Static and Runtime API Usage Analysis on .NET

Rufus Linke

18. August 2011
Outline

1. Introduction
2. Methodology
3. Results
4. Conclusion and Further Work
Introduction
Introduction

Overview

Analysis of

- the shape of existing APIs.
- their usage by "projects in the wild."

In terms of

- reusability of APIs.
- OO-specific usage.
- general acceptance and coverage.
Introduction

Overview

Analysis of

- the shape of existing APIs.
- their usage by "projects in the wild."

In terms of

- reusability of APIs.
- OO-specific usage.
- general acceptance and coverage.
Introduction

Overview

Analysis of

- the shape of existing APIs.
- their usage by "projects in the wild."

In terms of

- reusability of APIs.
  - OO-specific usage.
  - general acceptance and coverage.
Introduction

Overview

Analysis of

- the shape of existing APIs.
- their usage by "projects in the wild."

In terms of

- reusability of APIs.
- OO-specific usage.
- general acceptance and coverage.
Introduction
Overview

Analysis of
- the shape of existing APIs.
- their usage by "projects in the wild."

In terms of
- reusability of APIs.
- OO-specific usage.
- general acceptance and coverage.
Introduction
Motivation

- provide understanding of existing API design
- support research on API migration
- facilitate development of new APIs
- show up refactoring options
Introduction
Motivation

- provide understanding of existing API design
- support research on API migration
- facilitate development of new APIs
- show up refactoring options
Introduction

Motivation

- provide understanding of existing API design
- support research on API migration
- facilitate development of new APIs
- show up refactoring options
Introduction

Motivation

- provide understanding of existing API design
- support research on API migration
- facilitate development of new APIs
- show up refactoring options
Introduction

Contributions

- combined static and dynamic analysis of API design and usage
- application of analysis to a corpus of selected open-source software
- composition of metrics that are suitable to measure API related software usage
Introduction

Contributions

- combined static and dynamic analysis of API design and usage
- application of analysis to a corpus of selected open-source software
- composition of metrics that are suitable to measure API related software usage
Introduction

Contributions

- combined static and dynamic analysis of API design and usage
- application of analysis to a corpus of selected open-source software
- composition of metrics that are suitable to measure API related software usage
Methodology
Explorative analysis approach:

1. framework design analysis
Explorative analysis approach:

1. framework design analysis
2. framework usage analysis
Methodology

Analysis Overview
Methodology

Corpus

![Graph showing the relationship between number of IL instructions and number of calls to external methods for various libraries and frameworks in .NET. The x-axis represents the number of IL instructions, ranging from 0 to 350,000, while the y-axis represents the number of calls to external methods, ranging from 0 to 30,000. The graph includes points for libraries such as Rhino.Mocks, Moq, Prism, xUnit, MEF, log4net, Unity, Windsor, Json.NET, MonoRail, NUnit, Lucene.Net, NAnt, Spring.NET, NHibernate, and ActiveRecord. Each library is represented by a symbol on the graph, with the number of IL instructions and corresponding number of calls to external methods.]
API

APIs are defined by their namespace. Namespaces are grouped for easier handling.

Example

The XML API is covered by the namespaces

- System.Xml
- System.Xml.Schema
- System.Xml.Serialization
- ...

All these namespaces are grouped and represented as

- System.Xml.*
APIs are defined by their namespace. Namespaces are grouped for easier handling.

Example

The XML API is covered by the namespaces
- System.Xml
- System.Xml.Schema
- System.Xml.Serialization
- ...

All these namespaces are grouped and represented as
- System.Xml.*
Methodology

APIs

API Usage

The term “API usage” refers to “usage by the developer”.

Example

class SomeClass { public SomeClass() { } }

Implicitly calls System.Object::.ctor() ⇒ not API usage.
The term “API usage” refers to “usage by the developer”.

