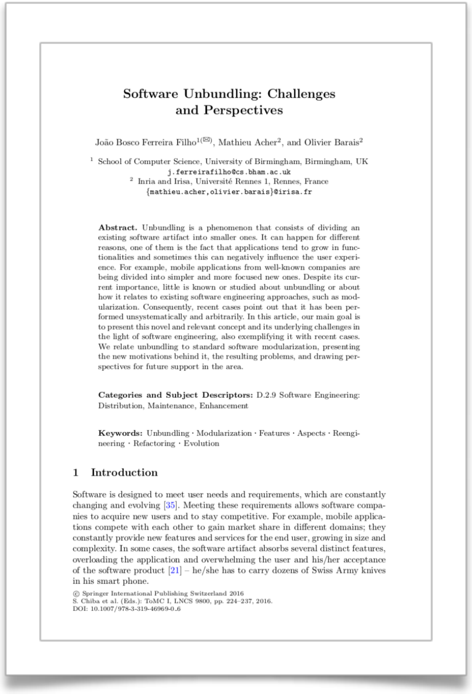
Splitting APIs: An Exploratory Study of Software Unbundling

UFC

Anderson Severo de Matos
João Bosco Ferreira Filho
Lincoln Souza Rocha

Software Unbundling

- *“Unbundling consists of dividing an existing software artifact into smaller ones, each one serving to different end use purposes”[1]*



Software Unbundling: Challenges and Perspectives

João Bosco Ferreira Filho[1](✉), Mathieu Acher[2], and Olivier Barais[2]

[1] School of Computer Science, University of Birmingham, Birmingham, UK
j.ferreirafilho@cs.bham.ac.uk
[2] Inria and Irisa, Université Rennes 1, Rennes, France
{mathieu.acher,olivier.barais}@irisa.fr

Abstract. Unbundling is a phenomenon that consists of dividing an existing software artifact into smaller ones. It can happen for different reasons, one of them is the fact that applications tend to grow in functionalities and sometimes this can negatively influence the user experience. For example, mobile applications from well-known companies are being divided into simpler and more focused new ones. Despite its current importance, little is known or studied about unbundling or about how it relates to existing software engineering approaches, such as modularization. Consequently, recent cases point out that it has been performed unsystematically and arbitrarily. In this article, our main goal is to present this novel and relevant concept and its underlying challenges in the light of software engineering, also exemplifying it with recent cases. We relate unbundling to standard software modularization, presenting the new motivations behind it, the resulting problems, and drawing perspectives for future support in the area.

Categories and Subject Descriptors: D.2.9 Software Engineering: Distribution, Maintenance, Enhancement

Keywords: Unbundling · Modularization · Features · Aspects · Reengineering · Refactoring · Evolution

1 Introduction

Software is designed to meet user needs and requirements, which are constantly changing and evolving [35]. Meeting these requirements allows software companies to acquire new users and to stay competitive. For example, mobile applications compete with each other to gain market share in different domains; they constantly provide new features and services for the end user, growing in size and complexity. In some cases, the software artifact absorbs several distinct features, overloading the application and overwhelming the user and his/her acceptance of the software product [21] – he/she has to carry dozens of Swiss Army knives in his smart phone.

© Springer International Publishing Switzerland 2016
S. Chiba et al. (Eds.): ToMC I, LNCS 9800, pp. 224–237, 2016.
DOI: 10.1007/978-3-319-46969-0_6

Software Unbundling

- *“Unbundling consists of dividing an existing software artifact into smaller ones, each one serving to different end use purposes”*[1]

Software Unbundling: Challenges and Perspectives

João Bosco Ferreira Filho[1](✉), Mathieu Acher[2], and Olivier Barais[2]

[1] School of Computer Science, University of Birmingham, Birmingham, UK
j.ferreirafilho@cs.bham.ac.uk
[2] Inria and Irisa, Université Rennes 1, Rennes, France
{mathieu.acher,olivier.barais}@irisa.fr

Abstract. Unbundling is a phenomenon that consists of dividing an existing software artifact into smaller ones. It can happen for different reasons, one of them is the fact that applications tend to grow in functionalities and sometimes this can negatively influence the user experience. For example, mobile applications from well-known companies are being divided into simpler and more focused new ones. Despite its current importance, little is known or studied about unbundling or about how it relates to existing software engineering approaches, such as modularization. Consequently, recent cases point out that it has been performed unsystematically and arbitrarily. In this article, our main goal is to present this novel and relevant concept and its underlying challenges in the light of software engineering, also exemplifying it with recent cases. We relate unbundling to standard software modularization, presenting the new motivations behind it, the resulting problems, and drawing perspectives for future support in the area.

Categories and Subject Descriptors: D.2.9 Software Engineering: Distribution, Maintenance, Enhancement

Keywords: Unbundling · Modularization · Features · Aspects · Reengineering · Refactoring · Evolution

1 Introduction

Software is designed to meet user needs and requirements, which are constantly changing and evolving [35]. Meeting these requirements allows software companies to acquire new users and to stay competitive. For example, mobile applications compete with each other to gain market share in different domains; they constantly provide new features and services for the end user, growing in size and complexity. In some cases, the software artifact absorbs several distinct features, overloading the application and overwhelming the user and his/her acceptance of the software product [21] – he/she has to carry dozens of Swiss Army knives in his smart phone.

© Springer International Publishing Switzerland 2016
S. Chiba et al. (Eds.): ToMC I, LNCS 9800, pp. 224–237, 2016.
DOI: 10.1007/978-3-319-46969-0_6

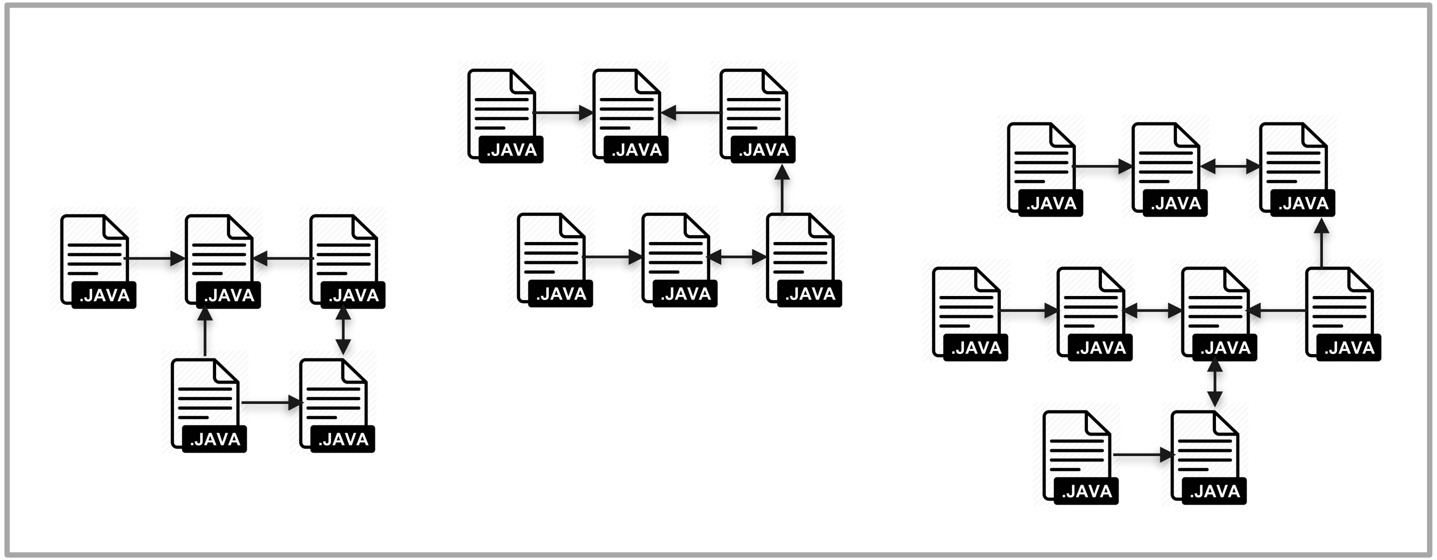
Why unbundling is important?

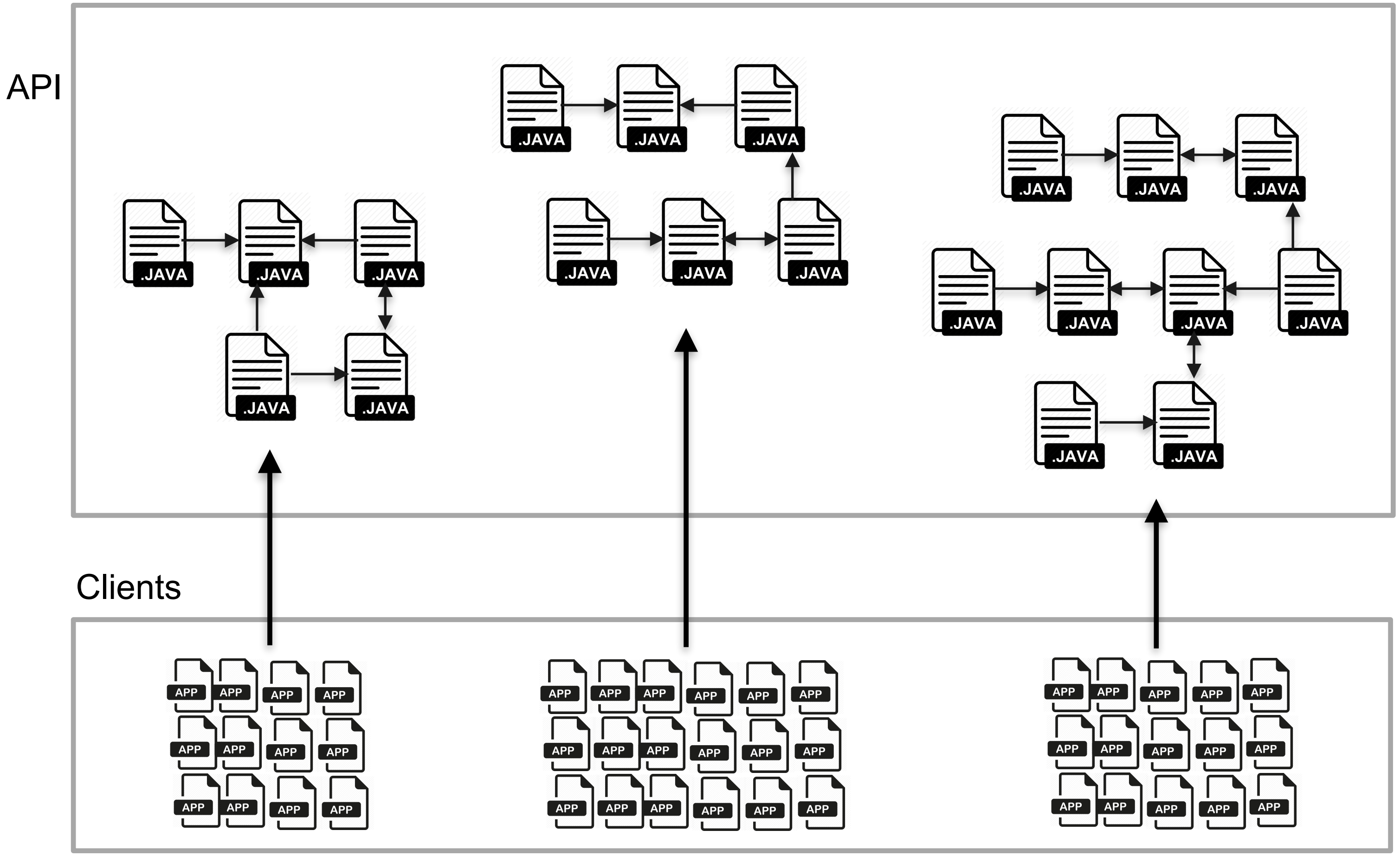
- It removes dead code injected through the software evolution
- It isolates features that may be diverging from the goals of the original software
- It promotes an emerging feature to become an application

