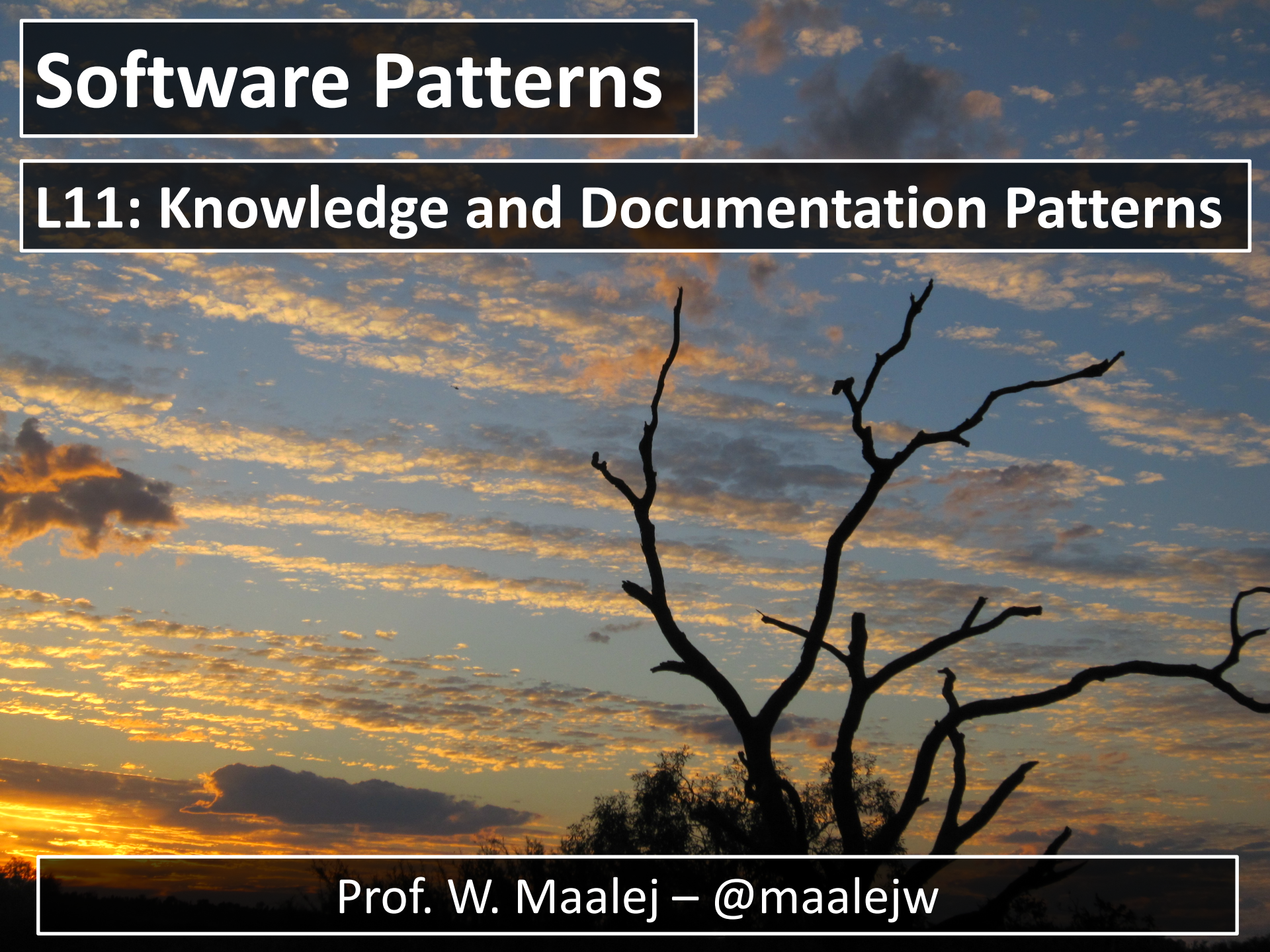
Software Patterns

L11: Knowledge and Documentation Patterns



Prof. W. Maalej – @maalejw

Patterns of Knowledge in API Reference Documentation

Walid Maalej and Martin P. Robillard

Abstract—Reading reference documentation is an important part of programming with application programming interfaces (APIs). Reference documentation complements the API by providing information not obvious from the API syntax. To improve the quality of reference documentation and the efficiency with which the relevant information it contains can be accessed, we must first understand its content. We report on a study of the nature and organization of knowledge contained in the reference documentation of the hundreds of APIs provided as a part of two major technology platforms: Java SDK 6 and .NET 4.0. Our study involved the development of a taxonomy of knowledge types based on grounded methods and independent empirical validation. Seventeen trained coders used the taxonomy to rate a total of 5,574 randomly sampled documentation units to assess the knowledge they contain. Our results provide a comprehensive perspective on the *patterns of knowledge* in API documentation: observations about the types of knowledge it contains and how this knowledge is distributed throughout the documentation. The taxonomy and patterns of knowledge we present in this paper can be used to help practitioners evaluate the content of their API documentation, better organize their documentation, and limit the amount of low-value content. They also provide a vocabulary that can help structure and facilitate discussions about the content of APIs.

Index Terms—API documentation, software documentation, empirical study, content analysis, grounded method, data mining, pattern mining, Java, .NET

1 Introduction

Application programming interfaces (APIs) enable the reuse of libraries and frameworks in software development. In essence, an API is a contract between the component providing a functionality and the component using that functionality (the client). The syntactic information is, in all but the most trivial cases, insufficient to allow a developer to correctly use the API in a programming task. First, interfaces abstract complex behavior, knowledge of which may be necessary to understand a feature. Second, even if the behavior of a component could be completely specified by its interface, developers often need ancillary knowledge about that element: how it relates to domain terms, how to combine it with other elements, and so on [30]. This knowledge is generally provided by documentation, in particular by the API's *reference documentation*.

We define API reference documentation as a set of documents indexed by API element name, where each document specifically provides information about an element (class, method, etc.). For example, the API documentation of the Java Development Toolkit (JDK) is a set of web pages, one for each package or type in the API.

Reference documentation is a necessary and significant part of a framework. For example, the reference documentation of the JDK 6 (SE and EE) totals over three million words, or six times the length of Tolstoy's epic novel *War and Peace*. Reference documentation also plays a crucial role in how developers learn and use an API, and developers can have high expectations about the information they should find therein [14], [30]. Empirical studies have described how developers have numerous and varied questions about the use of APIs (see Section 8). Efficient representation and access of knowledge in API reference documentation is therefore a likely factor for improving software development productivity.

Most technology platforms exposing APIs provide a documentation system with a uniform structure and look-and-feel for presenting and organizing the API documentation. For example, Java APIs are documented through Javadocs, documentation for Python modules can be generated with the pydoc utility, and Microsoft technologies, whose documentation is available through the MSDN website, follow the same look-and-feel. Unfortunately, no

