UNIVERSITÄT BERN

Adherence of class comments to style guidelines

Suada Abukar

Bachelor Thesis Final Presentation

Supervised by Pooja Rani

04 May 2021

Motivation

- Comments help developers understand code
- Contain different types of information (summary, description, …)
- Comments are free-form text
- Style guidelines for code documentation

… but do developers follow documentation style guidelines?

```
/** Event fired when a spacer element is hidden or shown in Escalator.
 *
 * @author Vaadin Ltd
 * @since 7.7.13
 */
```

Class: **SpacerVisibilityChangedEvent**.java

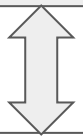
Style guidelines to write class comments

- First sentence is a summary.
- Use phrases instead of complete sentences.
- Use 3rd person instead of 2nd person.

```
/** Event fired when a spacer element is hidden or shown in Escalator.
 *
 * @author Vaadin Ltd
 * @since 7.7.13
 */
```

Class: **SpacerVisibilityChangedEvent**.java

Compare



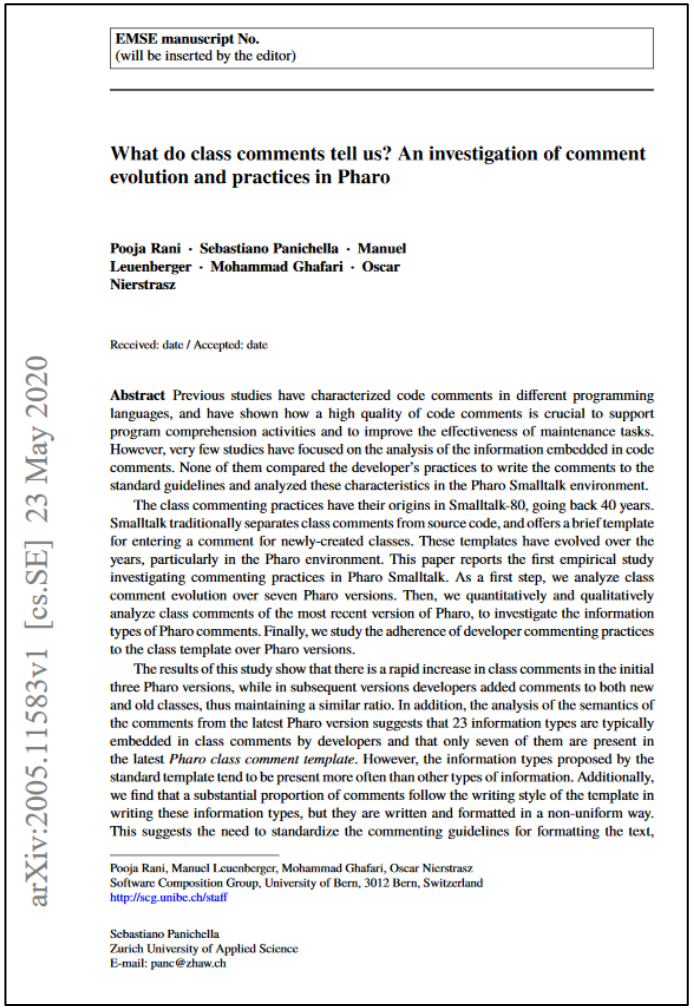
Style guidelines to write class comments

- First sentence is a summary.
- Use phrases instead of complete sentences.
- Use 3rd person instead of 2nd person.

Motivation

- Analysis of class comments in Pharo
- 60% of comments follow the style guideline

→ **We want to investigate this in Java and Python**



EMSE manuscript No.
(will be inserted by the editor)

What do class comments tell us? An investigation of comment evolution and practices in Pharo

Pooja Rani · Sebastiano Panichella · Manuel Leuenberger · Mohammad Ghafari · Oscar Nierstrasz

Received: date / Accepted: date

Abstract Previous studies have characterized code comments in different programming languages, and have shown how a high quality of code comments is crucial to support program comprehension activities and to improve the effectiveness of maintenance tasks. However, very few studies have focused on the analysis of the information embedded in code comments. None of them compared the developer's practices to write the comments to the standard guidelines and analyzed these characteristics in the Pharo Smalltalk environment.

The class commenting practices have their origins in Smalltalk-80, going back 40 years. Smalltalk traditionally separates class comments from source code, and offers a brief template for entering a comment for newly-created classes. These templates have evolved over the years, particularly in the Pharo environment. This paper reports the first empirical study investigating commenting practices in Pharo Smalltalk. As a first step, we analyze class comment evolution over seven Pharo versions. Then, we quantitatively and qualitatively analyze class comments of the most recent version of Pharo, to investigate the information types of Pharo comments. Finally, we study the adherence of developer commenting practices to the class template over Pharo versions.

The results of this study show that there is a rapid increase in class comments in the initial three Pharo versions, while in subsequent versions developers added comments to both new and old classes, thus maintaining a similar ratio. In addition, the analysis of the semantics of the comments from the latest Pharo version suggests that 23 information types are typically embedded in class comments by developers and that only seven of them are present in the latest *Pharo class comment template*. However, the information types proposed by the standard template tend to be present more often than other types of information. Additionally, we find that a substantial proportion of comments follow the writing style of the template in writing these information types, but they are written and formatted in a non-uniform way. This suggests the need to standardize the commenting guidelines for formatting the text,

Pooja Rani, Manuel Leuenberger, Mohammad Ghafari, Oscar Nierstrasz
Software Composition Group, University of Bern, 3012 Bern, Switzerland
http://scg.unibe.ch/staff

Sebastiano Panichella
Zurich University of Applied Science
E-mail: panc@zhaw.ch

What has been done?

- Analysis of class comments in Java and Python
- Focus on information types present in class comments
- Proposes techniques to automatically identify class comment information types

How to Identify Class Comment Types?
A Multi-language Approach for Class Comments Classification

Pooja Rani[a], Sebastiano Panichella[b], Manuel Leuenberger[a], Andrea Di Sorbo[c], Oscar Nierstrasz[a]

[a]Software Composition Group, University of Bern, Switzerland
[b]Zurich University of Applied Science, Switzerland
[c]Department of Engineering, University of Sannio, Italy

Abstract

Most software maintenance and evolution tasks require developers to understand the source code of their software systems. Software developers usually inspect class comments to gain knowledge about program behavior, regardless of the programming language they are using. Unfortunately, (i) different programming languages present language-specific code commenting notations and guidelines; and (ii) the source code of software projects often lacks comments that adequately describe the class behavior, which complicates program comprehension and evolution activities. To handle these challenges, this paper investigates the different language-specific class commenting practices of three programming languages: Python, Java, and Pharo. In particular, we systematically analyze the similarities and differences of the information types found in class comments of projects developed in these languages. We propose an approach that leverages two techniques — namely Natural Language Processing and Text Analysis — to automatically identify *class comment types*, *i.e.*, the specific types of semantic information found in class comments. To the best of our knowledge, no previous work has provided a comprehensive taxonomy of class comment types for these three programming languages with the help of a common automated approach. Our results confirm that our approach can classify frequent class comment information types with high accuracy for the Python, Java, and Pharo programming languages. We believe this work can help to monitor and assess the quality and evolution of code comments in different programming languages, and thus support maintenance and evolution tasks.

