Exploratory research on API usage

Ralf Lämmel
Software Languages Team, University of Koblenz-Landau

“Exploratory case studies are used as initial investigations of some phenomena to derive new hypotheses and build theories.”
API = Application Programming Interface
## What’s an API?

These are core and 3rd party APIs for the Java platform.

<table>
<thead>
<tr>
<th>API</th>
<th>Domain</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Collections</td>
<td>Collections</td>
<td>yes</td>
</tr>
<tr>
<td>AWT</td>
<td>GUI</td>
<td>yes</td>
</tr>
<tr>
<td>Swing</td>
<td>GUI</td>
<td>yes</td>
</tr>
<tr>
<td>Reflection</td>
<td>Other</td>
<td>yes</td>
</tr>
<tr>
<td>Core XML</td>
<td>XML</td>
<td>yes</td>
</tr>
<tr>
<td>DOM</td>
<td>XML</td>
<td>yes</td>
</tr>
<tr>
<td>SAX</td>
<td>XML</td>
<td>no</td>
</tr>
<tr>
<td>log4j</td>
<td>Logging</td>
<td>no</td>
</tr>
<tr>
<td>JUnit</td>
<td>Testing</td>
<td>no</td>
</tr>
<tr>
<td>Comm.Logging</td>
<td>Logging</td>
<td>no</td>
</tr>
<tr>
<td>Namespace</td>
<td># Types</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>System.Web.*</td>
<td>2327</td>
<td></td>
</tr>
<tr>
<td>System.Windows.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.ServiceModel.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Windows.Forms.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Data.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Activities.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.ComponentModel.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Workflow.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Xml.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Net.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.DirectoryServices.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Microsoft.VisualBasic.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Runtime.InteropServices.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Microsoft.JScript.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Drawing.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Runtime.Remoting.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Configuration.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Diagnostics.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.IO.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Reflection.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.EnterpriseServices.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.CodeDom.*</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>System.Management.*</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

What’s an API?

Each .NET namespace may be regarded as an API.
API domains

• Programming domains
  ‣ XML
  ‣ GUI
  ‣ BCE (bytecode engineering)
  ‣ Testing
  ‣ ...

• Application domains
  ‣ Financial exchange
  ‣ Human resources
  ‣ ...

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators
Research challenges related to APIs

• Help programmers to understand APIs.
• Help API developers to evolve APIs.
• Help API users to do API migration.
API migration = to eliminate an application's dependencies on a given API (the “original” API) and to make it depend instead on another API (the “replacement” API).
Incentives for API migration

• The replacement API is
  ‣ more modern, or
  ‣ more typed, or
  ‣ more compliant, or
  ‣ more efficient, or
  ‣ more usable, etc.

• The original API is
  ‣ no longer supported, or
  ‣ too costly in terms of license costs.

• The application must use less APIs per domain.

• Help with language migration.
API migration is a hard problem!


Large-scale, AST-based API-usage analysis of open-source Java projects

Ralf Lämmel¹,² and Ekaterina Pek² and Jürgen Starek¹

¹ Software Languages Team, Universität Koblenz-Landau, Germany
² ADAPT Lab, Universität Koblenz-Landau, Germany

Published in SAC’11
Large-scale, AST-based API-usage analysis of open-source Java projects

Ralf Lämmel\textsuperscript{1,2} and Ekaterina Pek\textsuperscript{2} and Jürgen Starek\textsuperscript{1}

\textsuperscript{1} Software Languages Team, Universität Koblenz-Landau, Germany
\textsuperscript{2} ADAPT Lab, Universität Koblenz-Landau, Germany

Research on API migration and language conversion can be informed by empirical data about API usage. For instance, such data may help with designing and defending mapping rules for API migration in terms of relevance and applicability. We describe an approach to large-scale API-usage analysis of open-source Java projects, which we also instantiate for the SourceForge open-source repository in a certain way. Our approach covers checkout, building, tagging with metadata, fact extraction, analysis, and synthesis with a large degree of automation. Fact extraction relies on resolved (type-checked) ASTs. We describe a few examples of API-usage analysis; they are motivated by API migration. These examples are concerned with analysing API footprint (such as the numbers of distinct APIs used in a project), API coverage (such as the percentage of methods of an API used in a corpus), and framework-like vs. class-library-like usage.
Large-scale, AST-based API-usage analysis of open-source Java projects

Ralf Lämmel$^{1,2}$ and Ekaterina Pek$^2$ and Jürgen Starek$^1$

$^1$ Software Languages Team, Universität Koblenz-Landau, Germany
$^2$ ADAPT Lab, Universität Koblenz-Landau, Germany

• 6K+ projects from SourceForge
• 1.5K projects built (200k classes)
• 60 reference projects
• Pool of 77 known APIs aggregated
• Additional API packages detected automatically
Many APIs per project

![Graph showing the number of used APIs versus project size. The graph compares built projects, unbuilt projects, and reference projects. The x-axis represents project size (in NCLOC), and the y-axis represents the number of used known APIs. The data points are color-coded to differentiate between built, unbuilt, and reference projects.](image-url)
Many distinct API methods

Figure 9 shows the numbers of distinct API methods used in projects, without distinguishing among APIs. The graph illustrates a logarithmic relationship between project size and the number of distinct API methods used. The distribution of data points indicates a trend where larger projects tend to use more distinct API methods, with a notable clustering of points around certain levels of project size and API method usage. This suggests that there is a significant variation in how projects leverage APIs, with some projects using a small number of distinct methods while others utilize many. The logarithmic scale on both axes (project size in MC and number of distinct API methods) helps to visualize the relationship across a wide range of project sizes, highlighting the scale at which API usage grows with project complexity.
Low usage of API methods

Cumulative coverage of API methods is 24.10% (if we ignore JDOM sources).
Few subclass derivations for APIs

Figure 3. Tag cloud of overridden API classes.

