How do you programmatically identify the number of references to a method with C#

Asked 16 years, 3 months ago Modified 11 years, 7 months ago

Viewed 3k times

6

I've recently inherited C# console application that is in need of some pruning and clean up. Long story short, the app consists of a single class containing over 110,000 lines of code. Yup, over 110,000 lines in a single class. And, of course, the app is core to our business, running 'round the clock updating data used on a dynamic website. Although I'm told my predecessor was "a really good programmer", it obvious he was not at all into OOP (or version control).

Anyway... while familiarizing myself with the code I've found plenty of methods that are declared, but never referenced. It looks as if copy/paste was used to version the code, for example say I have a method called getSomethingImportant(), chances are there is another method called getSomethingImortant_July2007() (the pattern is functionName_[datestamp] in most cases). It looks like when the programmer was asked to make a change to getSomethingImportant() he would copy/paste then rename to getSomethingImortant_Date, make changes to getSomethingImortant_Date, then change

any method calls in the code to the new method name, leaving the old method in the code but never referenced.

I'd like to write a simple console app that crawls through the one huge class and returns a list of all methods with the number of times each method was referenced. By my estimates there are well over 1000 methods, so doing this by hand would take a while.

Are there classes within the .NET framework that I can use to examine this code? Or any other usefull tools that may help identify methods that are declared but never referenced?

(Side question: Has anyone else ever seen a C# app like this, one reeeealy big class? It's more or less one huge procedural process, I know this is the first I've seen, at least of this size.)

c# refactoring

Share

Improve this question

Follow

asked Sep 19, 2008 at 23:15

Duffy

527 ● 1 ● 7 ● 10

10 Answers

Sorted by: Highest score (default)

You could try to use NDepend if you just need to extract some stats about your class. Note that this tool relies on

14

Mono.Cecil internally to inspect assemblies.

