Empirical research on the frameworkiness of open-source projects

Jan Baltzer
jbaltzer@uni-koblenz.de

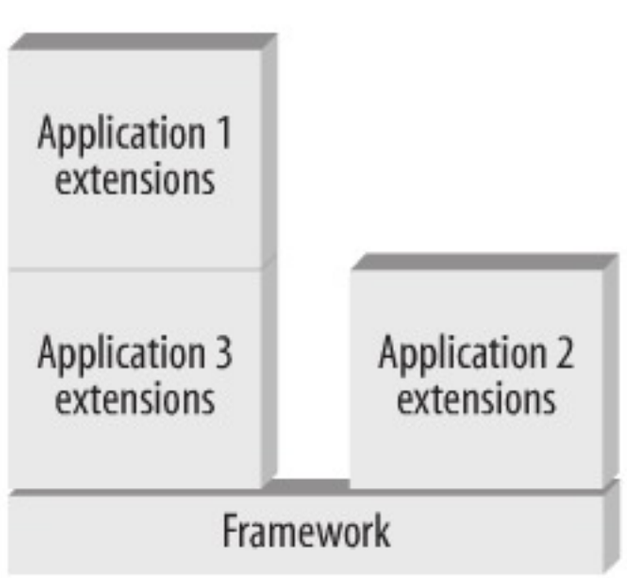
Agenda

- Motivation
- Frameworkiness
- Methods
- Measurements
- Conclusion

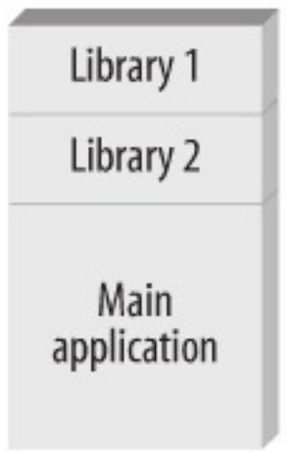
Framework vs. Library

- „A software library contains functions (...) that your application can **invoke**. A framework, on the other hand, provides generic (...) components that your application **extends** to provide a particular set of functions.“

(Cavaness, C.: *Programming Jakarta Struts*. 1st edition, Sebastopol: O‘Reilly Media, 2004. page 11)



Using a framework



Using a library

Basic questions

- What do we know about the handling of APIs?
  - How can an API be used?
    - **Design:** Provided functionality
  - How is an API used?
    - **Usage:** Used functionality

Basic assumptions

- Framework indicators
  - **Design:** Possibilities for extensions
  - **Usage:** Extensions
- Library indicators
  - **Design:** Inapplicability of extensions
  - **Usage:** Instantiations

Example: DOM - Design

- How can DOM be used?
  - Over 90% of the available types are interfaces
  - All XML types (document, node, element etc.) are interfaces

- Conclusion:
  - Usage of DOM only by interface-based programming
  - DOM is a framework

Example: DOM - Usage

- How is DOM used?

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.newDocument();

Element e = doc.createElement("foo");
```

Example: DOM - Usage

- How is DOM used?

