



Inspection of a system without actually running the code.

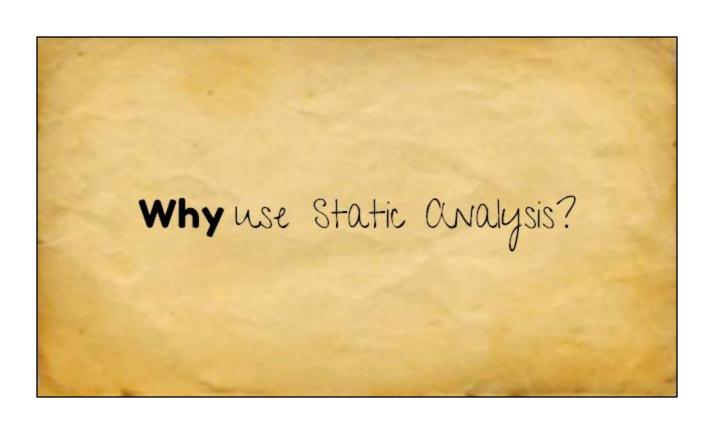




Image Credit: 401(K) 2012 @ Flickr http://www.flickr.com/photos/68751915@N05/6355811869/

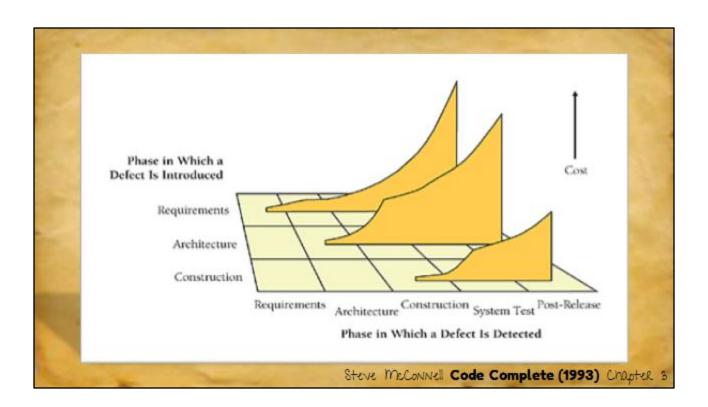


If you can easily eliminate boring bugs, you get to spend you time in more interesting ways.

Like writing new bugs

Image Credit: SuperFantastic @ Flicker

http://www.flickr.com/photos/superfantastic/1547089906/



Removal Step	Lowest Rate	Modal Rate	Highest Rate
Informal design Reviews	25%	35%	40%
Formal design inspections	45%	55%	U5%
Informal code Reviews	2.0%	2.5%	35%
Formal code inspections	45%	U0%	70%
Modeling or prototyping	35%	65%	<i>8</i> 0%
Personal desk-checking of code	2.0%	40%	U0%
unit test	15%	30%	50%
New function (component) test	2.0%	30%	35%
Integration test	2.5%	35%	40%
Regression test	1524	2.5%	30%
System test	2.5%	40%	55%



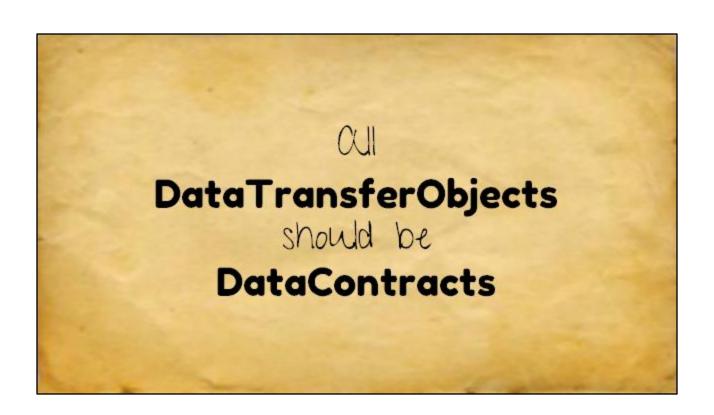
"The upshot is that defect—detection methods work better in combination than they do singly."

