

- 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...

Code Smell

Constructor and destructor declarations should not be redundant

Code Smell

Method parameters should be declared with base types

Code Smell

The simplest possible condition syntax should be used

Code Smell

Redundant parentheses should not be used

Code Smell

"GC.SuppressFinalize" should not be invoked for types without destructors

Code Smell

Members should not be initialized to default values

Code Smell

Sequential tests should not check the same condition

Code Smell

Redundant modifiers should not be used

Code Smell

Methods and properties that don't access instance data should be static

Code Smell

"Exception" should not be caught when not required by called methods

Code Smell

"sealed" classes should not have "protected" members

Code Smell

Flags enumerations should explicitly initialize all their members

Analyze your code

Bug Minor ?

Flags enumerations should not rely on the language to initialize the values of their members. Implicit initialization will set the first member to 0, and increment the value by one for each subsequent member. This implicit behavior does not allow members to be combined using the bitwise or operator in a useful way.

Instead, 0 and powers of two (i.e. 1, 2, 4, 8, 16, ...) should be used to explicitly initialize all the members.

Noncompliant Code Example

```
[Flags]
enum FruitType    // Noncompliant
{
    None,
    Banana,
    Orange,
    Strawberry
}
class Program
{
    static void Main()
    {
        var bananaAndStrawberry = FruitType.Banana | FruitTy
        // Will display only Strawberry!
        Console.WriteLine(bananaAndStrawberry.ToString());
    }
}
```

Compliant Solution

```
[Flags]
enum FruitType
{
    None = 0,
    Banana = 1,
    Orange = 2,
    Strawberry = 4
}
class Program
{
    static void Main()
    {
        var bananaAndStrawberry = FruitType.Banana | FruitTy
        // Will display Banana and Strawberry, as expected.
        Console.WriteLine(bananaAndStrawberry.ToString());
    }
}
```

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

Exceptions

The default initialization of 0, 1, 2, 3, 4, ... matches 0, 1, 2, 4, 8 ... in the first three values, so no issue is reported if the first three members of the enumeration is not initialized.

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)