Keywords: Natural Language Processing Technique, Code Comment Analysis, Software Documentation

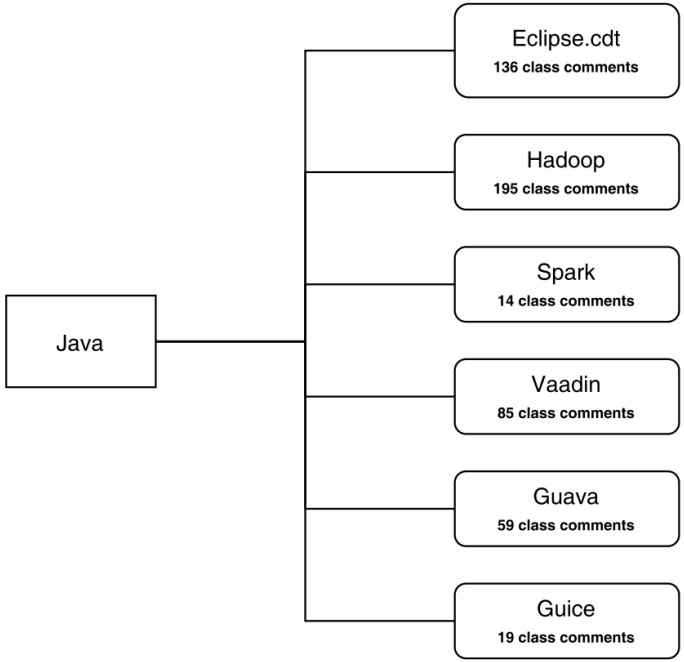
1. Introduction

Software maintenance and evolution tasks require developers to perform program comprehension activities [1, 2]. To understand a software system, developers usually refer to the software documentation of the system [3, 4]. Previous studies have demonstrated that developers trust code comments more than other forms of documentation when they try to answer program comprehension questions [5, 6, 4]. In addition, recent work has also demonstrated that *"code documentation"* is the most used source of information for bug fixing, implementing features, communication, and even code review [7]. In particular, well-documented code simplifies software maintenance activities, but many programmers often overlook or delay code commenting tasks [8].

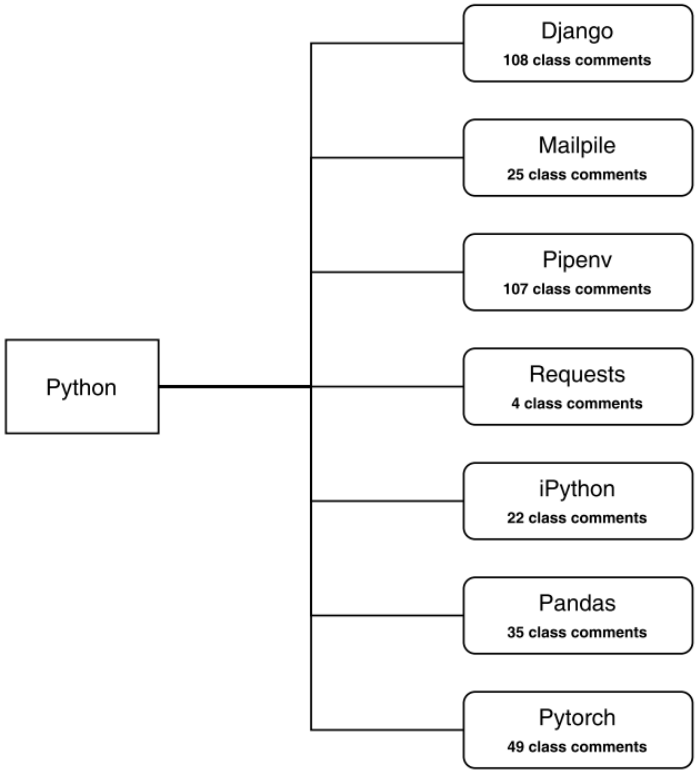
Class comments play an important role in obtaining a high-level overview of the classes in object-oriented languages [9]. In particular, when applying code changes, developers using different programming languages can inspect class comments to achieve most or majority of the high-level insights about the

Email addresses: pooja.rani@inf.unibe.ch (Pooja Rani), panc@zhaw.ch (Sebastiano Panichella), manuel.leuenberger@inf.unibe.ch (Manuel Leuenberger), disorbo@unisannio.it (Andrea Di Sorbo), oscar.nierstrasz@inf.unibe.ch (Oscar Nierstrasz)

What has been done?