Steve MICCONNEIl Code Complete (1993) Crapter 20



Image Credit: bayasaa @ Flickr http://www.flickr.com/photos/bayasaa/2693171833/







# Should Never directly create a PersistentEntity With New

```
var badMethodQuery
= from type in facadeAssembly.GetTypes()
    from method in type.GetMethods(allMethods)
    let inspector = new MethodInspector(method, false)
    from calledMethod in inspector.AllCalledMethods
    where calledMethod.IsConstructor
    where IsPersistentEntity(calledMethod.DeclaringType)
    select new {
        DeclaringType = type,
        Method = method,
        ConstructedType = calledMethod.DeclaringType
    };
```

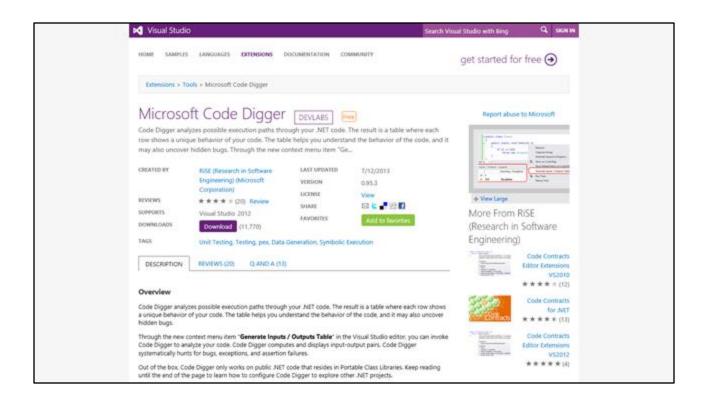




CamelCase.ToDashedName(camelCase)
StringSequenceExtensions.JoinWith(sequence, separator, lastSeparator)

Show how the columns are sortable – but that this is useless

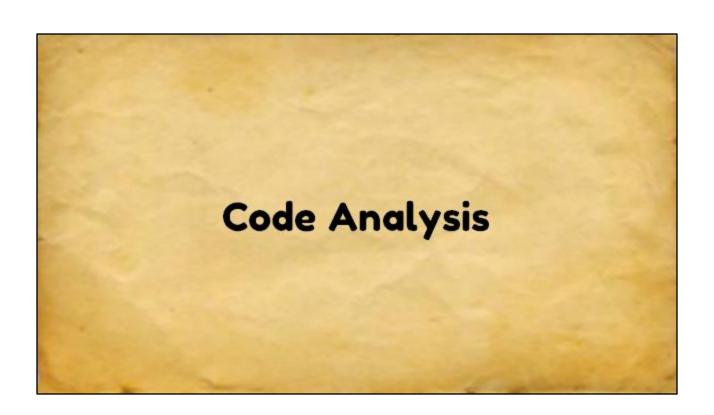


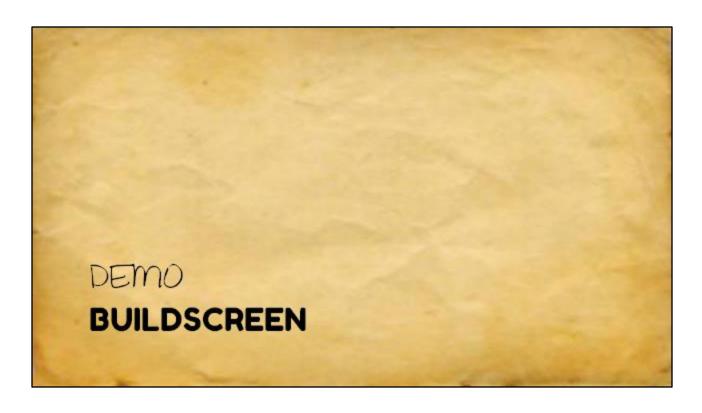




CamelCase.ToDashedName(camelCase)
StringSequenceExtensions.JoinWith(sequence, separator, lastSeparator)

Show how the columns are sortable – but that this is useless





Application that I use at work, though with modifications
This version unchanged from the latest version on CodePlex, except to add in Code
Analysis

CA1001 – implement IDisposable to ensure the background worker is cleaned up properly

CA1008 – no defined value for '0', so any variables initialised with illegal value

