

Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

Manuel Naujoks

Agenda

1. Anforderungen
2. Usus für Java
3. Technologie Evaluierung
4. Usus.NET Visual Studio-Erweiterung
5. Clean Code Unterstützung
6. andrena-Softwarequalitätsindex
7. Evaluation
8. Zusammenfassung
9. Ausblick

Zentrale Begriffe

Metrik

- *Metrik* ist eine Eigenschaft oder der Wert dieser Eigenschaft und wird für Methoden, Typen oder Namespaces bestimmt.

Statische Code-Analyse

- *Statische Code-Analyse* bezeichnet die Analyse eines Softwareprogramms, ohne dass dieses ausgeführt werden muss. Als Ergebnis wird ein Bericht, beispielsweise über Metriken, erstellt.

Clean Code

- *Clean Code* bezeichnet strukturierteren, wartbareren und verständlicheren Quellcode. „Clean Code“ ist der Titel von Robert C. Martin's Buch.

Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

ANFORDERUNGEN

Anforderungen / Aufgabe

- Visual Studio-Erweiterung (Usus für Eclipse als Vorlage)
- direktes Feedback anhand von Softwaremetriken
- Softwareentwickler unterstützen bei Clean Code-Entwicklung

- Heuristiken der Histogramme erkennen
- Softwarequalitätsindex (SQI) berechnen
- Evaluierung von Beispielaufgabe (andrena-Kurs)

Anforderungen / Ziele

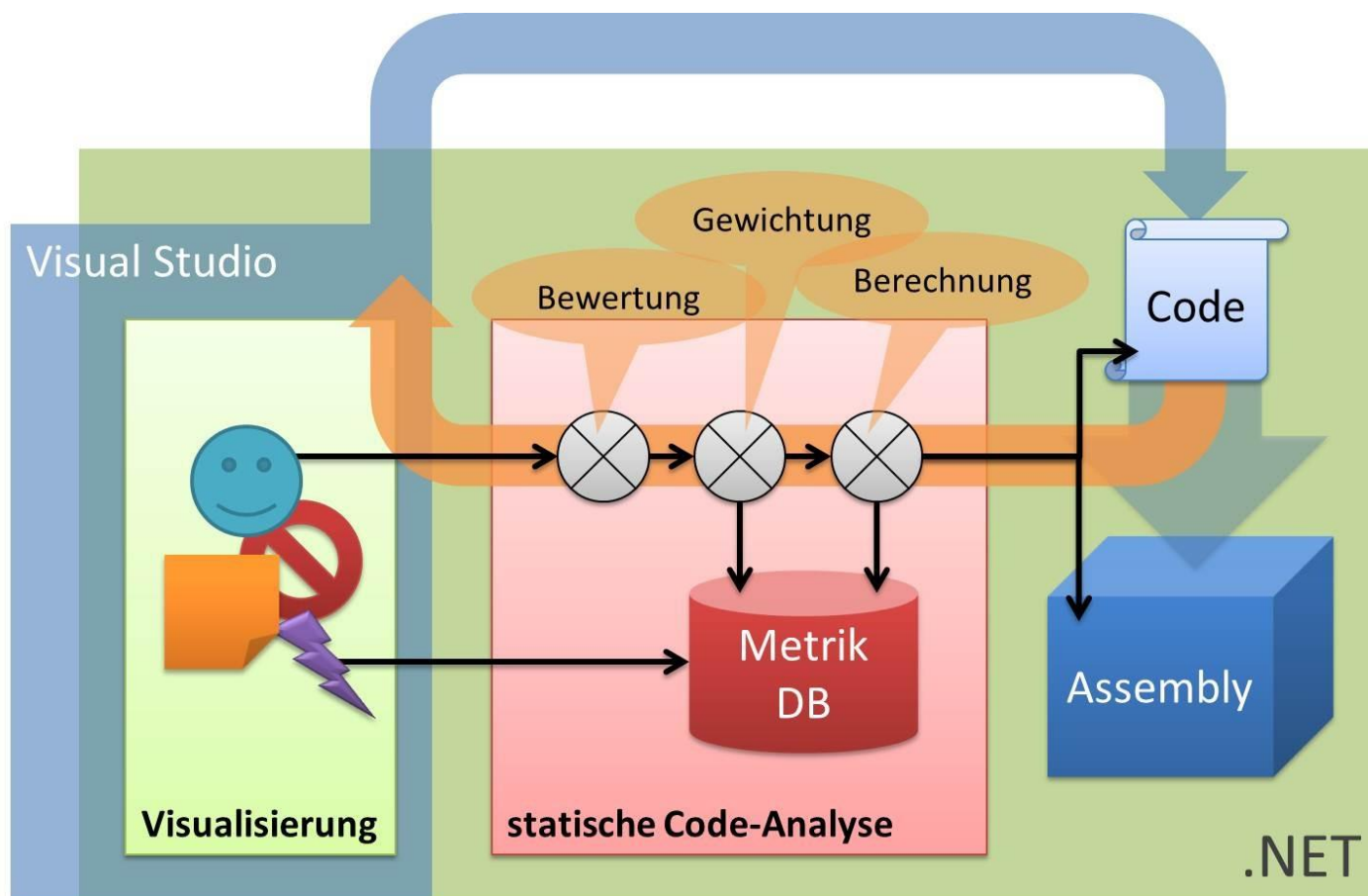
Einsicht in die
Codebasis

Erkennen von
Problemfällen

Förderung von
Clean Code

Interpretation
der
Softwarequalität

Anforderungen / Architektur



Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

USUS FÜR JAVA

Usus für Java / Allgemein

- Usus (lat. „das, was üblich ist“)
- Eclipse Plugin
 - Installation über Software Sites
 - <http://projectusus.googlecode.com/svn/updates/>
- Analyse pro Speichervorgang

Usus für Java / Fenster

Usus Cockpit
Snapshot taken at 07.03.12 10:26 (less than a minute ago)

Indicator	Avg...	H...	Total	T...
<input type="checkbox"/> Code proportions				
Average componen	100.0	0	1 classes	
Class size	0.0	0	1 classes	
Cyclomatic compl	0.0	0	1 methods	
Method length	11.1	1	1 methods	⊖
Number of non-sta	0.0	0	1 classes	
Packages with cycl	0.0	0	1 packages	

Method length: Hotspots are methods with more than 9 statements. Rating fu

Value	Name	Path	Trend
24	ArtikelpreisKalkulator...	\BadPractice_J\src\misc	0
13	DogHandlerFuerBew...	\BadPractice_J\src\coding	0
12	DogHandlerTest.train...	\BadPractice_J\src\coding\tests	0
10	DogHandlerTest.train...	\BadPractice_J\src\coding\tests	0

void main(String[])

- Code proportions
- Class size: 1
- Cumulative Component Dependency (of class): 1
- Cyclomatic complexity: 1
- Method length: 2