Only 7 APIs are used in more than 10 projects with “generalization”.

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators
Few interfaces implementations for APIs

Figure 1. Ratio of API method calls to all method calls.

Fig. 2 shows the usage of known API methods relative to all methods in a project—both in terms of calls. The smaller the ratio, the closer to zero, the lower the contribution of API calls. The quartiles show that in most projects, about each second method call is an API call.

As far as instance method calls are concerned, the figure distinguishes API vs. project-based method calls solely on the grounds of the static receiver type of methods.

Fig. 9 shows the relative frequency of API, interface implementations for all the known APIs, including Core APIs. The picture is dominated by AWT handler types, the interface for iterators, and a few XML-related types.

Only 7 APIs are used in more than 10 projects with "generalization".
A Framework Profile of .NET

Ralf Lämmel, Rufus Linke, Ekaterina Pek, and Andrei Varanovich

Software Languages Team & ADAPT Lab
Universität Koblenz-Landau, Germany

Published in WCRE’12
### Patterns of API usage

#### Table and Diagram

<table>
<thead>
<tr>
<th>Namespaces</th>
<th>Projects</th>
<th>Usage</th>
<th>Dynamic dispatch</th>
<th>Type specialization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rows:** top-10 .NET namespaces, in terms of number of types.

**Middle block of columns:** actual reuse by project of the corpus.

**Leftmost column:** potential reuse in terms of specializability.

**Rightmost column:** summary of actual reuse.

---

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators
A Framework Profile of .NET

Ralf Lämmel, Rufus Linke, Ekaterina Pek, and Andrei Varanovich

Software Languages Team & ADAPT Lab
Universität Koblenz-Landau, Germany

• 17 projects from different repositories
  • well known, widely used, mature
  • amenable to dynamic analysis
• API = .NET namespace
  • 401 in total
  • 69 after grouping
A Framework Profile of .NET

Ralf Lämmel, Rufus Linke, Ekaterina Pek, and Andrei Varanovich

Software Languages Team & ADAPT Lab
Universität Koblenz-Landau, Germany

Research questions

1) What are the interesting and helpful high-level characteristics of frameworks with regard to their potential and actual reuse?

2) To what extend can those characteristics be computed with simple metrics subject to simple static and dynamic program analysis?
Research method

We applied an explorative approach such that a larger set of metrics of mainly structural properties was incrementally screened until a smaller set of key metrics and derived classifiers emerged. We use infographics (such as Figure 1) to visualize metrics, classifiers, and other characteristics of frameworks and projects that use them. The resulting claims are subject to validation by domain experts for the framework under study.
Corpus of the .NET study