```
// abstract class
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
// abstract class
DocumentBuilder builder = factory.newDocumentBuilder();
// interface
Document doc = builder.newDocument();
// interface
Element e = doc.createElement("foo");
```

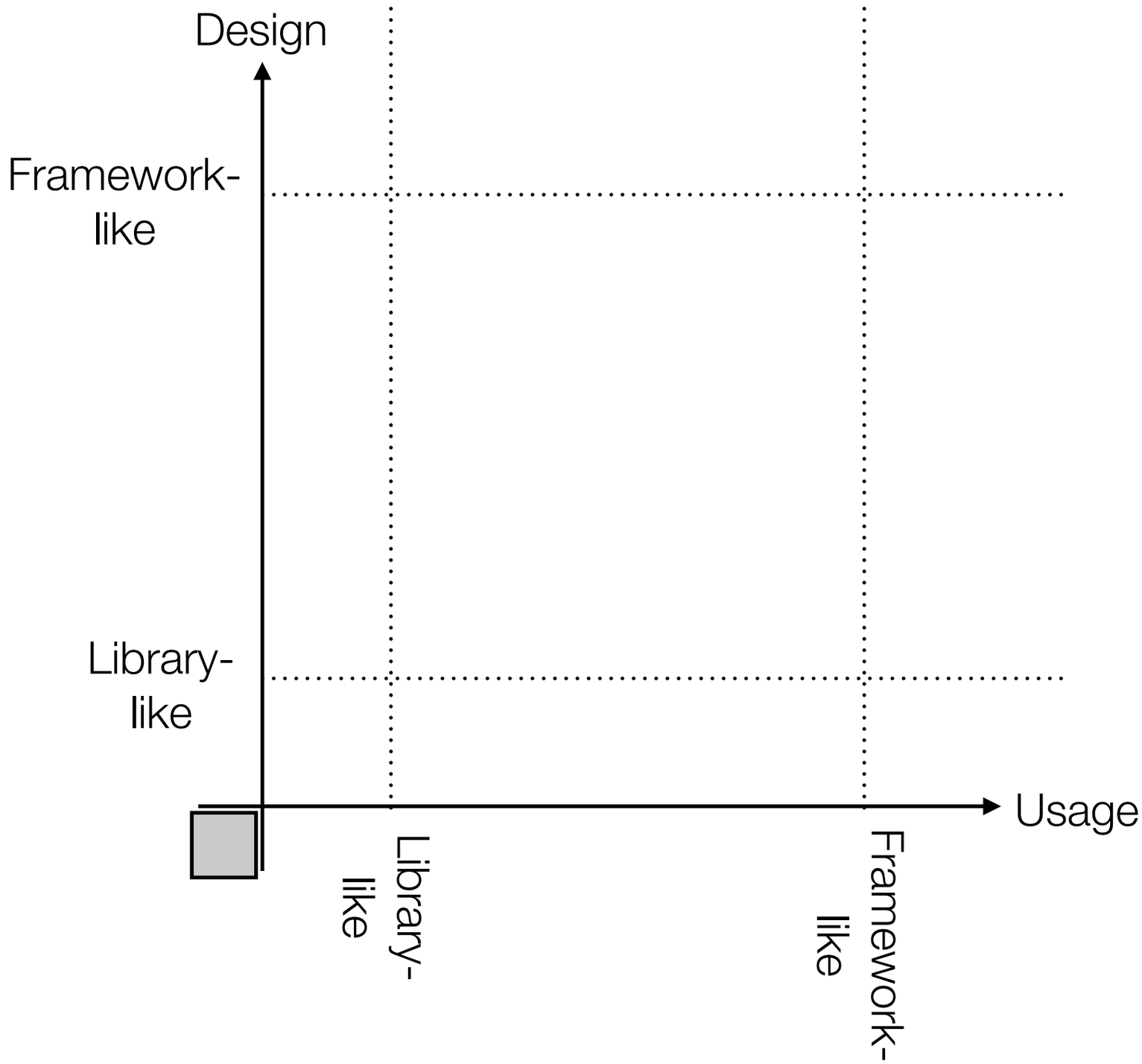
- Who provides the implementation, if no concrete class is used?

Is it a framework or a library?

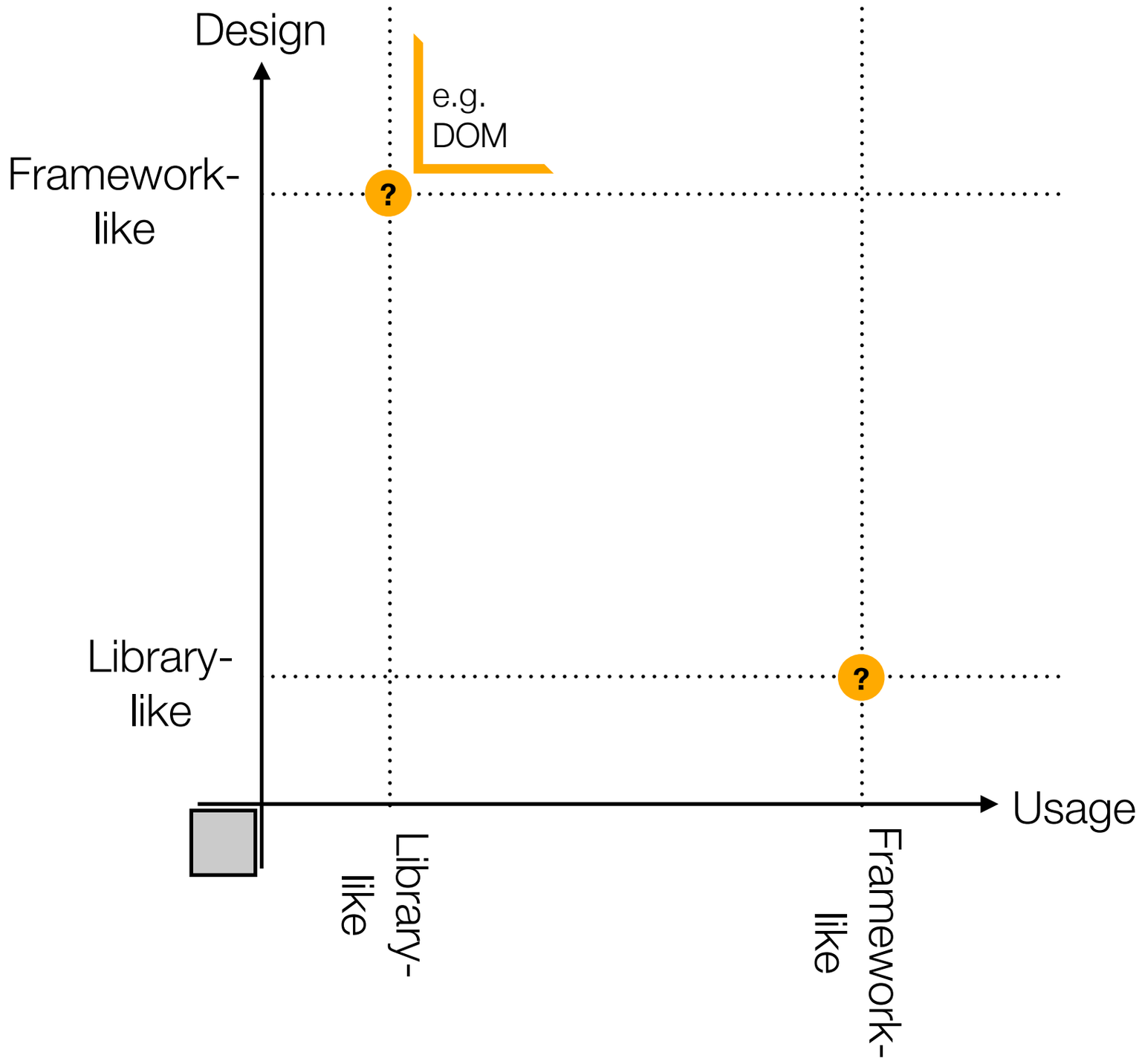
Research objectives

- Objective measurement of framework level
- Framework-like **design**
  - Providing many extendable classes (possibility of extensions)