Project Usus Infos

Count

Method length

Package Graph

Layout: Tree (vertical)

ArtikelpreisTest

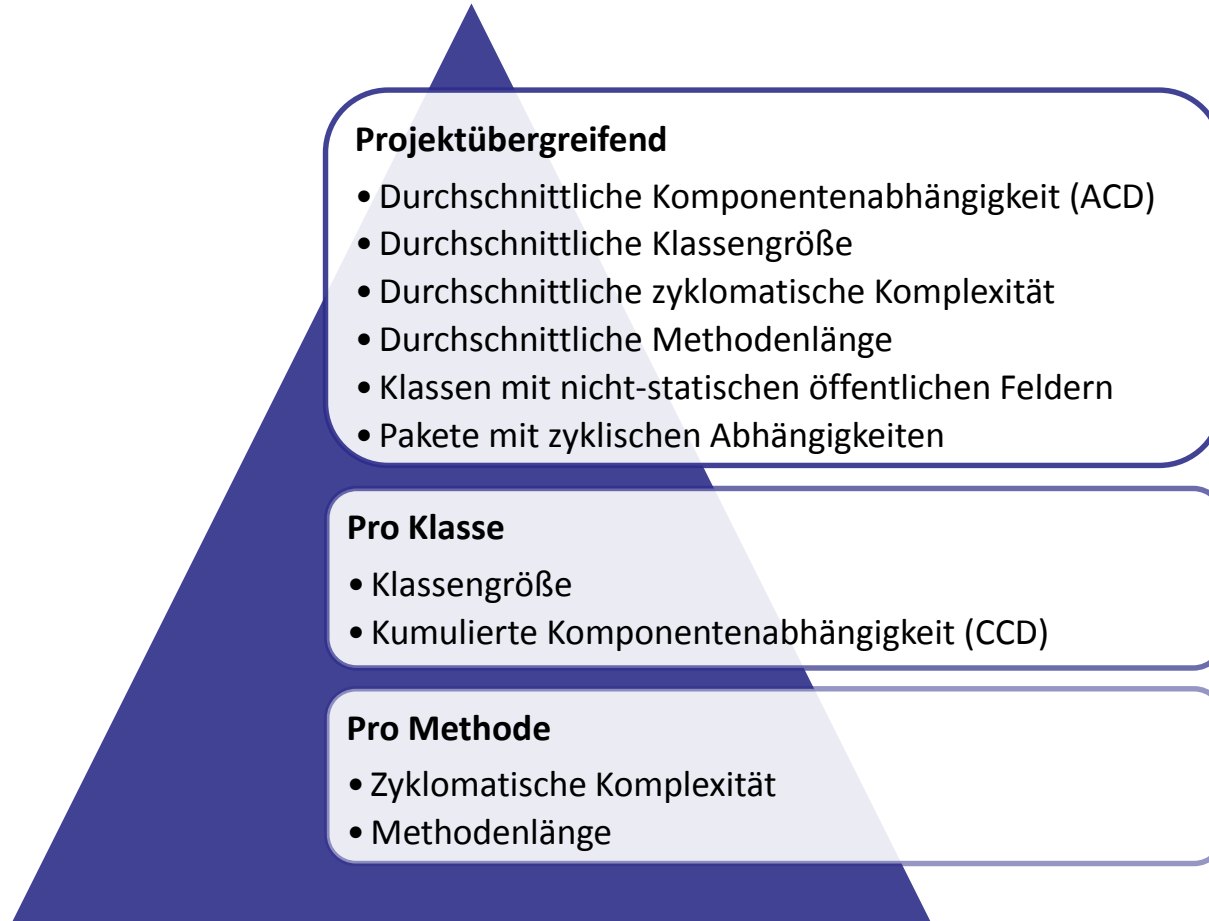
ArtikelPreisLoader

IArtikelPreisLoader

Artikelpreis

Loader

Usus für Java / Metriken

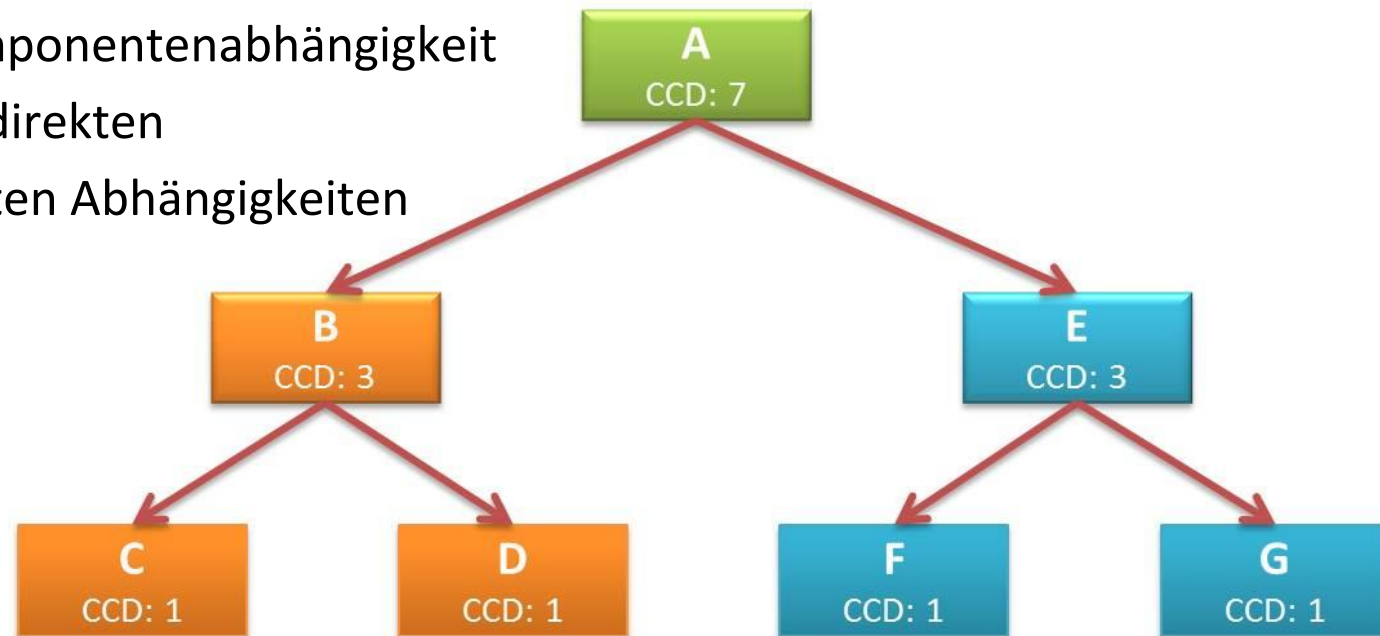


Usus für Java / Metriken / Pro Methode

- Zyklomatische Komplexität
 - Anzahl der unabhängigen Möglichkeiten eine Methode zu durchlaufen
 - Anzahl aller entscheidungstreffenden Stellen in der Methode
- Methodenlänge
 - Anzahl der Anweisungen