Outline of my talk

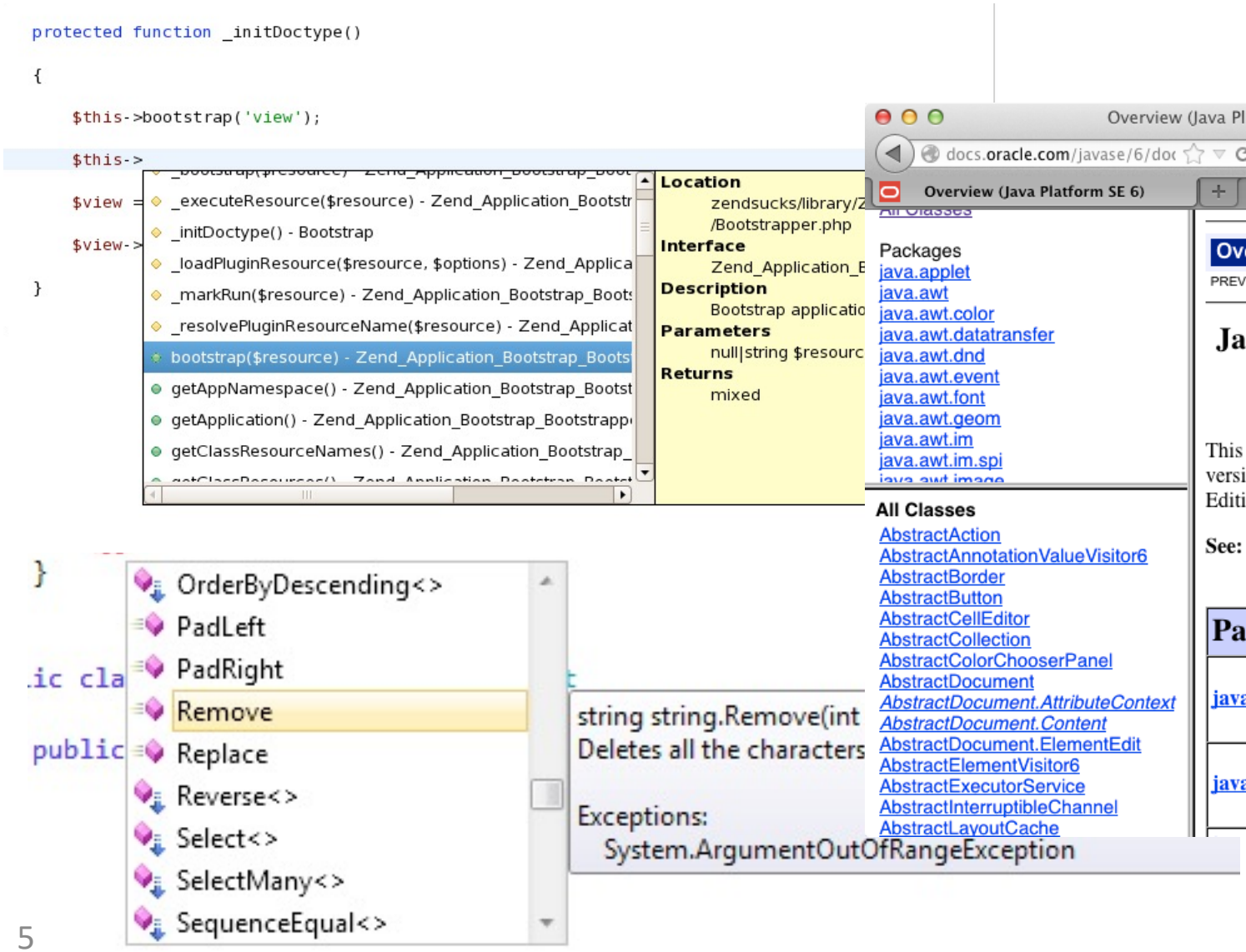
1. **Motivation**
2. Research Setting
3. Patterns of Knowledge
4. Findings & Implications

Software development is a knowledge work

Developers spend more than 50% of their time looking for information

 [Fjeldstad and Hamlen 1979][Ko et al. 2006] [Sillito et al 2007] [Zimmermann & Begel 2014] …

API reference documentation is an important source of knowledge



A .NET example

ServiceModelActivationSectionGroup Class

msdn

.NET Framework 4 | Other Versions

Contains the configuration section for the SMSvcHost.exe tool. This class cannot be inherited.

Inheritance Hierarchy

Namespace: System.ServiceModel.Activation.Configuration
Assembly: System.ServiceModel (in System.ServiceModel.dll)

Syntax

C# | C++ | F# | VB

```
public sealed class ServiceModelActivationSectionGroup : ConfigurationSectionGroup
```

org.apache.http

Interface HttpMessage

Java Example

All Known Subinterfaces:

HttpEntityEnclosingRequest, HttpRequest, HttpResponse

All Known Implementing Classes:

AbstractHttpMessage, BasicHttpEntityEnclosingRequest, BasicHttpRequest, BasicHttpResponse

Reference Documentation of a Type

```
public interface HttpMessage
```

HTTP messages consist of requests from client to server and responses from server to client.

```
HTTP-message   = Request | Response     ; HTTP/1.1 messages
```

HTTP messages use the generic message format of RFC 822 for transferring entities (the payload of the message). Both types of message consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and possibly a message-body.