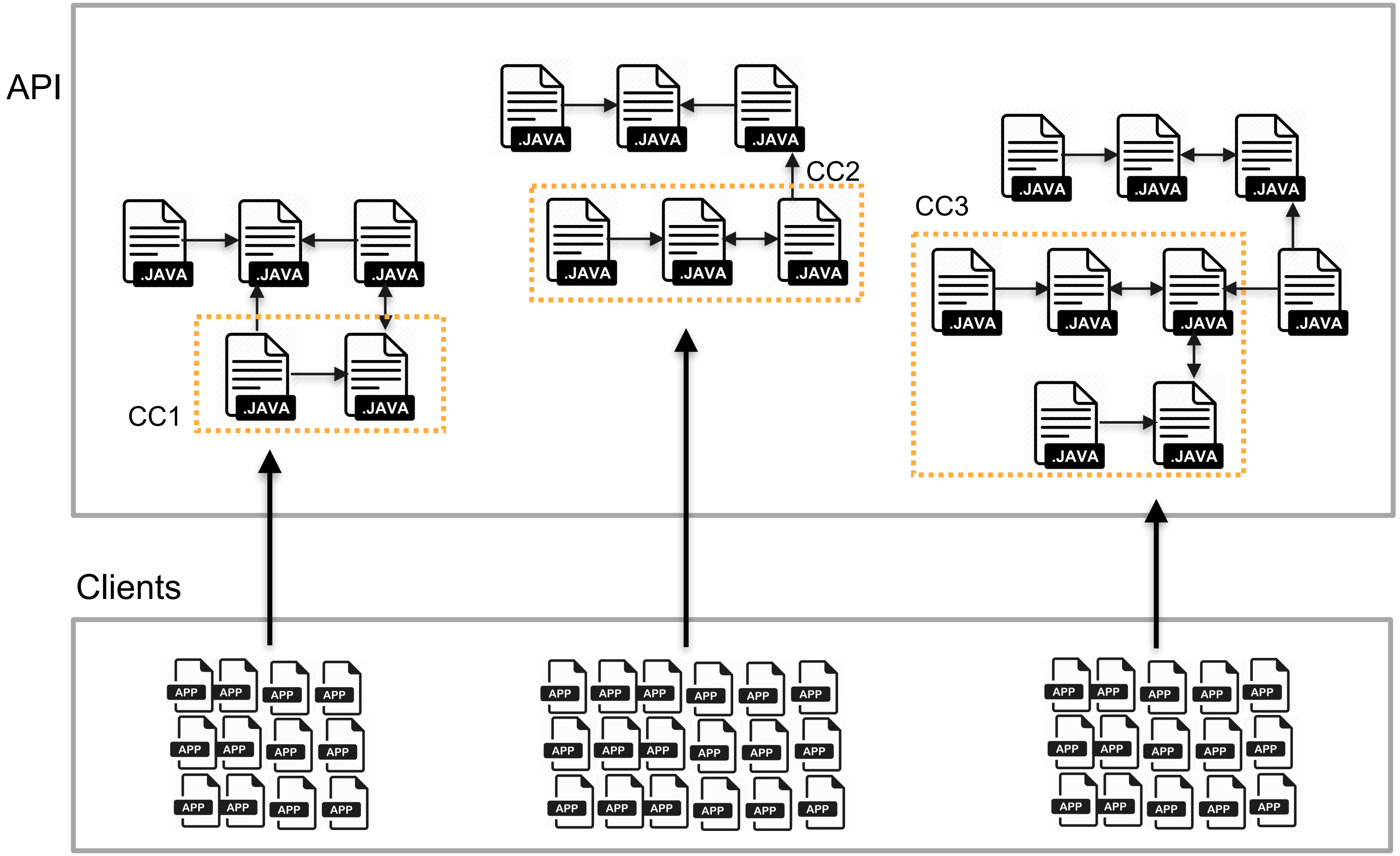
How do you decide which part of the code should become a new application?