Usus für Java / Metriken / Pro Klasse

- Klassengröße
 - Anzahl der Instanzmethoden,
 - der Klassenmethoden
 - und der Konstruktoren
- Kumulierte Komponentenabhängigkeit
 - Anzahl der direkten
 - und indirekten Abhängigkeiten



Usus für Java / Metriken / Projektübergreifend

- Durchschnittliche Komponentenabhängigkeit
 - Durchschnitt der kumulierten Komponentenabhängigkeiten aller Klassen
- Durchschnittliche Klassengröße
- Durchschnittliche zyklomatische Komplexität
- Durchschnittliche Methodenlänge
- Klassen mit nicht-statischen öffentlichen Feldern
- Pakete mit zyklischen Abhängigkeiten

Usus für Java / Hotspots

- Kumulierte Komponentenabhängigkeit
 - Schwellwertfunktion anhand einer Menge an Klassen

$$L_{ccd}(Cs) = \frac{1,5}{2(\log_5 |Cs|)}$$

- Klassengröße (Schwellwert **12**)
- Zyklomatische Komplexität (Schwellwert **4**)
- Methodenlänge (Schwellwert **9**)
- Klassen mit nicht-statischen öffentlichen Feldern (Schwellwert **0**)
- Pakete mit zyklischen Abhängigkeiten (Größe des Kreis) (Schwellwert **1**)

Usus für Java / Zusammenfassung

Einsicht in die
Codebasis

Erkennen von
Problemfällen

Förderung von
Clean Code

Interpretation
der
Softwarequalität

Usus.NET

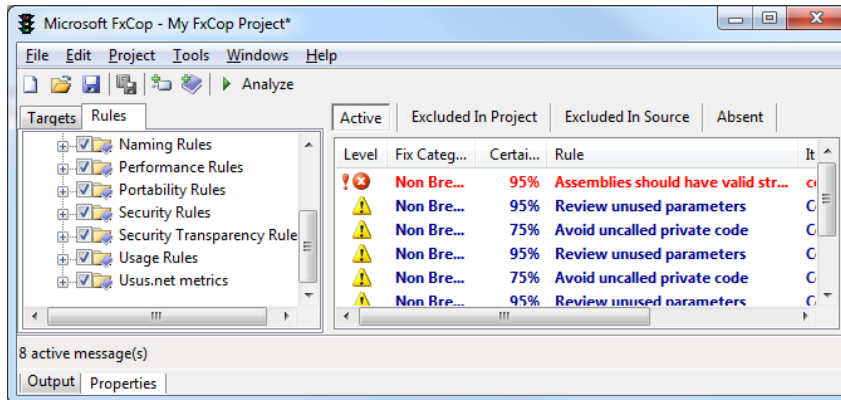
Visual Studio-Erweiterung zur statischen Code-Analyse

TECHNOLOGIE EVALUIERUNG

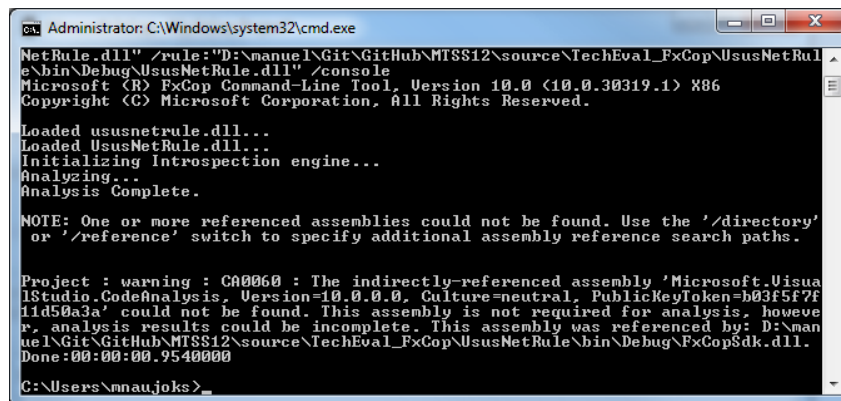
Technologie Evaluierung / Kriterien

- Metrik-Informationen bestimmbar?
- Verfügbar?
- Kostenfrei?
- Einfach einsetzbar?
- Für alle .NET-Versionen?
- Unabhängige Komponente?
- Tauglich für unvollständige Software?
- Für C# und VB.NET?
- Für Assembly?
- Für Code?

Technologie Evaluierung / FxCop



- Assembly-Analyse
- FxCop nutzt Regeln
- Eigene Regel
- Ergebnisse weiterverarbeiten? ☹️



Technologie Evaluierung / Common Compiler Infrastructure

CCI Metadata

CCI Code and
AST
Components

- Assembly-Analyse
- von Microsoft
- Einfach 😊
- Direkt 😊
- FxCop nutzt CCI

Technologie Evaluierung / NRefactory

- Assembly-Analyse (mit Mono.Cecil)
- Quellcode-Analyse
- von IC#code
- Teil von SharpDevelop
- Aktuell nur für C#
- Semantische Analyse (Method Binding) erforderlich -> kompilieren ☹️

Technologie Evaluierung / Project Roslyn

- Quellcode-Analyse
- von Microsoft
- Neue .NET-Compiler APIs
- Semantische Analyse (Method Binding) erforderlich -> kompilieren ☹️
- Workspace API 😊
 - Syntaxbaum und semantische Analyse auf Solution-Ebene
 - wenn in Visual Studio alles automatisch
 - neue Möglichkeiten Erweiterung zu entwickeln
- Aktuell nur CTP (Community Technology Preview) ☹️

Technologie Evaluierung / Zusammenfassung

	FxCop	CCI	NRefactory	Project Roslyn
Metrik-Informationen bestimmbar?	✓	✓	!	!
Verfügbar?	✓	✓	✓	✗
Kostenfrei?	✓	✓	✓	✓
Einfach einsetzbar?	✗	✓	✓	✓
Für alle .NET Versionen?	✓	✓	✓	✓
Unabhängige Komponente?	✗	✓	✓	✗
Tauglich für unvollständige Software?	✓	✓	!	!
Für C# und VB.net?	✓	✓	✗	✓
Für Assembly?	✓	✓	✓	✗
Für Code?	✗	✗	✓	✓