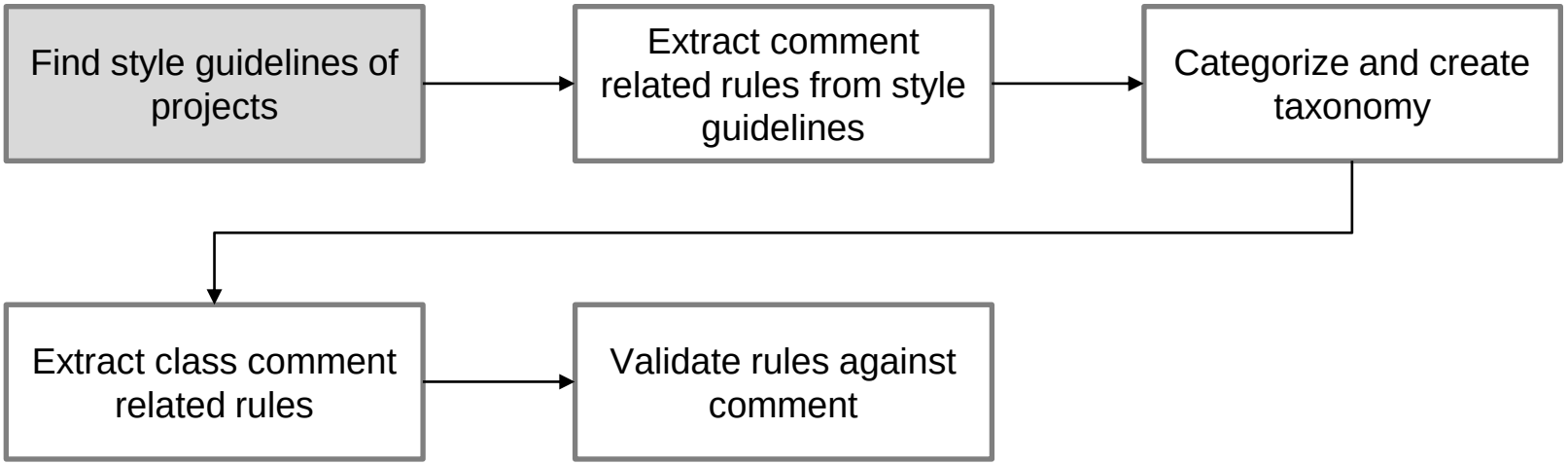


6 open-source Java projects

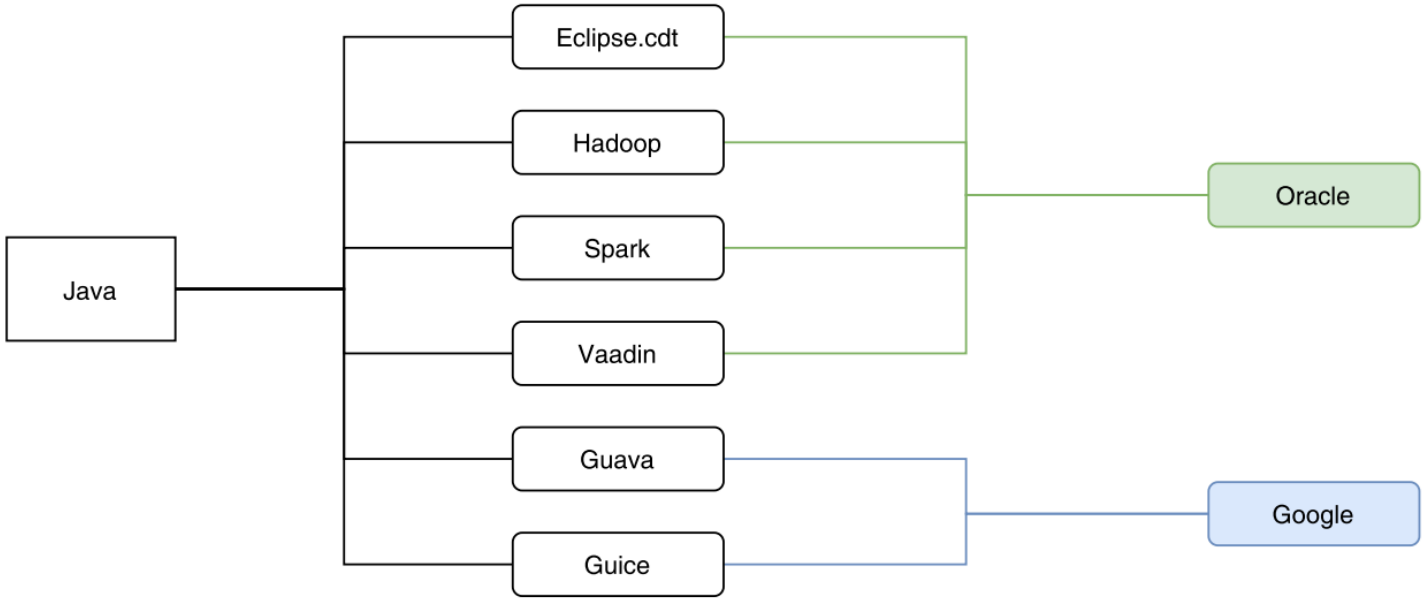


7 open-source Python projects

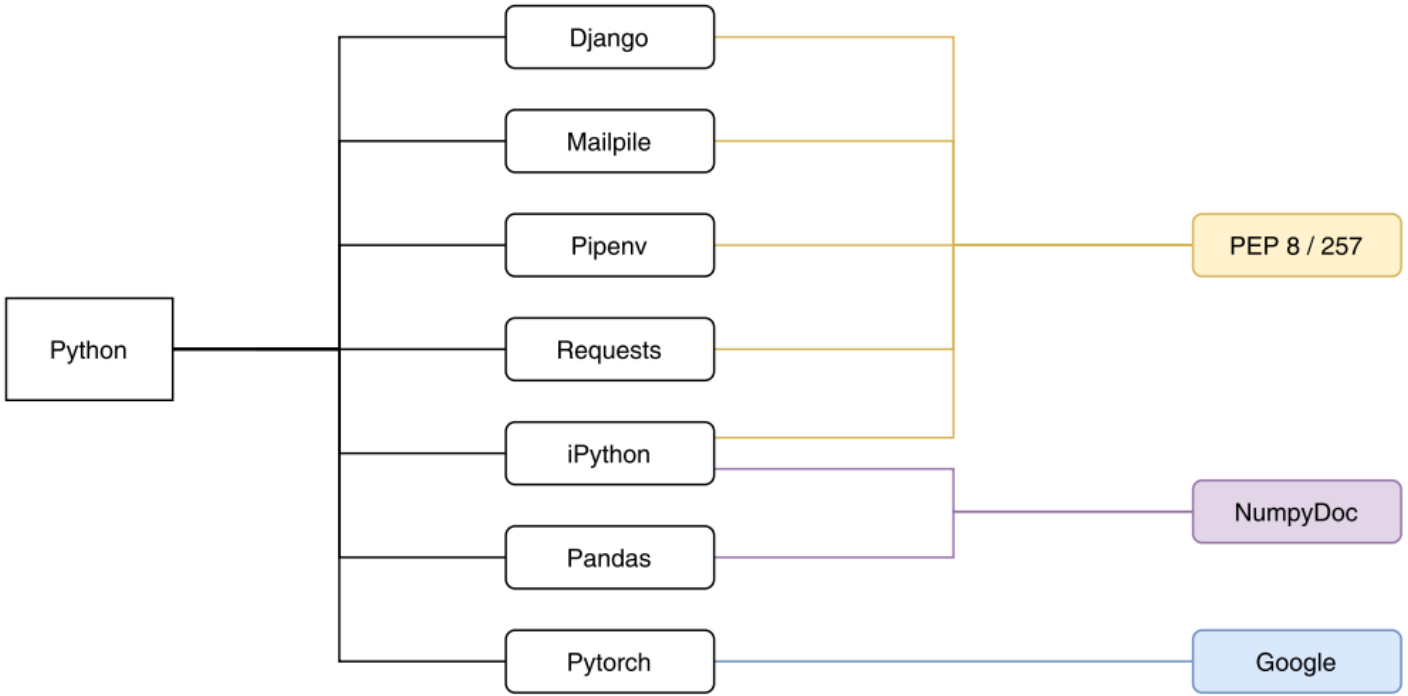
Approach


Find style guidelines of projects
Extract comment related rules from style guidelines
Categorize and create taxonomy
Extract class comment related rules
Validate rules against comment