Unbundling by usage

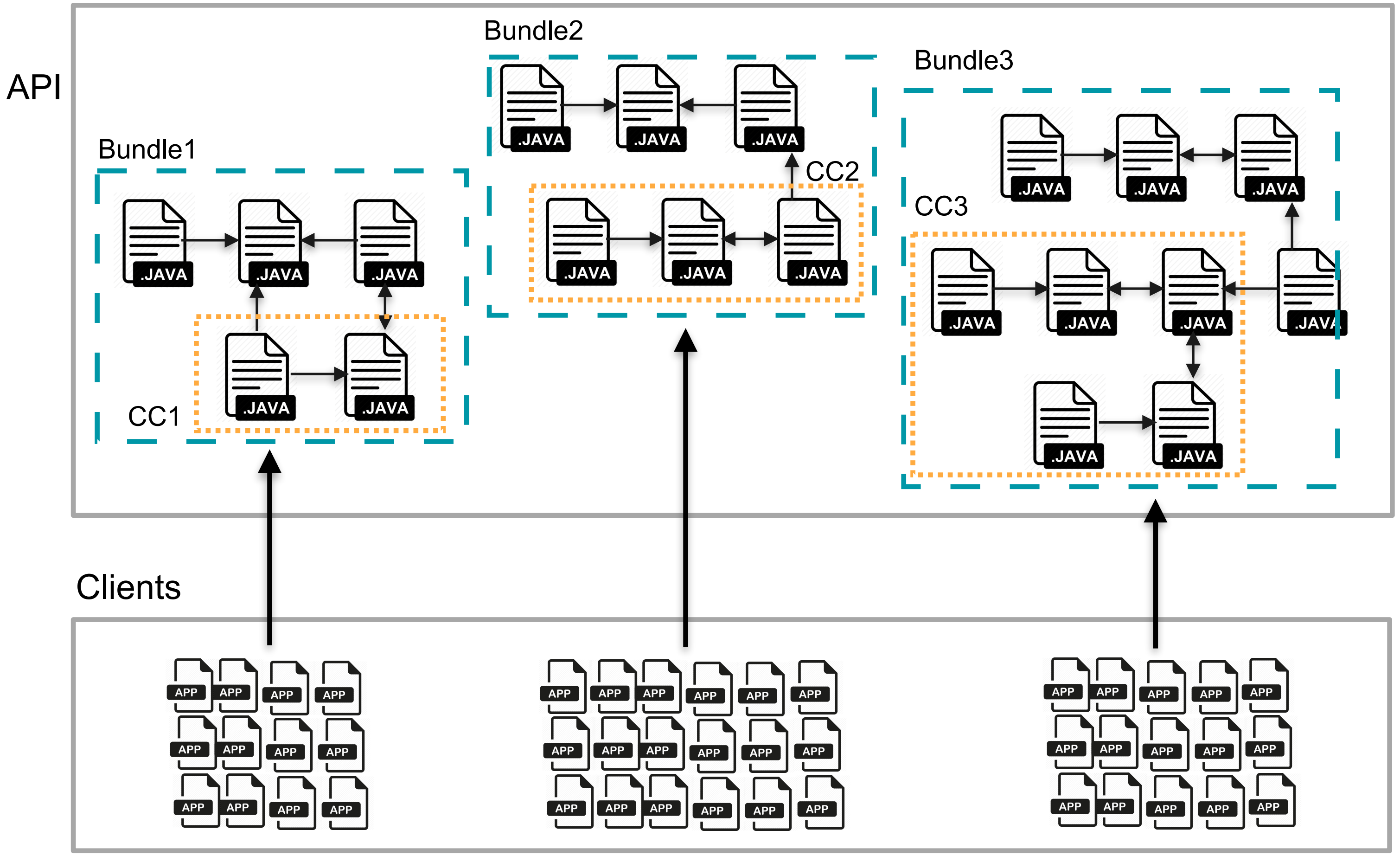
API




API
.JAVA
Clients
APP

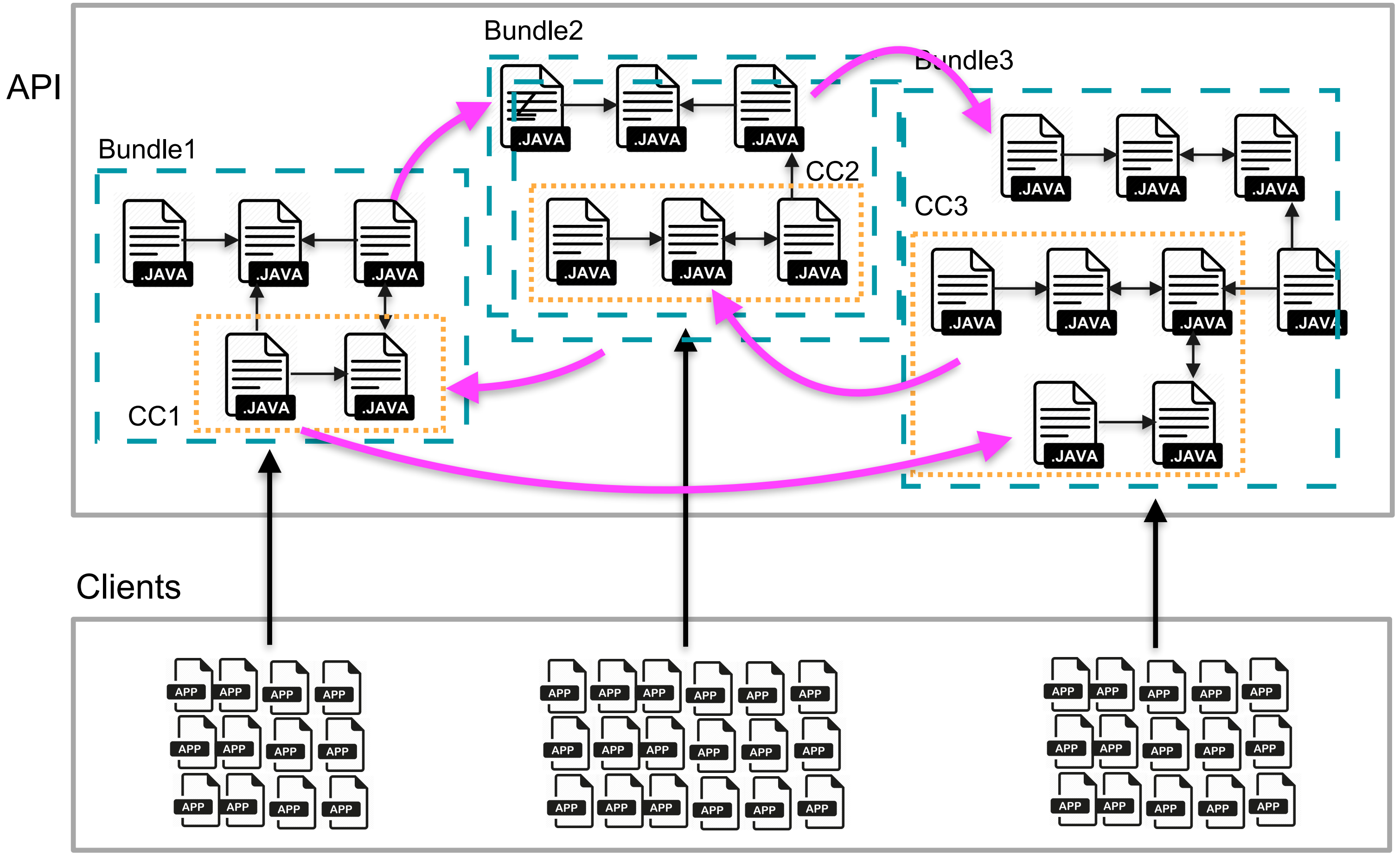


CC: Cluster of Classes



CC: Cluster of Classes

Bundle: a set of Java classes required to compile a cluster of classes



CC: Cluster of Classes

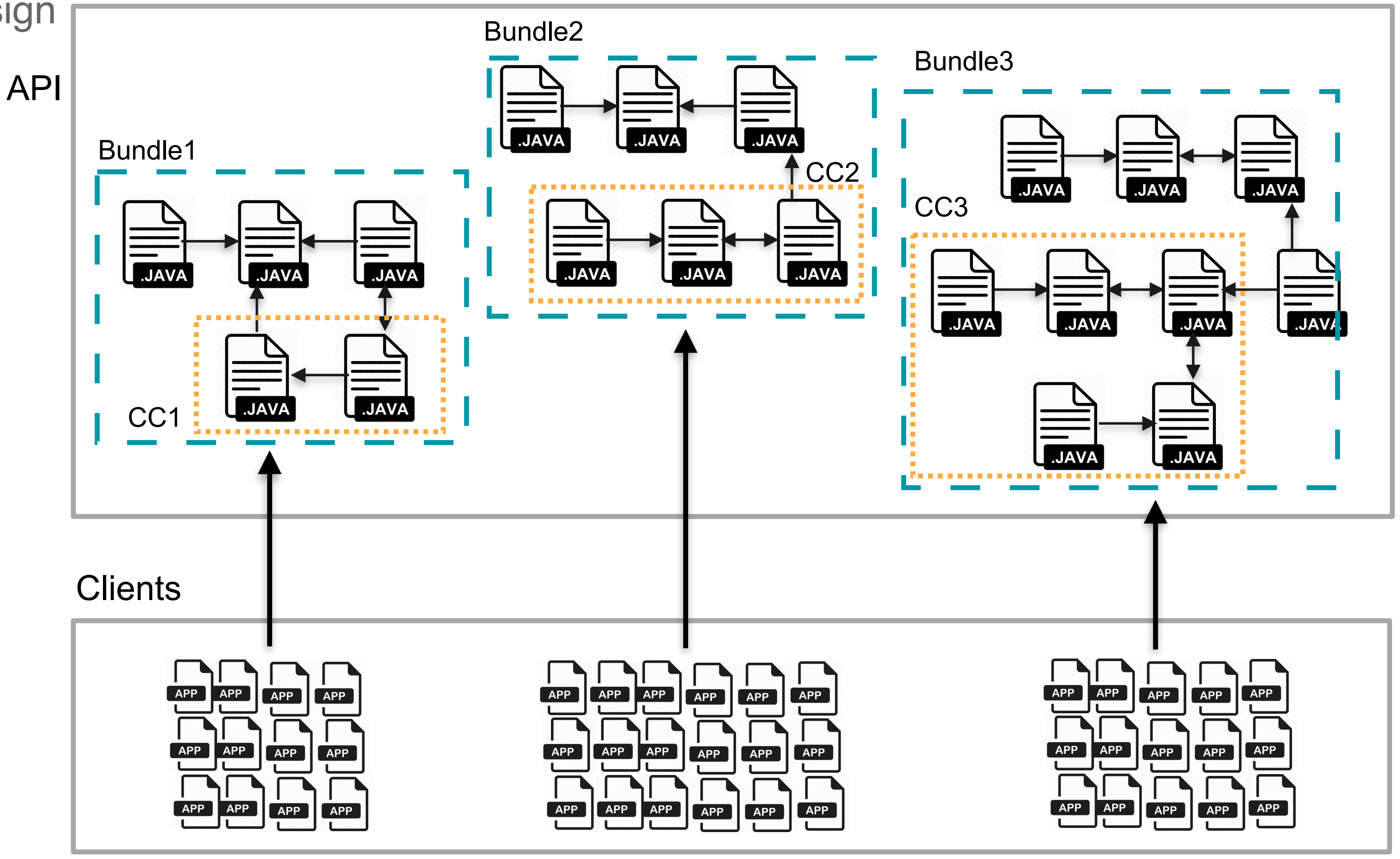
Bundle: a set of Java classes required to compile a cluster of classes

Representativeness

Uniqueness

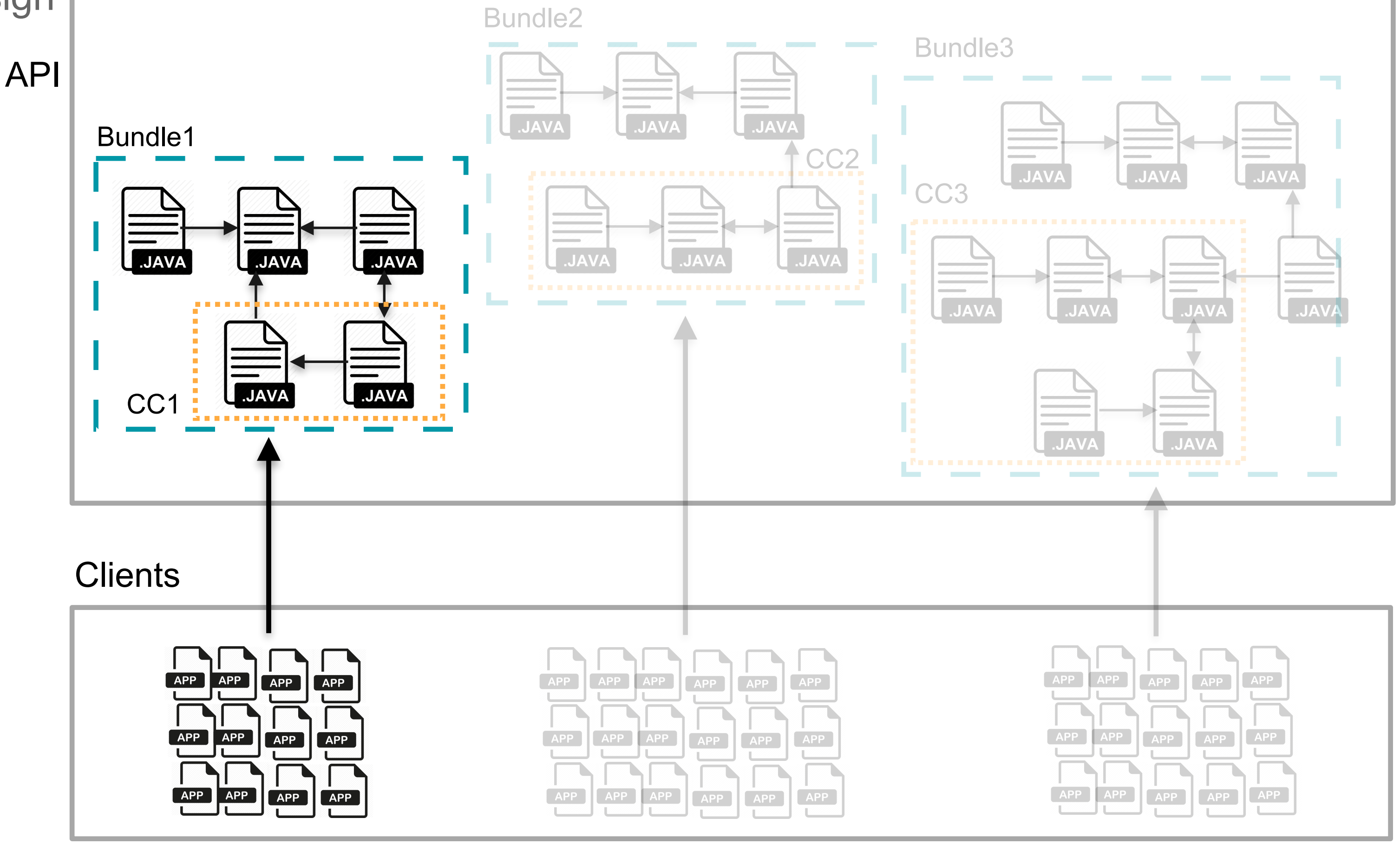
Uniqueness: the percentage of classes that intersect with different bundles