Ja ✓
Eingeschränkt !
Nein ✗

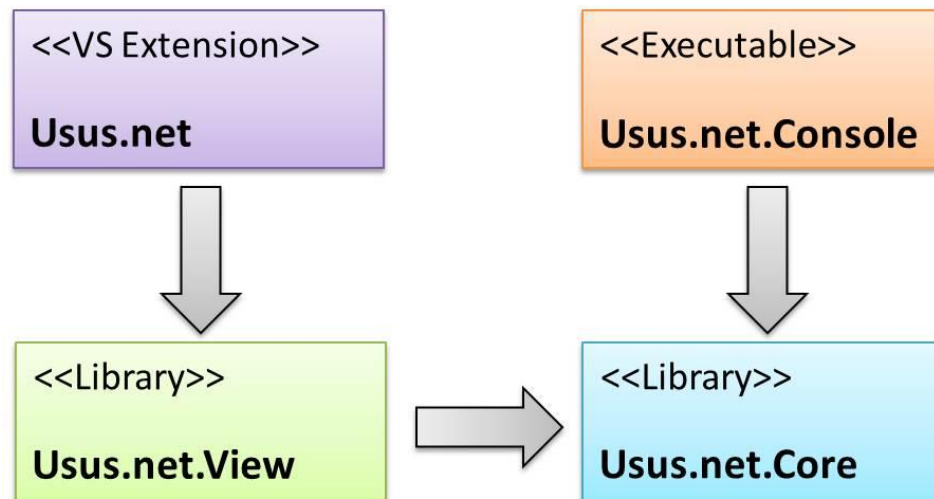
Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

USUS.NET VISUAL STUDIO- ERWEITERUNG

Usus.NET Visual Studio-Erweiterung / Allgemein

- Addin oder VSIX-Erweiterung?
- Assembly-Analyse mit CCI nach jedem Kompiliervorgang



Usus.NET Visual Studio-Erweiterung / Core

```
//var metrics = Analyze.PortableExecutables(assemblyToAnalyze);  
var metrics = Analyze.Me();  
foreach (var method in metrics.Methods)  
{  
    Console.WriteLine("Signature: " + method.Signature);  
    Console.WriteLine("CC: " + method.CyclomaticComplexity);  
}
```

**CODE
DEMO**

Usus.NET Visual Studio-Erweiterung / Fenster

The screenshot shows the Visual Studio interface with the Usus.NET extension menu open. The menu items are: Cockpit, Hotspots, Histograms, CleanCode, Method Info, and SQI. The code editor shows a snippet of C# code:

```

public void Main(string[] args)
{
    //This is a comment
    //and this is the next line
    System.Console.WriteLine("ddfdsd");
    System.ArithmeticException a = new
    bool c1 = true;
    bool c2 = true;
    bool c3 = true;
}
    
```

Usus.NET Info

Name	Main
Length	16
Cyclomatic Complexity	11
(Class) Size	8
(Class) Non-static public fields	0
(Class) Direct Dependencies	1
(Class) Cumulative Component Dependency	1

Usus.NET Cockpit

Metric	Average	Total	Hotspots	Distribution
Average Component Dependency	4,8%	255 types	32	0,08
Class Size	5,2%	255 types	20	0,21
Cyclomatic Complexity	1,4%	1003 methods	14	0,71
Method Length	0,8%	1003 methods	8	0,39
Non-Static Public Fields	3,1%	255 types	8	1,00
Namespaces with Cycles	6,5%	46 namespaces	3	0,88

3598 Relevant Lines Of Code (11:32)

Usus.NET Visual Studio-Erweiterung / Fenster

Usus.NET Hotspots

Classes with more than 12 methods.

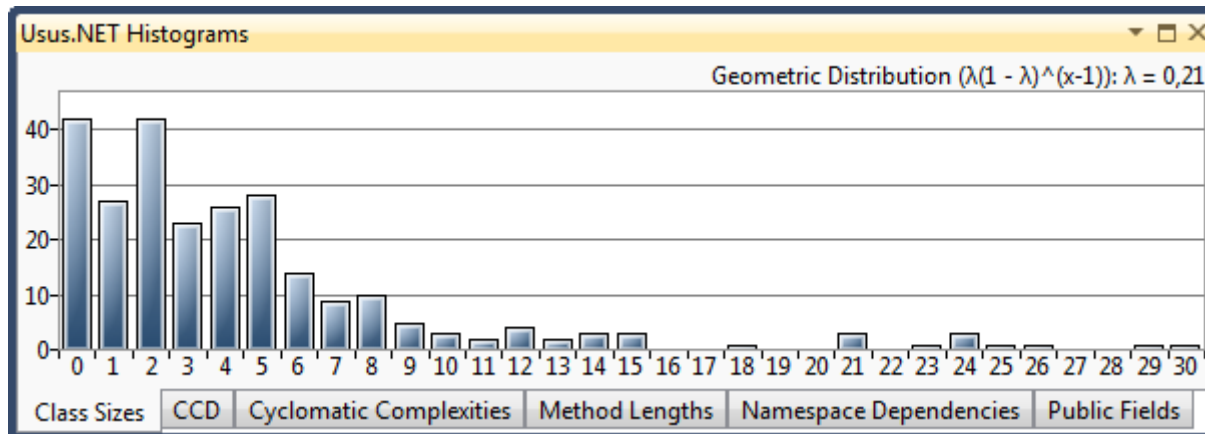
Size	Class	Fullname
30	RatedMetrics	andrena.Usus.net.Core.Hotspots.RatedMetrics
29	SolutionEventsAwareToolWindowPane	andrena.Usus.net.ExtensionHelper.SolutionEventsAwareToolWindowPane
26	CyclomaticComplexities	Usus.net.Core.IntegrationTests.MethodMetrics.CyclomaticComplexities
25	TypeDependencies	Usus.net.Core.IntegrationTests.MethodMetrics.TypeDependencies
24	HotspotsCollection	andrena.Usus.net.View.ViewModels.Hotspots.HotspotsCollection
24	MetricsHotspots	andrena.Usus.net.Core.Hotspots.MetricsHotspots

Class Sizes CCD Cyclomatic Complexities Method Lengths Cyclic Namespaces Public Fields

Namespace cycle

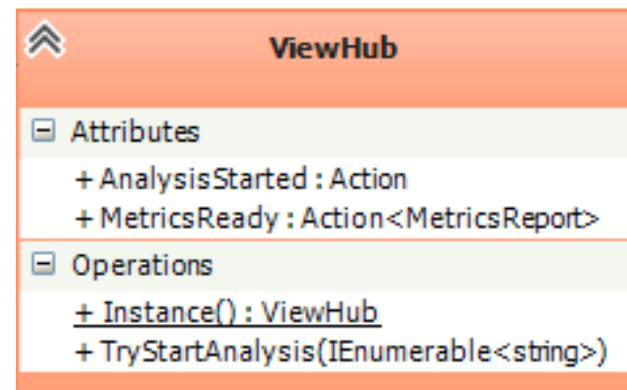
all namespaces in cycle	classes that reference classes in other namespaces
Usus.net.Core.IntegrationTests.NamespaceMetrics.Test4	Usus.net.Core.IntegrationTests.NamespaceMetrics.Test6.Class6
Usus.net.Core.IntegrationTests.NamespaceMetrics.Test5	Usus.net.Core.IntegrationTests.NamespaceMetrics.Test4.Class4 (.ctor)
Usus.net.Core.IntegrationTests.NamespaceMetrics.Test6	