Example

```csharp
class SomeClass { public SomeClass() { } }
```

Implicitly calls `System.Object::.ctor()` ⇒ not API usage.
Results
General statistics

- namespaces: 74 (3 with only sealed types)
- number of classes: 10457 (4295 sealed classes)
- interfaces: 957
Corpus
Usage Metrics

Specialization of types from the .NET APIs

General statistics

- namespaces: 74 (3 with only sealed types)
  - 19 namespaces have types that were specialized (26.76%)
- number of classes: 10457 (4295 sealed classes)
- interfaces: 957
.NET types specialized by corpus
General statistics

- namespaces: 74 (3 with only sealed types)
  - 19 namespaces have types that were specialized (26.76%)
- number of classes: 10457 (4295 sealed classes)
  - 100 types are specialized (1.62%)
- interfaces: 957
Implemented interfaces

- System.IDisposable
- System.Collections.IEnumerable
- System.Collections.IEnumerator
- System.Collections.Generic.IEnumerable`1
- System.Collections.Generic.IEnumerator`1
- System.ICloneable
- System.Collections.ICollection
- System.Runtime.Serialization.ISerializable
- System.ComponentModel.INotifyPropertyChanged
- System.Collections.Generic.IEqualityComparer`1
- System.Collections.Generic.IDictionary`2
- System.Runtime.Serialization.IObjectReference
- System.ComponentModel.Design.IServiceContainer
- System.Collections.IList
- System.Collections.IComparer
- System.Collections.IDictionary
- System.Collections.Generic.ICollection`1
- System.Collections.Generic.IEnumerator`1
- System.Collections.Generic.IEnumerable`1
- System.Collections.IEnumerator
- System.Collections.IEnumerable
- System.IDisposable
- System.Runtime.Serialization.IDeserializationCallback
- System.IServiceProvider
- System.Collections.IList
- System.Collections.IComparer
- System.Collections.IDictionary
- System.IComparable
- System.Collections.IDictionaryEnumerator
- System.IEquatable`1
- System.Runtime.Serialization.ISerializable
- System.Collections.ICollection
- System.ICloneable
- System.Collections.Generic.IEnumerable`1
- System.Collections.Generic.IEnumerator`1
- System.Collections.Generic.IEnumerable`1
- System.Collections.IEnumerator
- System.Collections.IEnumerable
General statistics

- namespaces: 74 (3 with only sealed types)
  - 19 namespaces have types that were specialized (26.76%)
- number of classes: 10457 (4295 sealed classes)
  - 100 types are specialized (1.62%)
- interfaces: 957
  - 70 interfaces are implemented (7.31%)
A simple number generator in the form of an `IEnumerable` can be generated with the following code:

```csharp
IEnumerable<int> GetSomeNumbers(int num)
{
    Random r = new Random();
    for (int i = 0; i < num; i++)
    {
        yield return r.Next();
    }
}
```
Corpus
Usage Metrics

