Statikus kódanalízis c# nyelvhez

Sok kódanalizátort lehet találni c# nyelvhez, bár ezek minősége igen vegyes. Három fő kategóriára oszthatóak. A legmegbízhatóbbak a Microsoft saját termékei, az FxCop, vagy a már Visual Studioban is elérhető Code Metrics funkció. Kevésbé megbízhatóak a fizetős szoftverek, melyeknek csak a demóját tudtam kipróbálni. Ilyen például az NDepend. A sor végén pedig az ingyenes programok állnak, melyekkel vagy nagyon kevés funkció érhető el vagy gyakran lefagynak.

Mivel a C# a .NET keretrendszer fő nyelve, ezért a keret rendszer köztes nyelvét (IL) is lehet elemezni, mely egyszerűbb mint általában a gépi kódok és rendelkezésre állnak a kódelemek nevei. A köztes nyelv elemzőknek a DLL-ekre van szükségük, így F#, vagy Visual Basic nyelven íródott programokat is tudnak elemezni. Az általam vizsgált IL elemző az NDepend, FxCop és a Nitriq.

A közvetlenül a forráskódot elemző programok általában más nyelveket is megértenek, tehát nem kötődnek a .NET keretrendszerhez. Ilyen például az Understand és a SourceMonitor.

Az elemzők két további osztályra bonthatóak. Az egyik csoport kódmetrikákat számol, a másik szabályok ellenőriz. Vannak olyanok is, melyek a szabályokba kódmetrikákat is belevesznek. A SourceMonitor például csak metrikákat számol, és a Visual Studio Premium Edition-jében lévő kód analizátor is csak ennyit tud. A szabály ellenörzők egyrészt stílusbeli hibákat is ellenőrizhetnek, például, hogy a bináris operátorok mellet van-e szóköz, ilyen például a StyleCop. A többség viszont tervezési, vagy fenntarthatósági hibákat keres, például, azon függvények megkeresése, melyek lehetnének statikusak, de nem azok. Egyes programok azt is támogatják, hogy saját szabályokat definiáljunk, ehhez általában egy SQL-hez hasonló nyelv áll rendelkezésre (NDepend, Nitriq).

Fontosabb kód metrikák

Cyclomatic Complexity

Egymástól különböző végrehajtási utak száma. Pontosabban, a program folyamatábráján lévő lineárisan független utak száma. Élek száma - Pontok száma + 2 * Összefüggő komponensek száma.

Coupling / Dependency

Csatoltság. Egyes elemek mennyire függenek más elemek működésétől. Többféleképpen is lehet mérni, például az egyik modul a másik modul hány függvényét hívja meg.

Class Coupling / Coupling Between Objects (CBO)

Hány másik osztályt használ egy adott osztály. Minél nagyobb ez a szám annál rosszabb. [S2010] szerint a 9 az optimális felső korlát.

Efferent Coupling (Ce) / Afferent Coupling (Ca)

A Coupling kétfelé bontható. Az Afferent Coupling azt méri, hogy az adott kódelemet hány másik kódelem használja. Az Efferent Coupling pedig azt méri, hogy a kérdéses kódelem, hány másik kódelemet használ. A kódelem lehet assembly, namespace, metódus, mező, stb..

Cohesion

Megmutatja, hogy egy modul mennyire összefüggő.

Halstead Complexity Mértékek

Először ezeket ki kell számolni a kód alapján:

- η_1 = hány különböző fajta operátor van
- η_2 = hány operandusként használt entitás van
- N_1 = operátorok száma
- N_2 = operandusok száma

Ebből számos mérőszám számolható:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$
- Number of delivered bugs: $B = \frac{E^{2/3}}{3000}$
- Time required to program: $T = E/18$ seconds

A difficulty azt próbálja mérni, hogy mennyire nehéz megérteni a kódot. Az effort azt méri, hogy mekkora munka lekódolni.

Maintainability Index (MI)

Azt méri mennyire karbantartható a kód. Valójában a méretére utaló különböző mennyiségek súlyozott átlaga. Maximális értéket csak az üres kód érheti el, mivel annál karbantarthatóbb nincs. Ha valaki ugyanazt a funkcionalitást tömörebben lekódolja, az magasabb értéket kap.

Először ezeket kell kiszámolni a kód alapján:

- V = Halstead Volume
- G = Cyclomatic Complexity
- LOC = Lines Of Code (SLOC)

Ezekből ki lehet számolni az indexet:

Az eredeti formula:

$$MI = 171 - 5.2 * ln(V) - 0.23 * (G) - 16.2 * ln(LOC)$$

A Visual Studio által használt formula át van skálázva 0-tól 100-ig:

$$MI^{VS} = MAX(0, MI) * 100 \ / \ 171)$$

Depth of Inheritance (DIT)

A Depth of inheritance három alapvető feltevésen alapszik:

1. Minél mélyebben van egy osztály a hierarchiában, annál több metódust örökölhet, ami megnehezíti a viselkedésének előrejelzését.
2. Mélyebb osztályhierarchia fák nagyobb tervezési komplexitást vonnak maguk után, mivel több osztályt és metódust tartalmaznak.
3. Mélyebb osztályok nagyobb eséllyel hasznosítják újra az örökölt metódusokat.

Az első két feltevés alapján a nagy mélység rossz, viszont a harmadik szerint hasznos a potenciális kód újrahasznosítás miatt. Tehát ebben nincs ideális érték.