  - Providing classes that must be extended (necessity of extensions)
- Framework-like **usage**
  - Extensions in the code of applications
- Comparison between usage and design

Dimensions of frameworkiness

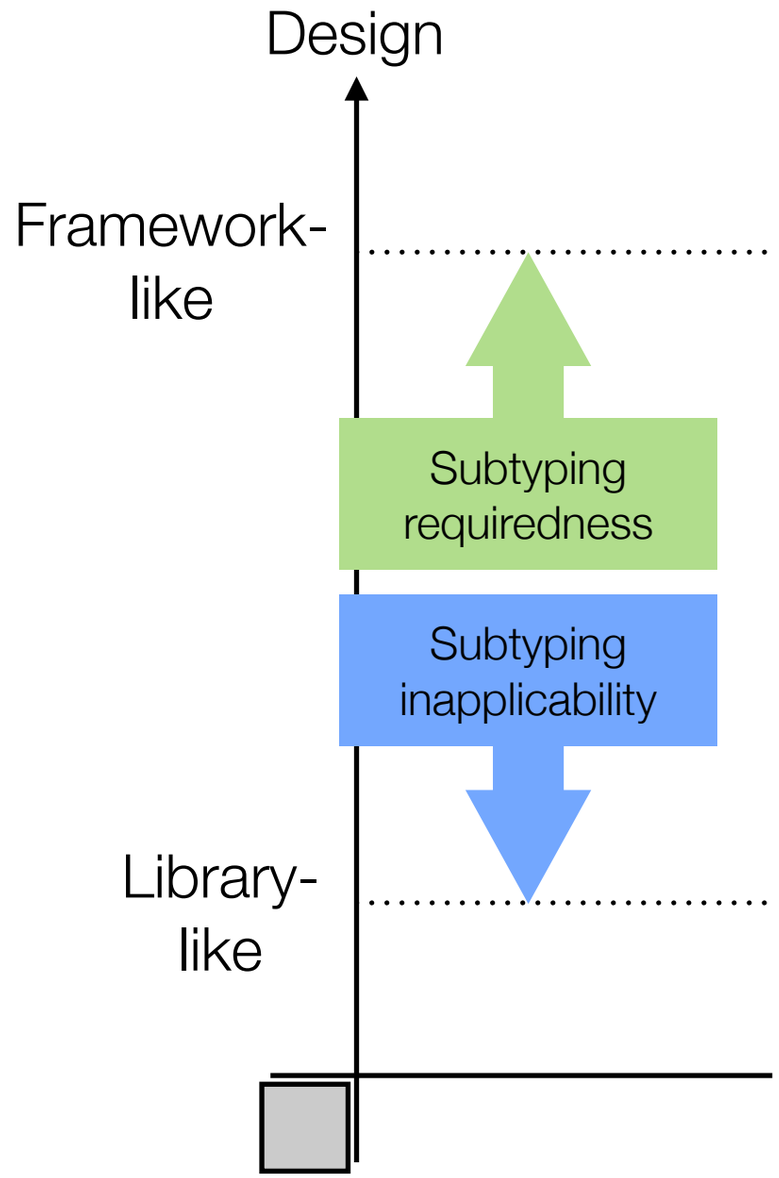


Dimensions of frameworkiness



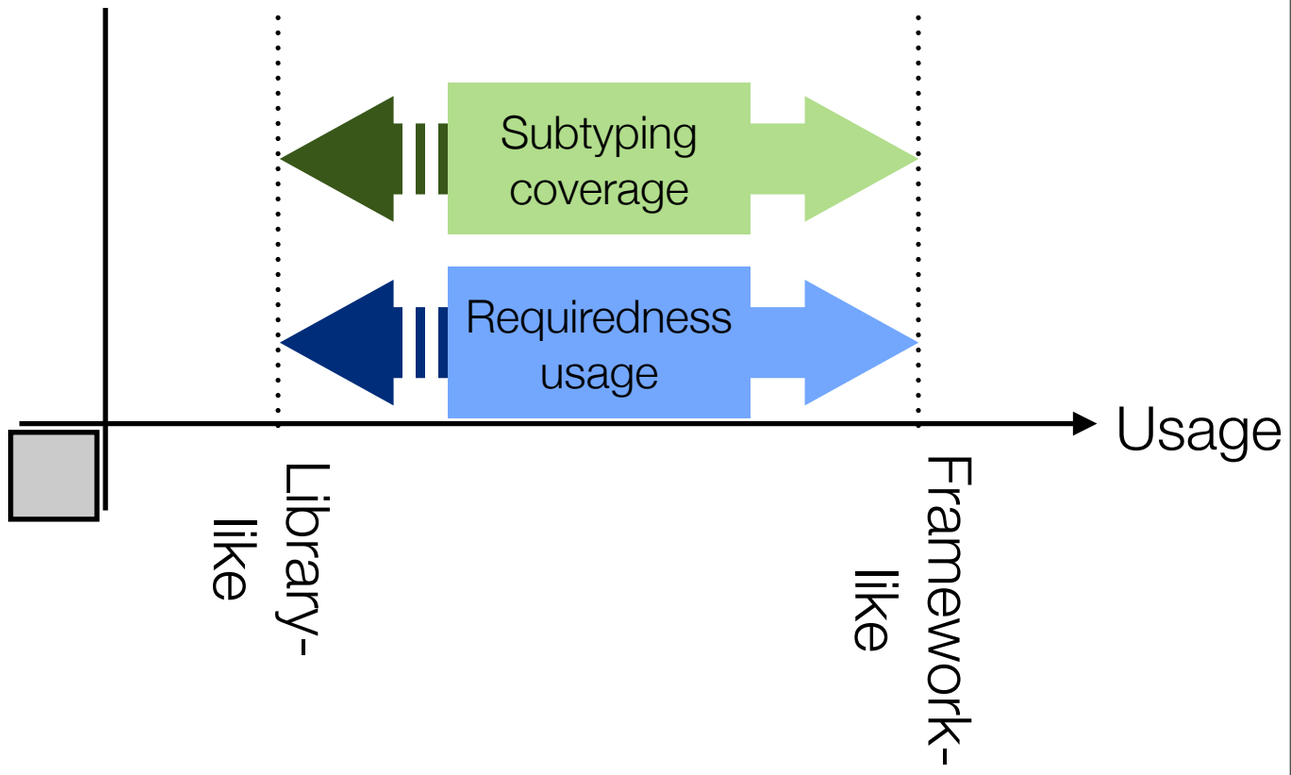
Specific indicators: Design

- *Subtyping requiredness*
  - Abstract types without concrete implementation within the API
- *Subtyping inapplicability*
  - Sealed types



Specific indicators: Usage

- *Subtyping coverage*
  - Extended and implemented types
- *Requiredness usage*
  - Used requiredness types



Project selection

- Small hand-picked corpus
  - Java projects with "good" object-oriented code
  - Based on the paper "Understanding the Shape of Java software" (Baxter et al., 2006)
  - 32 projects

API selection

- Fixed API pool
  - Main APIs of the runtime environment
  - Most "used" Java archives in the projects
    - Simple counting of occurrences
- 75 APIs

Knowledge base

- Prolog facts
  - class / interface declarations
  - method / constructor declarations
  - field declarations
  - local variable declarations
  - method / constructor calls

Sample facts