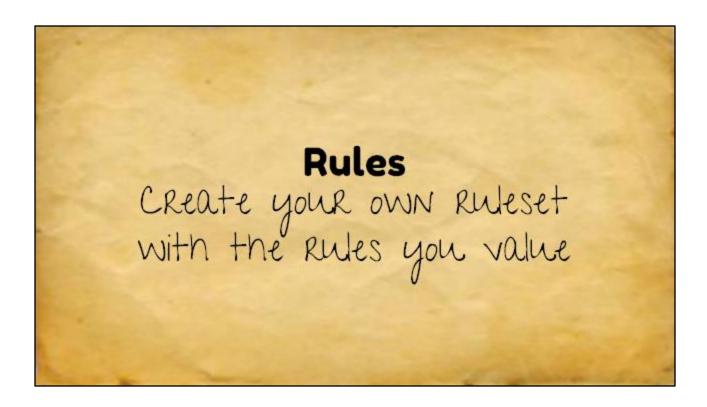
CA1704 – misspelled namespace

CA2001 – a naïve plugin framework is vulnerable, might not load the expected assembly

CA2235 – member isn't serializable, need to ensure it's not included in serialization

Show how to suppress a warning

Can you find a list of suppressed messages?



Here are some interesting rules ... use the ones of value to you ...

### CA1006: Do not nest generic types in member signatures

Dates back to the original introduction of generics, but of little use to teams that are thoroughly comfortable with their use.

Would flag useful things like Lazy<List<string>>

### CA1062 Validate arguments of public methods

http://esmithy.net/2011/03/15/suppressing-ca1062/ http://geekswithblogs.net/terje/archive/2010/10/14/making-static-code-analysisand-code-contracts-work-together-or.aspx

## CA1704 Identifiers should be spelled correctly

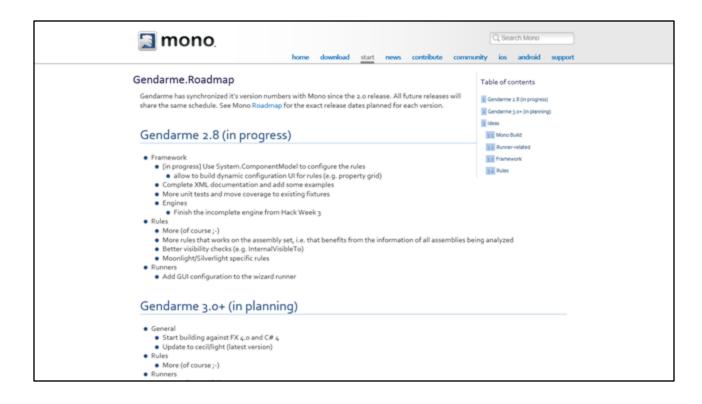
Detail adding a new dictionary file with new terms Share it across multiple projects

# CA2000 Dispose objects before losing scope

# CA2210 Assemblies should have valid strong names

Do you use strong names? If so, this is a useful reminder; if not, disable it.









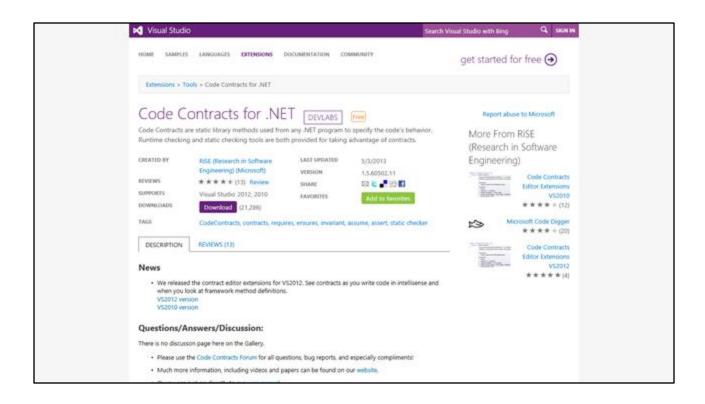
Preconditions that have to hold for the routine to funtion PostConditions that will hold afterwards Invariants that are always true

Code Contracts make those explicit



Invented a programming language – Eiffel – that includes Code Contracts as a first class concept

Image credit:





StringExtensions.Before()

Pin error list open

Build Solution (Analysis configuration) – note longer build time, 'tis doing a lot of work

Show how contracts for interfaces

Unit Tests
Code Metrics
Code Digger
Code Analysis
Gendarme
Code Contracts