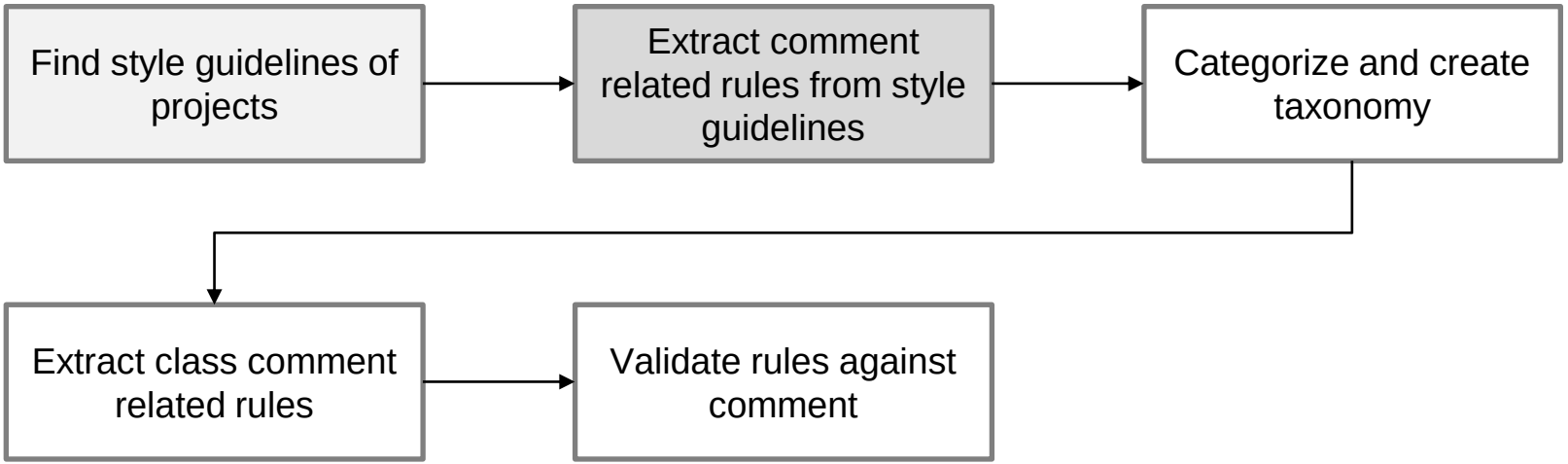
Find style guidelines of projects



Find style guidelines of projects

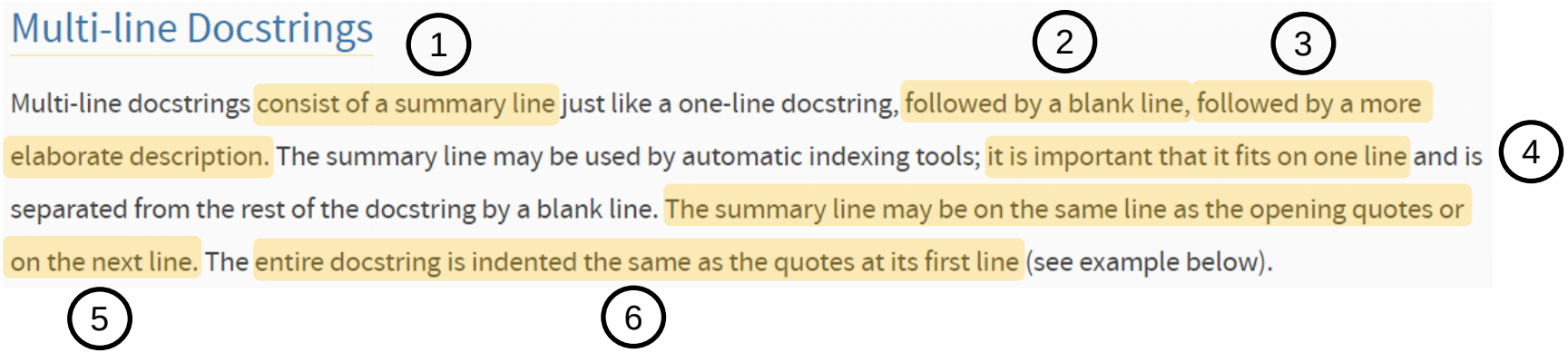


Approach

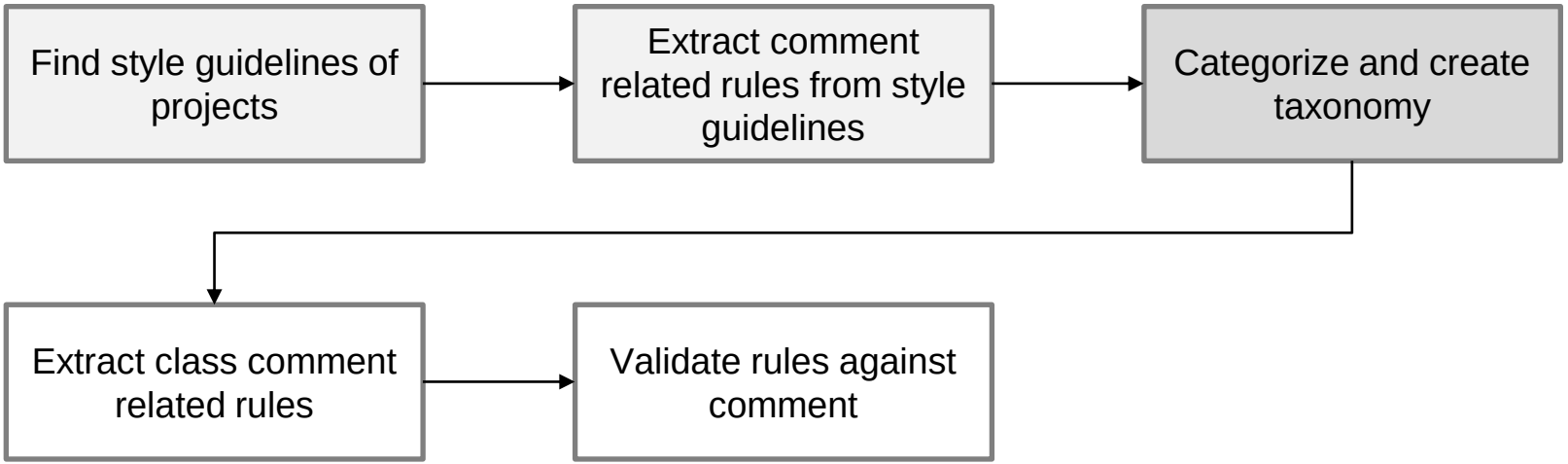

Find style guidelines of projects
Extract comment related rules from style guidelines
Categorize and create taxonomy
Extract class comment related rules
Validate rules against comment