Study Design



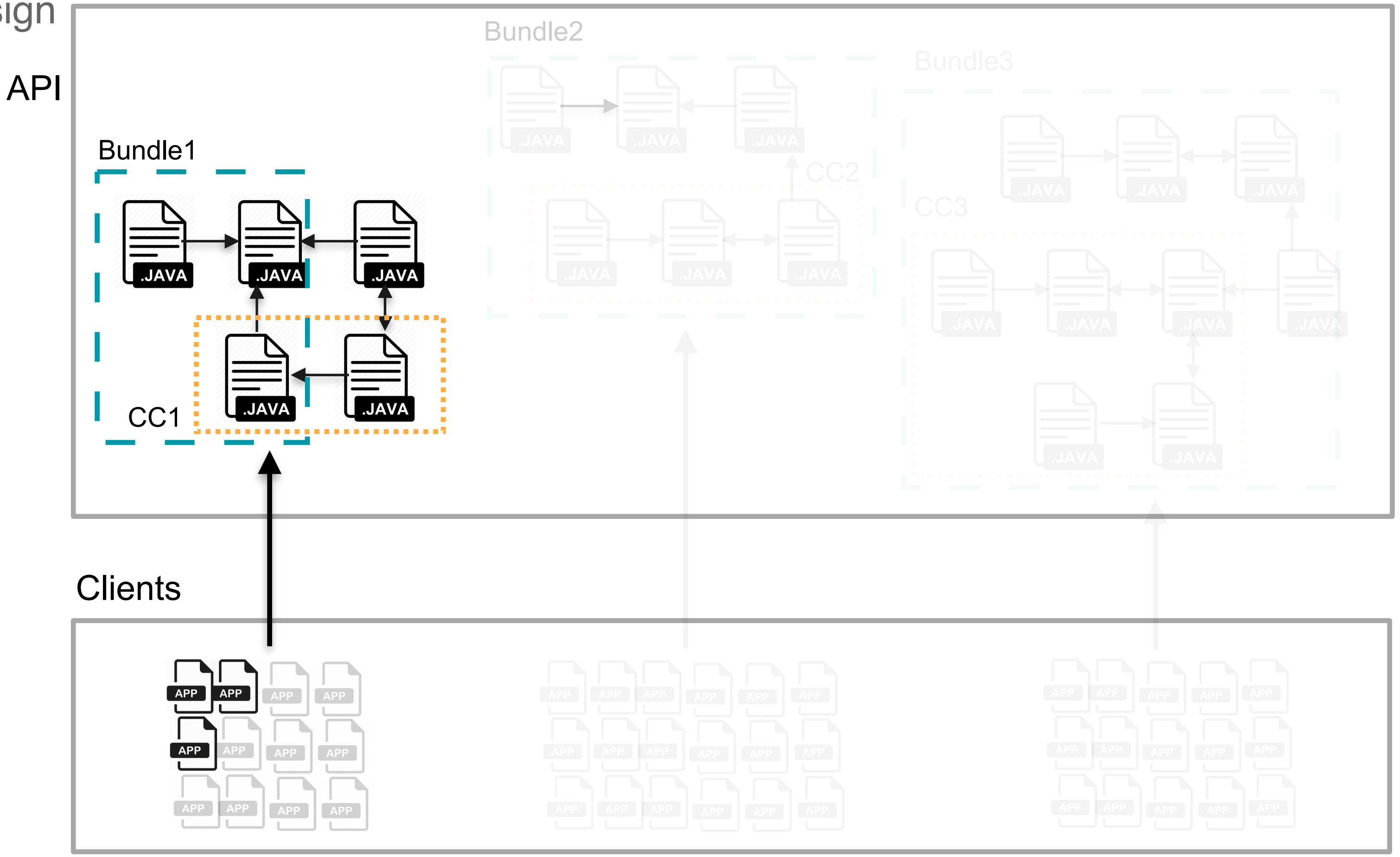
Representativeness: the percentage of clients a bundle could serve

Study Design

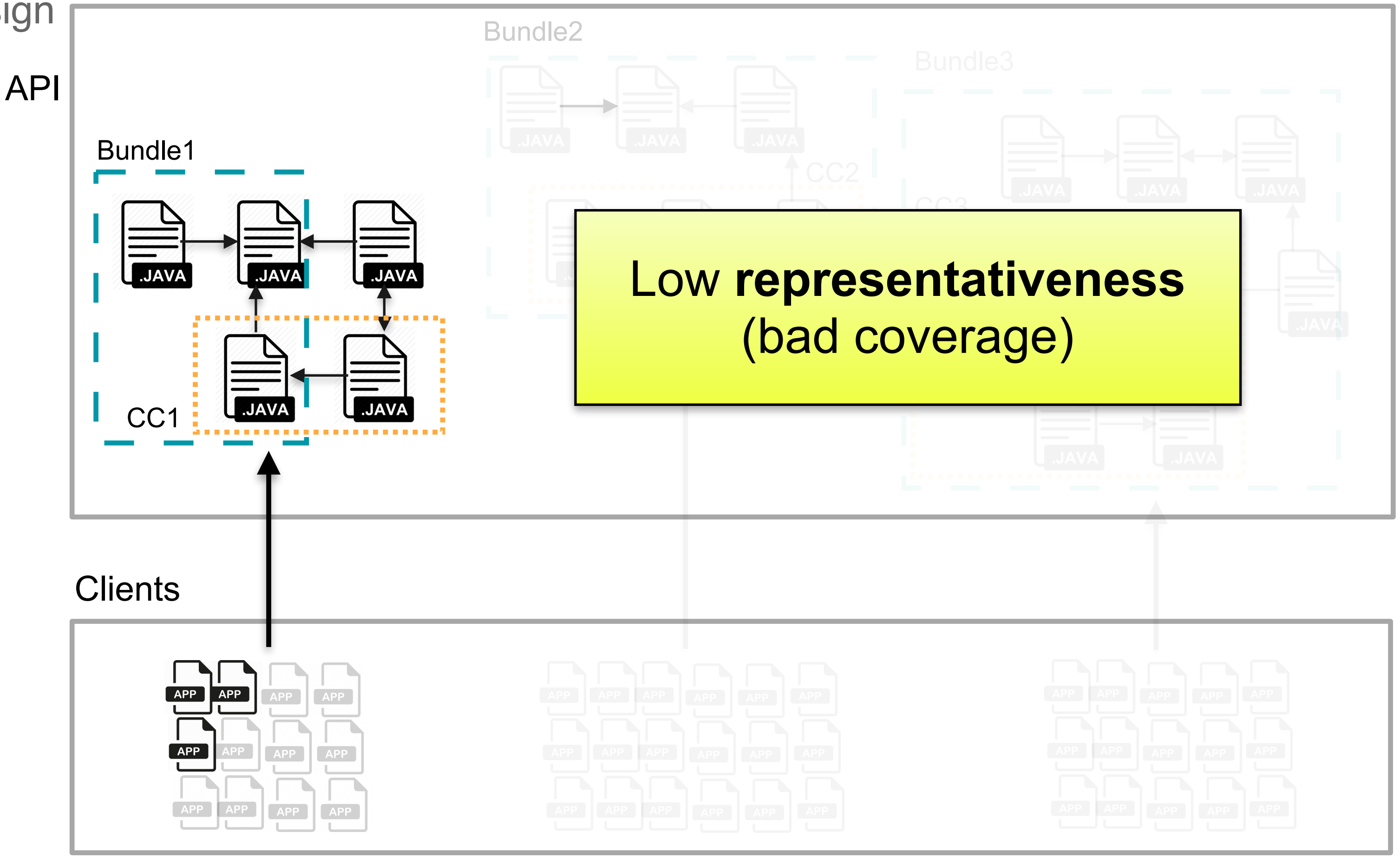


Representativeness: the percentage of clients a bundle could serve

Study Design



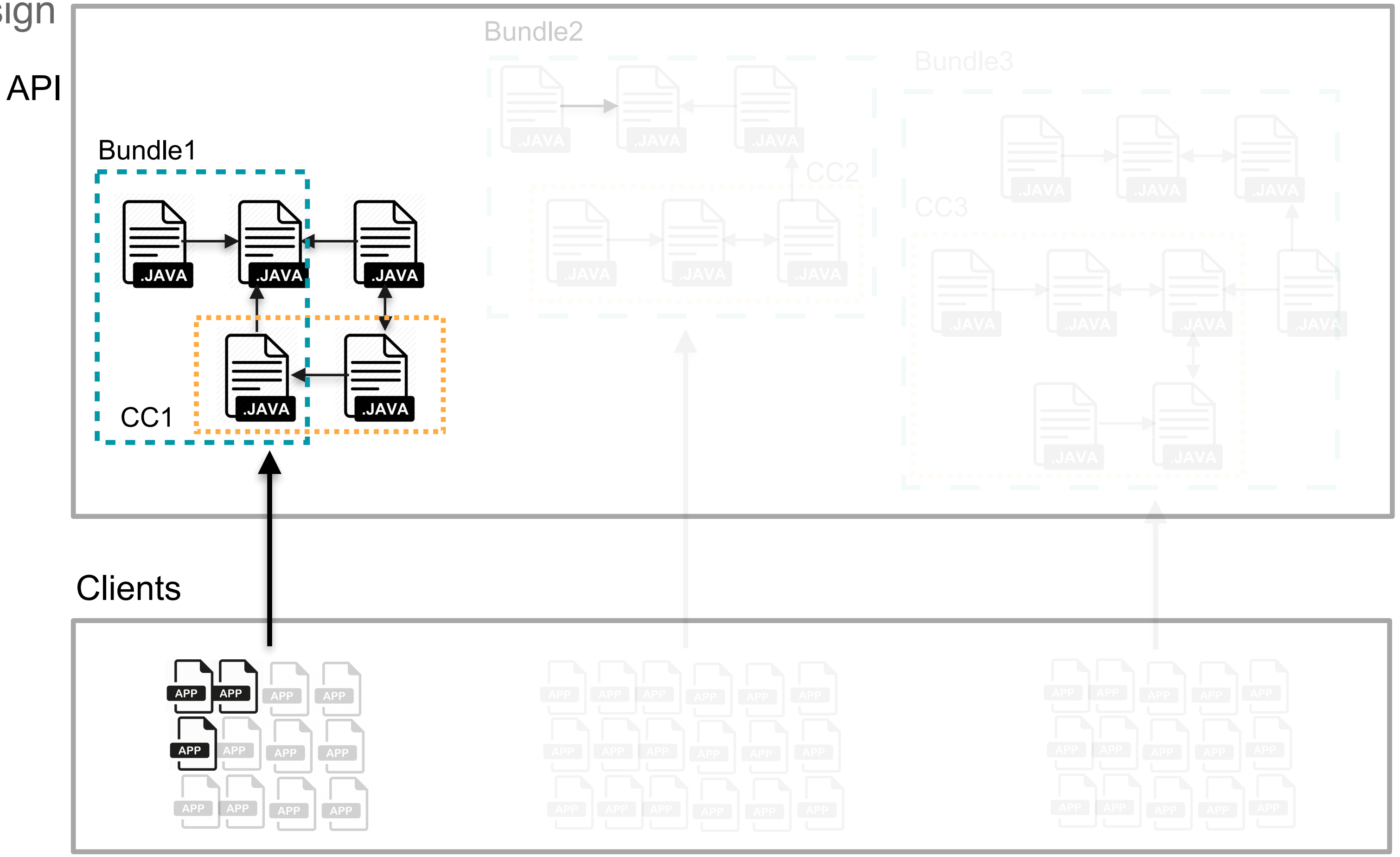
Representativeness: the percentage of clients a bundle could serve



Low **representativeness** (bad coverage)

