Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

C# static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C# code

All rules 409 | Vulnerability 34 | Bug 76 | Security Hotspot 28 | Code Smell 271 | Quick Fix 52

Tags

Search by name...

"protected" members
Code Smell

Underscores should be used to make large numbers readable
Code Smell

"ToString()" calls should not be redundant
Code Smell

"==" should not be used when "Equals" is overridden
Code Smell

An abstract class should have both abstract and concrete methods
Code Smell

Multiple variables should not be declared on the same line
Code Smell

Culture should be specified for "string" operations
Code Smell

"switch" statements should have at least 3 "case" clauses
Code Smell

break statements should not be used except for switch cases
Code Smell

String literals should not be duplicated
Code Smell

Files should contain an empty newline at the end
Code Smell

Unused "using" should be removed
Code Smell

Use a testable date/time provider.

Analyze your code

Code Smell | Major

One of the principles of a unit test is that it must have full control of the system under test. This is problematic when production code includes calls to static methods, which cannot be changed or controlled. Date/time functions are usually provided by system libraries as static methods.

This can be improved by wrapping the system calls in an object or service that can be controlled inside the unit test.

Noncompliant Code Example

```
public class Foo
{
    public string HelloTime() =>
        $"Hello at {DateTime.UtcNow}";
}
```

Compliant Solution

There are different approaches to solve this problem. One of them is suggested below. There are also open source libraries (such as NodaTime) which already implement an `IClock` interface and a `FakeClock` testing class.

```
public interface IClock
{
    DateTime UtcNow();
}

public class Foo
{
    public string HelloTime(IClock clock) =>
        $"Hello at {clock.UtcNow()}";
}

public class FooTest
{
    public record TestClock(DateTime now) : IClock
    {
        public DateTime UtcNow() => now;
    }

    [Fact]
    public void HelloTime_Gives_CorrectTime()
    {
        var dateTime = new DateTime(2017, 06, 11);
        Assert.Equal((new Foo()).HelloTime(new TestClock(dat
    }
}
```

A close curly brace should be located at the beginning of a line

Code Smell

Tabulation characters should not be used

Code Smell

Methods and properties should be named in PascalCase

Code Smell

Track uses of in-source issue suppressions

Code Smell

Another possible solution is using an adaptable static class, ideally supports an IDisposable method, that not only adjusts the time behaviour for the current thread only, but also for scope of the using.

```
public static class Clock
{
    public static DateTime UtcNow() { /* ... */ }
    public static IDisposable SetTimeForCurrentThread(Func<D
}

public class Foo
{
    public string HelloTime() =>
        $"Hello at {Clock.UtcNow()}";
}

public class FooTest
{
    [Fact]
    public void HelloTime_Gives_CorrectTime()
    {
        var dateTime = new DateTime(2017, 06, 11);
        using (Clock.SetTimeForCurrentThread(() => dateTime)
        {
             Assert.Equal((new Foo()).HelloTime(), $"Hello a
        }
    }
}
```

See

NodaTime testing

Available In:

sonarlint | sonarcloud | sonarqube