Method Rank

A google pagerank algoritmusát futtatja a metódusok gráfján, és ez alapján méri le, hogy melyik milyen fontos. Ha ez magas értéket ad, akkor arra a metódusra jobban oda kell figyelni.

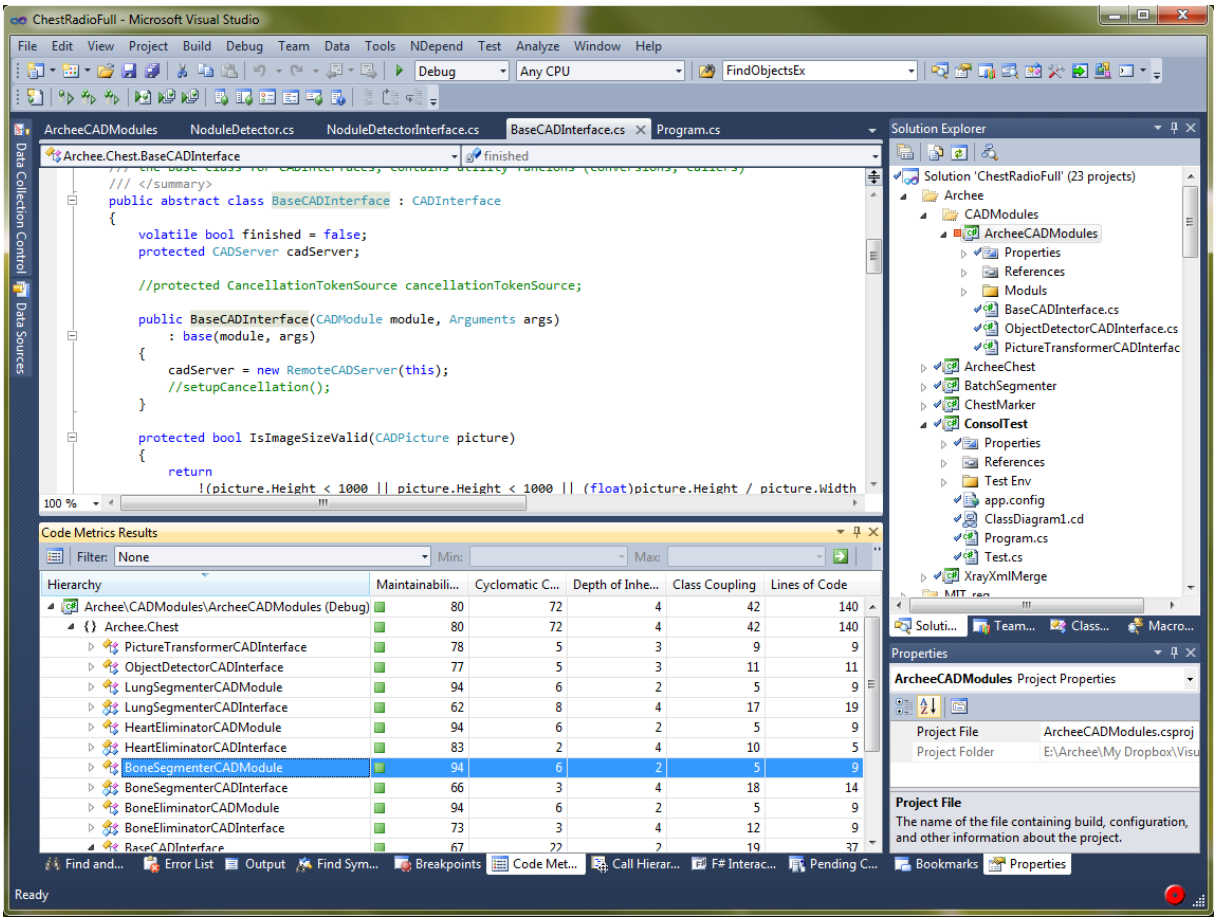
Kipróbált analizátorok

MS Visual Studio

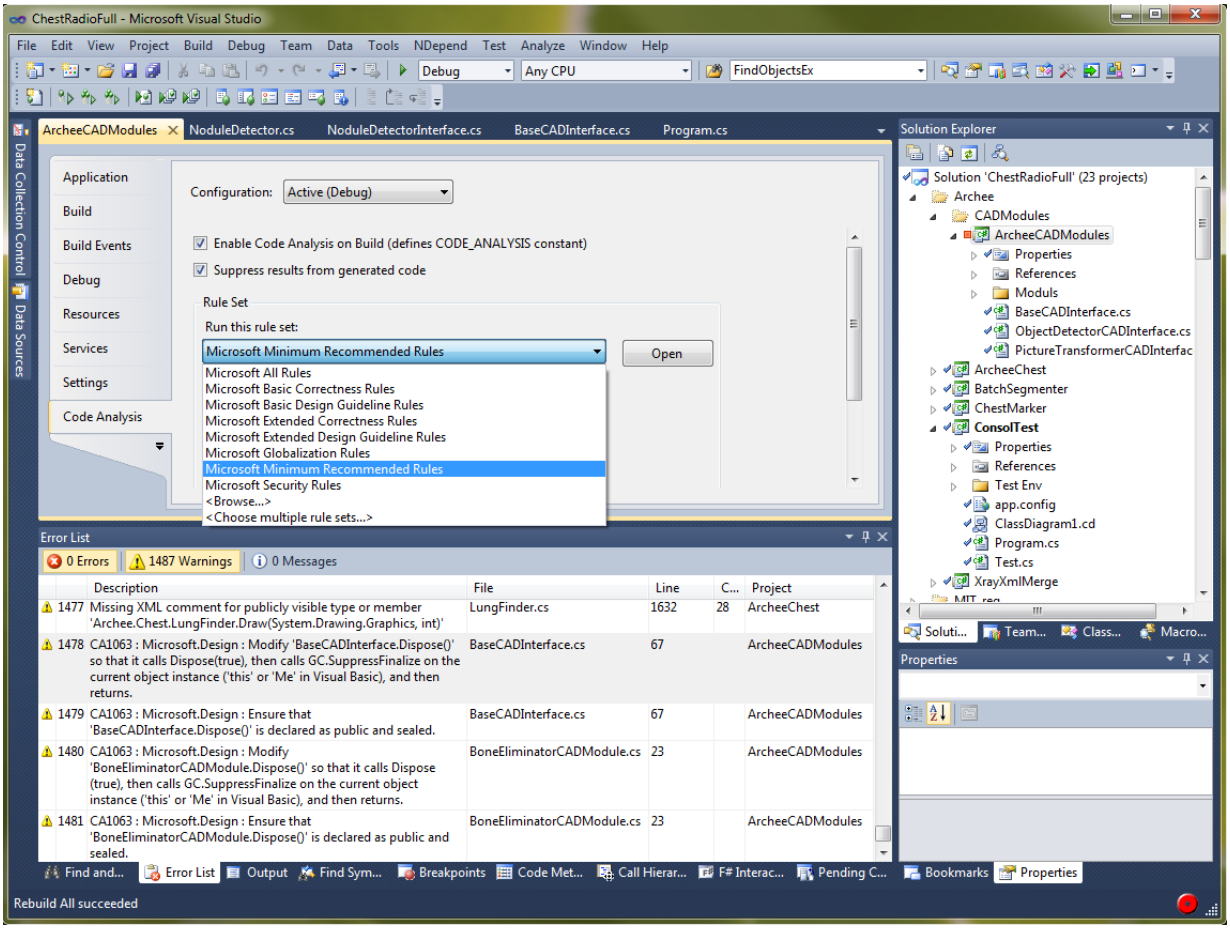
A 2008 Premium Editiontől felfelé be van építve a Code Metrics számítás. Csak a legfőbb mennyiségeket méri:

- Maintainability Index
- Cyclomatic Complexity
- Depth of Inheritance
- Class Coupling
- Lines of Code

Nagyon kényelmes, mivel teljesen integrálva van a fejlesztőkörnyezetbe és a Microsofttól elvárhatóan még sose volt vele problémám.



Ha a Code Analysis-t futtatjuk, akkor szabályellenőrzést végez a Visual Studio. Rendelkezésre állnak előre definiált szabály halmazok, és mi is kiválaszthatjuk hogy nekünk pontosan mik kellenek. Az eredményeket a fordítási hibák ablakában írja ki a Warningok közé, kattintásra odavisz a problémás részre.