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
start-line      = Request-Line | Status-Line
```

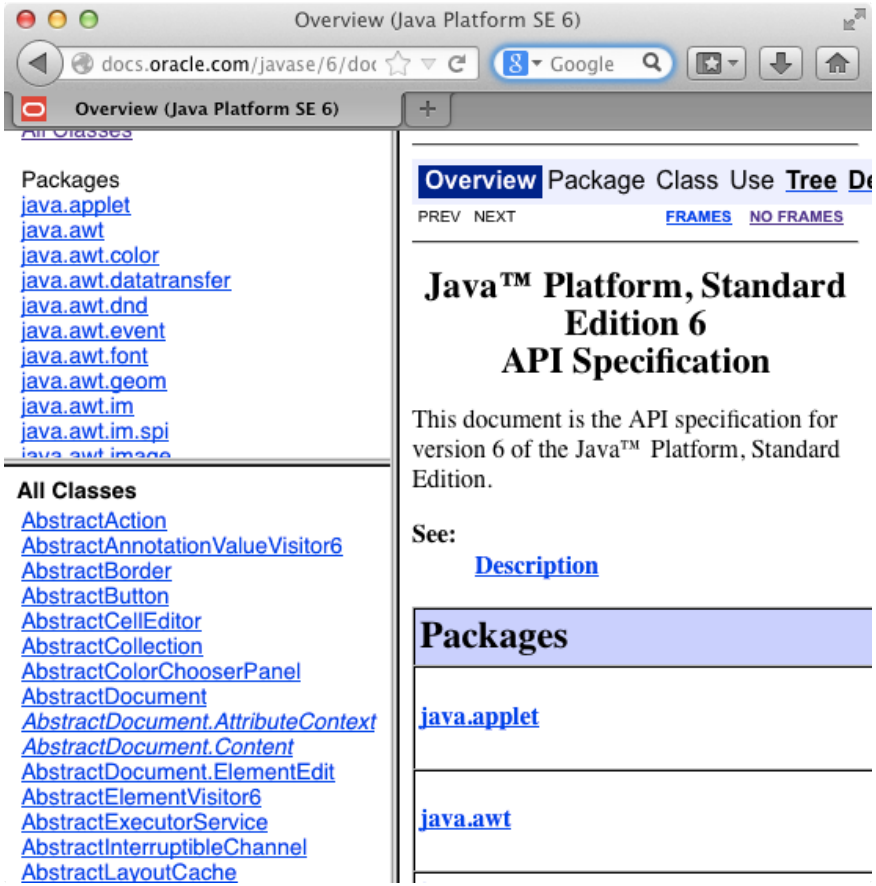
Since:

4.0

Method Summary

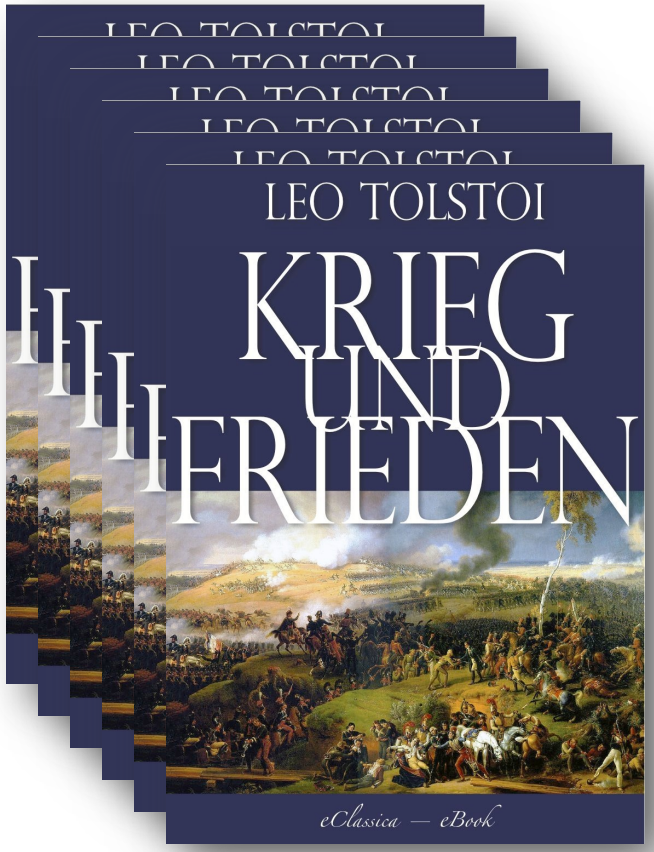
void	**addHeader**(Header header) Adds a header to this message.
void	**addHeader**(String name, String value)

How many words does JDK 6 (SE & EE) includes?



≈

x6!



Efficient knowledge representation and access impacts productivity

Value?
Quality?

What is inside?
How it is organized

Outline of my talk

1 Motivation

2 **Research Setting**

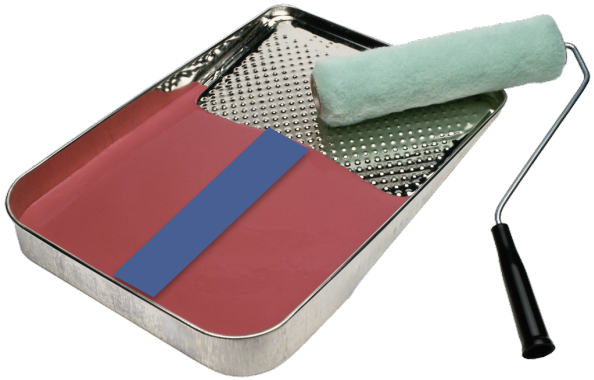
3 Patterns of Knowledge

4 Findings & Implications

Summary of research questions

1. What **types of knowledge** exists in reference documentation?

2. How are these types **distributed** across classes, fields, methods…?



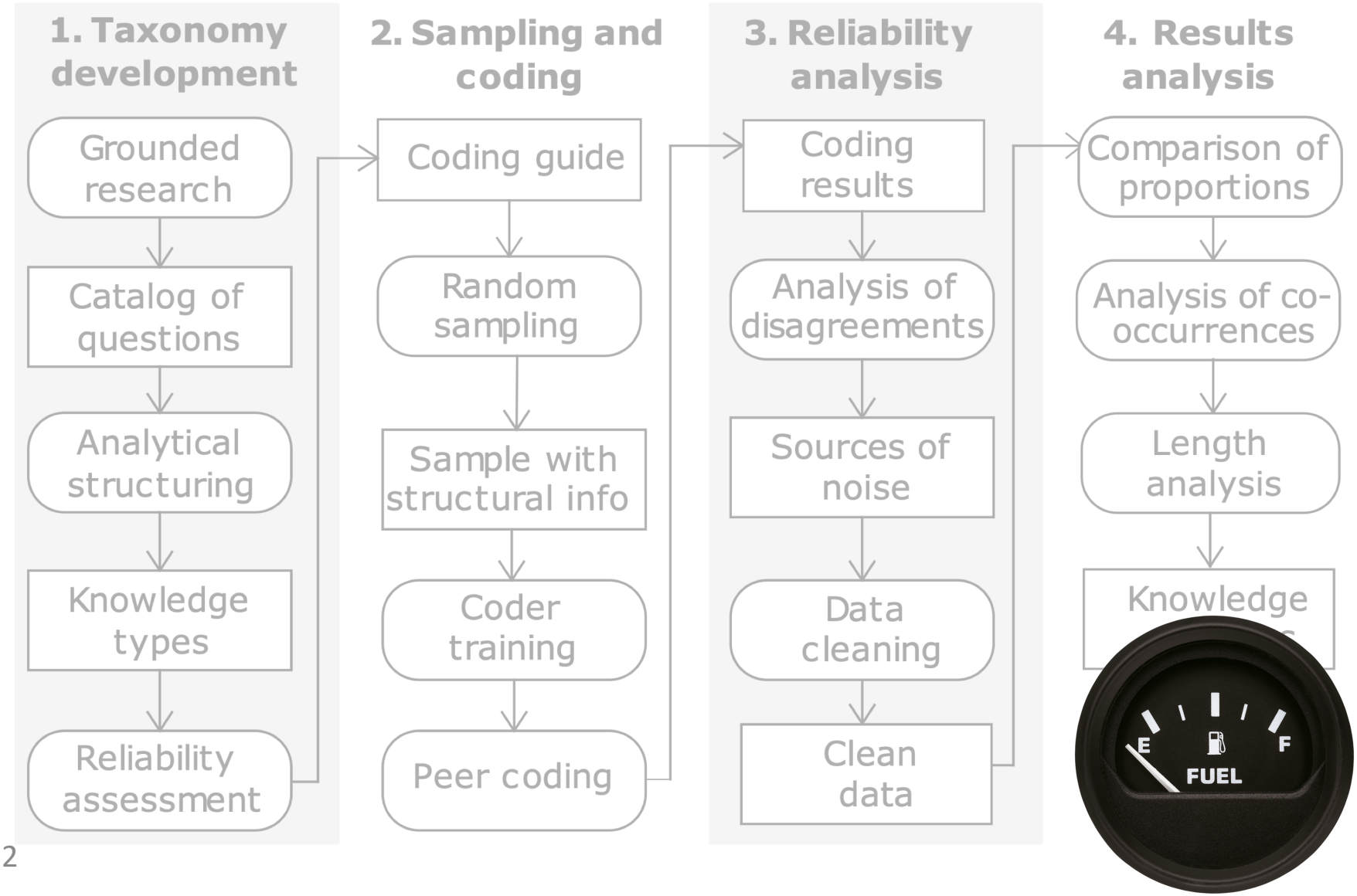
3. How are the different type of knowledge **combined**?

Java

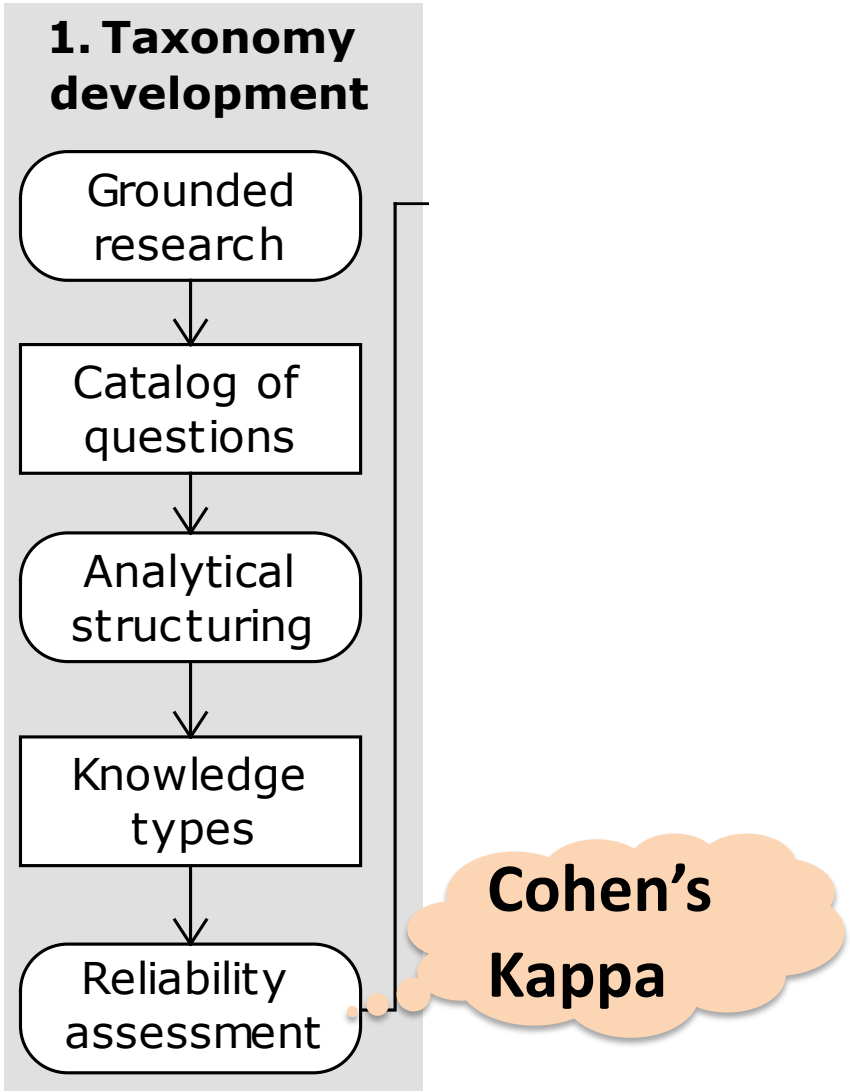
Microsoft® .net™

4. Are there **differences** between different technology platforms?

Overview of research



Overview of research



Coding guide

- Describes the coding task
- Give clear definitions of knowledge types and how to interpret the data
- Gives examples

API Knowledge Coding Guide Version 7.2

You will be presented with **documentation blocks** extracted from API reference documentation (Javadocs and the like). For each block, you will be also presented with the name of its corresponding package/namespace, class, method, or field. Your task is to read each block carefully and evaluate whether the block contains knowledge of the different types described below. You will need to evaluate whether each block contains **knowledge of each different type**. Rate the knowledge type as true only if there is clear evidence that knowledge of that type is present in the block. If you **hesitate** about whether or not to rate a knowledge type as true, leave it as false.

