
































-  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...



"switch" statements should not be nested

Code Smell

Methods and properties should not be too complex

Code Smell

Control flow statements "if", "switch", "for", "foreach", "while", "do" and "try" should not be nested too deeply

Code Smell

"switch/Select" statements should contain a "default/Case Else" clauses

Code Smell

"if ... else if" constructs should end with "else" clauses

Code Smell

Control structures should use curly braces

Code Smell

Expressions should not be too complex

Code Smell

ASP.NET HTTP request validation feature should not be disabled

Vulnerability

Serialization constructors should be secured

Vulnerability

Calculations should not overflow

Bug

Floating point numbers should not be tested for equality

Bug

Increment (++) and decrement (--) operators should not be used in a

"ServiceContract" and "OperationContract" attributes should be used together

Analyze your code

Code Smell Major api-design

The `ServiceContract` attribute specifies that a class or interface defines the communication contract of a Windows Communication Foundation (WCF) service. The service operations of this class or interface are defined by `OperationContract` attributes added to methods. It doesn't make sense to define a contract without any service operations; thus, in a `ServiceContract` class or interface at least one method should be annotated with `OperationContract`. Similarly, WCF only serves `OperationContract` methods that are defined inside `ServiceContract` classes or interfaces; thus, this rule also checks that `ServiceContract` is added to the containing type of `OperationContract` methods.

Noncompliant Code Example

```
[ServiceContract]
interface IMyService // Noncompliant
{
    int MyServiceMethod();
}
```

Compliant Solution

```
[ServiceContract]
interface IMyService
{
    [OperationContract]
    int MyServiceMethod();
}
```

Available In:

sonarlint | sonarcloud | sonarqube

operators should not be used in a method call or mixed with other operators in an expression

 Code Smell

Use a testable date/time provider.

 Code Smell

Property names should not match get methods

 Code Smell

Locales should be set for data types

 Code Smell

Literals should not be passed as