```
% fact type
interface(
    % API identifier
    "DOM",
    % package; encoded as list
    ["org", "w3c", "dom"],
    % simple name
    "Element",
    % extended interfaces; encoded as a list of lists
    [["org", "w3c", "dom", "Node"]],
    % modifiers
    ["public"]).
```

Evidence of framework-like usage

Counted occurrences

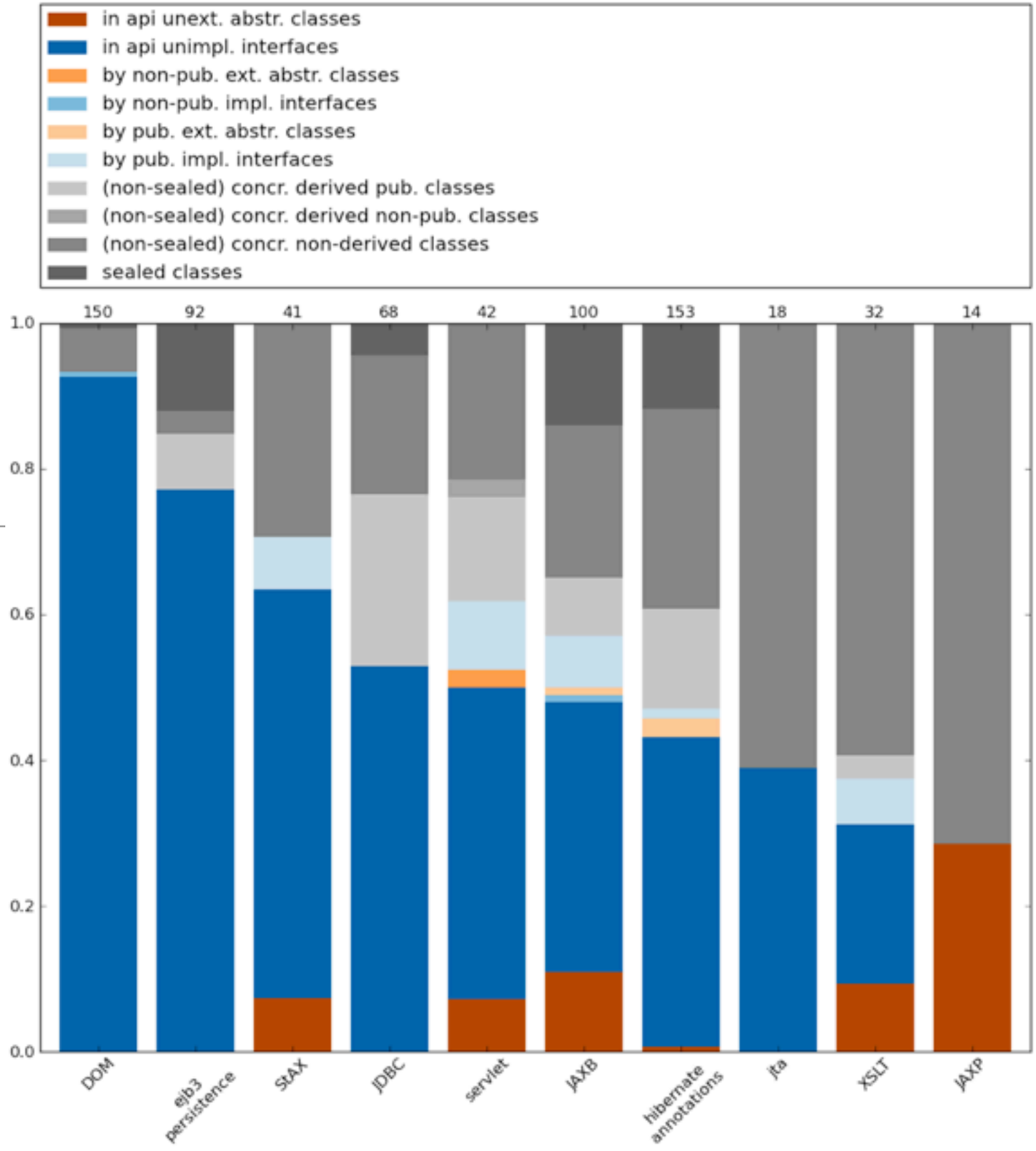
API	# projects			# types		ratio (# types / # projects)	
	interf.	class	any	interf.	class	interf.	class