Extract comment related rules from style guidelines

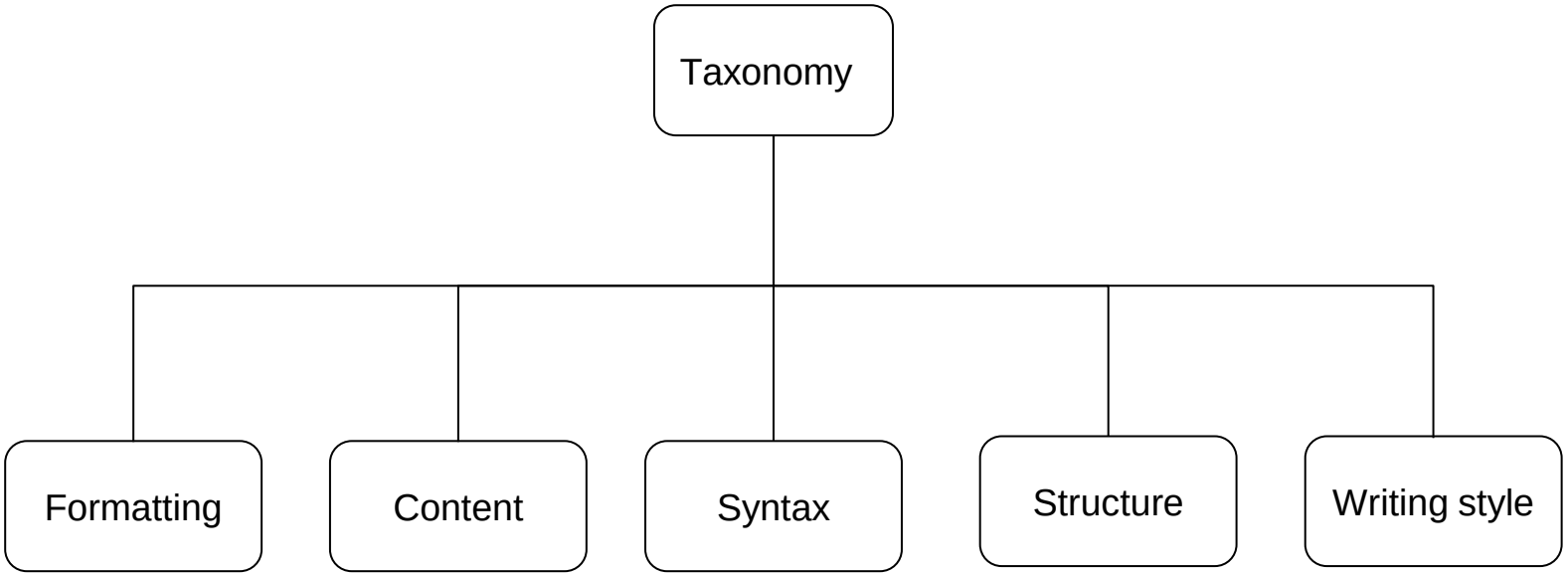
Example PEP257



Approach

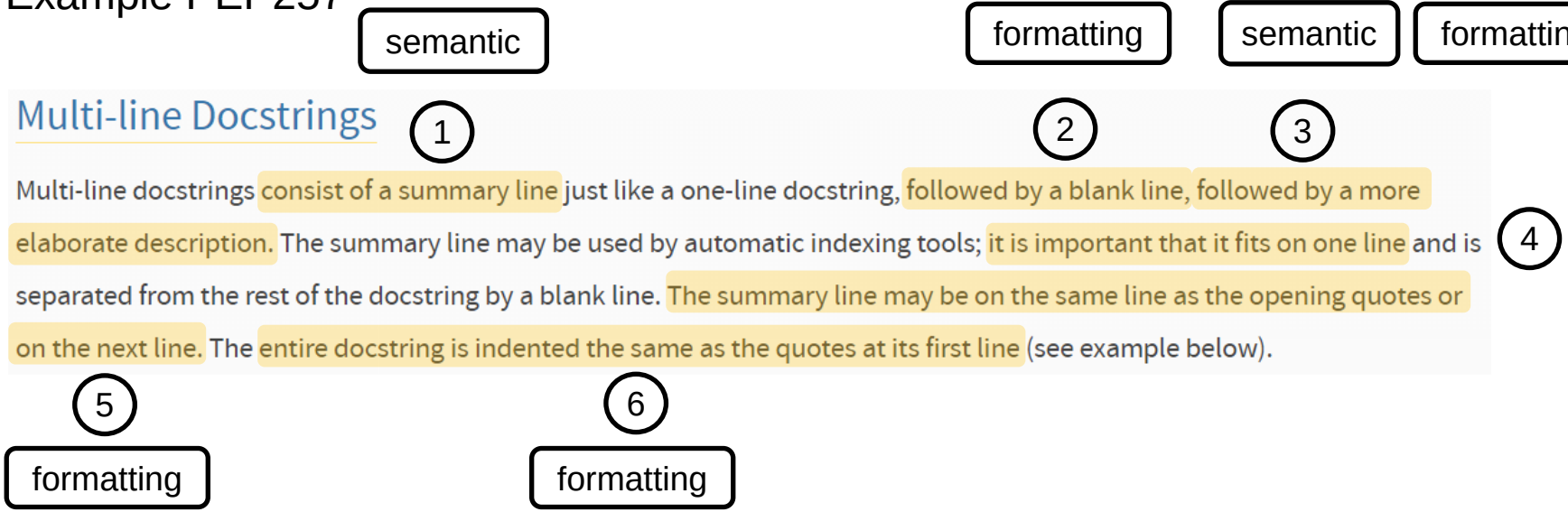

Find style guidelines of projects
Extract comment related rules from style guidelines
Categorize and create taxonomy
Extract class comment related rules
Validate rules against comment