Share Improve this answer

Follow

answered Sep 19, 2008 at 23:22

Romain Verdier

13k ● 7 ● 59 ● 77

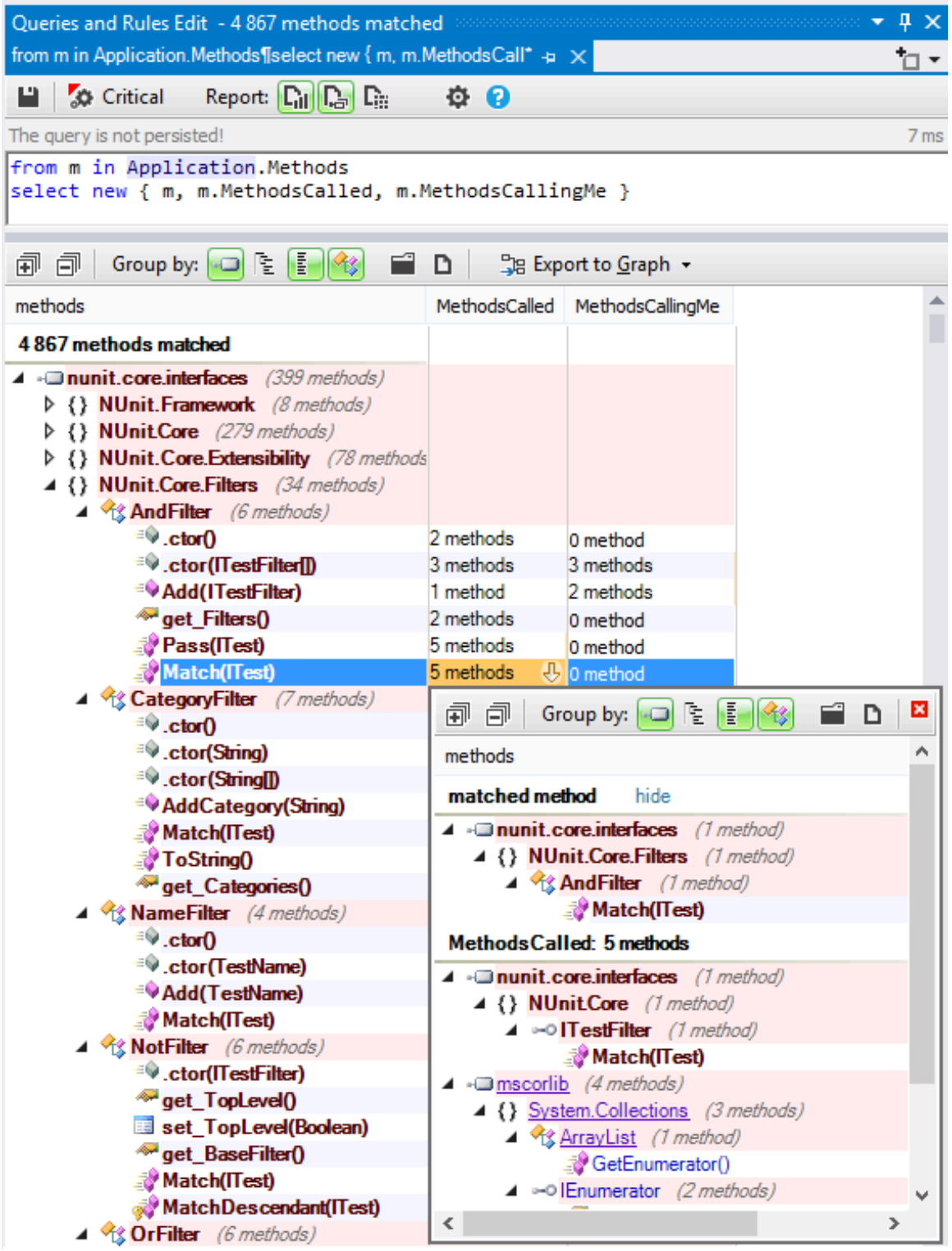
6

To complete the *Romain Verdier* answer, lets dig a bit into what NDepend can bring to you here. (*Disclaimer: I am a developer of the NDepend team*)

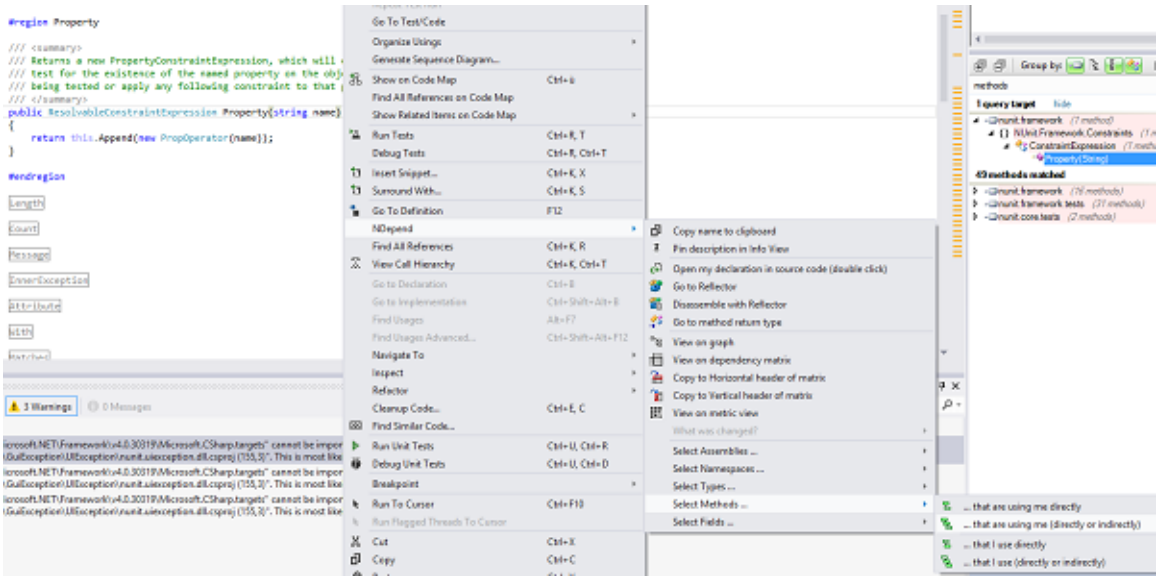
NDepend lets query your .NET code with some LINQ queries. Knowing which methods call and is called by which others, is as simple as writing the following LINQ query:

```
from m in Application.Methods
select new { m, m.MethodsCalled, m.MethodsCallingMe }
```

The result of this query is presented in a way that makes easy to browse callers and callees (and its 100% integrated into Visual Studio).