Do not evaluate automatically generated information such as the declaration of an element (e.g. extends MyInterface), or generated links in "specified by". Only evaluate human created documentation in the block (see last section in page 5 for more details).

Read the following description very carefully. It explains how to rate each knowledge type for a given block.

Knowledge Types

Functionality and Behavior

Describes **what** the API does (or does not do) in terms of **functionality** or **features**. The block describes **what happens** when the API **is used** (a field value is set, or a method is called). This **also** includes **specified behavior** such as what an element does, given special input values (for example, null) or what may cause an exception to be raised.

Functionality and behavior knowledge can also be found in the **description of parameters** (e.g., what the element does in response to a specific input), **return values** (e.g., what the API element returns), and **thrown exceptions**.

- *Detects stream close and notifies the watcher*
- *Obtains the SSL session of the underlying connection, if any. If this connection is open, and the underlying socket is an SSLSocket, the SSL session of that socket is obtained. This is a potentially blocking operation.*

Only rate this type as true if the block contains information that actually adds to what is obvious given the complete signature of the API element associated with the block. If a description of functionality only repeats the name of the method or field, it does not contain this type of knowledge and you should rate it as false, and instead rate the knowledge type non-information as true. For example, this would be the case if the documentation for a method called getTitle was

- *Returns the title.*

Similarly for constructors, if the documentation simply states "*Constructs a new X*", "*Instantiates a new object*", or something similar the value is false (with non-information coded as true). In some cases non-information will be phrased to look like a description of functionality, for examples with sentences that start with verbs like "gets", "adds", "determines", "initializes". Carefully read the name and signature of the API element and only assign a value of true for this knowledge type if the block adds something to the description of the element.

However, if any other details are provided, rate this type as true. For example:

- *Creates a new MalformedChallengeException with a null detail message.*

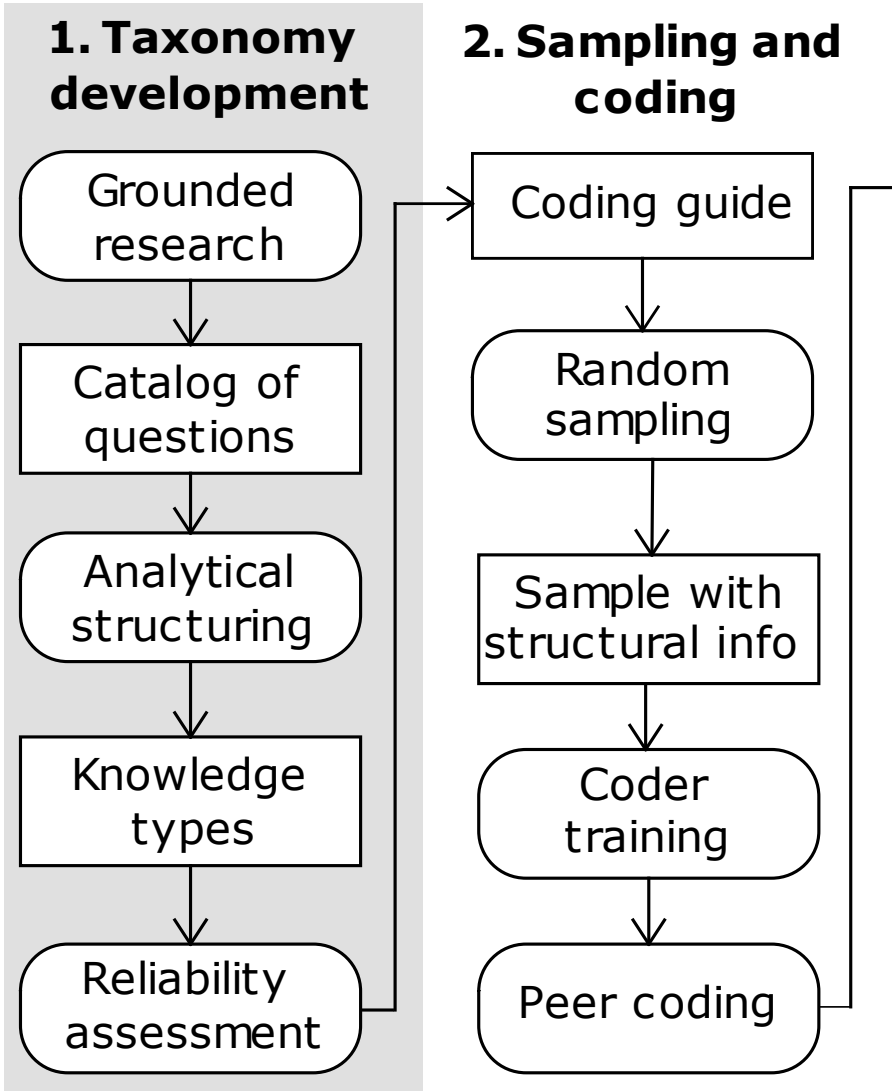
Should get a value of true because of the additional information about the value of the message field.

Mentioning that a **value can be obtained** from a field, property, or getter method does not constitute a description of functionality, except the API performs some additional functions when the value is accessed. For example, the block below does not represent a description of functionality. The Non-information type for this block should be rated as true.

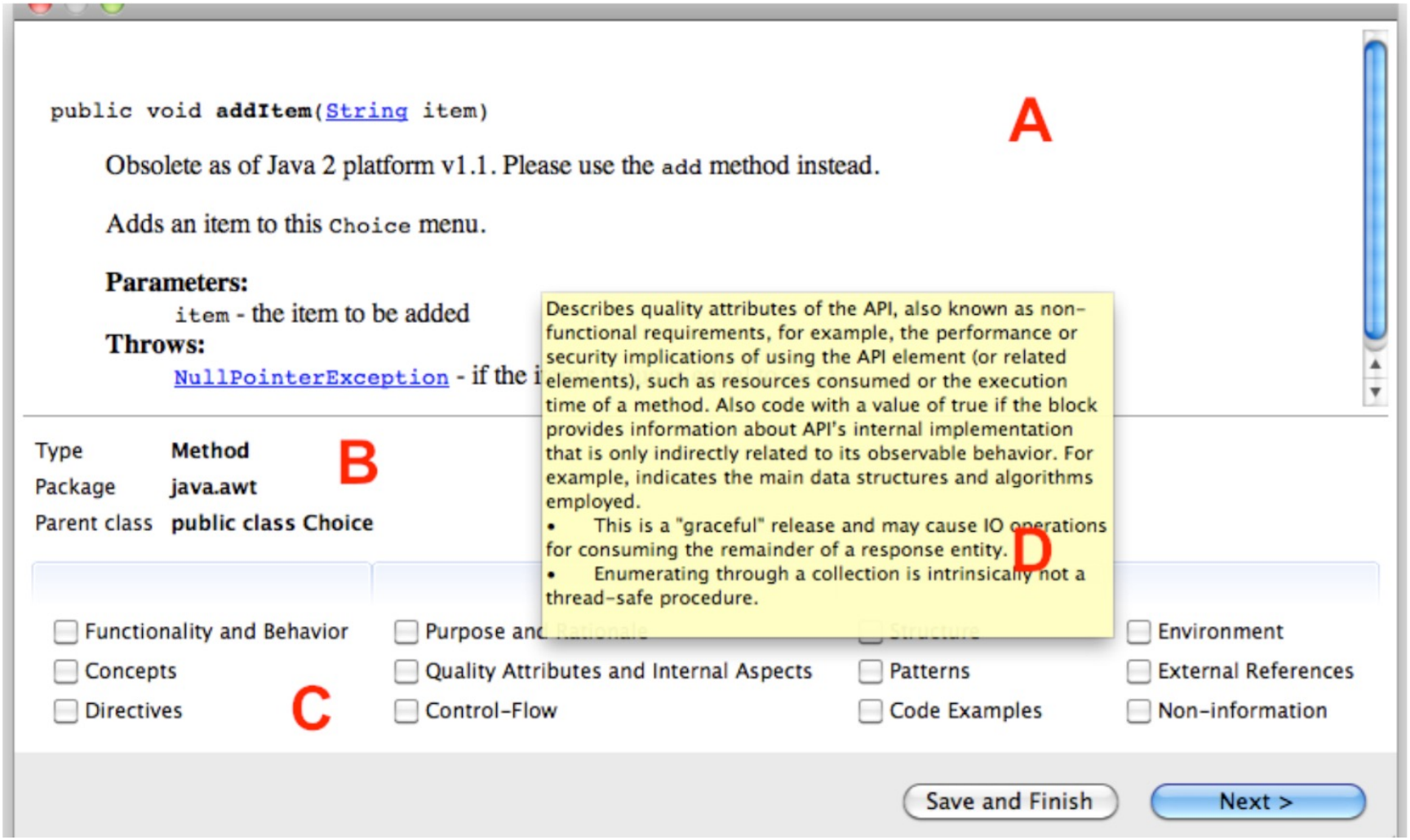
- *[LoggerDescription.Verbosity Property] Gets the verbosity level for the logger.*