Categorize and create taxonomy

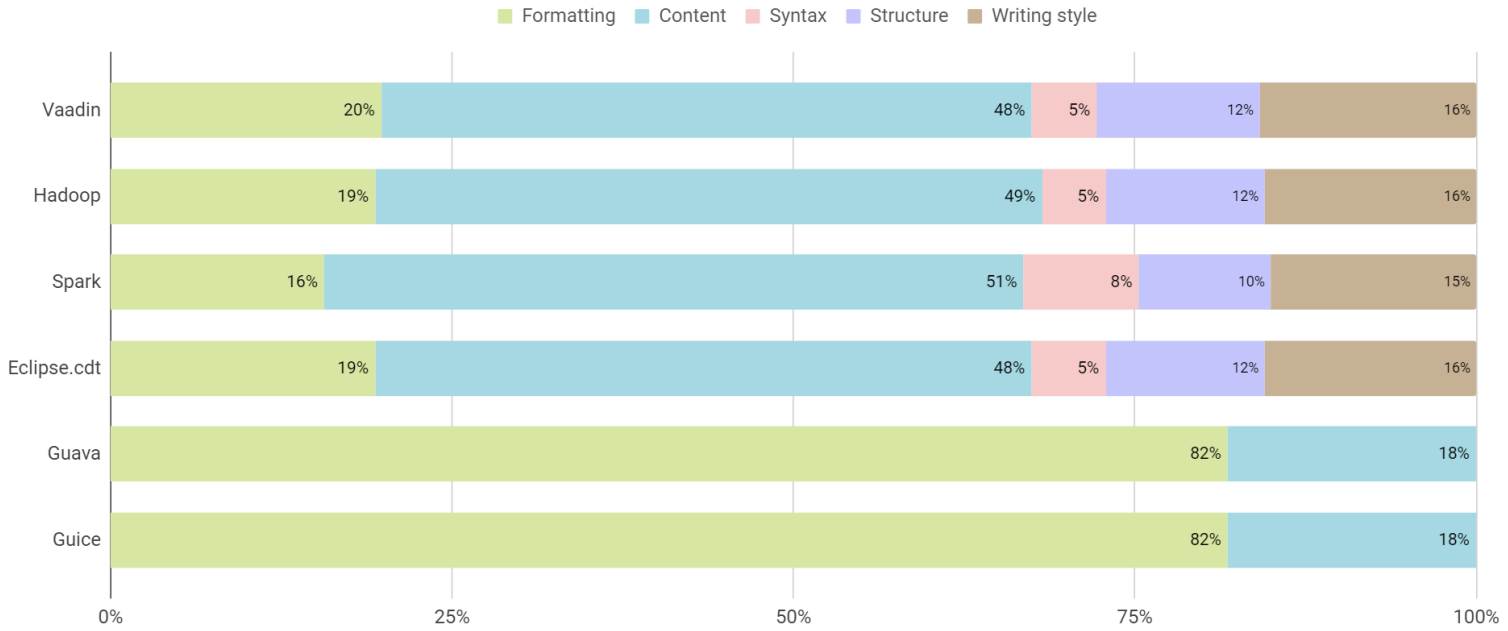


Categorize and create taxonomy

Example PEP257

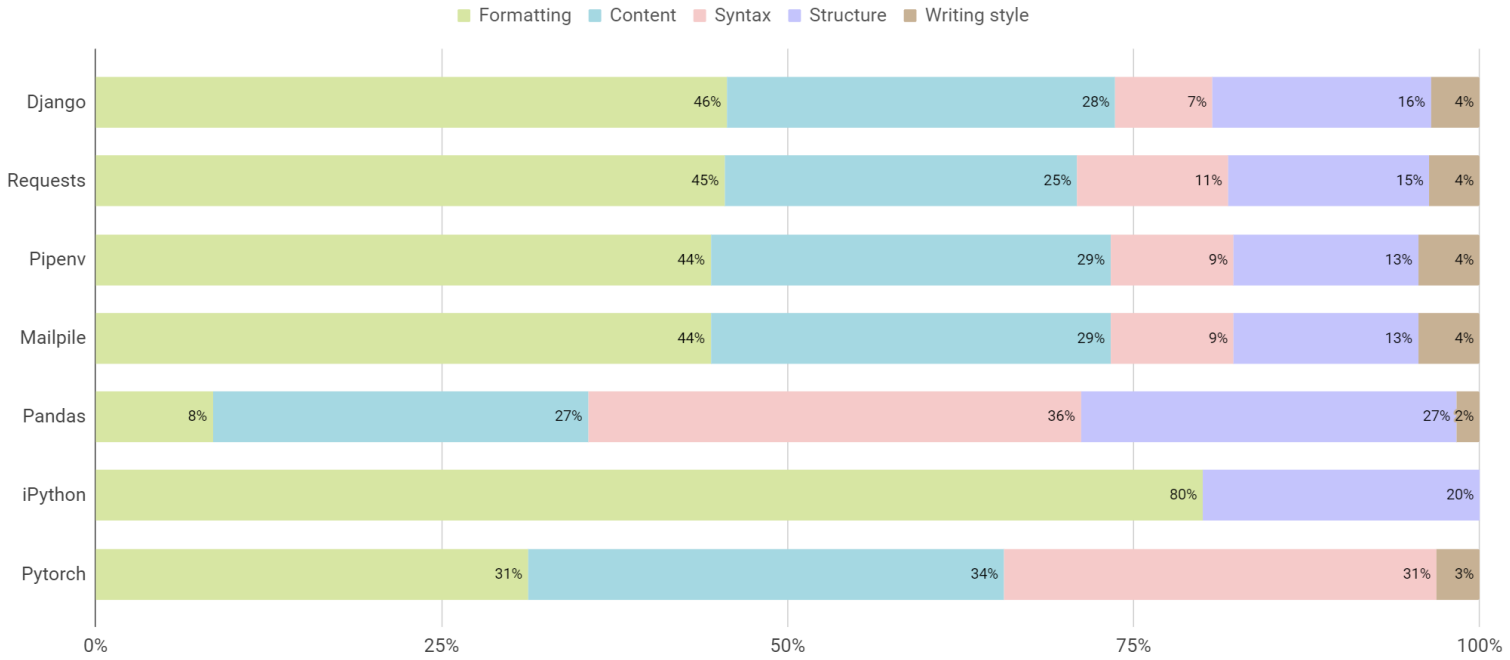


Categorize and create taxonomy



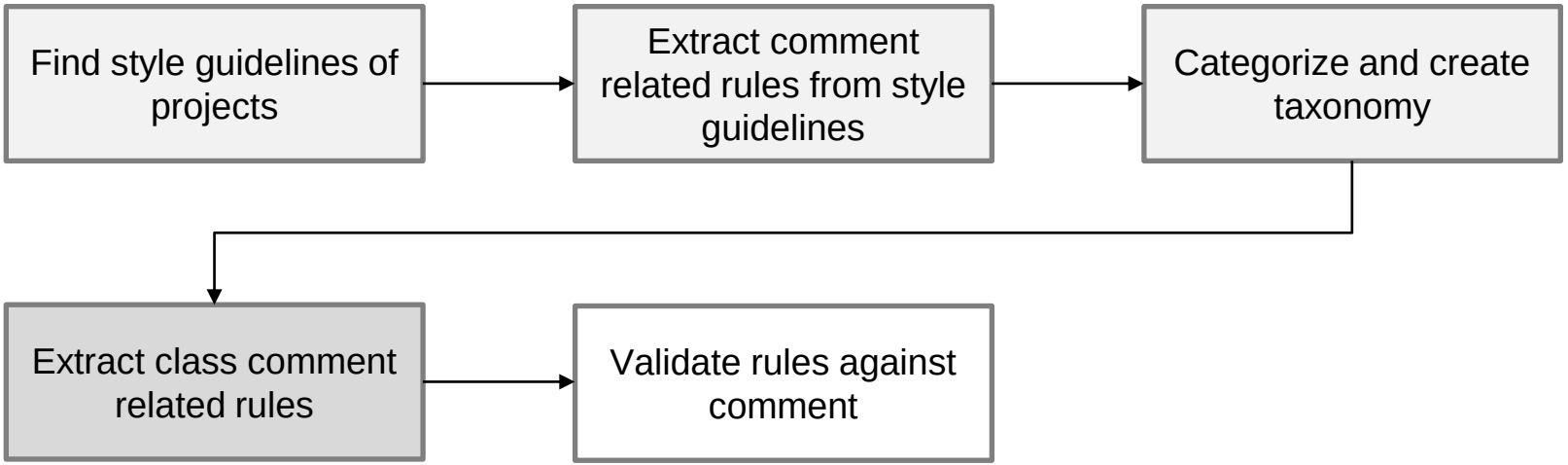
Majority of the rules are about formatting and content.

Categorize and create taxonomy



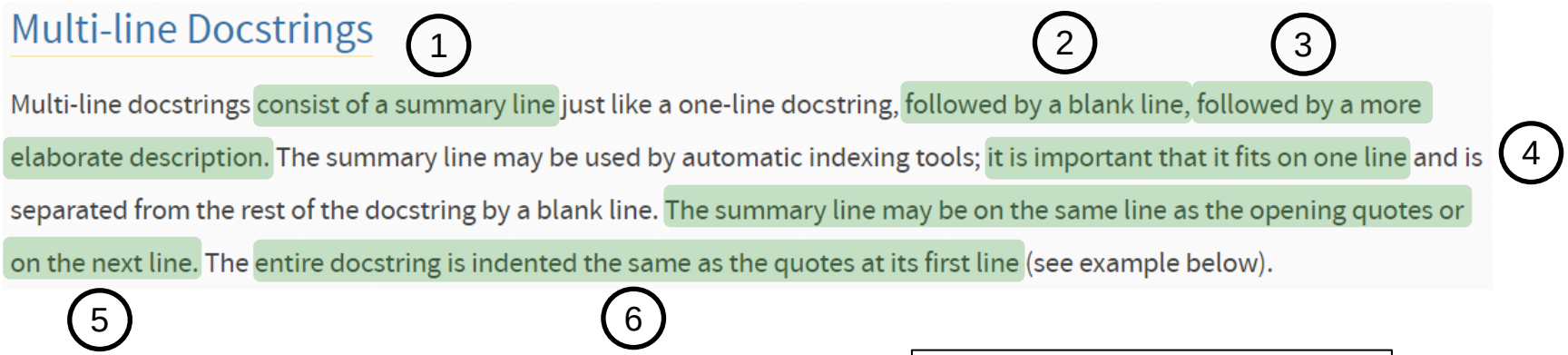