<table>
<thead>
<tr>
<th>.NET</th>
<th>Project</th>
<th>Repository</th>
<th>LOC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>Castle ActiveRecord</td>
<td>GitHub</td>
<td>30,303</td>
<td>Object-relational mapper</td>
</tr>
<tr>
<td>4.0</td>
<td>Castle Core Library</td>
<td>GitHub</td>
<td>36,659</td>
<td>Core library for the Castle framework</td>
</tr>
<tr>
<td>3.5</td>
<td>Castle MonoRail</td>
<td>GitHub</td>
<td>58,121</td>
<td>MVC Web framework</td>
</tr>
<tr>
<td>4.0</td>
<td>Castle Windsor</td>
<td>GitHub</td>
<td>50,032</td>
<td>Inversion of control container</td>
</tr>
<tr>
<td>4.0</td>
<td>Json.NET</td>
<td>Codeplex</td>
<td>43,127</td>
<td>JSON framework</td>
</tr>
<tr>
<td>2.0</td>
<td>log4net</td>
<td>Sourceforge</td>
<td>27,799</td>
<td>Logging framework</td>
</tr>
<tr>
<td>2.0</td>
<td>Lucene.Net</td>
<td>Apache.org</td>
<td>158,519</td>
<td>Search engine</td>
</tr>
<tr>
<td>4.0</td>
<td>Managed Extensibility Framework</td>
<td>Codeplex</td>
<td>149,303</td>
<td>Framework for extensible applications and components</td>
</tr>
<tr>
<td>4.0</td>
<td>Moq</td>
<td>GoogleCode</td>
<td>17,430</td>
<td>Mocking library</td>
</tr>
<tr>
<td>2.0</td>
<td>NAnt</td>
<td>Sourceforge</td>
<td>56,529</td>
<td>Build tool</td>
</tr>
<tr>
<td>3.5</td>
<td>NHibernate</td>
<td>Sourceforge</td>
<td>330,374</td>
<td>Object-relational mapper</td>
</tr>
<tr>
<td>3.5</td>
<td>NUnit</td>
<td>Launchpad</td>
<td>85,439</td>
<td>Unit testing framework</td>
</tr>
<tr>
<td>4.0</td>
<td>Patterns &amp; Practices - Prism</td>
<td>Codeplex</td>
<td>146,778</td>
<td>Library to build flexible WPF and Silverlight applications</td>
</tr>
<tr>
<td>3.5</td>
<td>RhinoMocks</td>
<td>GitHub</td>
<td>23,459</td>
<td>Mocking framework</td>
</tr>
<tr>
<td>2.0</td>
<td>SharpZipLib</td>
<td>Sourceforge</td>
<td>25,691</td>
<td>Compression library</td>
</tr>
<tr>
<td>2.0</td>
<td>Spring.NET</td>
<td>GitHub</td>
<td>183,772</td>
<td>Framework for enterprise applications</td>
</tr>
<tr>
<td>2.0</td>
<td>xUnit.net</td>
<td>Codeplex</td>
<td>23,366</td>
<td>Unit testing framework</td>
</tr>
<tr>
<td>Namespace</td>
<td># Types</td>
<td># Methods</td>
<td>MAX size class tree</td>
<td>MAX size interface tree</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------</td>
<td>-----------</td>
<td>--------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>System.Web.*</td>
<td>2327</td>
<td>29315</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Windows.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.ServiceModel.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Windows.Forms.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Data.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Activities.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.ComponentModel.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Workflow.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Xml.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Net.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.DirectoryServices.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Microsoft.VisualBasic.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Runtime.InteropServices.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Microsoft.JScript.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Drawing.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Runtime.Remoting.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>System.Configuration.*</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>“Potential reuse”</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Types</td>
<td># Methods</td>
<td>MAX size class tree</td>
<td>MAX size interface tree</td>
<td># Referenced namespaces</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>---------------------</td>
<td>------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>2327</td>
<td>29315</td>
<td>●</td>
<td>●</td>
<td>43</td>
</tr>
<tr>
<td>82</td>
<td>82</td>
<td>●</td>
<td>●</td>
<td>43</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>●</td>
<td>●</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>●</td>
<td>●</td>
<td>12</td>
</tr>
<tr>
<td>69</td>
<td>69</td>
<td>●</td>
<td>●</td>
<td>69</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>●</td>
<td>●</td>
<td>38</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>●</td>
<td>●</td>
<td>23</td>
</tr>
</tbody>
</table>
Namespace categories with regard to ‘inter-namespaces reuse’:
- *application* if \#*Referring namespaces* = 0.
- *core* if \#*Referring namespaces* is ‘exceptional’.

Namespace categories with regard to ‘specializability’:
- *open* if \%*Specializable types* is ‘exceptional’.
- *closed* if \%*Sealed classes* is ‘exceptional’.
- *incomplete* if \%*Orphan types* is ‘exceptional’.

Namespace categories with regard to ‘class-inheritance trees’:
- *branched* if MAX size class tree is ‘exceptional’.
- *flat* if MAX size class tree = 0.

Namespace categories with regard to ‘intensiveness’:
- *interface-intensive* if \%*Interface arguments* is ‘exceptional’.
- *delegate-intensive* if \%*Delegate arguments* is ‘exceptional’.

A sub-category for delegate-intensive namespaces:
- *event-based* if \%*Delegate types* is ‘exceptional’.

Occurrences of ‘exceptional’ are essentially configurable. In this paper, we assume though that “\(x\) is ‘exceptional’ for a namespace” proxies for the statement that the metric \(x\) for the given namespace is in the [75, 100) percentage interval with regard to the distribution for metric \(x\) over all namespaces.
## Classification of selected namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Application</th>
<th>Core</th>
<th>Open</th>
<th>Closed</th>
<th>Incomplete</th>
<th>Branched</th>
<th>Flat</th>
<th>Interface-intensive</th>
<th>Delegate-intensive</th>
<th>Event-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Web.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Data.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Activities.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.ComponentModel.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Xml.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.DirectoryServices.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.EnterpriseServices.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.CodeDom.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Linq.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.AddIn.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft.VisualBasic.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Transactions.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Collections</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Device.Location</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Runtime.ExceptionServices</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namespace</td>
<td># Types</td>
<td>% Specializable types</td>
<td>ActiveRecord</td>
<td>CastleCore</td>
<td>MonoRail</td>
<td>Windsor</td>
<td>Json.NET</td>
<td>log4net</td>
<td>MEF</td>
<td>Moq</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------</td>
<td>-----------------------</td>
<td>--------------</td>
<td>------------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>---------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Framework</td>
<td></td>
<td></td>
<td>3.5</td>
<td>4.0</td>
<td>3.5</td>
<td>4.0</td>
<td>2.0</td>
<td>4.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>System.Web.*</td>
<td>2327</td>
<td>•</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>System.Windows.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.ServiceModel.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Windows.Forms.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Data.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.ComponentModel.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Xml.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Net.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Runtime.InteropServices.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft.VisualBasic.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Drawing.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Runtime.Remoting.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Configuration.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Diagnostics.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.IO.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.Reflection.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.EnterpriseServices.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.CodeDom.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft.Build.*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**“Actual reuse”**
### Table: Project Reuse Summary