Collection	29	23	**30**	480	123	16,55	5,35
AWT	20	18	21	2500	606	**125**	33,67
Swing	16	19	19	617	1561	38,56	**82,16**
SAX	8	14	17	37	63	4,63	4,5

Evidence of framework-like usage

Extended and implemented features:
Splitting CamelCase notation

Investigation of framework-like design (1)

Structural classification of API types



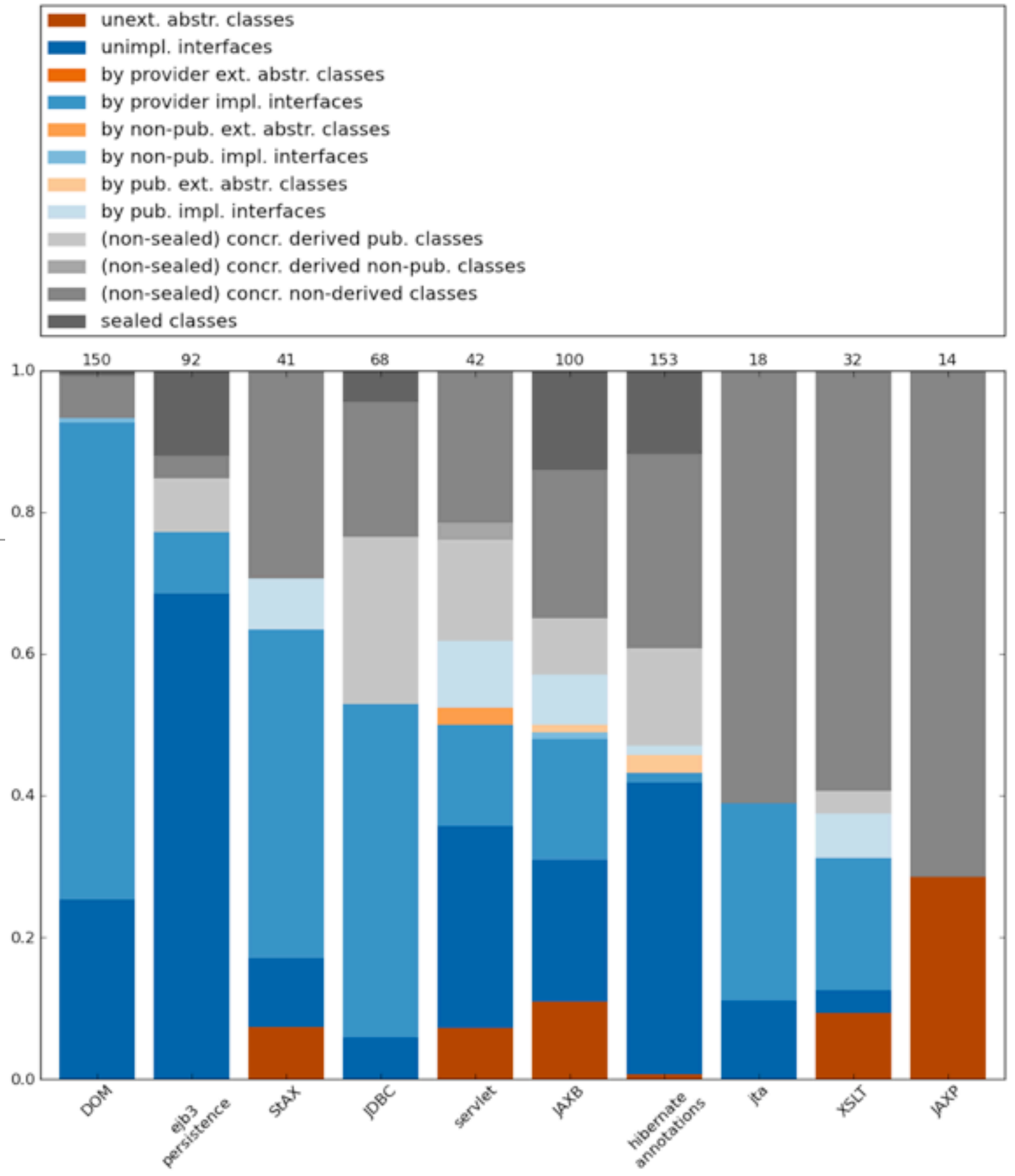
Example DOM - revisited

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element e = doc.createElement("foo");
```