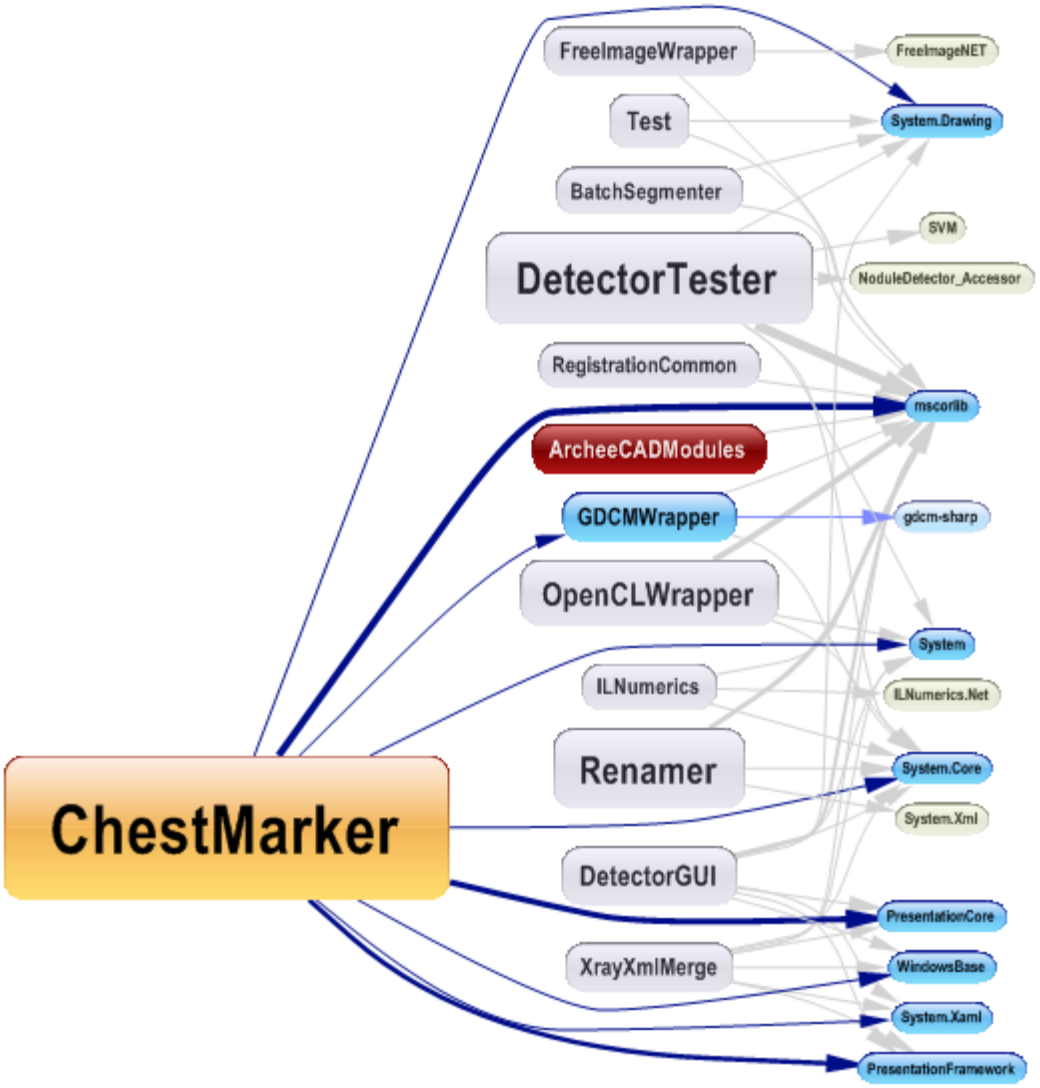


NDepend

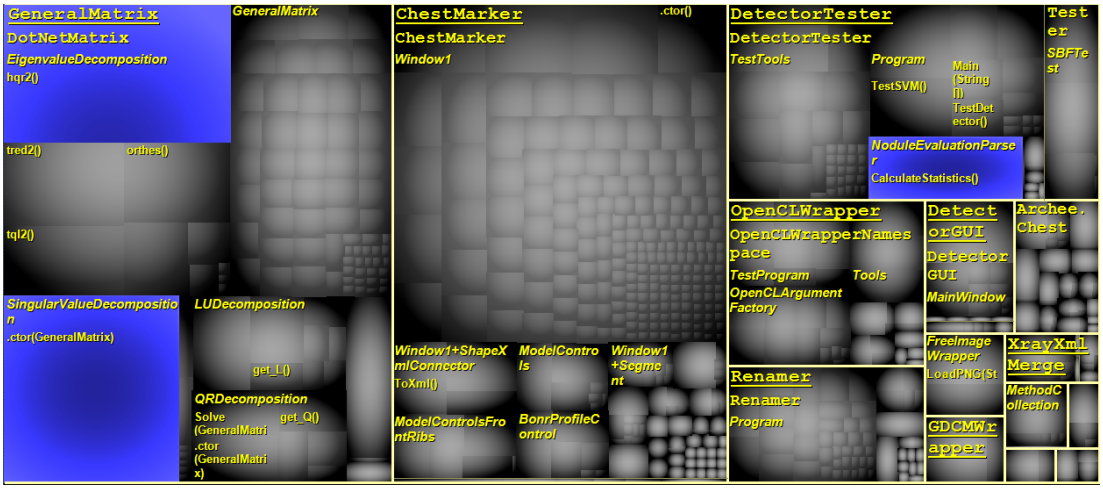
Ennek az analizátornak van egy fizetős változata, vagy pedig Open Source, Academic Licence vagy 2 hét demo az ami ingyen jár, de mindegyik gyakorlatilag ugyanazokat a korlátozásokat tartalmazza.

Hasonlít a Visual Studio-val járó kód analizátorhoz, ez is beépül a fejlesztő környezetbe, viszont természetesen nem annyira az eredeti. Például nagyon belassítja az ablak átméretezését és az ablakainak az elhelyezésivel is vannak problémák, gyakorlatilag az egész GUI elég rosszul van megírva. Cserébe viszont sokkal gazdagabb a funkcionalitása.

Az NDepend a lefordított DLL-eket analizálja, így az elemzést nem a forráskódon, hanem a köztes nyelven (IL) végzi. Be lehet állítani, hogy mely assambly-ket szeretnénk bevenni az analízisbe, majd le lehet futtatni. Az egyik leglátványosabb eszköze a Dependecy Graph, mely az egyes assambly-k közötti függőségeket ábrázolja. Ki lehet választani, hogy az egyes élek vastagságai a típus hivatkozások, vagy metódushivatkozások, stb.. alapján változzanak. Az assambly-ket jelölő téglalapok mérete is függővé tehető a kódsorok mennyisége vagy a ciklikus komplexitás vagy egyéb mennyiségektől.



Egy másik szemléletes diagram típus a Metrics, amely egy Tree Map, egyszerre ábrázolja az osztályokat és azok tagjait. A tagok kiválasztott kódmetrikájától függ az őket jelölő téglalap mérete. Ez a megjelenítés típus a gyors navigálást segíti, mivel egyszerre minden ábrázolva van. Egy adott téglalap fölé állva az egérrel megjeleníti a mért mennyiségeket. Dupla kattintásra a forráskódhoz navigálna, de ez csak a fizetős verzióban érhető el.