<table>
<thead>
<tr>
<th># Types</th>
<th>% Specializable types</th>
<th>ActiveRecord</th>
<th>CastleCore</th>
<th>MonoRail</th>
<th>Windsor</th>
<th>Json.NET</th>
<th>log4net</th>
<th>MEF</th>
<th>Moq</th>
<th>NAnt</th>
<th>NHibernate</th>
<th>NUnit</th>
<th>Prism</th>
<th>Rhino-Mocks</th>
<th>Spring.NET</th>
<th>xUnit</th>
<th>SharpZipLib</th>
<th>Lucene.Net</th>
<th>Dominator</th>
<th>% Referenced OO types (rel.)</th>
<th>% Specialized types (rel.)</th>
<th>% Late-bound types (rel.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2327</td>
<td>•</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Code namespaces contribute to the

### “Actual reuse”

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators
Data interpretation

Let us compare potential reuse in terms of specializability with actual reuse in terms of the dominator. There are eight namespaces with dominator ‘▲’ or ‘■’. Half of these namespaces contribute to the `System.Collections.*` hierarchy and the associated specializability is ‘exceptional’. However, specializability is ‘non-exceptional’ for the remaining cases; specializability is, in fact, in the percentage interval (0,25) for two cases; see namespaces `System.Configuration.*` and `System.Runtime.Serialization.*`. This observation further confirms that high specialization is not predicted by high specializability in any obvious sense.
Top-30 inherited .NET classes

- System.Attribute: 59
- System.Exception: 51
- System.MarshalByRefObject: 42
- System.Collections.CollectionBase: 38
- System.ApplicationException: 25
- System.EventArgs: 23
- System.Windows.Forms.Form: 17
- System.ComponentModel.TypeConverter: 17
- System.IO.Stream: 12
- System.Collections.Hashtable: 12
- System.IO.TextWriter: 11
- System.SystemException: 11
- System.ComponentModel.EnumConverter: 7
- System.IO.IOException: 6
- System.Web.UI.Control: 6
- System.Collections.ArrayList: 5
- System.Web.UI.WebControls.WebControl: 4
- System.Linq.Expressions.Expression: 4
- System.Resources.ResourceManager: 4
- System.Configuration.ConfigurationElement: 3
- System.Configuration.ConfigurationElementCollection: 3
- System.Web.UI.Page: 3
- System.Collections.Specialized.NameValueCollection: 2
- System.Collections.DictionaryBase: 2
Top-30 implemented .NET interfaces

- System.IDisposable: 303
- System.Collections.IEnumerable: 256
- System.Collections.IEnumerator: 185
- System.Collections.Generic.IEnumerable<T>: 176
- System.Collections.Generic.IEnumerator<T>: 118
- System.ICloneable: 83
- System.Collections.IEnumerable: 65
- System.Runtime.Serialization.ISerializable: 38
- System.Collections.Generic.ICollection<T>: 33
- System.Collections.IDictionary: 28
- System.Collections.IComparer: 25
- System.Collections_IList: 19
- System.IServiceProvider: 18
- System.Runtime.Serialization.IDeserializationCallback: 16
- System.Collections.IDictionaryEnumerator: 14
- System.IEquatable<T>: 14
- System.IComparable: 14
- System.Configuration.IConfigurationSectionHandler: 13
- System.Collections.Generic.IDictionary<TKey,TValue>: 12
- System.Collections.Generic.IEqualityComparer<T>: 11
- System.ComponentModel.INotifyPropertyChanged: 10
- System.Collections.Generic_IList<T>: 9
- System.ComponentModel.IContainer: 7
- System.Web.IHttpModule: 7
- System.Web.IHttpHandler: 6
- System.Data.IDataReader: 5
- System.ComponentModel.IContainer: 5
- System.ComponentModel.IContainer: 5

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators
Multi-dimensional exploration of API usage

Coen De Roover
Software Languages Lab
Vrije Universiteit Brussel

Ralf Lämmel
Software Languages Team
University of Koblenz-Landau

Ekaterina Pek
ADAPT Lab
University of Koblenz-Landau

Published in ICPC’13
Multi-dimensional exploration of API usage

Coen De Roover  
Software Languages Lab  
Vrije Universiteit Brussel

Ralf Lämmel  
Software Languages Team  
University of Koblenz-Landau

Ekaterina Pek  
ADAPT Lab  
University of Koblenz-Landau

We identify abstract exploration insights as they are expected by API developers and project developers with regard to their overall intention to understand API usage. These expected insights rely on multiple dimensions of exploration, e.g., hierarchical organization of scopes and project-versus API-centric perspectives. Existing methods such as code completion and searching API documentation do not serve these insights.
Exploration insights

• The API Dispersion insight
• The API Distribution insight
• The API Footprint insight
• The Sub-API Footprint insight
• The API Cocktail insight
• The API Coupling insight
• The API Profile insight
We use the following format:

- The hierarchical breakdown of the project scopes with
- The listing of projects with associated API usage metrics for quantitative comparison and API facets for qualitative comparison and
- Understand API distribution across project scopes
- Understand an API's dispersion in a corpus by comparing API usage across the projects in the corpus
- Project developer

A. Format of insight descriptions

- The insight is about the significance of API usage across corpus
- The insight benefits the API developer
- The insight may help a developer to decide on the feasibility of an API migration, as we discussed in
- The projects are ordered by the
- The listing of used API packages, types, and methods
- The listing is further refined to show details per API

B. The intelligence supported by the insight

- The listing of projects with associated API usage metrics for quantitative comparison and API facets for qualitative comparison and metrics not aligning
- The projects are ordered by

C. The on suitable 'test projects' just like that

- The insight describes in abstract terms how API usage

D. The intelligence supported by the insight

- The insight identifies whether

JDOM

Fig. 4. JDOM's API Dispersion in QUAATLAS (project-centric table).