- Method `newInstance()`:
  - ordered lookup procedure and fallback value in the code
  - Fallback class:
    `com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl`
- Conclusion
  - Some APIs provide functionality to other APIs

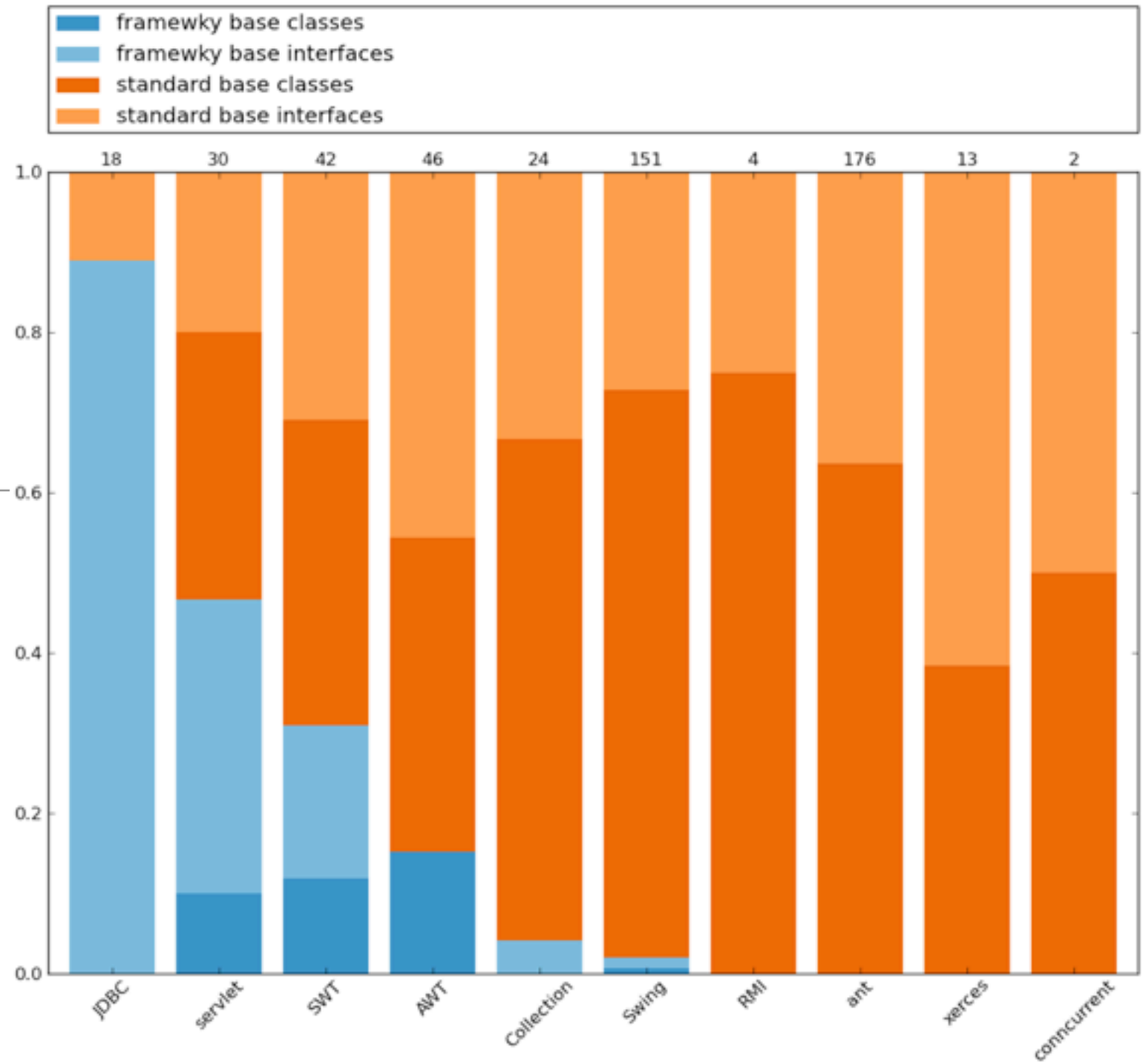
Investigation of framework-like design (2)