Usus.NET Visual Studio-Erweiterung / Fenster



Usus.NET Visual Studio-Erweiterung / Integration

- Visual Studio-Kontext ermitteln (in einer Fensteroberklasse)
 - `var dt2 = base.GetService(typeof(SDTE)) as EnvDTE80.DTE2;`
- Oberfläche dem Visual Studio-Fenster zuweisen
 - `base.Content = ViewFactory.CreateCockpit(ViewHub.Instance);`
- Auf Events reagieren und Analyse starten
 - `BuildSuccessfull += files => ViewHub.Instance.TryStartAnalysis(files);`



Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

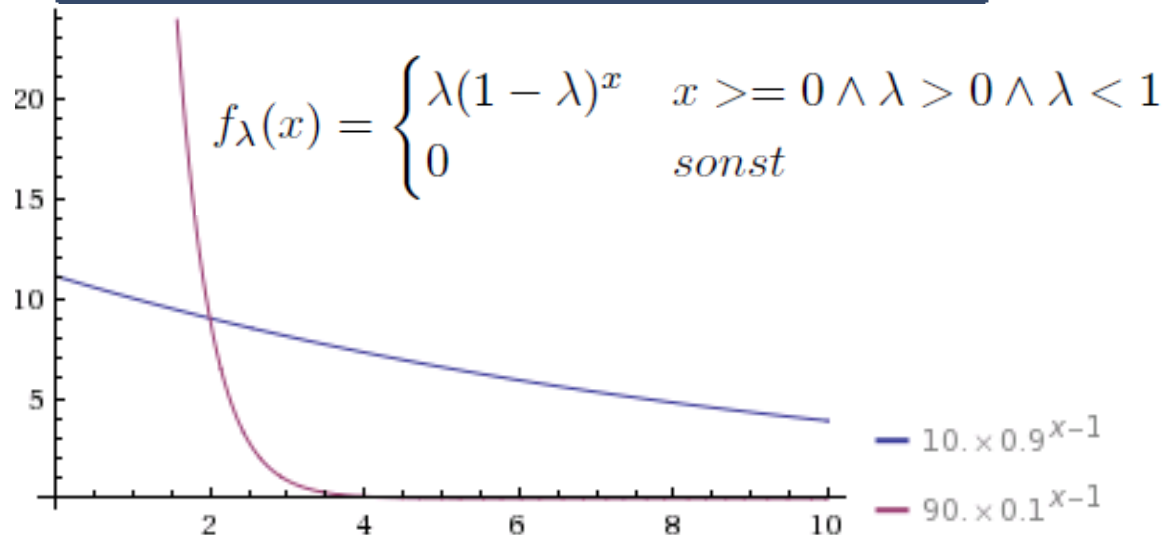
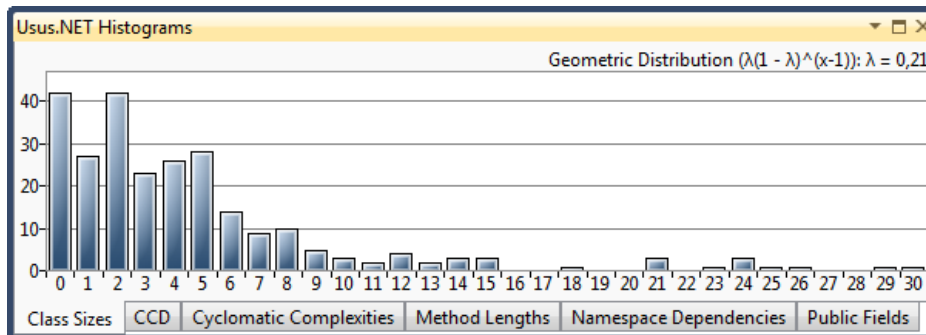
CLEAN CODE UNTERSTÜTZUNG

Clean Code Unterstützung / Bekannte Metriken

- CRAP
 - Kombination von Testabdeckung und zyklomatischer Komplexität
- The Braithwaite Correlation
 - nutzt logarithmische Skalen, lineare Regression und Pareto-Verteilung
- Neue Metrik?
 - Bedeutung von CRAP oder Braithwaite Correlation?
 - Bezug zu Clean Code? Nicht offensichtlich ☹
 - Grundlegende Clean Code-Metriken -> kleine Metriken
 - Bsp. Funktionen: „Small!“ („Clean Code“ Seite 34)
 - **Lambda der geometrischen Verteilung**

Clean Code Unterstützung / Metrik-Histogramm

- Approximation der geometrischen Verteilung



Je größer das λ , desto eher entsprechen die Werte der betrachteten Metrik dem Clean Code-Paradigma.

Clean Code Unterstützung / Metrik-Histogramm

- Annäherung mit der Maximum Likelihood-Methode

- Schätzer

$$\lambda = \frac{n}{\sum_{i=1}^n x_i} = \frac{1}{\bar{x}}$$

- Reziproker Mittelwert aller Werte des Histogramms

- **Lambda der geometrischen Verteilung**

- Neue Metrik mit direktem Bezug zu Clean Code
- Mehr kleinere Metriken, größeres Lambda
 - Bsp.: kleine Methodenlängen von Robert C. Martin explizit befürwortet!
- Veränderungen des λ klassifizieren Refactorings
 - λ -verbessernde Refactorings fördern Clean Code

Je größer das λ , desto eher entsprechen die Werte der betrachteten Metrik dem Clean Code-Paradigma.

Clean Code Unterstützung / Zusammenfassung

Einsicht in die
Codebasis

Erkennen von
Problemfällen

Förderung von
Clean Code

Interpretation
der
Softwarequalität

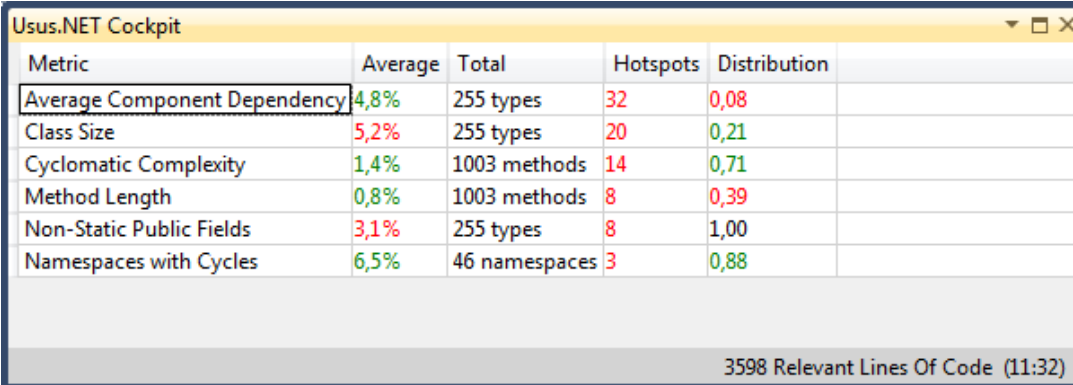
Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