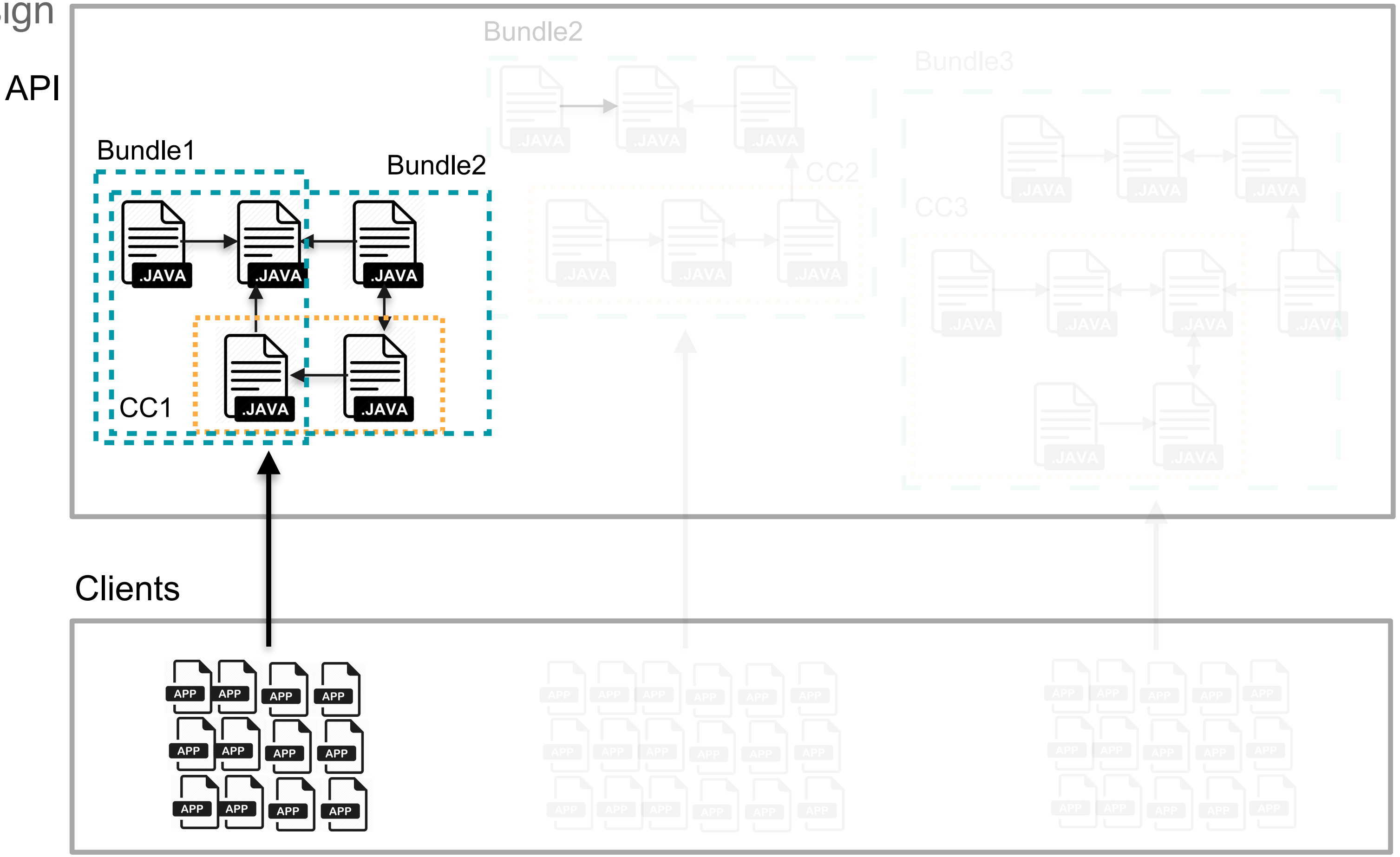
Representativeness: the percentage of clients a bundle could serve

Study Design



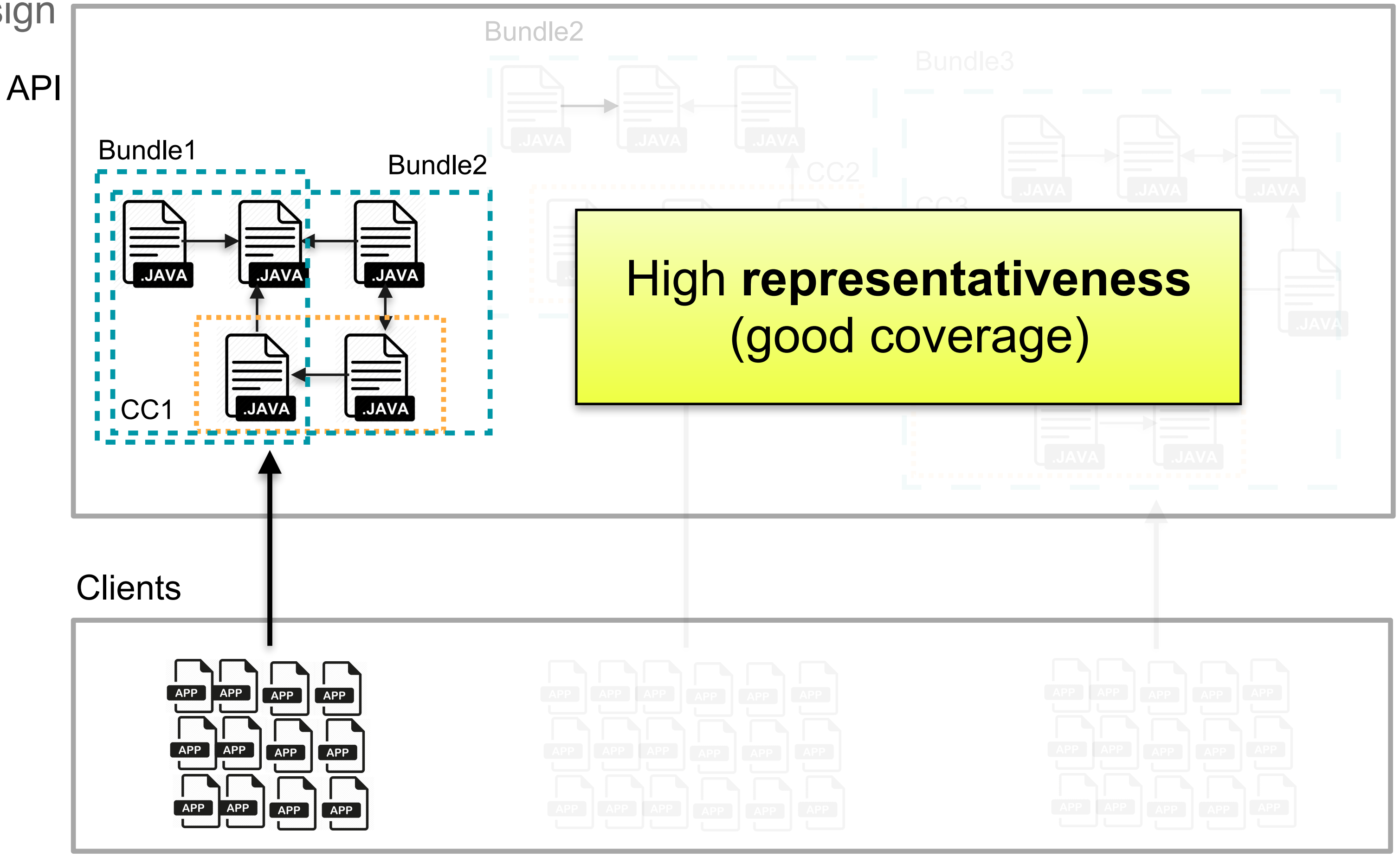
Uniqueness: the percentage of classes that intersect with different bundles

Study Design



Uniqueness: the percentage of classes that intersect with different bundles

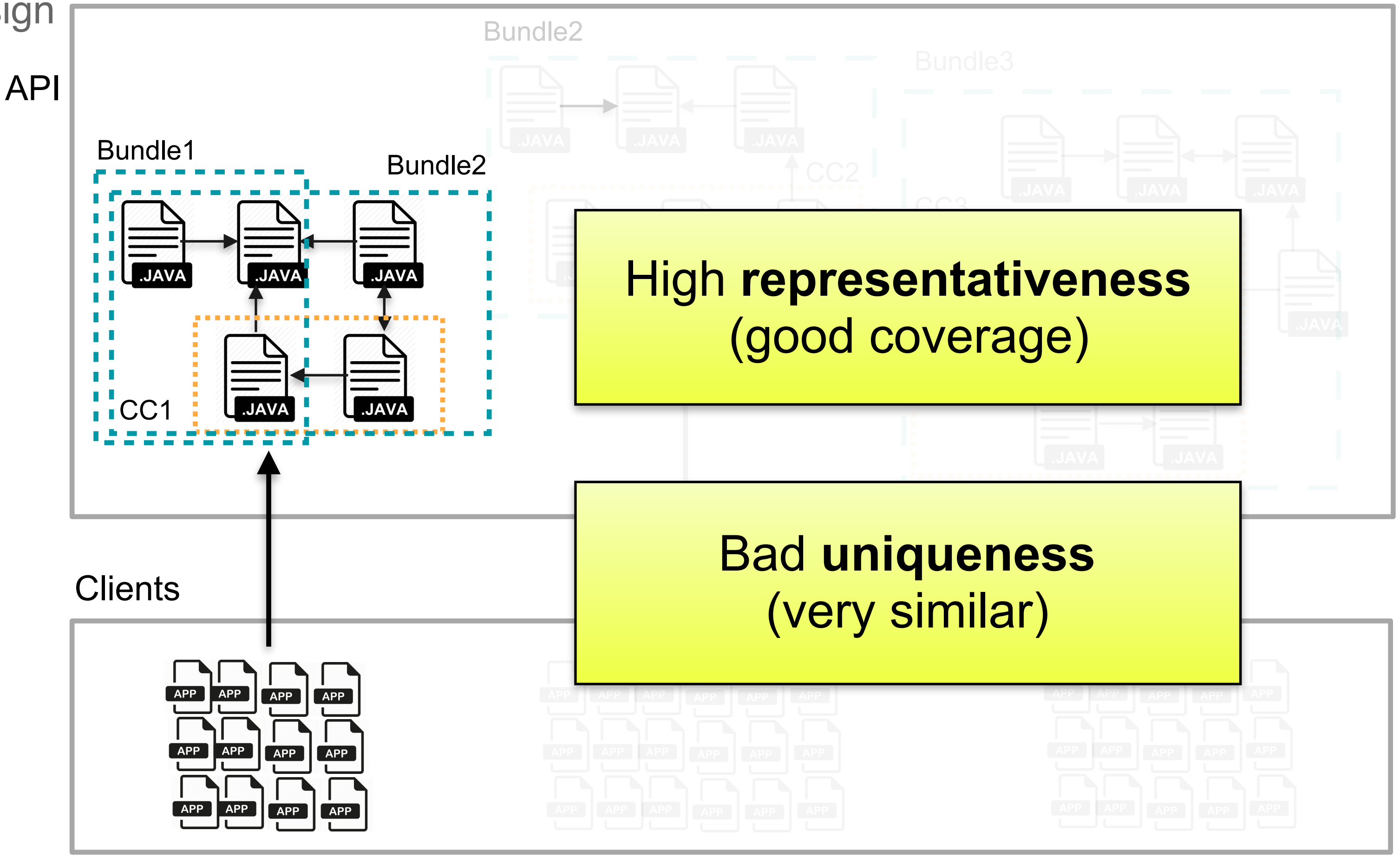
Study Design



High **representativeness** (good coverage)

Uniqueness: the percentage of classes that intersect with different bundles

Study Design



High **representativeness**
(good coverage)

Bad **uniqueness**
(very similar)

Uniqueness: the percentage of classes that intersect with different bundles

We are interested in bundles with **high representativeness** and **(almost) unique**

Uniqueness: the percentage of classes that intersect with different bundles

Research Questions

RQ1. Can we automatically create smaller APIs based on the client usage?

