
































-  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

"Any()" should be used to test for emptiness

Analyze your code

Code Smell Minor Quick Fix clumsy

Using `.Count()` to test for emptiness works, but using `.Any()` makes the intent clearer, and the code more readable. However, there are some cases where special attention should be paid:

- if the collection is an `EntityFramework` or other ORM query, calling `.Count()` will cause executing a potentially massive SQL query and could put a large overhead on the application database. Calling `.Any()` will also connect to the database, but will generate much more efficient SQL.
- if the collection is part of a LINQ query that contains `.Select()` statements that create objects, a large amount of memory could be unnecessarily allocated. Calling `.Any()` will be much more efficient because it will execute fewer iterations of the enumerable.

Noncompliant Code Example

```
private static bool HasContent(IEnumerable<string> strings)
{
    return strings.Count() > 0; // Noncompliant
}

private static bool HasContent2(IEnumerable<string> strings)
{
    return strings.Count() >= 1; // Noncompliant
}

private static bool IsEmpty(IEnumerable<string> strings)
{
    return strings.Count() == 0; // Noncompliant
}
```

Compliant Solution





```
private static bool HasContent(IEnumerable<string> strings)
{
    return strings.Any();
}

private static bool HasContent2(IEnumerable<string> strings)
{
    return strings.Any();
}

private static bool IsEmpty(IEnumerable<string> strings)
{
    return !strings.Any();
}
```

Available In:



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