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators
**Intent** – Understand an API’s dispersion in a corpus by comparing API usage across the projects in the corpus.

**Stakeholder** – API developer.

**API usage** – One API.

**View** – The listing of projects with associated API-usage metrics for quantitative comparison and API facets for qualitative comparison.

**Illustration** – Fig. 4 summarizes JDOM’s dispersion quantitatively in QUAATLAS. 6 projects in the corpus exercise JDOM. The projects are ordered by the #ref metric with the other metrics not aligning. Only 2 projects (jspwiki and velocity) exercise type derivation at the boundary of API and project.

**Intelligence** – The insight is about the significance of API usage across corpus. In the figure, arguably, project jspwiki shows the most significant API usage because it references the most API elements. Project jmeter shows the least significant API usage. Observation of significance helps an API developer in picking hard and easy projects for compliance testing along API evolution—an easy one to get started; a hard one for a solid proof of concept. For instance, development of a wrapper-based API re-implementation for API migration relies on suitable ‘test projects’ just like that [6], [7].
The view only shows packages and types with -PI references to DOM. Out of the top-level packages of JHotDraw, only one of them, the xml package and its subpackage css, reference -PI. There is a total of 2 class types that contain references -PI. The combined reference count is 92, where 9 unique -PI elements are referenced, which is a relatively small number of used -PI elements in the view of hundreds of -PI elements declared by the library.

public void applyStylesTo(Element elem) {
    for (CSSRule rule : rules) {
        if (rule.matches(elem)) {
            rule.apply(elem);
        }
    }
}

Let us focus on the requirement for replacing XML by JSON. In two XML -PIs show up DOM and SAX. Joanna begins with an exploration of DOM usage in JHotDraw. She summarizes DOM usage in JHotDraw as analyzed with X-PUS. Encouragingly, DOM's footprint in JHotDraw only covers a few types and methods. The corresponding view is shown in Fig. 1 and it strikingly reveals good news in so far that DOM usage is limited to the JHotDraw package org.jhotdraw.xml, which she shall explore further to prepare a possible XML-to-JSON migration.

### III. Related Work

We identify the following categories of related work. In discussion of the cited papers, we bring forward the aspects directly comparable to our effort.

#### A. Exploration of Projects

There are several conceptual styles of project comprehension. An example of interactive, human-involving effort can be found in work of (Fischuhl et al., [1] Fischuhl et al., [1]) where experts annotate project parts to capture human knowledge. They further use the emerged metamodel to analyze features, architecture, and design flaws of the project.

Query-driven comprehension can proceed through user-defined queries that identify code of interest, as in the work of Mens and Kellens ([2] Mens and Kellens, [3]) or Roover et al. ([4] Roover et al. [5]). Where a comprehensive tool suite facilitates defining and exploring query results. Lwis and Murphy in their work ([6] Lwis and Murphy [7]) identify and investigate predefined queries for exploration of a software system, e.g., “What calls this method?”

Visual summary of projects usually involves some sort of scaling, color coding, and hierarchical grouping, as discussed by Lanza and Casasse ([8] Lanza and Casasse [9]). Visualizations can be more involved, as in the work of Wettel et al. ([10] Wettel et al. [11]), where a city metaphor is used to represent a structure of projects based on the value of metrics.

Our approach combines these conceptual styles. We allow the user to accumulate and refine knowledge about -PI, their facets, and domains. The exploration activities explained in the paper are intuitive. Flexibility in their combination enables answering the typical questions like identified by Lwis and Murphy ([8] Lwis and Murphy [7]). Tag clouds, tables, and trees accompanied by metrics provide basic and familiar visual aid in exploration.

#### B. Exploration of APIs

1) Measuring usage: Research on -PI usage often leverages usage frequency or popularity of -PIs and their parts. For instance, Mileva et al. use popularity to identify most commonly used library versions ([2] Mileva et al., [3]) or to identify and predict -PI usage trends over time ([4] Mileva et al., [5]). Holmes et al. appeal to popularity as the main indicator for the -PI developer to be able to prioritize efforts and be informed about consumption of libraries for the -PI user to be able to identify libraries of.
The view only shows packages and types with -PI references to |OMx Out of the top-level packages of JHotDraw only few of them: the xml package and its subpackage css. There is a total of 2 class types that contain references. The combined reference count is 92, where 19 unique -PI elements are referenced, which is a relatively small number of used -PI elements in the view of hundreds of -PI elements declared by the JHotDraw package.

public void applyStylesTo(Element elem) {
    for (CSSRule rule : rules) {
        if (rule.matches(elem)) {
            rule.apply(elem);
        }
    }
}

Fig. 1. API usage in JHotDraw with scaling applied to numbers of API references
There are several conceptual styles of project comprehension. We identify the following categories of related work. 