A mért mennyiségek:

- IL Instructions
- Lines of Code
- Lines of Comment
- Percentage Comment
- Cyclomatic Complexity
- IL Cyclomatic Complexity
- IL Nesting Depth
- Overloads
- Parameters
- Variables
- Method Level
- Efferent Coupling
- Afferent Coupling
- Method Rank

Fontos eszköze az analizátornak a Code Query Language (**CQL**), mely egy SQL-hez hasonlító lekérdező nyelv, melyben a kóddal kapcsolatos lekérdezéseket lehet írni. Például ki lehet listáztatni a kritikus metódusokat, melyek ciklomatikus komplexitása nagyobb mint 40 és a beágyazottsági mélysége nagyobb mint 4:

```
WARN IF Count > 0 IN SELECT METHODS WHERE
  ILCyclomaticComplexity > 40 AND
  ILNestingDepth > 4
  ORDER BY ILCyclomaticComplexity DESC
```

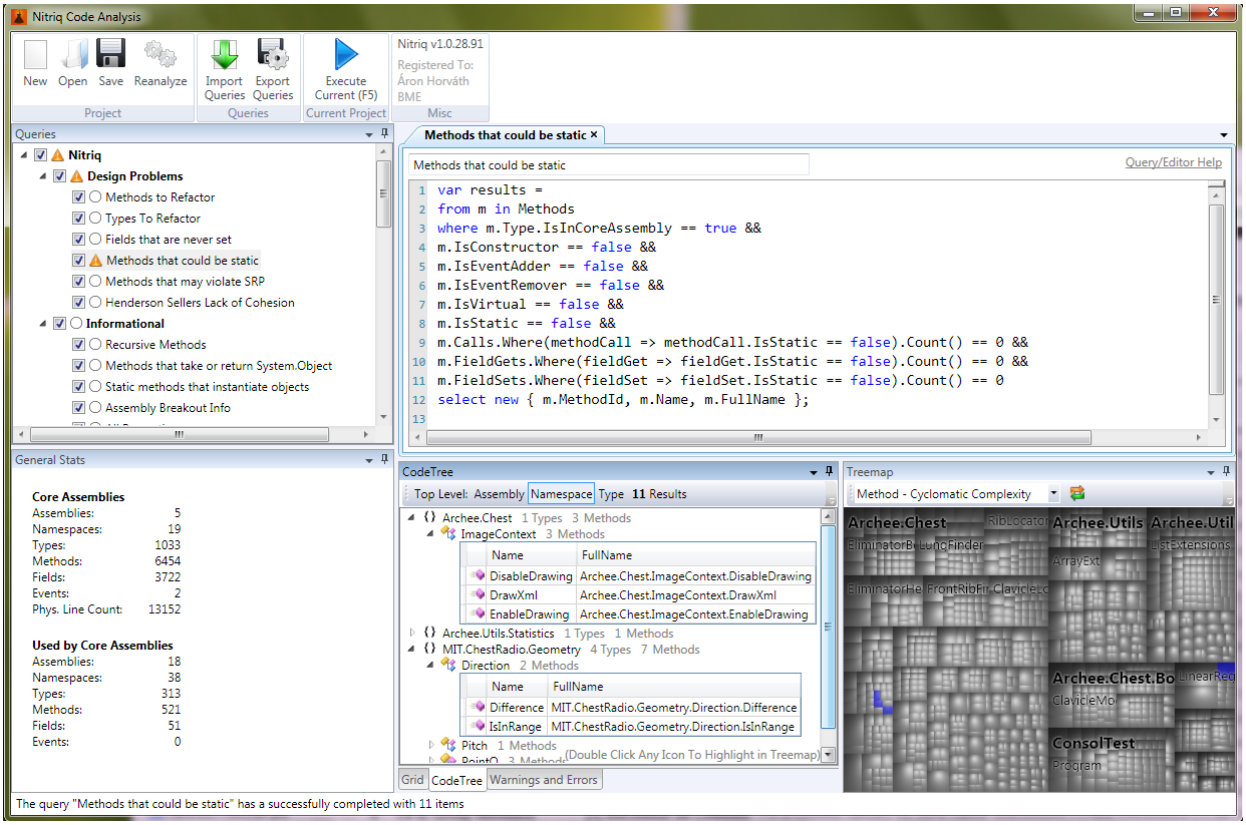
A fenti diagramon ennek eredményeit kékkel jelölte a rendszer. Az eredményül kapott metódusok tényleg nagyon bonyolult más programnyelvből automatikusan átfordított kódrészek.

Sajnos van egy komoly hibája. Sok assambly-t kizár a vizsgálatból, mert több helyen is megtalálja a DLL-jét és különböznek, ez akkor fordulhat elő, ha a különböző módokban (Debug, Release, x86, x64..) nincs mind újrafordítva az adott projekt. Ez gyakorlatilag lehetetlenné tette az érdemi kód metrika mérést, mert csak néhány projektre működött. Sajnos nem lehet neki

megadni egyesével a DLL-eket, mert a Solution Projectjeit veszi alapul.

Nitriq

A Nitriq egy VS-tól független letisztultnak tűnő kódanalizátor. Ez is IL alapon dolgozik, viszont előnye, hogy teljesen független a forráskódtól, elég csak az elemzendő DLL-eket kiválasztani, így az NDependnél meglévő problémám itt nem lépett fel.



Azábrán látható lekérdezés pl hasznosnak bizonyult, talált jópár olyan metódust, melyek lehetnének statikusak de nem azok.

Itt is lehet saját szabályokat definiálni. Nagy előny, hogy nem egy külön nyelvet kell megtanulni mint az NDependnél, hanem C# nyelven írhatunk lekérdezéseket. Ha a LINQ alnyelvet használjuk, akkor SQL-hez hasonló lekérdezéseket lehet készíteni. Ebben a lekérdezésben pl C#-os módon használnak Regex kifejezéseket:

```
//if a method name contains a conjunction like "And", "Or", or "Then",
//then it may be doing too many things and is violating the
//Single Responsibility Principle (SRP)
var results =
from m in Methods
where !m.IsPropertyGetter && !m.IsPropertySetter && m.IsInCoreAssembly &&
m.Name.Like(".*And[A-Z].*|.*Then[A-Z].*|.*Or[A-Z].*", false) //false => case sensitive
select new { m.MethodId, m.Name, m.FullName };
```

Warn(results, 0);

Az eredményeket itt is meg lehet jeleníteni egy Treemapen. Sajnos viszont nem lehet odanavigálni a forráskódhoz, mivel az nem áll rendelkezésre.