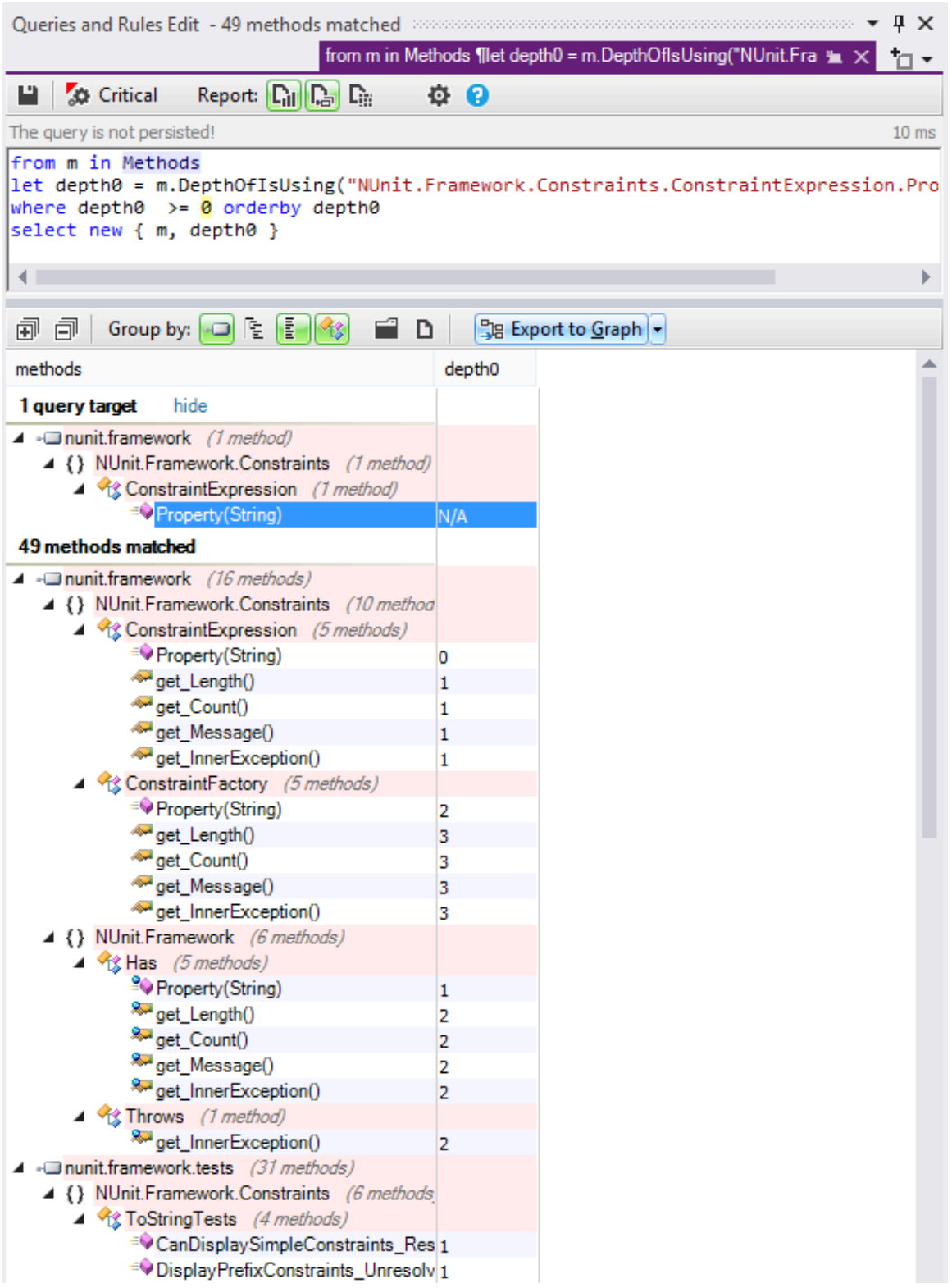
There are many other NDepend capabilities that can help you. For example you can *right click a method in Visual Studio > NDepend > Select methods... > that are using me (directly or indirectly)* ...