ANDRENA- SOFTWAREQUALITÄTSINDEX

andrena-Softwarequalitätsindex / Allgemein

- Werte des Usus.NET Cockpit erfordern viel manuelle Interpretation



The screenshot shows a window titled "Usus.NET Cockpit" containing a table with the following data:

Metric	Average	Total	Hotspots	Distribution
Average Component Dependency	4,8%	255 types	32	0,08
Class Size	5,2%	255 types	20	0,21
Cyclomatic Complexity	1,4%	1003 methods	14	0,71
Method Length	0,8%	1003 methods	8	0,39
Non-Static Public Fields	3,1%	255 types	8	1,00
Namespaces with Cycles	6,5%	46 namespaces	3	0,88

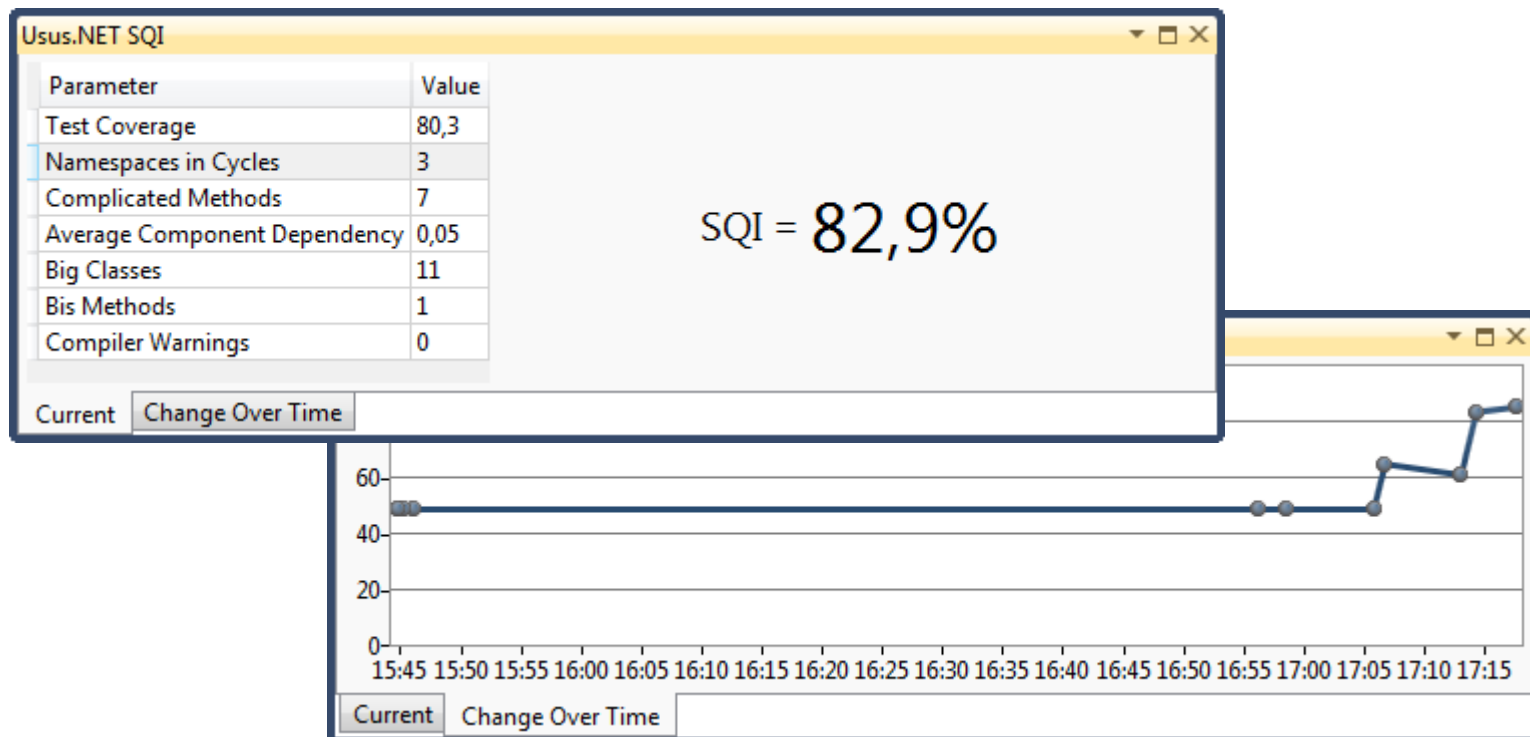
At the bottom right of the window, it states: 3598 Relevant Lines Of Code (11:32)

- Einschätzungen, Gewichtungen und Vergleiche basieren auf Erfahrung
 - Automatisierbar?

andrena-Softwarequalitätsindex / Allgemein

- Ein einziger transparenter Wert erleichtert Interpretation
 - andrena-**Softwarequalitätsindex** von Dr. Eberhard Kuhn
- Aktuelle Bestimmung
 - Software kompilieren
 - Externes Tool für statische Code-Analyse starten (NDepend)
 - Ergebnisbericht (Metriken) aufbereiten und in **Isis** importieren
- **Isis** verwaltet und visualisiert die **Softwarequalitätsindizes**
- Zu viele Context Switches erforderlich ☹️

andrena-Softwarequalitätsindex / Fenster



andrena-Softwarequalitätsindex / Berechnung

- Testabdeckung in Prozent
- Anzahl der Namespaces in Zyklen
- Anzahl der komplizierten Methoden
- Durchschnittliche Komponentenabhängigkeit in Prozent
- Anzahl der großen Klassen
- Anzahl der großen Große Methoden
- Anzahl der Compiler-Warnungen

andrena-Softwarequalitätsindex / Berechnung

- Gewichtete Softwarequalitätsniveaus für jeden Parameter m
 - (außer Testabdeckung)

$$SQNiveau(m) = \frac{100}{1,5 \left(\frac{RelativeGroesse(m)}{Zweidrittelkonstante(m)} \right)}$$

$$f_{mk}(m) = \frac{1}{1,5 \left(\frac{MittlereGroesse(m)}{Zweidrittelkonstante_{f_{mk}}(m)} \right)^3}$$