Ennek a programnak is vannak hibái. Az egyik assembly-mnél elszáll, így ezt sem tudom használni.

SourceMonitor

Ez egy nagyon fapados program. Több nyelvet is támogat: C++, C#, Java, Delphi, C, HTML, Visual Basic. Elég nehézkes a kezelőfelülete főleg az új projekt létrehozásánál, viszont könnyen ki lehet választani, hogy pontosan mely fileokat vegyen figyelembe. Csak kód metrikákat mér, de azok között vannak egész érdekesek, pl a Stmts/Method.

A teljes lista:

- Lines
- Statements
- % Comments
- % Docs,
- Classes
- Methods/Class
- Calls/Method
- Stmts/Method
- Max complexity
- Max Depth
- Avg Depht
- Avg Complexity

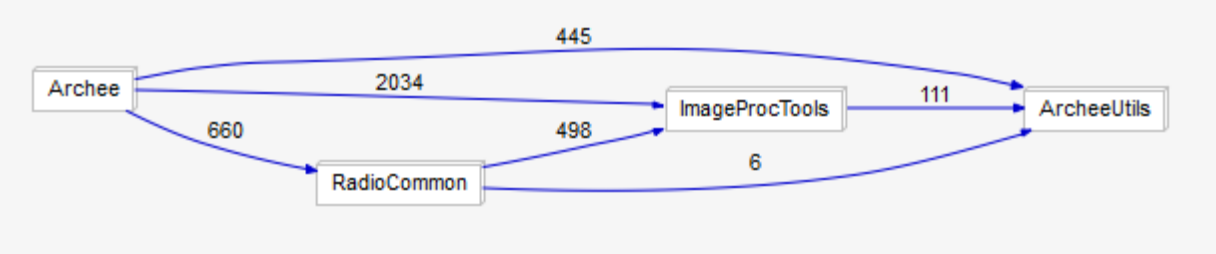
Understand

Ez is egy több nyelvet támogató program (Ada, C/C++, C#, FORTRAN, Java, JOVIAL, Pascal, PL/M, VHDL, Cobol, PHP, HTML, CSS, JavaScipt, Python), így a programkódot, és nem az IL-kódot elemzi. Ez fizetős program, viszont 14 napos teljes funkcionalitású kipróbálási verziója elérhető.

Nem sikerült túl sokat kihoznom ebből az analizátorból. Az egyes assambly-khez ki tud írni alap dolgokat:

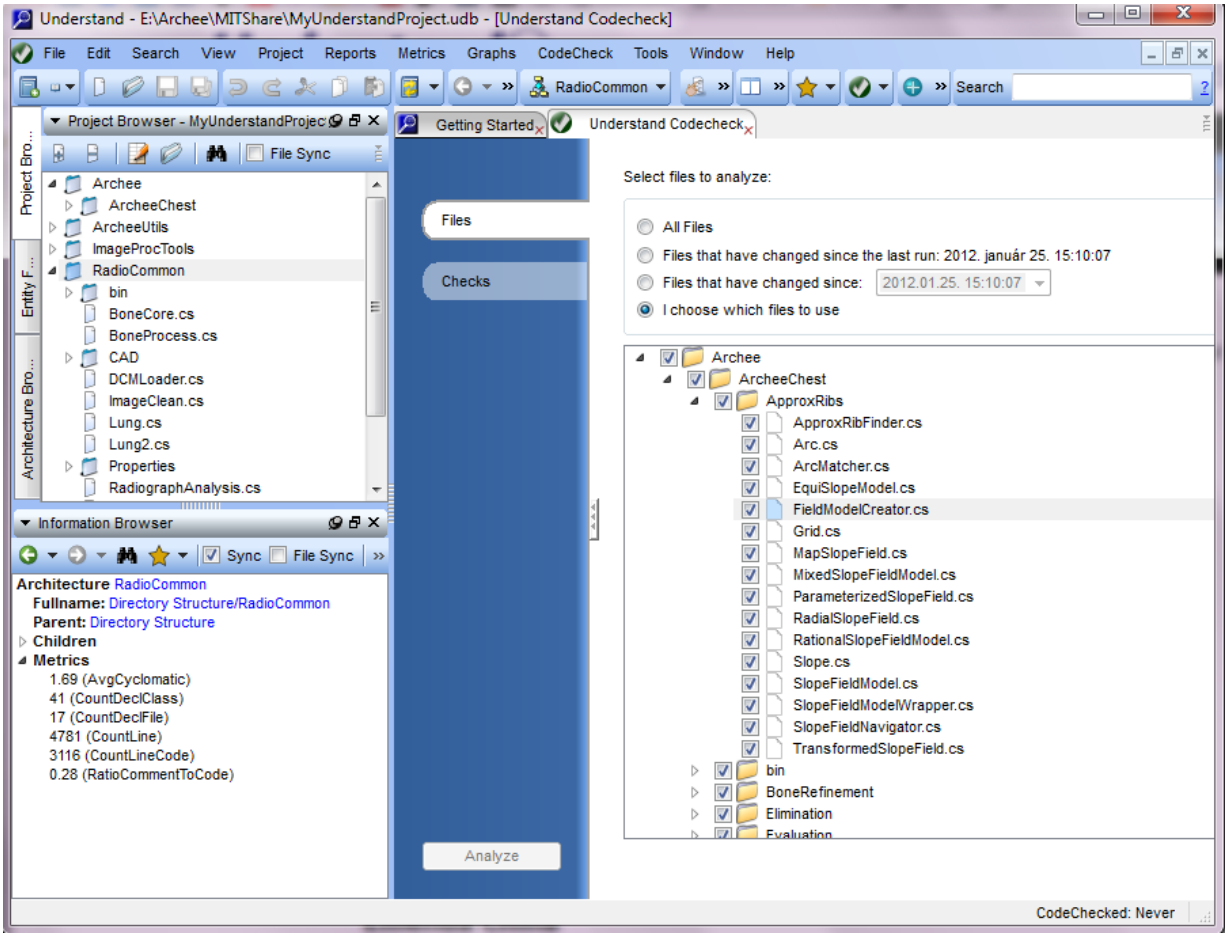
- AvgCyclomatic
- CountDeclClasa
- CountDeclFile
- CountLine
- CountLineCode
- RatioCommentToCode

