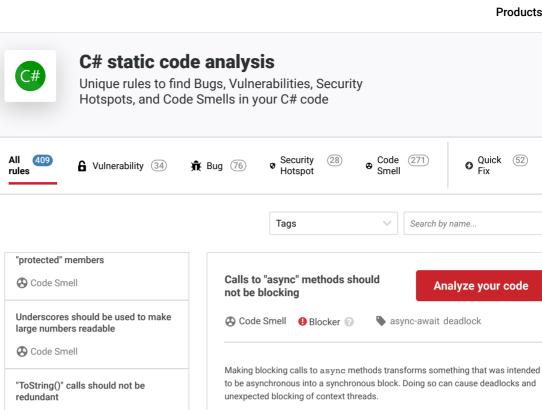
Q





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

```
According to the MSDN documentation:
  The root cause of this deadlock is due to the way await handles contexts. By
  default, when an incomplete Task is awaited, the current "context" is captured
  and used to resume the method when the Task completes. This "context" is
  the current SynchronizationContext unless it's null, in which case it's the
  current TaskScheduler. GUI and ASP.NET applications have a
  SynchronizationContext that permits only one chunk of code to run at a
  time. When the await completes, it attempts to execute the remainder of the
  async method within the captured context. But that context already has a
  thread in it, which is (synchronously) waiting for the async method to
  complete. They're each waiting for the other, causing a deadlock.
    To Do This ...
                            Instead of This ...
                                                          Use This
Retrieve the result of a Task.Wait, Task.Result or
                     Task.GetAwaiter.GetResult
background task
Wait for any task to
                     Task.WaitAny
                                                    await Task.WhenAny
complete
Retrieve the results of
                     Task.WaitAll
                                                    await Task.WhenAll
multiple tasks
Wait a period of time Thread. Sleep
                                                    await Task.Delay
Noncompliant Code Example
  public static class DeadlockDemo
      private static asvnc Task DelavAsvnc()
       {
           await Task.Delay(1000);
      // This method causes a deadlock when called in a GUI or
      public static void Test()
           // Start the delay.
           var delayTask = DelayAsync();
           // Wait for the delay to complete.
           delayTask.Wait(); // Noncompliant
  }
Compliant Solution
  public static class DeadlockDemo
```

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

```
private static async Task DelayAsync()
{
    await Task.Delay(1000);
}

public static async Task TestAsync()
{
    // Start the delay.
    var delayTask = DelayAsync();
    // Wait for the delay to complete.
    await delayTask;
}
```

Exceptions

- Main methods of Console Applications are not subject to this deadlock issue and so are ignored by this rule.
- Thread.Sleep is also ignored when it is used in a non-async method.
- Calls chained after Task.Run or Task.Factory.StartNew are ignored because they don't suffer from this deadlock issue

See

Async/Await - Best Practices in Asynchronous Programming

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