Let us focus on the requirement for replacing XML by JSON. A natural next step is to explore JHotDraw. The corresponding view is shown in Figure 1. An example of a slice of JHotDraw is shown in Figure 2. JHotDraw reveals good news in so far that this approach only covers a small part of the hold. A more interesting part are the remaining nodes for packages, which is a relatively small number of used -PI elements (rule.matches(elem)).

The view only shows packages and types with -PI usage in the JHotDraw. The total number of -PI references to the JHotDraw package and its subpackages is 92, where 41 unique -PI elements are referenced. The view only shows packages and types with -PI usage in the JHotDraw. The total number of -PI references to the JHotDraw package and its subpackages is 92, where 41 unique -PI elements are referenced.

Fig. 2. The slice of JHotDraw with DOM usage.
Let us focus on the requirement for replacing XML by JSON. In two XML -PIs show up. DOM and SAX.

Joanna begins with an exploration of DOM usage in JHotDraw. She summarizes DOM usage in JHotDraw as analyzed with X-PUS. Encouragingly, DOM’s footprint in JHotDraw only covers a few types and methods. The corresponding view is shown in Fig. 1 and it strikingly reveals good news in so far that DOM usage is limited to the JHotDraw package org.jhotdraw.xml, which she shall explore further to prepare a possible XML-to-JSON migration.

III. Related work

We identify the following categories of related work. For discussion of the cited papers, we bring forward the aspects directly comparable to our effort.

A. Exploration of projects

There are several conceptual styles of project comprehension. An example of interactive, human-involving effort can be found in work of (Fuhlmann et al., ...), where experts annotate project parts to capture human knowledge. They further use the emerged metamodel to analyze features, architecture, and design flaws of the project. Query-driven comprehension can proceed through user-defined queries that identify code of interest, as in the work of Mens and Kellens or (Roover et al., ...), where a comprehensive tool suite facilitates defining and exploring query results.

Lanza and (UCasse ...), where visualizations can be more involved, as in the work of Wettel et al., ... where a city metaphor is used to represent a project structure of projects based on the value of metrics.

Our approach combines these conceptual styles. We allow the user to accumulate and refine knowledge about -PIs, their facets, and domains. The exploration activities explained in the paper are intuitive. Flexibility in their combination enables answering the typical questions like identified by (Lis and Murphy, ...).

Tag clouds, tables, and trees accompanied by metrics provide basic and familiar visual aid in exploration.

B. Exploration of APIs

1) Measuring usage:

Research on -PI usage often leverages usage frequency or popularity of -PIs and their parts.

For instance, Mileva et al. use popularity to identify the most commonly used library versions or to identify and predict -PI usage trends over time. Holmes et al. appeal to popularity as the main indicator for the -PI developer to be able to prioritize efforts and be informed about consumption of libraries for the -PI user to be able to identify libraries of

Fig. 3. Minuscule view for DOM usage in JHotDraw: with leaves for methods, eggs for types, and the remaining nodes for packages.
view for DOM usage in JHotDraw

The view only shows packages and types with -PI references to the top level packages of JHotDraw. There is a total of 2 class types that contain references to XML. The combined reference count is 92, where 9 unique -PI elements are referenced, which is a relatively small number of used -PI elements in the view of hundreds of -PI elements declared by the package org.jhotdraw.xml.

```java
public void applyStylesTo(Element elem) {
    for (CSSRule rule : rules) {
        if (rule.matches(elem)) {
            rule.apply(elem);
        }
    }
}
```

The corresponding view is shown in the figure and it strikingly reveals good news in so far that DOM usage is limited to the JHotDraw package org.jhotdraw.xml, which she shall explore further to prepare a possible XML to JSON migration.

In discussion of the cited papers, we bring forward the aspects directly comparable to our effort. In the work of Lüthi et al., experts annotate project parts to capture human knowledge. They further use the emerged metamodel to analyze features, architecture, and design flaws of the project. Query-driven comprehension can proceed through user-defined queries that identify code of interest, as in the work of Mens and Kellens or Roover et al., where a comprehensive tool suite facilitates defining and exploring query results. Léwis and Murphy in their work identify and investigate predefined queries for exploration of a software system. "What calls this method?"

Visual summary of projects usually involves some sort of scaling, color coding, and hierarchical grouping, as discussed by Lanza and Lecasse. Visualizations can be more involved, as in the work of Wettel et al., where a city metaphor is used to represent a structure of projects based on the value of metrics. Our approach combines these conceptual styles. We allow the user to accumulate and refine knowledge about -PIs, their facets, and domains. The exploration activities explained in the paper are intuitive. Flexibility in their combination enables answering the typical questions like identified by Léwis and Murphy. Tag clouds, tables, and trees accompanied by metrics provide basic and familiar visual aid in exploration.