Implemented interfaces (without compiler generated)
# Abstractness of .NET namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>ConcreteTypes</th>
<th>AbstractTypes</th>
<th>Abstractness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility</td>
<td>1</td>
<td>6</td>
<td>0.86</td>
</tr>
<tr>
<td>System.Dynamic</td>
<td>4</td>
<td>16</td>
<td>0.80</td>
</tr>
<tr>
<td>System.AddIn.*</td>
<td>14</td>
<td>27</td>
<td>0.66</td>
</tr>
<tr>
<td>System.Collections</td>
<td>9</td>
<td>14</td>
<td>0.61</td>
</tr>
<tr>
<td>System.Runtime.InteropServices.*</td>
<td>68</td>
<td>80</td>
<td>0.54</td>
</tr>
<tr>
<td>Microsoft.VisualStudio.*</td>
<td>21</td>
<td>17</td>
<td>0.45</td>
</tr>
<tr>
<td>System.Collections.Generic</td>
<td>14</td>
<td>11</td>
<td>0.44</td>
</tr>
<tr>
<td>System.Collections.Concurrent</td>
<td>5</td>
<td>3</td>
<td>0.38</td>
</tr>
<tr>
<td>System.Security.AccessControl</td>
<td>33</td>
<td>17</td>
<td>0.34</td>
</tr>
<tr>
<td>System.Runtime.Remoting.*</td>
<td>108</td>
<td>54</td>
<td>0.33</td>
</tr>
<tr>
<td>System.Runtime.Caching.*</td>
<td>14</td>
<td>7</td>
<td>0.33</td>
</tr>
<tr>
<td>System.Runtime.ConstrainedExecution</td>
<td>2</td>
<td>1</td>
<td>0.33</td>
</tr>
<tr>
<td>System.Xaml.*</td>
<td>37</td>
<td>16</td>
<td>0.30</td>
</tr>
<tr>
<td>System.Text</td>
<td>17</td>
<td>7</td>
<td>0.29</td>
</tr>
<tr>
<td>System.ComponentModel.*</td>
<td>280</td>
<td>114</td>
<td>0.29</td>
</tr>
<tr>
<td>System.Runtime.Serialization.*</td>
<td>46</td>
<td>18</td>
<td>0.28</td>
</tr>
<tr>
<td>System.Security</td>
<td>19</td>
<td>7</td>
<td>0.27</td>
</tr>
<tr>
<td>Microsoft.JScript.*</td>
<td>140</td>
<td>51</td>
<td>0.27</td>
</tr>
<tr>
<td>System.Linq.*</td>
<td>39</td>
<td>14</td>
<td>0.26</td>
</tr>
<tr>
<td>System.IdentityModel.*</td>
<td>67</td>
<td>23</td>
<td>0.26</td>
</tr>
</tbody>
</table>
Comparison of API abstractness to number of specializations

Number of Subtypes in Projects

- System.Collections
- System.Collections.Generic
- System.Runtime.Remoting
- System.ComponentModel
- System.Runtime.Serialization
- System.Collections.Specialized
- System.IO
- System.Resources
- System.Diagnostics
- System.Linq
- System.Web
- System.Windows
- System.Configuration
- System.Security.Cryptography
- System.Collections.ObjectModel
- System.Xml
- System
- System.Windows.Forms
- System.Diagnostics.Diagnostics
- System.Threading

Corpus
Usage Metrics
Unimplemented abstract types

[Diagram showing the number of unimplemented abstract types for various namespaces in the .NET Framework, with System.Runtime.InteropServices having the most at 70, followed by System, System.Net, System.Configuration, and others with varying counts.]
Previously unimplemented abstract types
### Receiver Types

- static receiver type
- runtime receiver type

### Example

```csharp
Exception ex = new InvalidOperationException();
string s = ex.ToString();

static receiver type: Exception
runtime receiver type: InvalidOperationException
```
Corpus
Usage Metrics

**Receiver Types**
- static receiver type
- runtime receiver type

**Example**

```csharp
Exception ex = new InvalidOperationException();
string s = ex.ToString();
```

static receiver type: `Exception`
runtime receiver type: `InvalidOperationException`
Corpus
Usage Metrics

Receiver Types
- static receiver type
- runtime receiver type

Example
```
Exception ex = new InvalidOperationException();
string s = ex.ToString();
```

static receiver type: Exception
runtime receiver type: InvalidOperationException
Late-bound methods

