm. It's fast, scalable and integrates with modern test and build tooling.



Getting started with Stryker



mutmut mutmut - python mutation tester

build passing docs failing codecov 81% chat 17 online

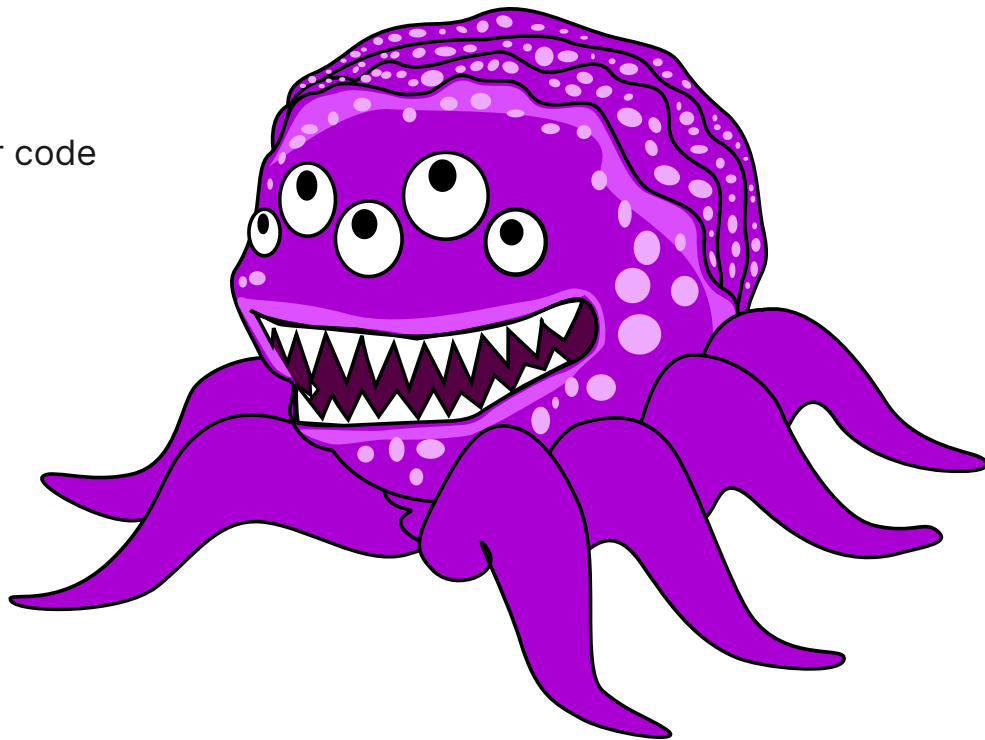
» Hackage :: [Package]

MuCheck: Automated Mutation Testing



Mutation Testing: Summary

- requires decent test coverage
- modifies production code to find corner cases
- requires a lot of resources: use deliberately!
- Don't include it in your CI/CD pipeline
- Use it as an exploratory tool to find bugs in your code
- Use it to find critical bugs in your code





The End

Any Questions?