RQ2. Can we reduce an API but keeping a high representativeness?

Study Design

Apache Commons™
http://commons.apache.org/

google-gson
A Java library to convert JSON to Java objects and vice-versa

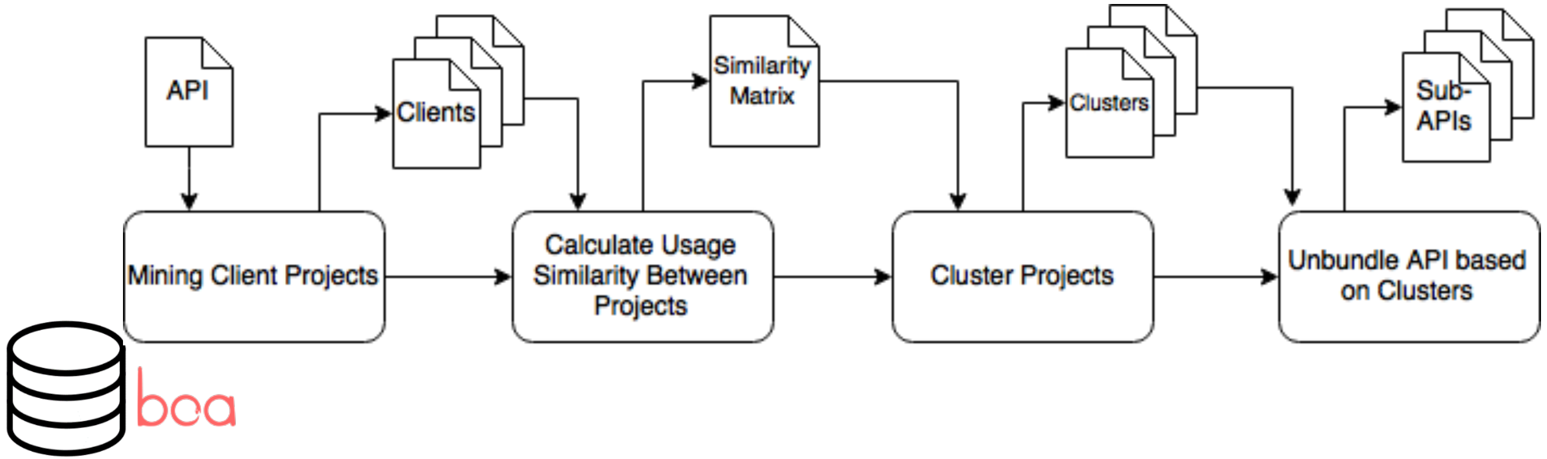
Hamcrest

JUnit

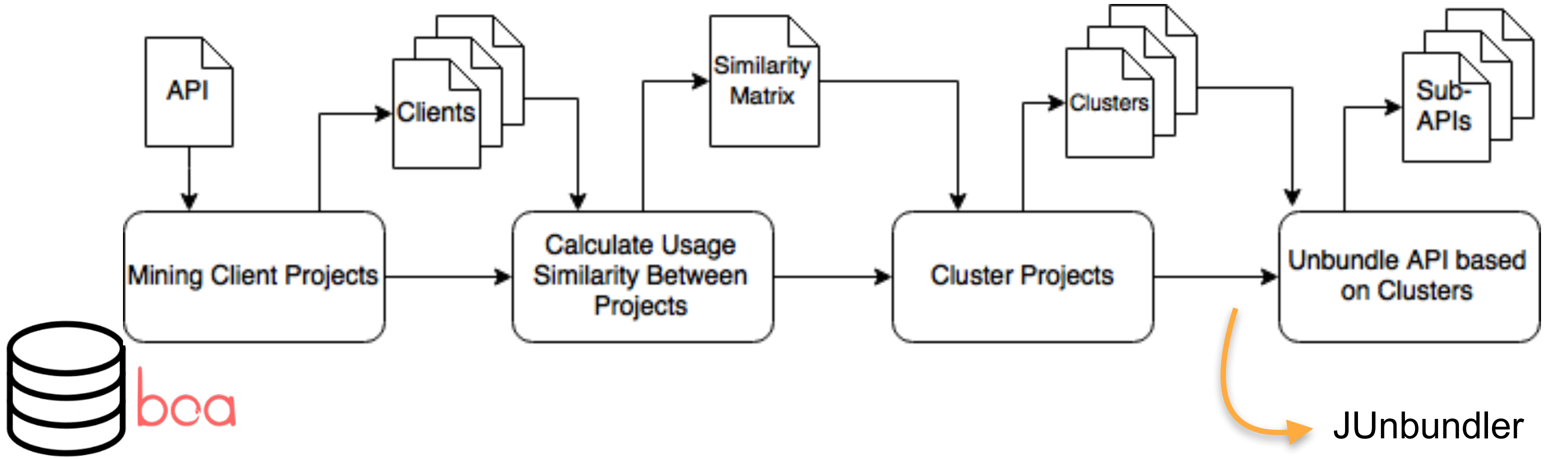
GUAVA

70k+ clients

Flowchart


API
Clients
Similarity Matrix
Clusters
Sub-APIs
Mining Client Projects
Calculate Usage Similarity Between Projects
Cluster Projects
Unbundle API based on Clusters
boa

Flowchart


API
Clients
Similarity Matrix
Clusters
Sub-APIs
Mining Client Projects
Calculate Usage Similarity Between Projects
Cluster Projects
Unbundle API based on Clusters
boa
JUnbundler

Results

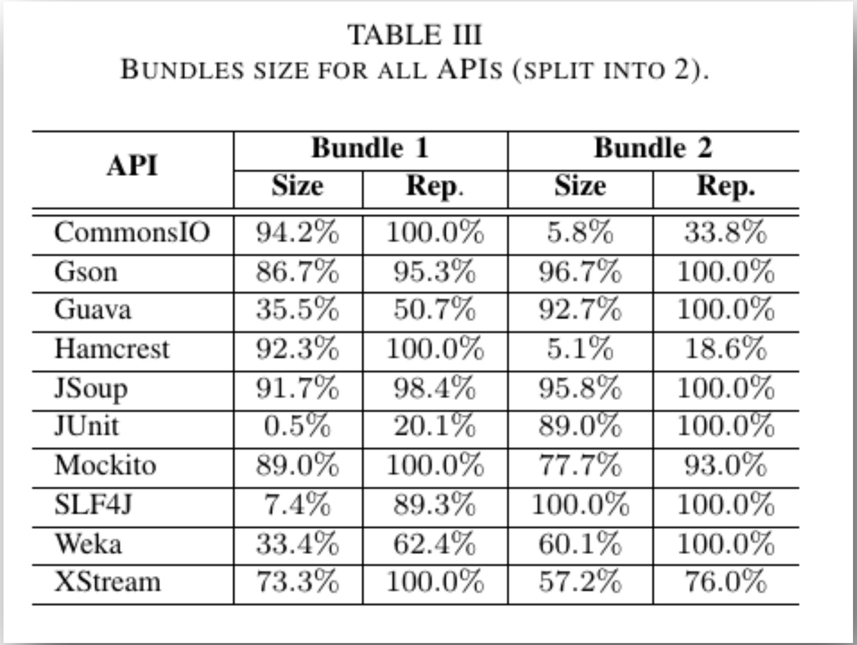


TABLE III
BUNDLES SIZE FOR ALL APIS (SPLIT INTO 2).

API	Bundle 1		Bundle 2	
	Size	Rep.	Size	Rep.
CommonsIO	94.2%	100.0%	5.8%	33.8%
Gson	86.7%	95.3%	96.7%	100.0%
Guava	35.5%	50.7%	92.7%	100.0%
Hamcrest	92.3%	100.0%	5.1%	18.6%
JSoup	91.7%	98.4%	95.8%	100.0%
JUnit	0.5%	20.1%	89.0%	100.0%
Mockito	89.0%	100.0%	77.7%	93.0%
SLF4J	7.4%	89.3%	100.0%	100.0%
Weka	33.4%	62.4%	60.1%	100.0%
XStream	73.3%	100.0%	57.2%	76.0%

Results

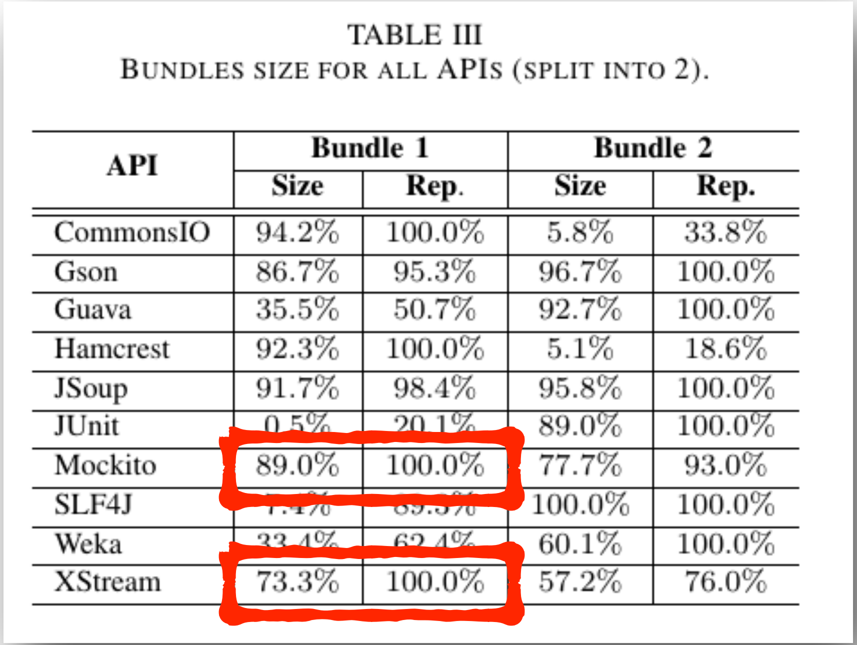


TABLE III
BUNDLES SIZE FOR ALL APIS (SPLIT INTO 2).