B. Exploration of APIs

1) Measuring usage: Research on -PI usage often leverages usage frequency or popularity of -PIs and their parts. For instance, Mileva et al. use popularity to identify most commonly used library versions, or to identify and predict -PI usage trends over time. Holmes et al. appeal to popularity as the main indicator for the -PI developer to be able to prioritize efforts and be informed about consumption of libraries. for the -PI user to be able to identify libraries of...
API-usage metrics

#proj: Number of projects referencing APIs.
#api: Number of APIs being referenced.
#ref: Number of references from projects to APIs.
#elem: Number of API elements being referenced.
#derive: Number of project types derived from API types.
#super: Number of API types serving as supertype for derivations.
#sub: Number of project types serving as subtype for derivations.
API domains

GUI: GUI programming, e.g., Swing and AWT.
XML: XML processing, e.g., DOM, JDOM, and SAX.
Data: Data structures incl. containers, e.g., java.util.
IO: File- and stream-based I/O, e.g., java.io and java.nio.
Component: Component-oriented programming, e.g., JavaBeans.
Meta: Meta-programming incl. reflection, e.g., java.lang.reflect.
Basics: Basic language support, e.g., java.lang.String.
API facets

Input / Output: De-/serialization for DOM trees.
Observation: Getter-like access and other ‘read only’ forms.
Addition: Addition of nodes et al. as part also of construction.
Removal: Removal of nodes et al. as a form of mutation.
Namespaces: XML namespace manipulation.
Nontrivial XML: Use of CDATA, PI, and other XML idiosyncrasies.
A. Format of insight descriptions

We use the following format:

1. **Intent** paragraph summarizes the insight.
2. **Stakeholder** paragraph identifies whether the insight benefits the API developer, the project developer, or both.
3. **API usage** paragraph quantifies API usage of interest, e.g., whether one API is considered or all APIs.
4. **View** paragraph describes, in abstract terms, how API usage data is to be rendered.
5. **Illustration** paragraph applies the abstract insight concretely to APIs and projects of QUAATLAS.
6. We use different forms of illustrations, such as tables, trees, and tag clouds.
7. **Intelligence** paragraph hints at the 'operational' intelligence supported by the insight.

B. The API Dispersion insight

1. **Intent** – Understand an API's dispersion in a corpus by comparing API usage across the projects in the corpus.
2. **Stakeholder** – API developer.
3. **API usage** – One API.
4. **View** – The listing of projects with associated API usage metrics for quantitative comparison and API facets for qualitative comparison.
5. **Illustration** – Fig. 4 summarizes JDOM's dispersion quantitatively in QUAATLAS projects in the corpus. The projects are ordered by the reference metric with the other metrics not aligning. Only two projects, jspwiki and velocity, exercise type derivation at the boundary of API and project. The intelligence of the insight is about the significance of API usage across corpus. In the figure, arguably, project jspwiki shows the most significant API usage because it references the most API elements. Project jmeter shows the least significant API usage. Observation of significance helps an API developer in picking hard and easy projects for compliance testing along API evolution—an easy one to get started and a hard one for a solid proof of concept. For instance, development of a wrapper-based API reimplementation for API migration relies on suitable 'test projects' just like that.

C. The API Distribution insight

1. **Intent** – Understand API distribution across project scopes.
2. **Stakeholder** – Project developer.
3. **API usage** – One API.
4. **View** – The hierarchical breakdown of the project scopes with associated API usage metrics for quantitative comparison and API facets for qualitative comparison.
5. **Illustration** – Remember JHotDraw's slice of DOM usage in Fig. 4 in §II. This view was suitable for efficient exploration of project scopes that directly depend DOM. The intelligence of the insight may help a developer to decide on the feasibility of an API migration, as we discussed in §II.

D. The API Footprint insight

1. **Intent** – Understand what API elements are used in a corpus or varying project scopes.
2. **Stakeholder** – Project developer and API developer.
3. **API usage** – One API.
4. **View** – The listing of used API packages, types, and methods.
5. **Illustration** – Remember the tree-based representation of the API footprint for JHotDraw as shown in Fig. 4: in §II. In a similar manner, while using a table-based representation, Fig. 4 summarizes JDOM usage across QUAATLAS. All JDOM packages are listed. The core package is heavily used and thus the listing is further refined to show details per API type. Ordering relies on the reference metric. Clearly, there is little usage of API elements outside the core package. The intelligence of the footprint describes the 'smaller' 'actual' API that needs to be understood as opposed to the 'full' 'official' API. For instance, many APIs enable nontrivial framework-like usage [7], but in the absence of actual framework-like usage, the project developer may entertain a much simpler view on the API. In the context of API evolution, an API developer consults an API's footprint to minimize changes that break actual usage or to make an impact analysis for changes. In the context of wrapper-based API reimplementation for API migration, an API developer or a project developer who develops a project-specific wrapper uses the footprint to limit the effort.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>#projs</th>
<th>#refs</th>
<th>#elems</th>
<th>#derives</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jdom</td>
<td>6</td>
<td>2391</td>
<td>84</td>
<td>5</td>
</tr>
<tr>
<td>Element</td>
<td>5</td>
<td>1912</td>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>Document</td>
<td>5</td>
<td>160</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Namespace</td>
<td>4</td>
<td>82</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Attribute</td>
<td>4</td>
<td>70</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Text</td>
<td>4</td>
<td>67</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>JDOMException</td>
<td>6</td>
<td>54</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Content</td>
<td>3</td>
<td>21</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>CDATA</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>DocType</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ProcessingInstruction</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>IllegalDataException</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Comment</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EntityRef</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Verifier</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>DefaultJDOMFactory</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom.input</td>
<td>6</td>
<td>103</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>org.jdom.output</td>
<td>5</td>
<td>101</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>org.jdom.xpath</td>
<td>2</td>
<td>50</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
API facets for qualitative comparison associated API usage metrics for quantitative comparison and View API usage Stakeholder Intent