Note IMPORTANT: Description of functionality is not limited to the functionality of the element associated with the block, but the API as a whole. **However, if the block explains a sequence of method calls or creation of particular objects (e.g. events) code this as Control-flow.** For example, if setting the value of a field results in some perceived behavior by the framework, this knowledge counts as functionality. If the block describes a resulting sequence of method calls or events fired, this is control flow. If the block contains both, then both should be coded as true.

Overview of research

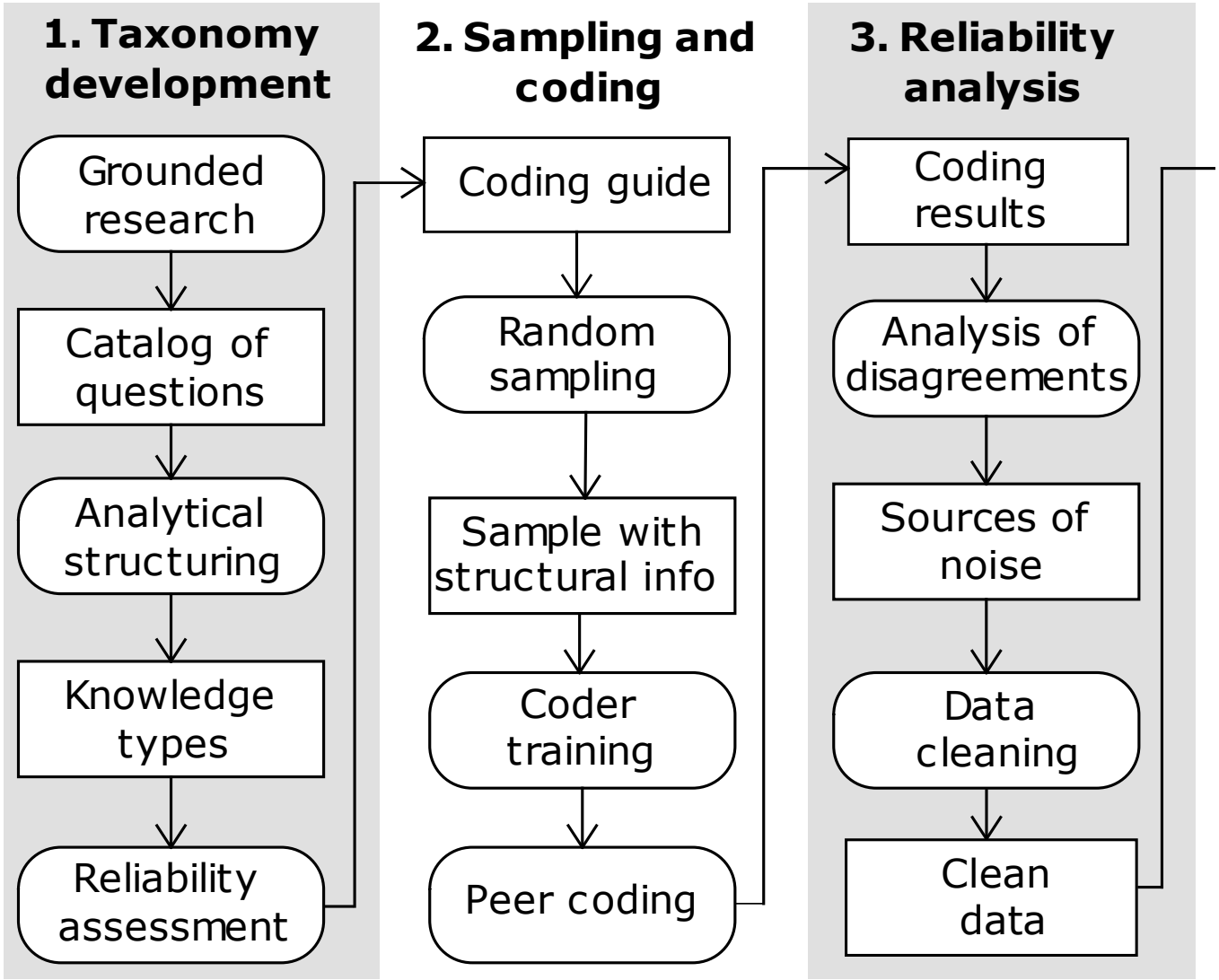


Coding tool - CADO

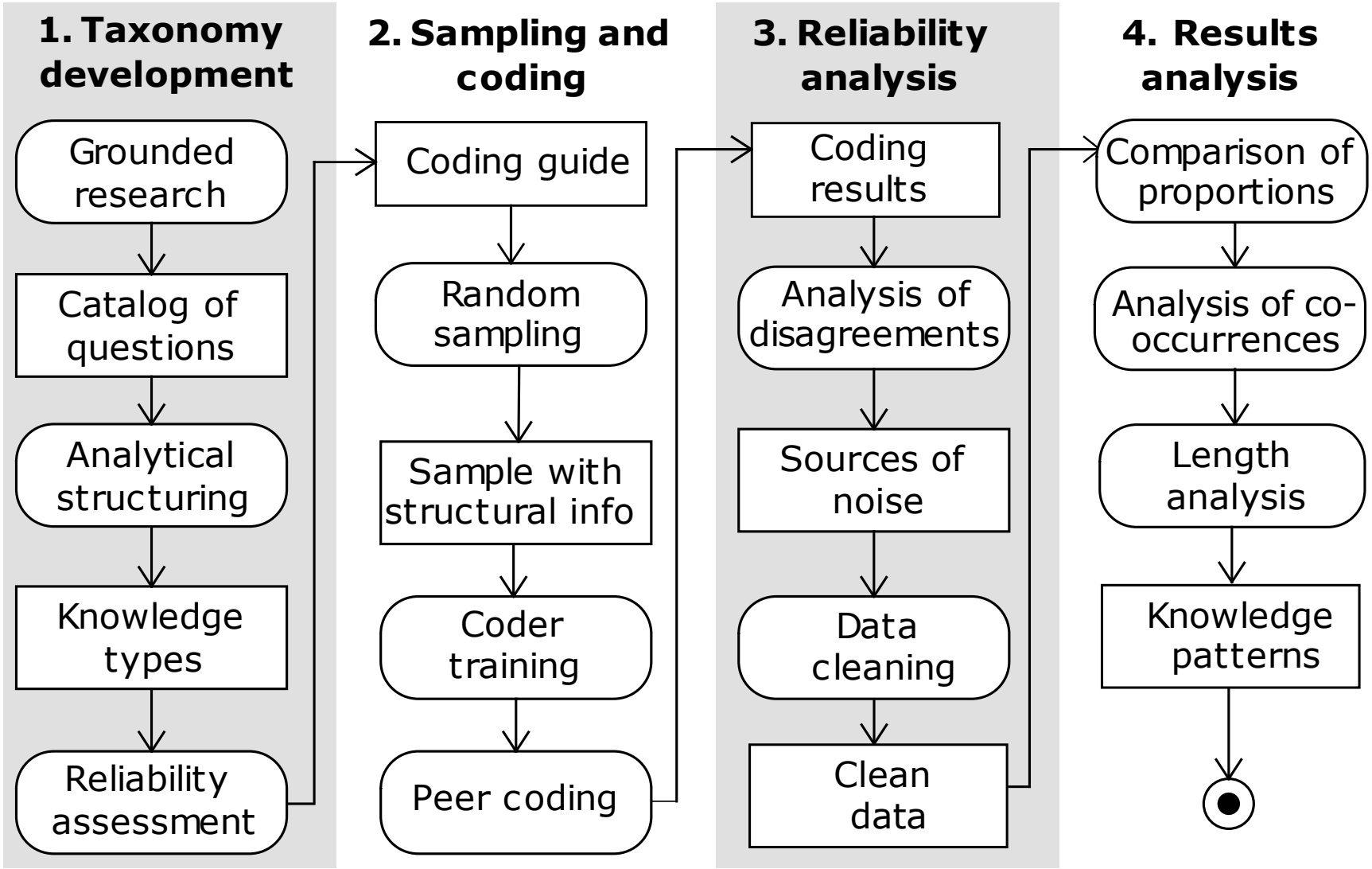


http://apidocumendation.org/

Overview of research



Overview of research



Outline of my talk

1 Motivation

2 Research Setting

3 **Patterns of Knowledge**

4 Findings & Implications

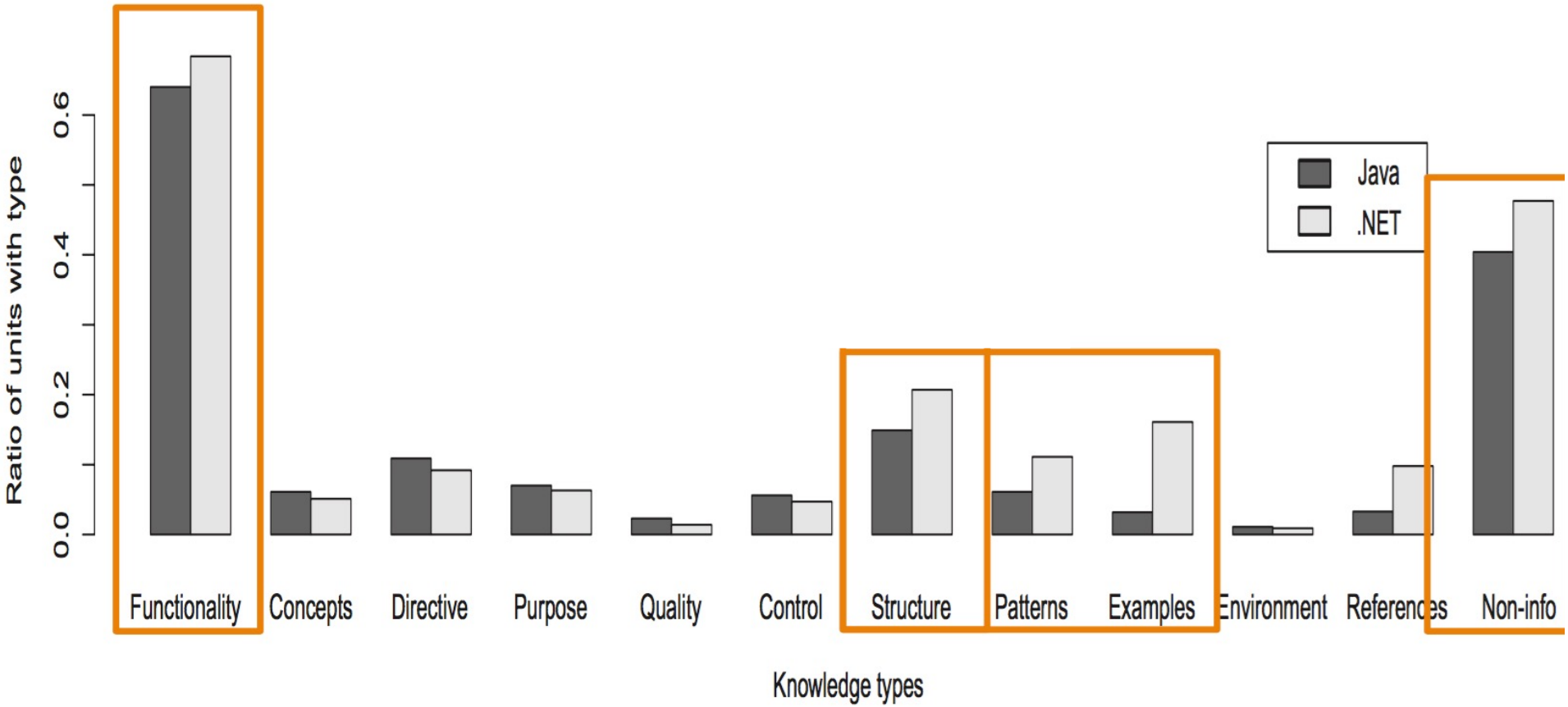
Knowledge types 1-6

Type	Example
Functionality and Behavior	*Detects stream close and notifies the watcher*
Concepts	*Plain sockets may be considered secure, e.g. if they are connected to a known host in the same network segment.*
Directives	*If this method returns false, the caller MUST close the connection to correctly comply with the HTTP protocol.*
Purpose and Rationale	*This allows for the header to be sent without another formatting step.*
Quality Attributes	*This is a "graceful" release and may cause IO operations for consuming the remainder of a response entity*
Control-Flow	*On the client side, this step [the callback] is performed before the request is sent to the server.*

Knowledge types 7-12

Type	Example
Structure	*A Node has five subtypes: Node_Blank, Node_Anon, Node_URI, Node_Variable, and Node_ANY.*
Patterns	*Nodes are only constructed by the node factory methods*
Examples	*The usual execution flow can be demonstrated by the code snippet below: + CODE SNIPPET*