API	Bundle 1		Bundle 2	
	Size	Rep.	Size	Rep.
CommonsIO	94.2%	100.0%	5.8%	33.8%
Gson	86.7%	95.3%	96.7%	100.0%
Guava	35.5%	50.7%	92.7%	100.0%
Hamcrest	92.3%	100.0%	5.1%	18.6%
JSoup	91.7%	98.4%	95.8%	100.0%
JUnit	0.5%	20.1%	89.0%	100.0%
Mockito	89.0%	100.0%	77.7%	93.0%
SLF4J	7.4%	89.3%	100.0%	100.0%
Weka	33.4%	62.4%	60.1%	100.0%
XStream	73.3%	100.0%	57.2%	76.0%

We identified a set of clients that could be fully served by a smaller subset of the API

(Negative?) Results

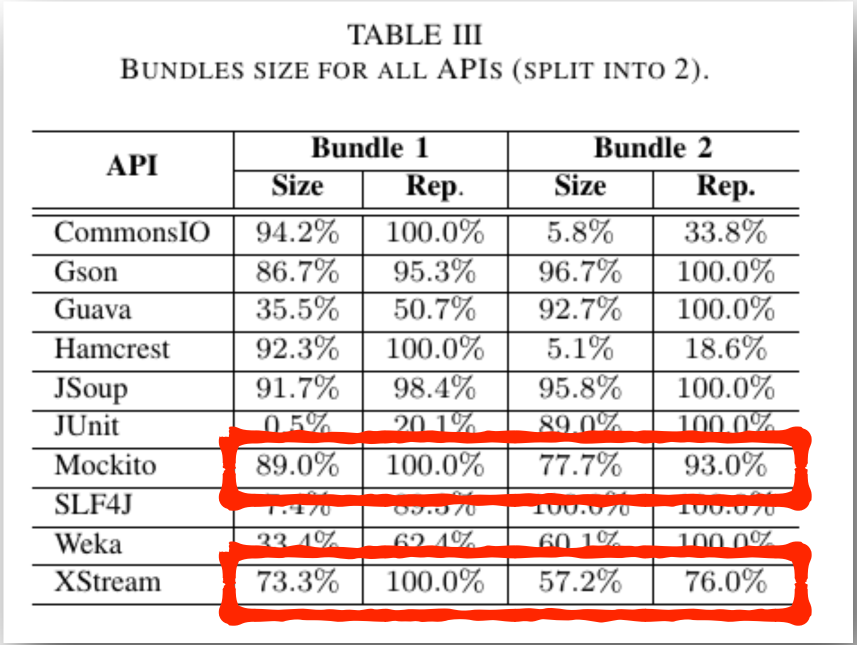


TABLE III
BUNDLES SIZE FOR ALL APIS (SPLIT INTO 2).

API	Bundle 1		Bundle 2	
	Size	Rep.	Size	Rep.
CommonsIO	94.2%	100.0%	5.8%	33.8%
Gson	86.7%	95.3%	96.7%	100.0%
Guava	35.5%	50.7%	92.7%	100.0%
Hamcrest	92.3%	100.0%	5.1%	18.6%
JSoup	91.7%	98.4%	95.8%	100.0%
JUnit	0.5%	20.1%	89.0%	100.0%
Mockito	89.0%	100.0%	77.7%	93.0%
SLF4J	7.4%	89.5%	100.0%	100.0%
Weka	33.4%	62.4%	60.1%	100.0%
XStream	73.3%	100.0%	57.2%	76.0%

We did not find a fully-disjoint usage for any studied API

(Negative?) Results

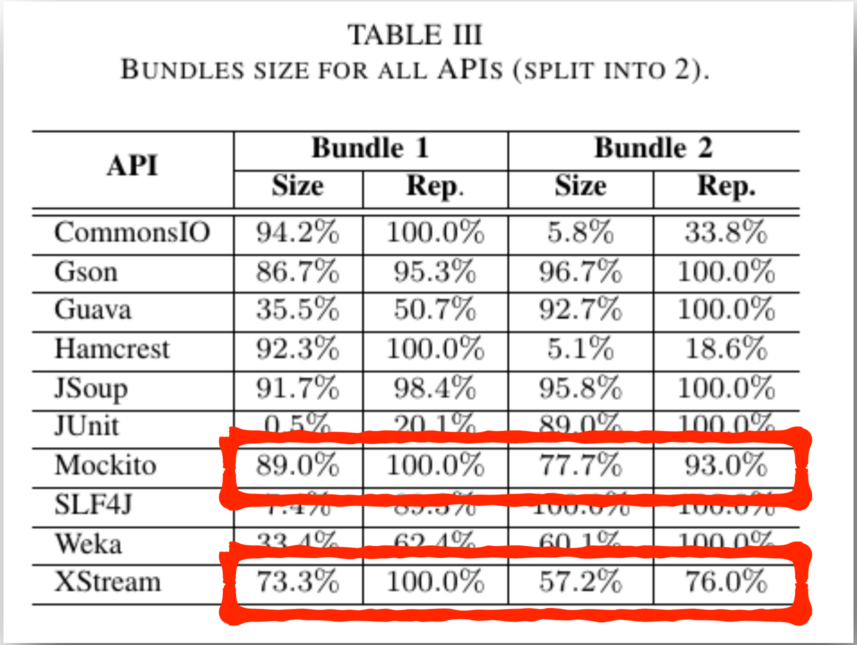
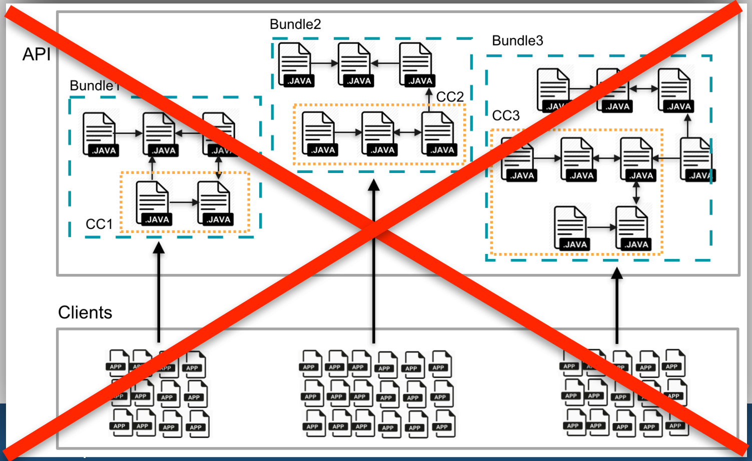


TABLE III
BUNDLES SIZE FOR ALL APIS (SPLIT INTO 2).

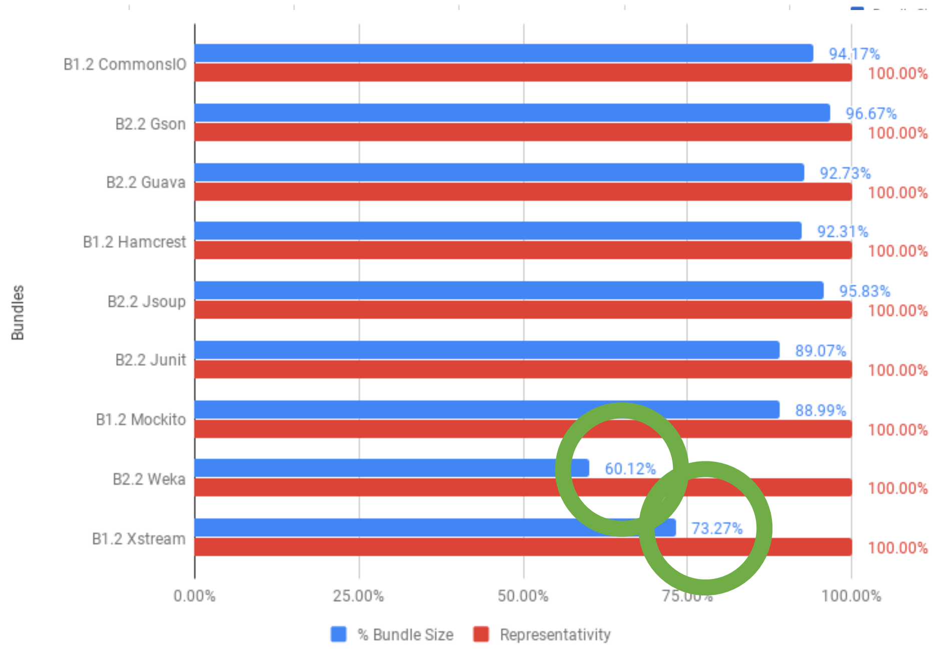
API	Bundle 1		Bundle 2	
	Size	Rep.	Size	Rep.
CommonsIO	94.2%	100.0%	5.8%	33.8%
Gson	86.7%	95.3%	96.7%	100.0%
Guava	35.5%	50.7%	92.7%	100.0%
Hamcrest	92.3%	100.0%	5.1%	18.6%
JSoup	91.7%	98.4%	95.8%	100.0%
JUnit	0.5%	20.1%	89.0%	100.0%
Mockito	89.0%	100.0%	77.7%	93.0%
SLF4J	7.4%	89.5%	100.0%	100.0%
Weka	33.4%	62.4%	60.1%	100.0%
XStream	73.3%	100.0%	57.2%	76.0%

We did not find a fully-disjoint usage for any studied API



Representativeness results

What if we want 100% of coverage?