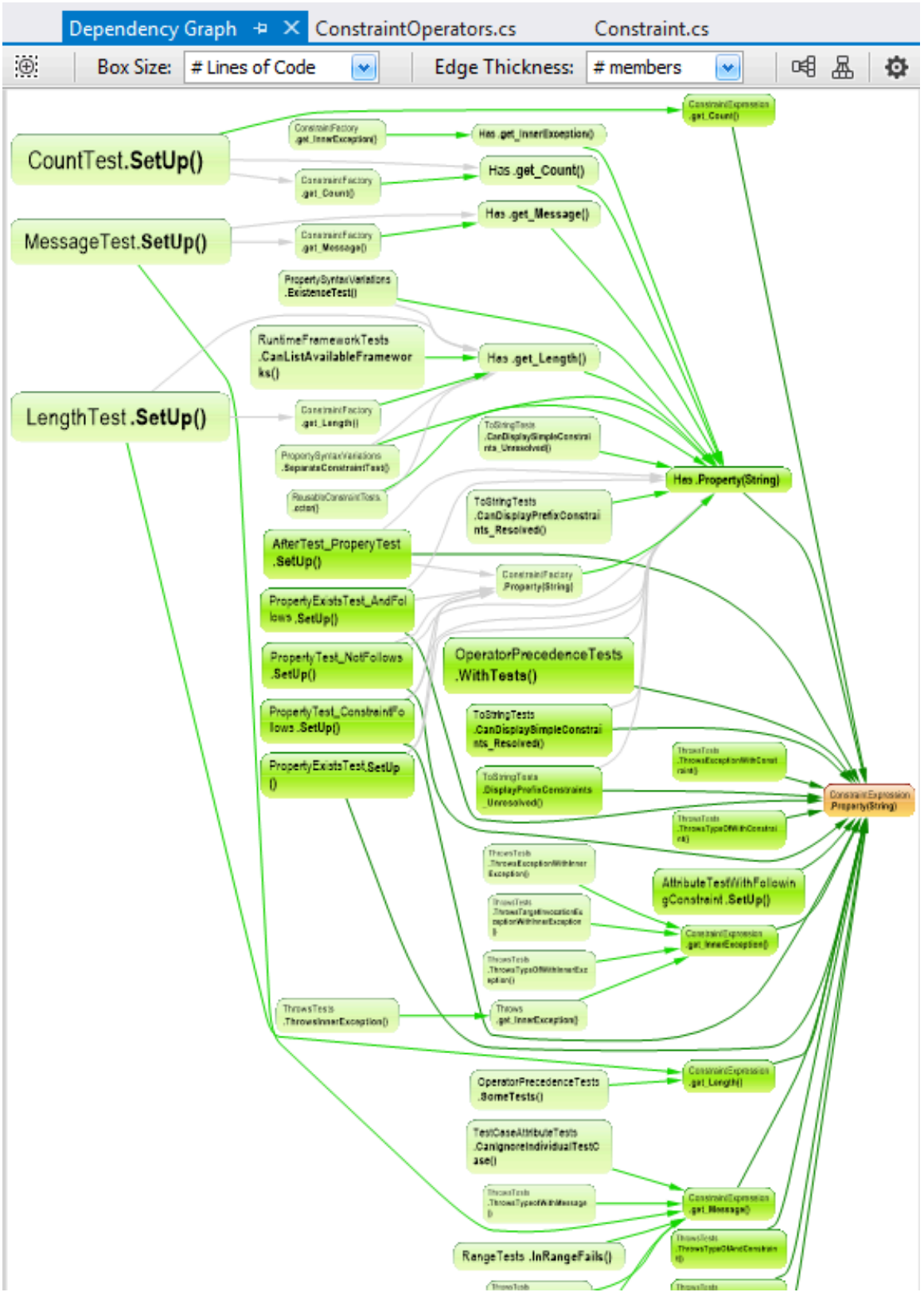
The following code query is generated...

```
from m in Methods
let depth0 =
m.DepthOfIsUsing("NUnit.Framework.Constraints.Constrai
where depth0  >= 0 orderby depth0
select new { m, depth0 }
```

... which matches direct and indirect callers, with the depth of calls (1 means direct caller, 2 means caller of direct callers and so on).



And then by clicking the button **Export to Graph**, you get a call graph of your pivot method (of course it could be the other way around, i.e method called directly or indirectly by a particular pivot method).



Share Improve this answer

Follow

edited May 4, 2013 at 11:16

answered Oct 26, 2008 at 17:29

Patrick from NDepend team

13.8k ● 6 ● 67 ● 106

When I was trying to generate graph it is taking huge time by using the following query `from m in Application.Methods select new { m, m.MethodsCalled, m.MethodsCallingMe }` but here I am selecting only one file method which is having only 2 dependencies. – Krish Dec 5, 2019 at 7:26

4

Download the *free trial* of Resharper. Use the Resharper->Search->Find Usages in File (Ctrl-Shift-F7) to have all usages highlighted. Also, a count will appear in the status bar. If you want to search across multiple files, you can do that too using Ctrl-Alt-F7.