$$SQNiveau_{f_{mk}}(m) = SQNiveau(m) \times f_{mk}(m)$$

- Summe aller gewichteten Softwarequalitätsniveaus

$$SQI = \sum_{m \in M} SQNiveau_{f_{mk}}(m) \times Gewicht(m) \times 0,1$$

andrena-Softwarequalitätsindex / Zusammenfassung

Einsicht in die
Codebasis

Erkennen von
Problemfällen

Förderung von
Clean Code

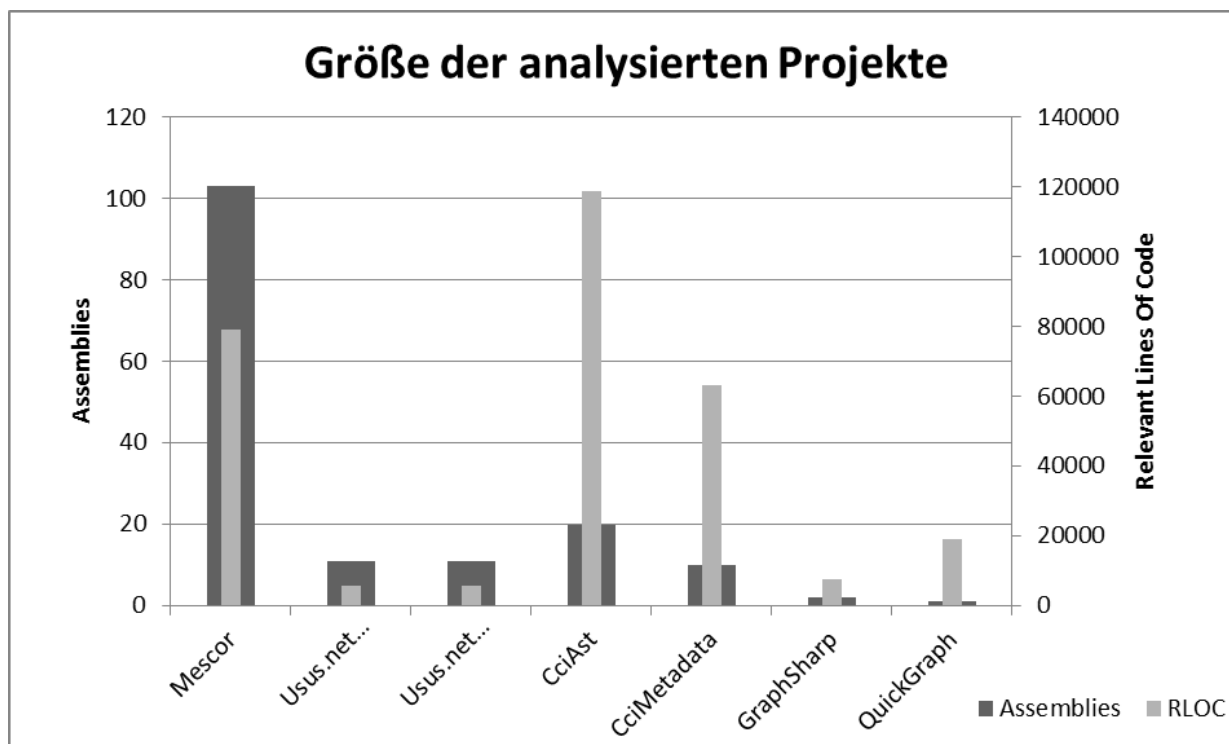
Interpretation
der
Softwarequalität

Usus.NET

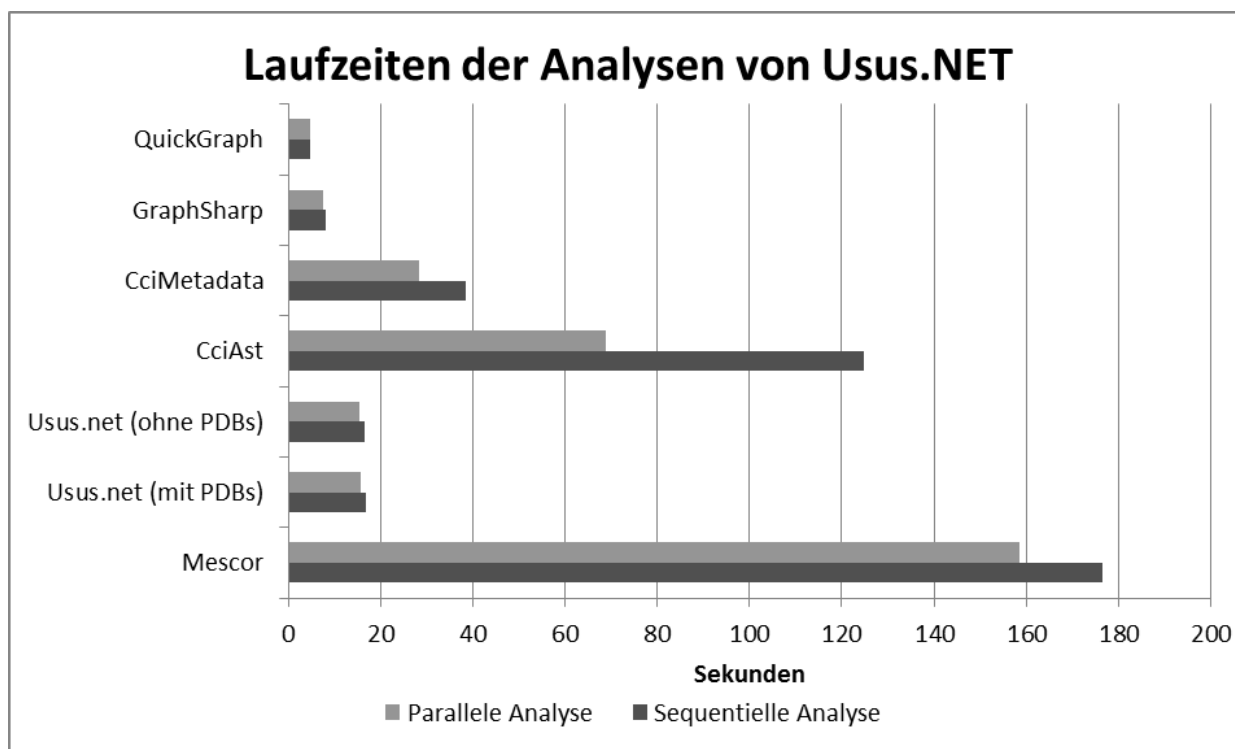
Visual Studio-Erweiterung zur statischen Code-Analyse

EVALUATION

Evaluation / Laufzeiten



Evaluation / Laufzeiten



Evaluation / Refactoring-Übung aus ASE-Kurs (vorher)

Usus.NET Cockpit

Metric	Average	Total	Hotspots	Distribution
Average Component Dependency	40,0%	5 types	0	0,50
Class Size	0,0%	5 types	0	0,20
Cyclomatic Complexity	5,0%	16 methods	1	0,62
Method Length	8,5%	16 methods	1	0,19
Non-Static Public Fields	0,0%	5 types	0	1,00
Namespaces with Cycles	0,0%	2 namespaces	0	1,00

96 Relevant Lines Of Code (11:25)

Usus.NET Hotspots

Methods with more than 4 different execution paths.