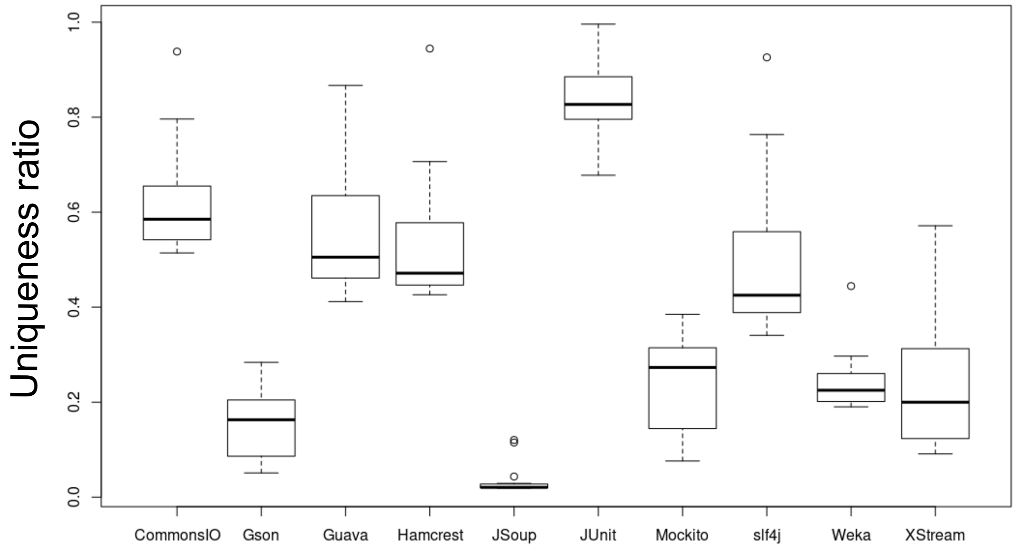


Green circles highlight bundles smaller than 50% of original API that cover 95% or more of its client set

Uniqueness results

Distribution of 20 different bundles

The higher the better

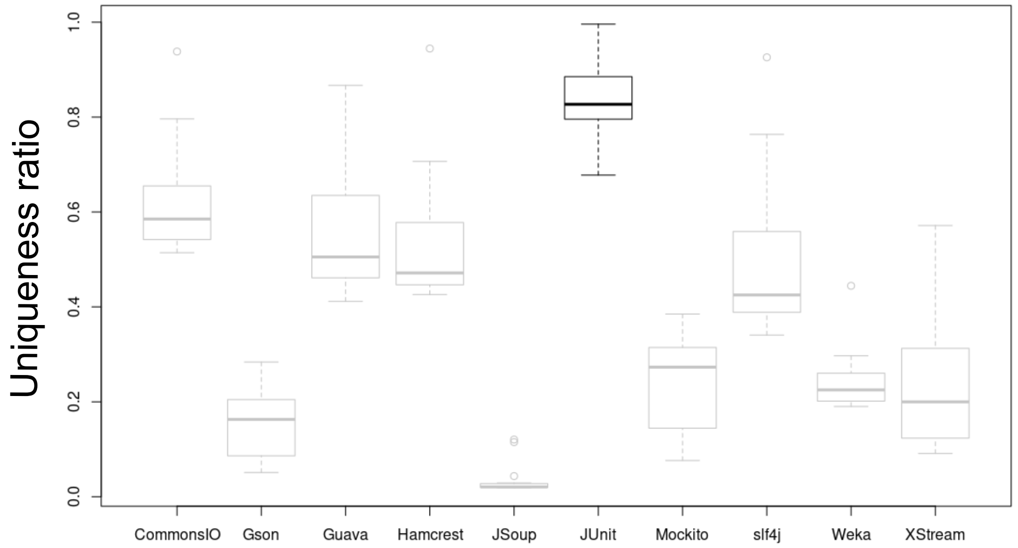


Boxplot of average uniqueness of API bundles

Uniqueness results

JUnit: good design?

The higher the better

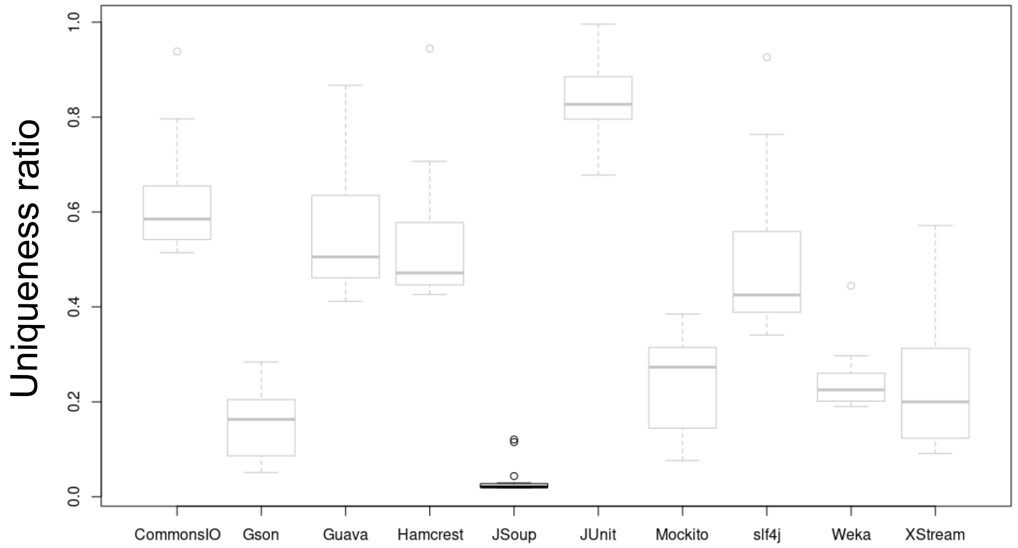


Boxplot of average uniqueness of API bundles

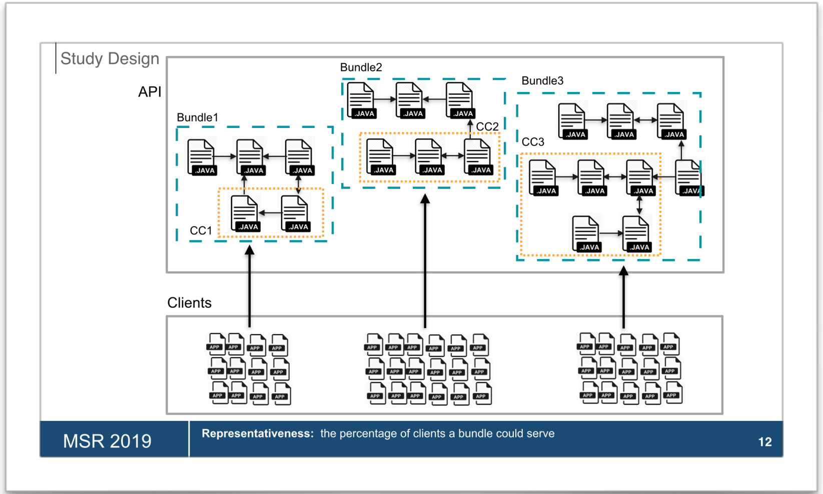
Uniqueness results

JSoup: bad design?

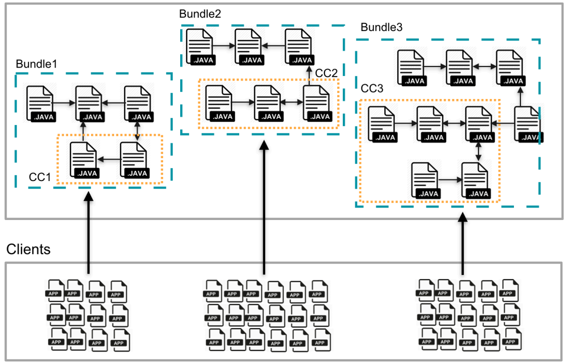
The higher the better



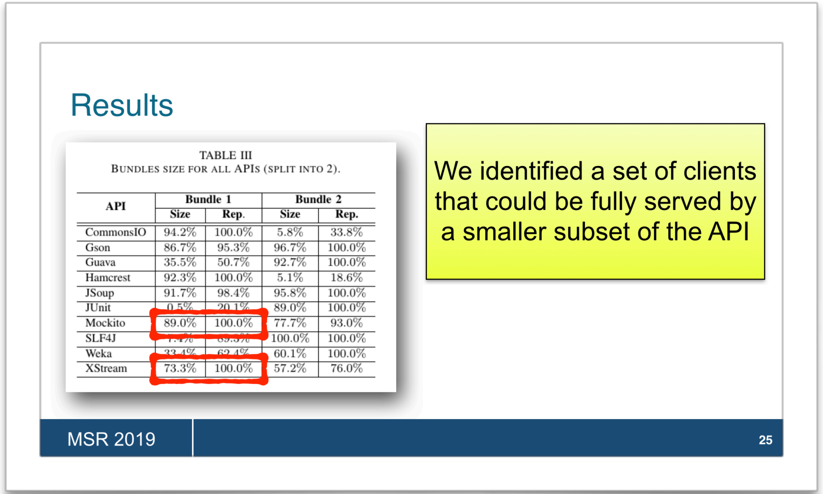
Boxplot of average uniqueness of API bundles



Study Design



MSR 2019 | **Representativeness:** the percentage of clients a bundle could serve | 12



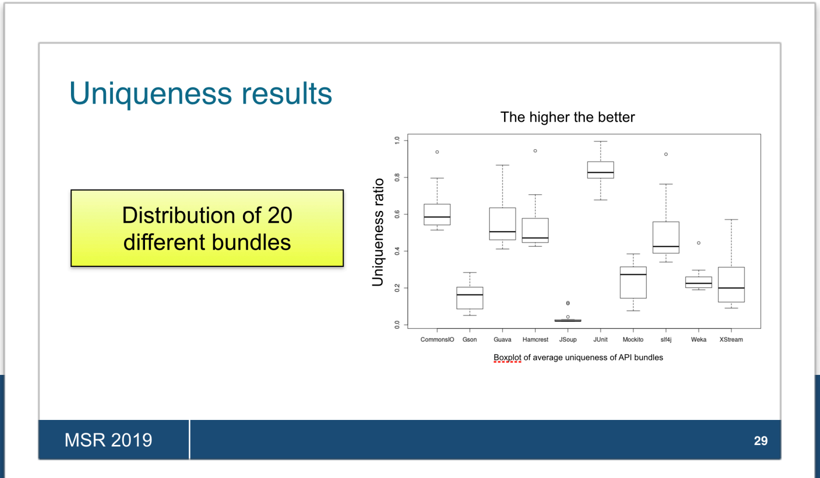
Results

TABLE III
BUNDLES SIZE FOR ALL APIS (SPLIT INTO 2).

API	Bundle 1		Bundle 2	
	Size	Rep.	Size	Rep.
CommonsIO	94.2%	100.0%	5.8%	33.8%
Gson	86.7%	95.3%	96.7%	100.0%
Guava	35.5%	50.7%	92.7%	100.0%
Hamcrest	92.3%	100.0%	5.1%	18.6%
JSoup	91.7%	98.4%	95.8%	100.0%
JUnit	0.5%	20.1%	89.0%	100.0%
Mockito	89.0%	100.0%	77.7%	93.0%
SLF4J	7.4%	89.3%	100.0%	100.0%
Weka	33.4%	62.4%	60.1%	100.0%
XStream	73.3%	100.0%	57.2%	76.0%

We identified a set of clients that could be fully served by a smaller subset of the API

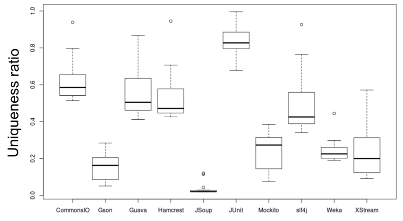
MSR 2019 | 25



Uniqueness results

Distribution of 20 different bundles

The higher the better



Boxplot of average uniqueness of API bundles

MSR 2019 | 29

UNIVERSIDADE FEDERAL DO CEARÁ

MDCC
Mestrado e Doutorado em Ciência da Computação

GREat
Grupo de Redes de Computadores
Engenharia de Software
e Sistemas

National Institute
of Science and Technology
in Software Engineering
ines

CAPES

Thank You!

Anderson Severo de Matos: severo@alu.ufc.br

João Bosco Ferreira Filho: bosco@dc.ufc.br

Lincoln Souza Rocha: lincoln@dc.ufc.br