Majority of the rules are about formatting and content.

Approach


Find style guidelines of projects
Extract comment related rules from style guidelines
Categorize and create taxonomy
Extract class comment related rules
Validate rules against comment

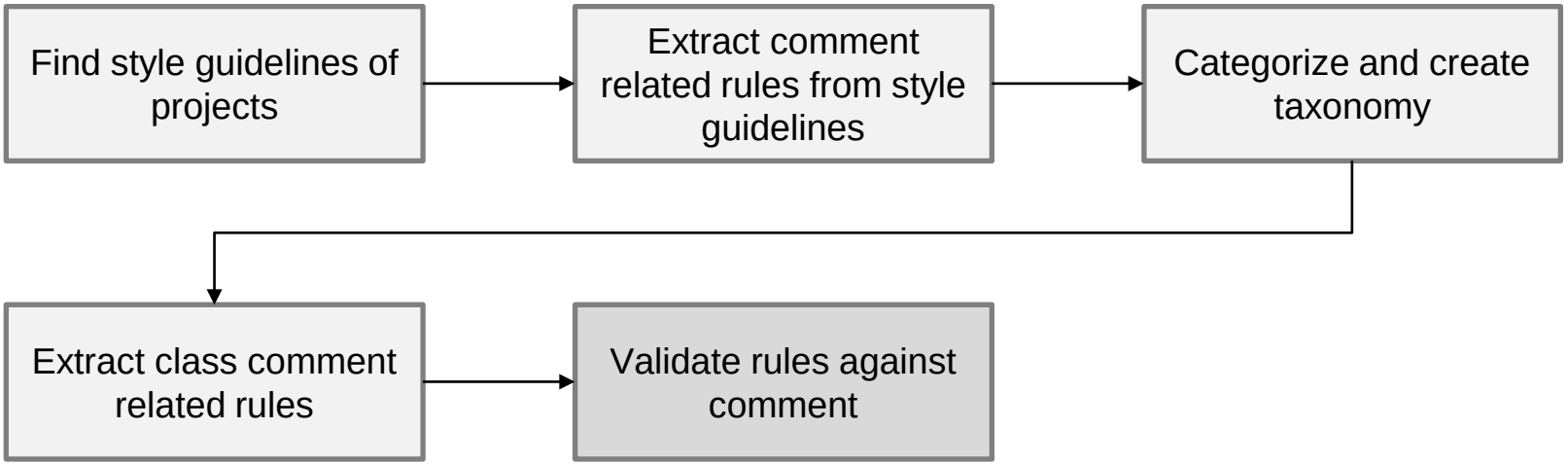
Extract class comment related rules

Example PEP257



All 6 rules are class comment related.