Az egyik fő probléma, hogy könyvtár struktúra alapján akarja kezelni a hierarchiákat, így csak Projektek közti függőségeket tudtam vele kirajzoltatni pl:

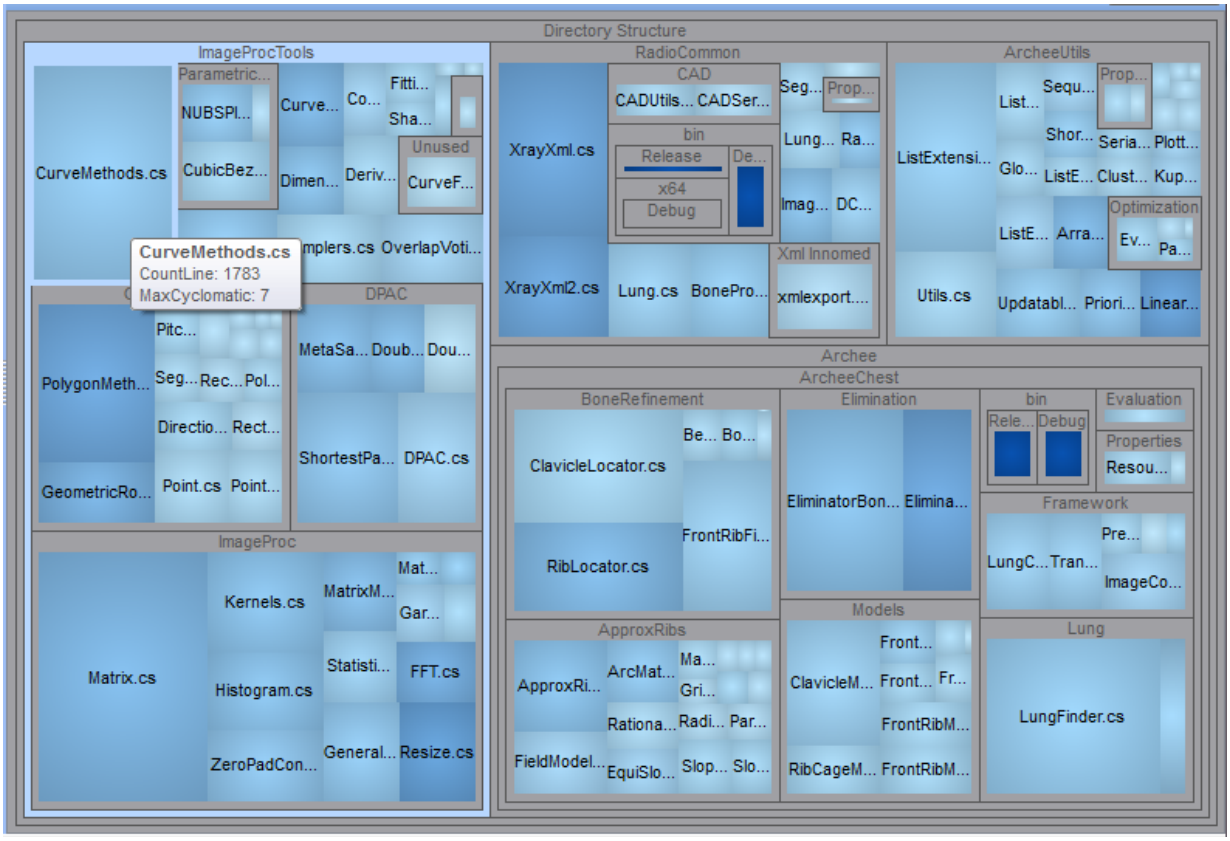


Itt is állítható, hogy az egyes nyilak mit jelentsenek.

Osztályokról nem sok mindent tudtam meg. Ki tud rajzolni egy osztálydiagramot, de az viszont elég nehezen értelmezhető, mert 1335 x 37425 pixeles lett.