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Static Data</th>
<th>Additional Runtime Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Methods</td>
<td>% Public API</td>
</tr>
<tr>
<td>System</td>
<td>493</td>
<td>506</td>
</tr>
<tr>
<td>System.Collections</td>
<td>130</td>
<td>488</td>
</tr>
<tr>
<td>System.Collections.Generic</td>
<td>105</td>
<td>327</td>
</tr>
<tr>
<td>System.Reflection.*</td>
<td>233</td>
<td>111</td>
</tr>
<tr>
<td>System.Data.*</td>
<td>123</td>
<td>97</td>
</tr>
<tr>
<td>System.Xml.*</td>
<td>354</td>
<td>82</td>
</tr>
<tr>
<td>System.IO.*</td>
<td>203</td>
<td>51</td>
</tr>
<tr>
<td>System.ComponentModel.*</td>
<td>79</td>
<td>25</td>
</tr>
<tr>
<td>System.Linq.*</td>
<td>55</td>
<td>16</td>
</tr>
<tr>
<td>System.Runtime.Remoting.*</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>System.CodeDom.*</td>
<td>47</td>
<td>9</td>
</tr>
<tr>
<td>System.Text</td>
<td>44</td>
<td>9</td>
</tr>
<tr>
<td>System.Configuration.*</td>
<td>31</td>
<td>8</td>
</tr>
<tr>
<td>System.Net.*</td>
<td>71</td>
<td>7</td>
</tr>
<tr>
<td>System.Windows.Forms.*</td>
<td>446</td>
<td>3</td>
</tr>
<tr>
<td>System.Web.*</td>
<td>382</td>
<td>3</td>
</tr>
<tr>
<td>System.Security.Principal</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>System.Windows.*</td>
<td>34</td>
<td>2</td>
</tr>
<tr>
<td>System.Runtime.Serialization.*</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>
Examples for late-bound receiver methods

**Public API**

```csharp
IDisposable file = new FileStream(...);
file.Close();
```

**Internal API**

```csharp
// Object.GetType() returns an object
// of type RuntimeType, which is internal.
Type type = file.GetType();
```
Examples for late-bound receiver methods

**Public API**

```csharp
IDisposable file = new FileStream(...);
file.Close();
```

**Internal API**

```csharp
// Object.GetType() returns an object
// of type RuntimeType, which is internal.
Type type = file.GetType();
```
Examples for late-bound receiver methods

**Project**

```csharp
class ProjectClass : ICloneable { ... }

ICloneable original = new ProjectClass();
object clone = c.Clone();
```
Corpus
Usage Metrics

Distribution of method calls in the corpus
Conclusion and Further Work
Conclusion

- not much framework type specialization
  - often limited to special types
- tendency towards interfaces over type specialization
  - late-binding used with interfaces
- corpus mostly does not meet usage expectations from framework design
- many calls into API code, but primarily “library-like” usage
Conclusion

- not much framework type specialization
  - often limited to special types
- tendency towards interfaces over type specialization
  - late-binding used with interfaces
- corpus mostly does not meet usage expectations from framework design
- many calls into API code, but primarily “library-like” usage
Conclusion

- not much framework type specialization
  - often limited to special types
- tendency towards interfaces over type specialization
  - late-binding used with interfaces
- corpus mostly does not meet usage expectations from framework design
- many calls into API code, but primarily “library-like” usage
Conclusion

- not much framework type specialization
  - often limited to special types
- tendency towards interfaces over type specialization
  - late-binding used with interfaces
- corpus mostly does not meet usage expectations from framework design
- many calls into API code, but primarily “library-like” usage
Further Work

- automated test suites for runtime analysis
- specialized analyses for smaller sets of APIs
- measure other API usage forms
- comparison to Java APIs
- acceptance of new framework features
Further Work

- automated test suites for runtime analysis
- specialized analyses for smaller sets of APIs
- measure other API usage forms
- comparison to Java APIs
- acceptance of new framework features
Further Work

- automated test suites for runtime analysis
- specialized analyses for smaller sets of APIs
- measure other API usage forms
- comparison to Java APIs
- acceptance of new framework features
Further Work

- automated test suites for runtime analysis
- specialized analyses for smaller sets of APIs
- measure other API usage forms
- comparison to Java APIs
- acceptance of new framework features
Further Work

- automated test suites for runtime analysis
- specialized analyses for smaller sets of APIs
- measure other API usage forms
- comparison to Java APIs
- acceptance of new framework features
Thanks for your attention!