Approach


Find style guidelines of projects
Extract comment related rules from style guidelines
Categorize and create taxonomy
Extract class comment related rules
Validate rules against comment

Validate rules against comments

Project	Nr. of rules	Applicable rules	Non applicable rules
Vaadin	59	28.58%	71.42 %
Hadoop	60	19.82%	80.18 %
Spark	59	20.17%	79.83 %
Eclipse.cdt	32	23.55%	76.45 %
Guava	6	33.90%	66.10 %
Guice	6	26.32%	73.68 %

Most rules were not applicable due to data unavailability.

The @deprecated description in the first sentence should at least tell the user when the API was deprecated.

@deprecated tag not present

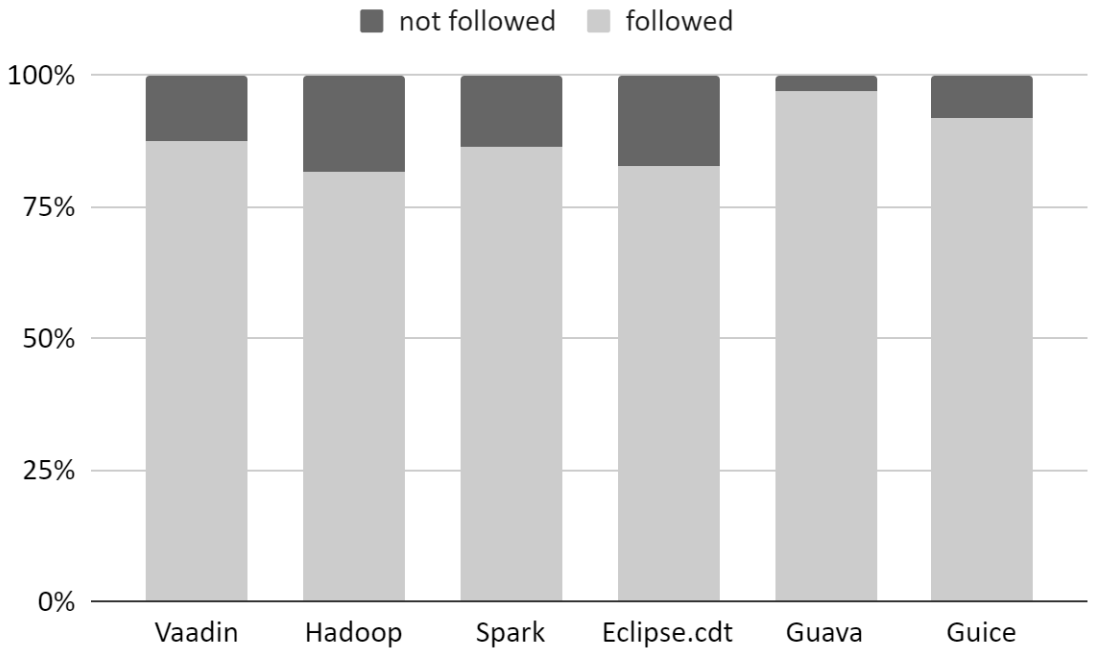
rule is not applicable

Validate rules against comments

Project	Nr. of rules	Applicable rules	Non applicable rules
Django	39	66.81%	33.19%
Requests	39	68.33%	31.67%
Pepenv	29	64.49%	35.51%
Mailpile	29	62.15%	37.85%
Pandas	140	15.83%	84.17%
iPython	72	65.38%	34.62%
Pytorch	27	20.82%	79.18%

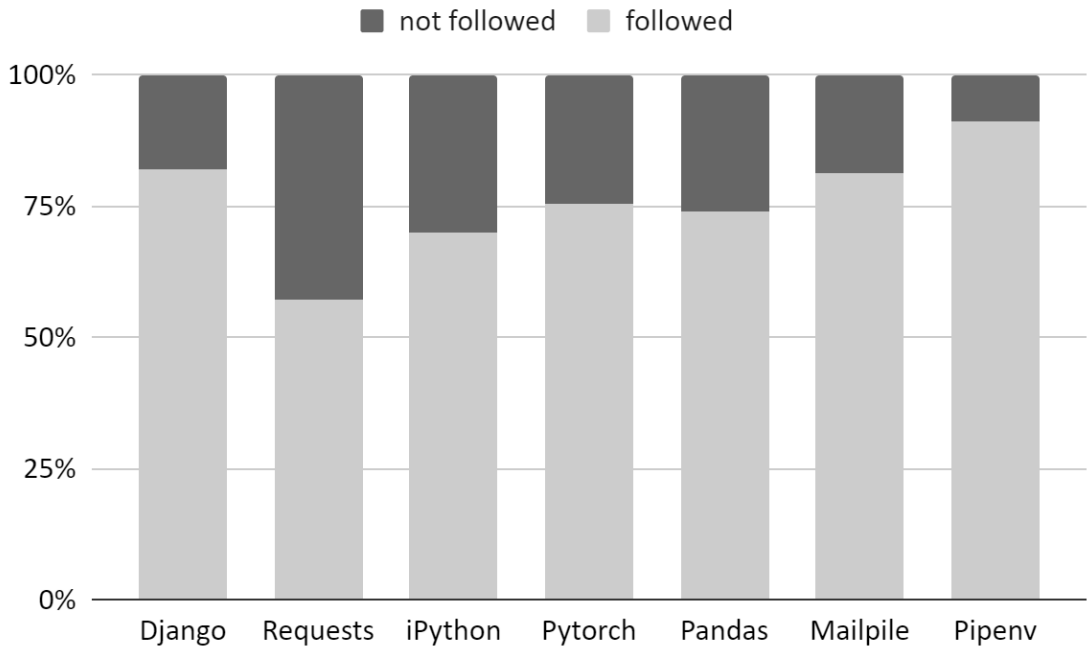
High % of non applicable rules only for Pandas and Pytorch observable.

Validate rules against comments



From the applicable rules, on average 87.91% are followed.

Validate rules against comments



From the applicable rules, on average 75.86% are followed.

Summary

- Most of the rules are not applicable due to data unavailability.
- From those applicable rules most are followed.
- Only small portion of applicable rules are not followed.

Future Work

- Comparison of style guide adherence between Java and Python
- Finish writing thesis