Environment	*This package includes code adapted from Xerces 2.6.0, which is marked and is Copyright ©….*
References	*Digest authentication scheme as defined in RFC2617.*
Non-information	*IOException – If something happens.*

Proportion of knowledge type by documentation unit



Co-occurrence of knowledge types (excerpt)

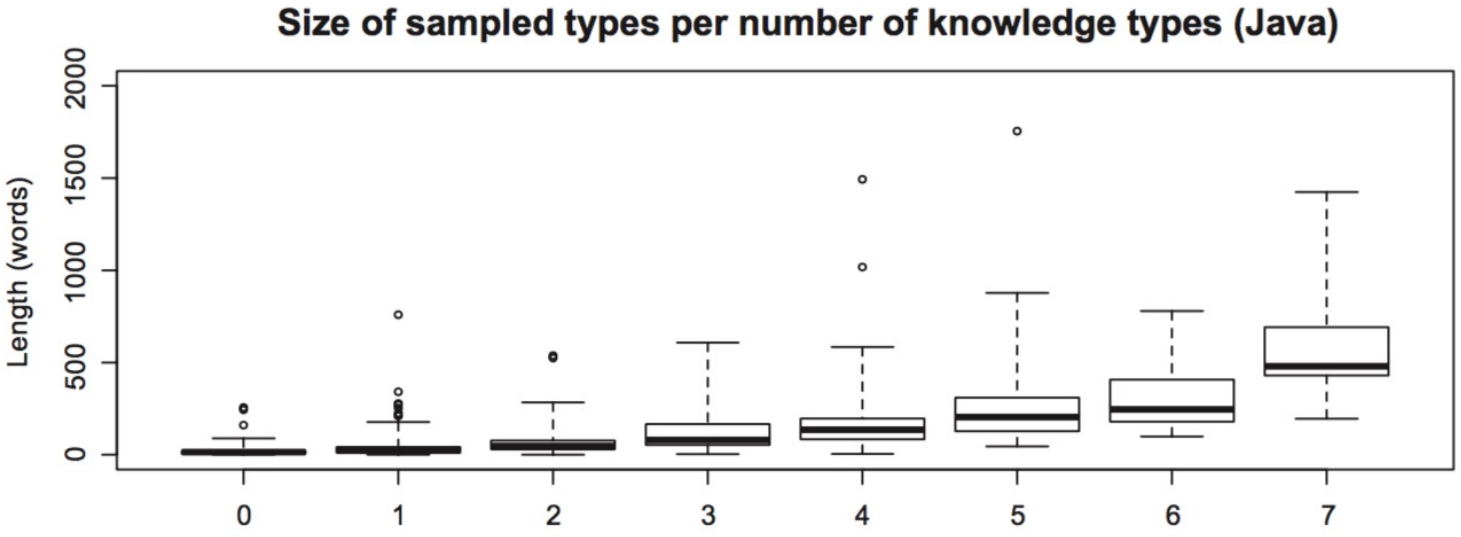
JDK Types (1243 Items)	
Itemset	**Support**
{**Functionality, Structure**}	0.265
{**Functionality, Patterns**}	0.160
{**Functionality, Purpose**}	0.154
{**Functionality, Directives**}	0.141
{**Structure, Patterns**}	0.121
{Functionality, Concepts}	0.120
{**Functionality, Examples**}	0.112
{**Structure, Purpose**}	0.104
{Structure, Directives}	0.103
{**Functionality, Structure, Patterns**}*	0.099
{Functionality, Structure, Directives}*	0.090
{**Functionality, Structure, Purpose**}*	0.083

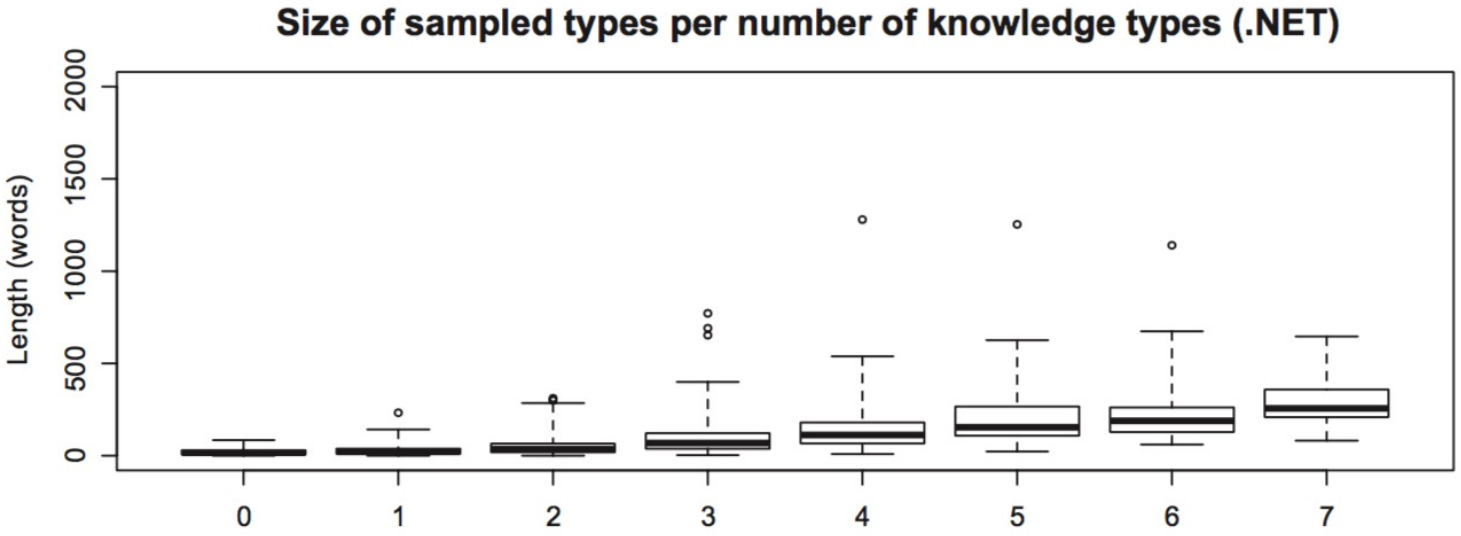
.NET Types (1283 Items)	
Itemset	**Support**
{**Functionality, Structure**}	0.394
{**Functionality, Examples**}	0.255
{Structure, Examples}	0.192
{**Functionality, Patterns**}	0.184
{Functionality, Structure, Examples}	0.163
{**Structure, Patterns**}	0.155
{Functionality, Non-information}	0.154
{Patterns, Examples}	0.136
{**Functionality, Purpose**}	0.133
{Functionality, References}	0.133
{**Functionality, Structure, Patterns**}	0.125
{**Structure, Purpose**}	0.105
{**Functionality, Directives**}	0.104
{Functionality, Patterns, Examples}	0.104
{Structure, Non-information}	0.104
{**Functionality, Structure, Purpose**}*	0.090
{Structure, Patterns, Examples}*	0.088
{Functionality, Structure, Non-information}*	0.081

JDK Members (1549 Items)	
Itemset	**Support**
{**Functionality, Non-information**}	0.267