Complexity	Method	Signature
7	GetMailMessage	RefactoringCodingDojo.C

III

Usus.NET SQI

Parameter	Value
Test Coverage	100
Namespaces in Cycles	0
Complicated Methods	1
Average Component Dependency	0,40
Big Classes	0
Bis Methods	1
Compiler Warnings	0

SQI = 82,2%

Usus.NET Histograms

Geometric Distribution $(\lambda(1 - \lambda)^{(x-1)})$; $\lambda = 0,19$

Evaluation / Refactoring-Übung aus ASE-Kurs (nachher)

Usus.NET Cockpit

Metric	Average	Total	Hotspots	Distribution
Average Component Dependency	32,8%	8 types	2	0,38
Class Size	0,0%	8 types	0	0,19
Cyclomatic Complexity	0,0%	35 methods	0	0,89
Method Length	0,0%	35 methods	0	0,32
Non-Static Public Fields	0,0%	8 types	0	1,00
Namespaces with Cycles	0,0%	2 namespaces	0	1,00

130 Relevant Lines Of Code (11:27)

Usus.NET Hotspots

Classes with more than 4 cumulated dependencies.

Dependencies	Class	Fullname
6	MailComposerTest	RefactoringCodingE
5	MailComposer	RefactoringCodingE

III

Usus.NET SQI

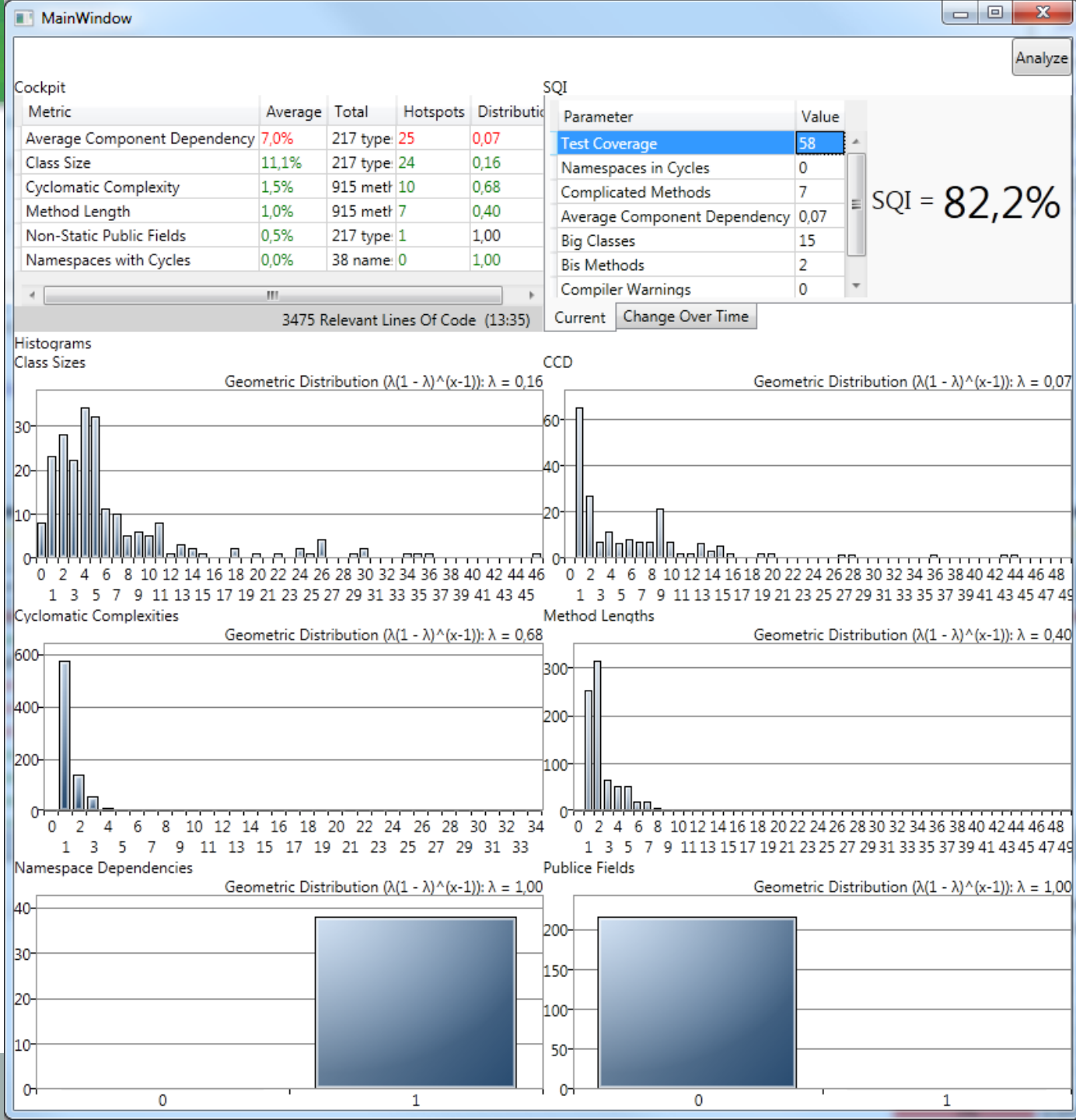
Parameter	Value
Test Coverage	99
Namespaces in Cycles	0
Complicated Methods	0
Average Component Dependency	0,33
Big Classes	0
Bis Methods	0
Compiler Warnings	0

SQI = 99,6%

Usus.NET Histograms

Geometric Distribution ($\lambda(1 - \lambda)^{(x-1)}$): $\lambda = 0,32$

Evaluation / Usus.NET



Evaluation / Zusammenfassung

Einsicht in die
Codebasis

Erkennen von
Problemfällen

Förderung von
Clean Code

Interpretation
der
Softwarequalität

Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

ZUSAMMENFASSUNG

Zusammenfassung

- **Usus.NET** - Visual Studio-Erweiterung zur statischen Code-Analyse
- Berechnung aller Softwaremetriken von Usus für Java pro Build
- Fast alle Funktionen von Usus für Java
- Clean Code-Unterstützung
- Berechnung des andrena-Softwarequalitätsindex
- Evaluierung der Refactoring-Übung des ASE-Kurs

Usus.NET

Visual Studio-Erweiterung zur statischen Code-Analyse

AUSBLICK

Ausblick

- ~~Visualisierung des Klassen- und Namespace-Graph~~
- Kleinsten Zyklus in einem Namespace-Zyklus finden
- Testabdeckung automatisch bestimmen (SQI, CRAP)
- Mehr Verteilungen (Pareto (The Braithwaite Correlation), Poisson)
- ~~Refactoring-Vorschläge~~
- ~~Automatisierte Refactorings~~

- Usus.NET als Grundlage weitere Trends zu implementieren/auszuprobieren

Realistischer Ausblick

- Visual Studio 2013 Portierung
- Analysis-Engine für Evaluierung auf Roslyn umbauen
- Klassen- und Methoden-Metriken besser visualisieren
- Ein- und Ausschalten der Analyse, ohne VS neuzustarten

</präsentation>