Structural classification of API types with provider analysis



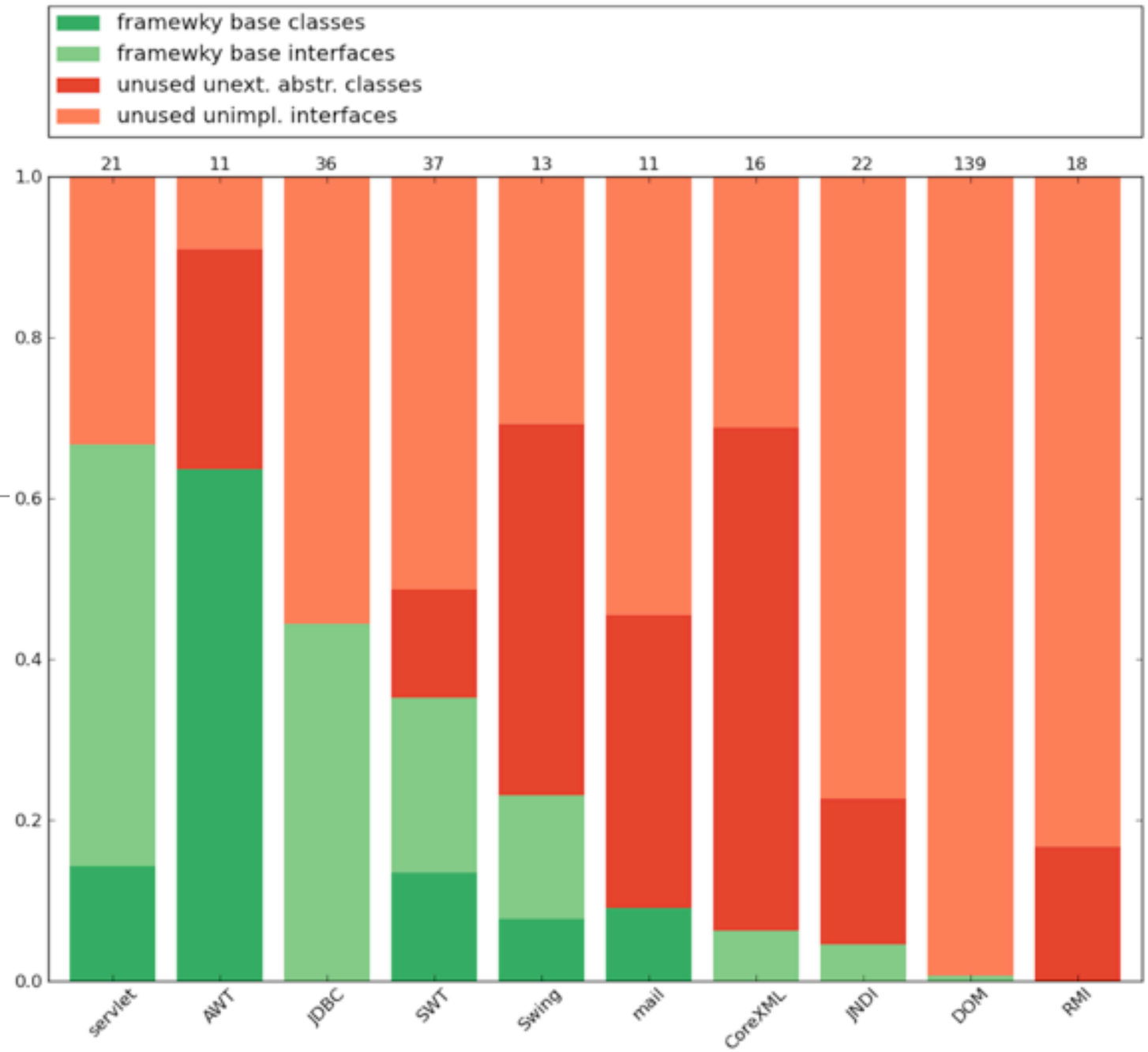
Usage of framework-like design (1)

Usage of requiredness types compared to the total number of extensions



Usage of framework-like design (2)

Usage of different requiredness types compared to the total number of requiredness types



Summary

- New scenario for API usage analysis
- Framework-like usage and design is countable
- Frameworkiness is significant for the domains XML and GUI
- Frameworkiness may be limited to certain areas
  - Hotspots like GUI listeners
- Requiredness types may be implemented by a provider (even unknowingly)

Related work

- Data mining approaches
  - Goals
    - Derivation of usage patterns
    - Detection of hotspots
    - Derivation of specifications
- General approaches
- No framework-specific investigations

Further work

- Different kinds of frameworks
  - Frameworks via annotations and XML configuration files
- Other framework aspects
- Project selection
- API selection

Further work

- Different kinds of frameworks
- Other framework aspects
  - Extensions of methods: Overridden, hidden, implemented methods
- Project selection
- API selection

Further work

- Different kinds of frameworks
- Other framework aspects
- Project selection
  - Larger handpicked corpus
  - More balanced corpus
- API selection

Further work

- Different kinds of frameworks
- Other framework aspects
- Project selection
- API selection
  - Verification of the API pool

Thank you for your attention