Unit Testing

How to Write a Unit Test in C# That Checks for an Exception

Kristijan Kralj

Life is unpredictable. And unexpected things happen.

It's the same with the code. There are places in the code where you need to use error handling to protect the rest of the code execution from bad inputs. Unit tests can be used to check for exceptions and to ensure that your code handles such exceptions the way you expect it to.

If you use the xUnit testing framework, you can check that the exception was thrown in two different ways:

- **Assert.Throws**
- **Using try-catch block**

This post will show you how to write unit tests for your C# code that checks for exceptions. It will also discuss why you want to write these unit tests.

Table of Contents

Should exceptions be unit tested?

Unit Testing (https://methodpoet.com/unit-testing-in-c/) is a way to test the smallest part of a program to see if it works and use it for troubleshooting. Unit tests are also a great way to refactor code and see which parts of a large application are currently working. When writing these tests, it's important to make sure that they cover all of the bases. For example, one of the scenarios that occur in code is throwing exceptions.

Exceptions are a controversial topic. Some developers think they should be unit tested, while others see exceptions as anti-patterns to be avoided (https://methodpoet.com/worst-anti-patterns/) at all costs.

But you definitely should test that your code throws an exception when bad input happens. If you don't test the exception cases, you will not cover all possible scenarios with your test suite. And that can lead to unexpected behavior in your code and bugs.

So how do you assert exceptions in unit testing with the xUnit framework?

Test for exceptions using Assert.Throws

When it comes to writing your code, it's important to know whether or not it's working. Assertions are a great way to do that. An assertion is a test that checks if something is true.

In this post, I will use the xUnit (https://xunit.net/) unit testing framework.

xUnit has an `Assert.Throws` method on the Assert class for testing that the expected exception occurs. The `Assert.Throws` method takes a function as a parameter, which will be called with the list of parameters to the method under test.

This is an example of throwing an exception:

```
[Fact]
public void Action_throws_exception()
{
    Action throwingAction = () =>
    {
        throw new ArgumentException();
    };

    Assert.Throws<ArgumentException>(throwingAction);
}
```

As you can see, it very simple. You wrap your code into the `Action` and pass that to the `Assert.Throws` method.

Let's see how this works when testing the production code. We want to test the `DocumentWriter` class. It has a method for writing paragraphs to the document.

```
class DocumentWriter
{
    public void Write(string paragraph)
    {
        if (paragraph.Length < 10)
        {
            throw new ArgumentException("You need to specify a bigger paragraph."
        }
        //other logic here...
    }
}
```

Going back to the test class, the following test case checks that the `Write` method throws an exception.

```
[Fact]
public void Write_throws_exception_for_too_short_paragraph()
{
    var documentWriter = new DocumentWriter();

    Action testWrite = () => documentWriter.Write("");

    Assert.Throws<ArgumentException>(testWrite);
}
```

Testing asynchronous code

In case you want to test the asynchronous method, you can use `ThrowsAsync`. The following test method provides an example.

```
[Fact]
public void AsyncAction_throws_exception()
{
    Func<Task> throwingAction = () =>
    {
        throw new ArgumentException();
    };

    Assert.ThrowsAsync<ArgumentException>(throwingAction);
}
```

In this case, you need to use `Func`, since the method's return type is `Task`. This means that you are calling the async method.

Checking exception message

How to check an exception message in unit tests?

It is straightforward to check the exception message in unit tests. You just need to capture the thrown exception.

```
[Fact]
public void Read_exception()
{
    Action throwingAction = () => { throw new ArgumentException("Argument is not w

    var exception = Assert.Throws<ArgumentException>(throwingAction);

    Assert.Equal("Argument is not within required range", exception.Message);
}
```

Test for exceptions using try-catch block

`Assert.Throws` is a preferred way to check for exceptions while using xUnit. But there is another way to achieve the same goal. In your unit test case, you can use a try-catch block.

Let me show you what I mean.

```
[Fact]
public void Write_throws_exception_for_too_short_paragraph2()
{
    var documentWriter = new DocumentWriter();

    try
    {
        documentWriter.Write("");
        Assert.True(false, "Should have thrown the exception");
    }
    catch (ArgumentException)
    {
        //the expected exception occurred
    }
}
```

You use the try block to execute the code under test. If the method throws an exception, that will be covered by the catch block. If the `Write` method doesn't work as the test code expects, the `Assert.True(false, message)` will be triggered, and the automated test fails.

Conclusion

Unit testing is an essential practice for any developer. It helps you find bugs and identify faulty code. Unit tests also help you maintain code quality and consistency. And you should also use them to check w exceptions happen in the code.