{**Functionality, Structure**}	0.112

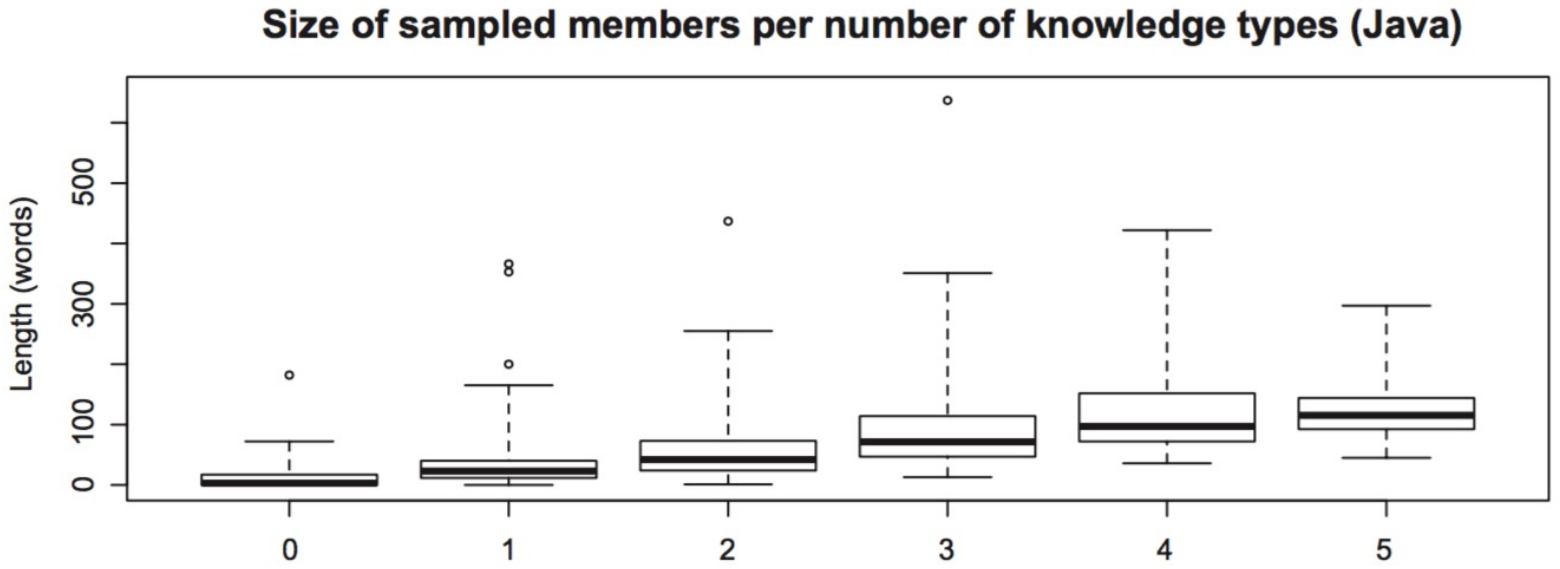
.NET Members (1499 Items)	
Itemset	**Support**
{**Functionality, Non-information**}	0.292
{**Functionality, Structure**}	0.142
{Functionality, Examples}	0.113

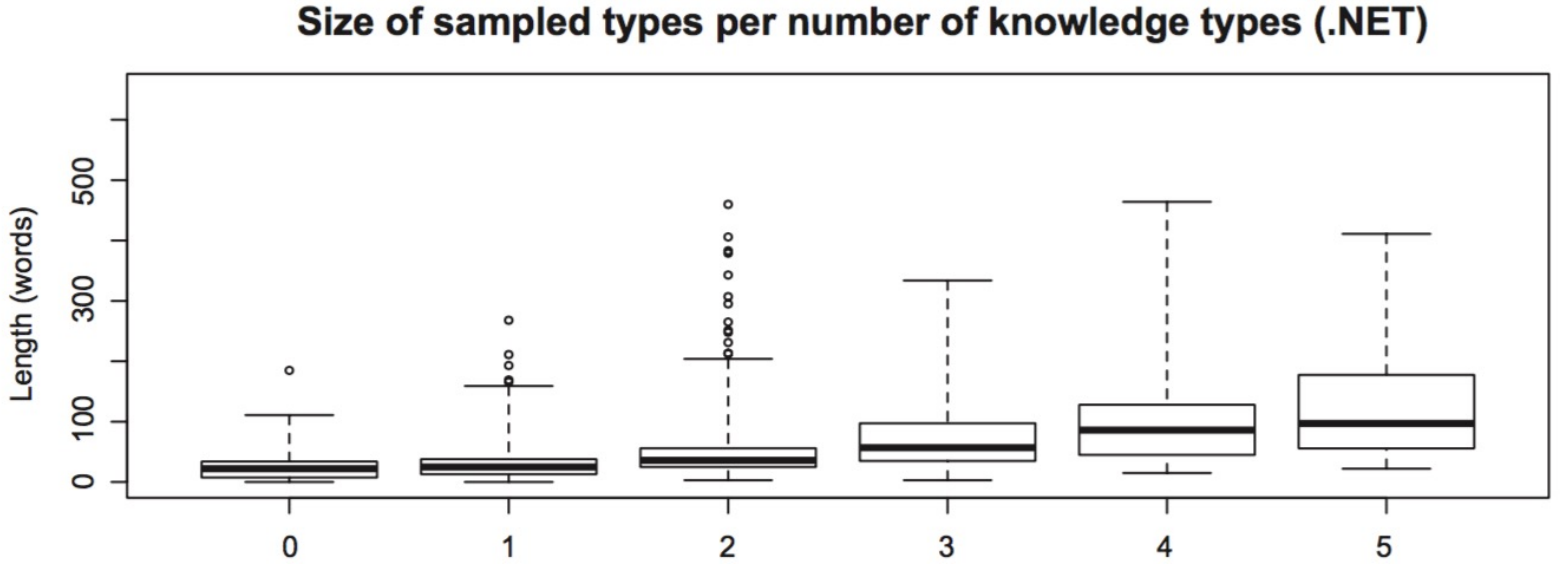
Length analysis – classes and interfaces





Length analysis – fields and methods





Outline of my talk

1 Motivation

2 Research Setting

3 Patterns of Knowledge

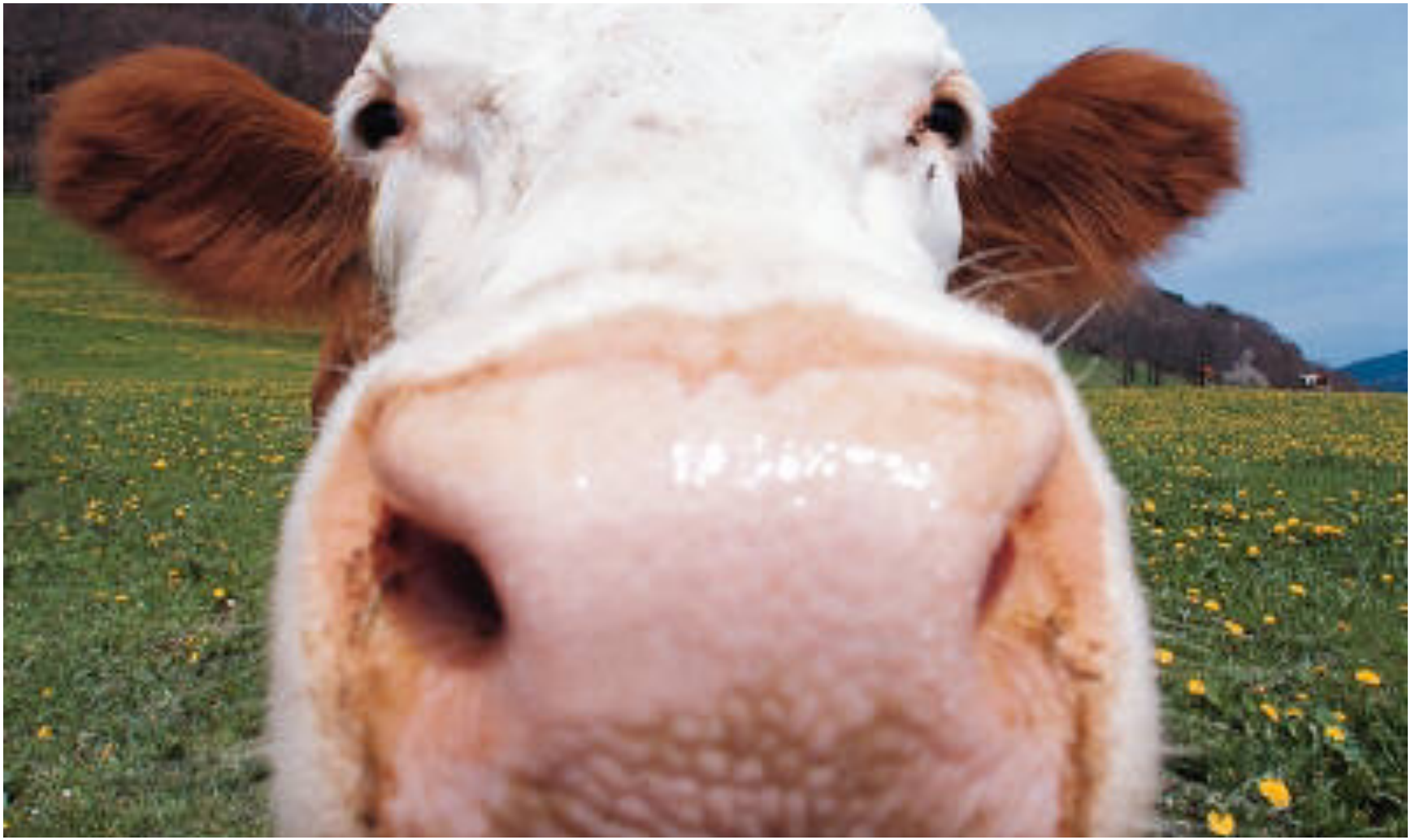
4 **Findings & Implications**

Main contributions

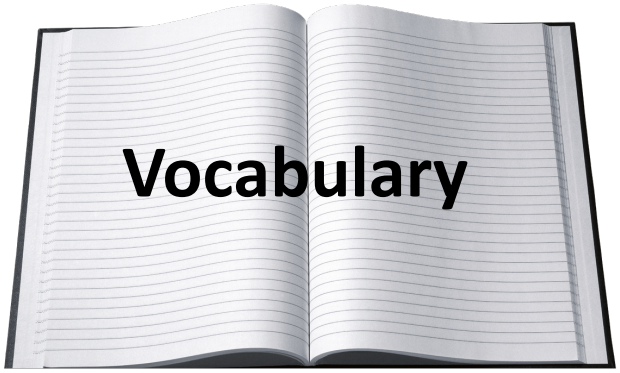
Validated taxonomy of knowledge types

Patterns of knowledge in API documentation

So what?

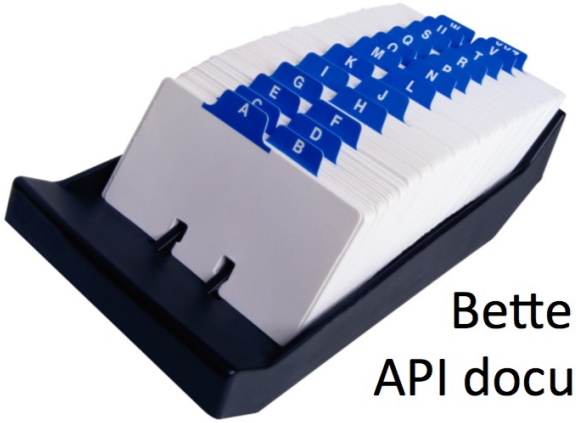


Implications



Structure content and facilitate **discussions** API documentation

Evaluate documentation content in relation to the knowledge types



Better **organize** API documentation

Insights into documentation **quality**

Differences in documentation styles

- More “conceptual” types of knowledge (Concepts and Purpose)
- More Directives
- Directives are correlated with Structure and Examples
- Concepts, Purpose, & Quality in interfaces (vs. classes)

- More “structural” types of knowledge (Structure and Patterns)
- More Examples
- Most of the fields contain documentation about the Functionality of the API

Which style is better?

Summary

1. A study of Java and .NET reference documentation resulting in a taxonomy of **knowledge types** and their **distribution**

2. Insights for API designers to improve the **quality** of documentation and the **efficiency** of accessing its information

3. Motivate the use of **content analysis** as a **reliable** and useful research methods in software engineering research