If you don't like that, do text search for the function name in Visual Studio (Ctrl-Shift-F), this should tell you how many occurrences were found in the solution, and where they are.

Share Improve this answer

Follow

answered Sep 19, 2008 at 23:42

tobinharris

2,559 ● 1 ● 21 ● 20

1

I don't think you want to write this yourself - just buy NDepend and use its Code Query Language

Share Improve this answer

Follow

answered Sep 19, 2008 at 23:23

Steve Eisner

2,005 ● 23 ● 35

1

There is no easy tool to do that in .NET framework itself. However I don't think you really need a list of unused methods at once. As I see it, you'll just go through the code and for each method you'll check if it's unused and then delete it if so. I'd use Visual Studio "Find References" command to do that. Alternatively you can use Resharper with its "Analize" window. Or you can just use Visual Studio code analysis tool to find all unused private methods.

Share Improve this answer

Follow

answered Sep 19, 2008 at 23:29

Ihar Bury

494 ● 3 ● 10

Your suggestion to delete as I go through the code was my first thought too. I'll be using this method plus NDepend, as others suggested, to do further analysis. – Duffy Sep 20, 2008 at 0:05

1

FXCop has a rule that will identify unused private methods. So you could mark all the methods private and have it generate a list.

FXCop also has a language if you wanted to get fancier http://www.binarycoder.net/fxcop/

Share Improve this answer

Follow

answered Sep 20, 2008 at 0:56

Bob D

1

If you don't want to shell out for NDepend, since it sounds like there is just a single class in a single assembly - comment out the methods and compile. If it compiles, delete them - you aren't going to have any inheritance issues, virtual methods or anything like that. I know it sounds primitive, but sometimes refactoring is just grunt work like this. This is kind of assuming you have unit tests you run after each build until you've got the code cleaned up (Red/Green/Refactor).

Share Improve this answer

Follow

answered Sep 22, 2008 at 22:01

Cade Roux

89.6k ● 40 ● 184 ● 266

1

The Analyzer window in Reflector can show you where a method is called (Used By).
Sounds like it would take a very long time to get the information that way though.
You might look at the API that Reflector provides for writing add-ins and see if you can get the grunt work of the analysis that way. I would expect that the source code for the code metrics add-in could tell you a bit about how to get information about methods from the reflector API.

Edit: Also the code model viewer add-in for Reflector could help too. It's a good way to explore the Reflector

API.

Share Improve this answer Follow

edited Oct 6, 2008 at 19:38

answered Sep 19, 2008 at 23:28

Hamish Smith

8,181 ● 1 ● 38 ● 49

0

I don't know of anything that's built to handle this specific case, but you could use Mono.Cecil. Reflect the assemblies and then count references in the IL. Shouldn't be too tough.

Share Improve this answer Follow

answered Sep 19, 2008 at 23:17

Serafina Brocious

30.6k ● 12 ● 91 ● 115

-1

Try having the compiler emit assembler files, as in x86 instructions, not .NET assemblies.

Why? Because it's much easier to parse assembler code than it is C# code or .NET assemblies.

For instance, a function/method declaration looks something like this:

```
    .string "w+"
    .text
    .type   create_secure_tmpfile, @function
```

```
create_secure_tmpfile:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $24, %esp
    movl    $-1, -8(%ebp)
    subl    $4, %esp
```

and function/method references will look something like this:

```
    subl    $12, %esp
    pushl   24(%ebp)
    call    create_secure_tmpfile
    addl    $16, %esp
    movl    20(%ebp), %edx
    movl    %eax, (%edx)
```

When you see "create_secure_tmpfile:" you know you have a function/method declaration, and when you see "call create_secure_tmpfile" you know you have a function/method reference. This may be good enough for your purposes, but if not it's just a few more steps before you can generate a very cute call-tree for your entire application.

Share Improve this answer Follow

edited Sep 20, 2008 at 0:27

answered Sep 20, 2008 at 0:17

mbac32768

11.6k ● 9 ● 37 ● 41

1 How is it "much easier" to parse assembler when .net includes reflection libraries? Not to mention 3rd party libraries

like Mono.Cecil. – Wesley Wiser Aug 6, 2010 at 17:41

Because there's a huge base of tools for processing line-based data in an ad hoc way. e.g. grep, sed, awk, etc. – mbac32768 Oct 22, 2010 at 15:00