C. The on suitable 'test projects' just like that ['']2 [-]4 wrapper based API re implementation for API migration relies a solid proof of concept For instance development of a API evolution—an easy one to get started) a hard one for in picking hard and easy projects for compliance testing along API usage Observation of significance helps an API developer most API elements Project shows the most significant API usage because it references the usage across corpus In the figure arguably project 

Intelligence exercise type derivation at the boundary of API and project metrics not aligning . projects y The projects are ordered by the t

Illustration comparison4

A. Format of insight descriptions

We use the following format4 The paragraph describes2 in abstract terms2 how API usage or both4 The

API Dispersion

The insight benefits the paragraph applies the insight concretely to APIs and projects of Q data is to be rendered4 The

API developer

The insight is about the significance of API usage of jspwiki show's dispersion quantita3

an API developer ywho develops a project's dispersion quantita3

project developer

API developer or a analysis for changes4 In the context of wrapper based API migration2 an API developer or a

evolution2 an API developer consults an API's footprint to minimize changes that break actual usage or to make an impact

framework3like usage2 the project developer may entertain framework3like usage [7]2 [.;]2 but in the absence of actual

full y'official'z API4 For instance2 many APIs enable nontrivial2 'actual' API that needs to be understood as opposed to the

usage of API elements outside the core package4

Intelligence

Type4 Ordering relies on the t

UAATLAS

JDOM

4 ' projects in the corpus exercise

JDOM

UAATLAS

§ II4 This view was suitable for efficient exploration

§ II4 In

Fig4 ; summarizes

– The hierarchical breakdown of the project scopes with
– The listing of projects with associated API usage met3
– Understand API distribution across project scopes4
– Understand an API's dispersion in a corpus by com3
– The insight may help a developer to decide on

UAATLAS

§

Fig4 ; in

‘api3centric tablez4

– Remember the tree based representation of the
– Remember

View

API usage

Stakeholder

B. The intelligence supported by the insight4

We use different forms of illustrations( tables2 trees2 and tag

abstract insight
defines the intelligence

Fig4 , summarizes

© 2009-2014 Ralf Lämmel, Software Languages Team and collaborators

Fig. 5.  *JDOM’s API Footprint in QUAAATLAS* (api-centric table).
Another visualization of API footprint

Top-level circles = packages
Inner circles = classes
Size of inner circle = LOC
Color = # API-type references
(Fields are ignored.)

"project centric"
(for 1 project)

Thanks are due to Victor Winter / SHIFT Lab @ UNO.
Another visualization of API footprint

Top-level circles = packages
Inner circles = classes
Size of inner circle = LOC
Color = #API-type references

"API centric" (for 1 project)

Thanks are due to Victor Winter / SHIFT Lab @ UNO.
<table>
<thead>
<tr>
<th>Scope</th>
<th>Tags incl. facets</th>
<th>#proj</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jdom</td>
<td>JDOM</td>
<td>2</td>
</tr>
<tr>
<td>Verifier</td>
<td>JDOM,Nontrivial API</td>
<td>1</td>
</tr>
<tr>
<td>DefaultJDOMFactory</td>
<td>JDOM,Nontrivial API</td>
<td>1</td>
</tr>
<tr>
<td>EXTENDS_CLASS</td>
<td>JDOM,Nontrivial API, AnakiaJDOMFactory</td>
<td>1</td>
</tr>
<tr>
<td>Document</td>
<td>JDOM</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 6. ‘Non-trivial API’ usage for package org.jdom in QUAATLAS.
Fig. 7. The API Cocktail of JHotDraw (cloud of API tags).
### Fig. 8. Cocktail of domains for *JHotDraw*.

<table>
<thead>
<tr>
<th>GUI Basics</th>
<th>Component</th>
<th>IO</th>
<th>Project <em>jhotdraw</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>Format</td>
<td>Component</td>
<td>Meta</td>
</tr>
<tr>
<td>XML</td>
<td>Distribution</td>
<td>Parsing</td>
<td>Control</td>
</tr>
<tr>
<td>GUI</td>
<td>Basics</td>
<td>Component</td>
<td>IO</td>
</tr>
</tbody>
</table>

**Project *jhotdraw***

**Package *org.jhotdraw.undo***
Fig. 9. API Coupling for JHotDraw’s interface `org.jhotdraw.app.View`. 

```java
import java.lang.*;
import java.net.*;
import javax.swing.*;
import org.jdesktop.swingtools.*;
import java.awt.*;
import java.io.*;
import java.util.*;
```
Fig. 10. *JDOM*’s API Profile in the *informa* project (cloud of facet tags).
In this course, we could carry out research designs to study API usage in an exploratory or more definitive manner (e.g., in experiments). Further, we could discuss research on API migration, which is informed by API-usage analysis. We could also look into language usage analysis as a similar research domain.

Rather we could do a separate course on „program comprehension“ or „mining software repositories“