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...

Number patterns should be regular
Code Smell

"out" and "ref" parameters should not be used
Code Smell

Unchanged local variables should be "const"
Code Smell

"ConfigureAwait(false)" should be used
Code Smell

"interface" instances should not be cast to concrete types
Code Smell

Literal boolean values should not be used in assertions
Code Smell

Optional parameters should not be used
Code Smell

Public constant members should not be used
Code Smell

Array covariance should not be used
Code Smell

"nameof" should be used
Code Smell

Modulus results should not be checked for direct equality
Code Smell

"for" loop increment clauses should modify the loops' counters
Code Smell

Objects should not be disposed more than once

Analyze your code

Code Smell | Major | confusing pitfall

Disposing an object twice, either with the `using` keyword or by calling `Dispose` directly, in the same method is at best confusing and at worst error-prone. The next developer might see only one of the `Dispose`/`using` and try to use an already-disposed object.

In addition, even if the documentation of `Disposable` explicitly states that calling the `Dispose` method multiple times should not throw an exception, some implementation still do it. Thus it is safer to not dispose an object twice when possible.

This rule raises an issue when, in the same method, the `Dispose` method is explicitly called twice on the same object, or when `using` is used with a direct call to `Dispose()`.

Noncompliant Code Example

```
using (var d = new Disposable()) // Noncompliant
{
    d.Dispose();
}
```

```
using var d = new Disposable();
d.Dispose(); // Noncompliant {{Refactor this code to make su
```

Compliant Solution

```
using var d = new Disposable();
```

Available In:

sonarlint | sonarcloud | sonarqube

"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