Ez a program is tudja ábrázolni Treemapen a metrikákat:



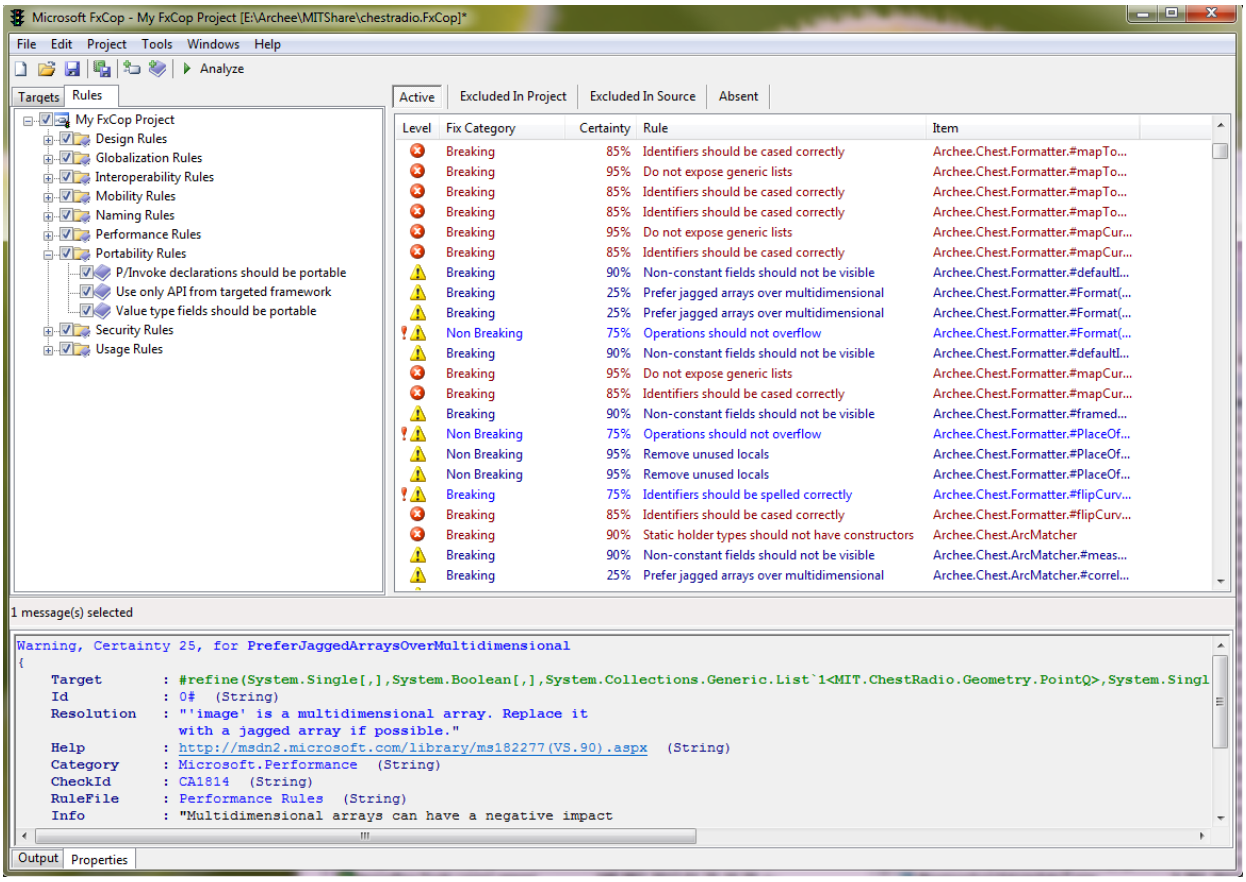
Habár ez az eszköz tűnt a Microsoftosokon kívül a legmegbízhatóbbnak, nem igazán sikerült olyan funkciót találnom amit használnék belőle.

Microsoft FxCop, StyleCop

Ezek mind a Microsoft ingyenes termékei.

Az FxCop részben beépült a Visual Studio újabb verzióiba Code Analytics menüpontként, de külön is elérhető. IL alapon történik az ellenörzés, és egy előre definiált szabályok halmazából lehet válogatni. Elég jól átlátható program, a különböző súlyosságú hibákat jól megkülönbözteti. Sajnos a forráskódhoz itt sem lehet navigálni.

A StyleCop nyílt forráskódú és a VisualStudióba pluginként lehet telepíteni. A forráskódot elemzni, pl olyan hibákat keres, hogy az összeadás jel két oldalán nincs szóköz.



Felhasznált irodalom

Maintainability Index:
http://www.projectcodemeter.com/cost_estimation/help/GL_maintainability.htm

Halsted Complexity Measures:
http://en.wikipedia.org/wiki/Halstead_complexity_measures

Kohézió:
http://en.wikipedia.org/wiki/Cohesion_(computer_science)

Coupling:
http://en.wikipedia.org/wiki/Coupling_(computer_programming)

Affernt / Efferent Coupling:
http://codebetter.com/patricksmacchia/2008/02/15/code-metrics-on-coupling-dead-code-design-flaws-and-re-engineering/

Class Coupling:
http://blogs.msdn.com/b/zainnab/archive/2011/05/25/code-metrics-class-coupling.aspx

Cyclomatic Complexity

http://en.wikipedia.org/wiki/Cyclomatic_complexity

Depth of Inheritance:
http://blogs.msdn.com/b/zainnab/archive/2011/05/19/code-metrics-depth-of-inheritance-dit.aspx

[S2010]
Chidamber, S. R. & Kemerer, C. F. (1994). *A Metrics Suite for Object Oriented Design* (IEEE Transactions on Software Engineering, Vol. 20, No. 6). Retrieved May 14, 2011, from the University of Pittsburgh web site: http://www.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf

FxCop:
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=6544
http://en.wikipedia.org/wiki/FxCop

StyleCop:
http://en.wikipedia.org/wiki/StyleCop