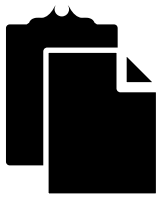
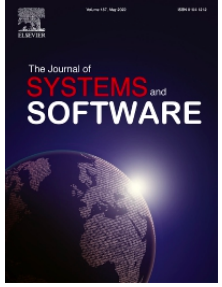
SANER 2022



Journal First



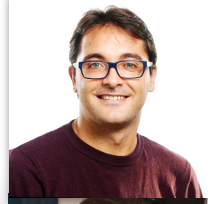
How to Identify Class Comment Types? A Multi-language Approach for Class Comment Classification

Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, Oscar Nierstrasz

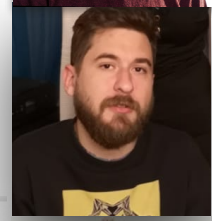
UNIVERSITÄT BERN

Zurich University of Applied Sciences

zhaw



UNIVERSITÄT BERN



Universita degli Studi del Sannio

UNIVERSITÄT BERN

Motivation

```
/**
* A class representing a window on the screen.
*
* For example:
* <pre>
*     Window win = new Window(parent);
*     win.show();
* </pre>
*
* @author Sami Shaio
* @version 1.13, 06/08/06
* @see java.awt.BaseWindow
*/
class Window extends BaseWindow {
  …
}
```

Trustworthy form of documentation

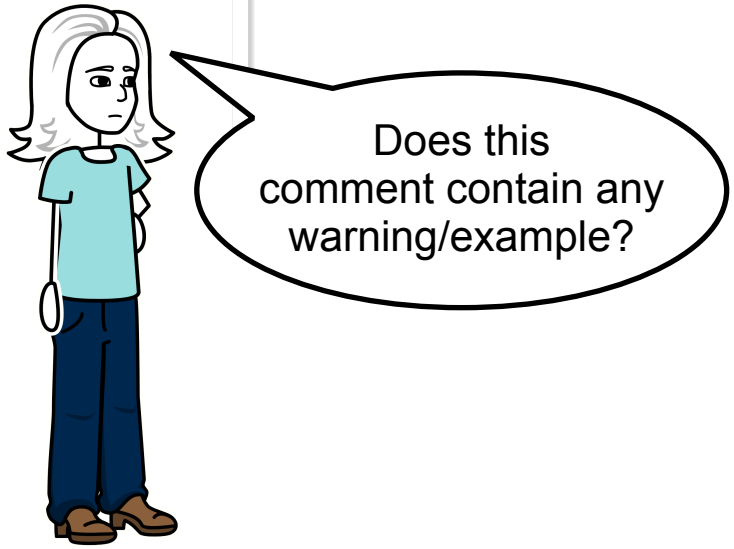
- McMillan et al. 2010

High-quality code comments assist developers

- Dekel et al. 2009

Problem

```
/**
* A class representing a window on the screen.
*
* For example:
* <pre>
*     Window win = new Window(parent);
*     win.show();
* </pre>
*
* @author Sami Shaio
* @version 1.13, 06/08/06
* @see java.awt.BaseWindow
*/
class Window extends BaseWindow {
  …
}
```



Challenges

No standard definition of comments

No strict syntax and structure conventions

Lack of quality assessment tools

Challenges

No standard definition of comments

No strict syntax and structure conventions

Lack of quality assessment tools

Challenges

No standard definition of comments

No strict syntax and structure conventions

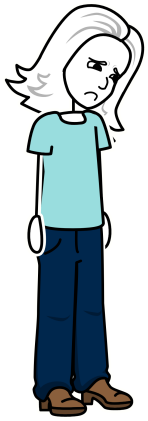
Lack of quality assessment tools

Challenges

No standard definition of comments

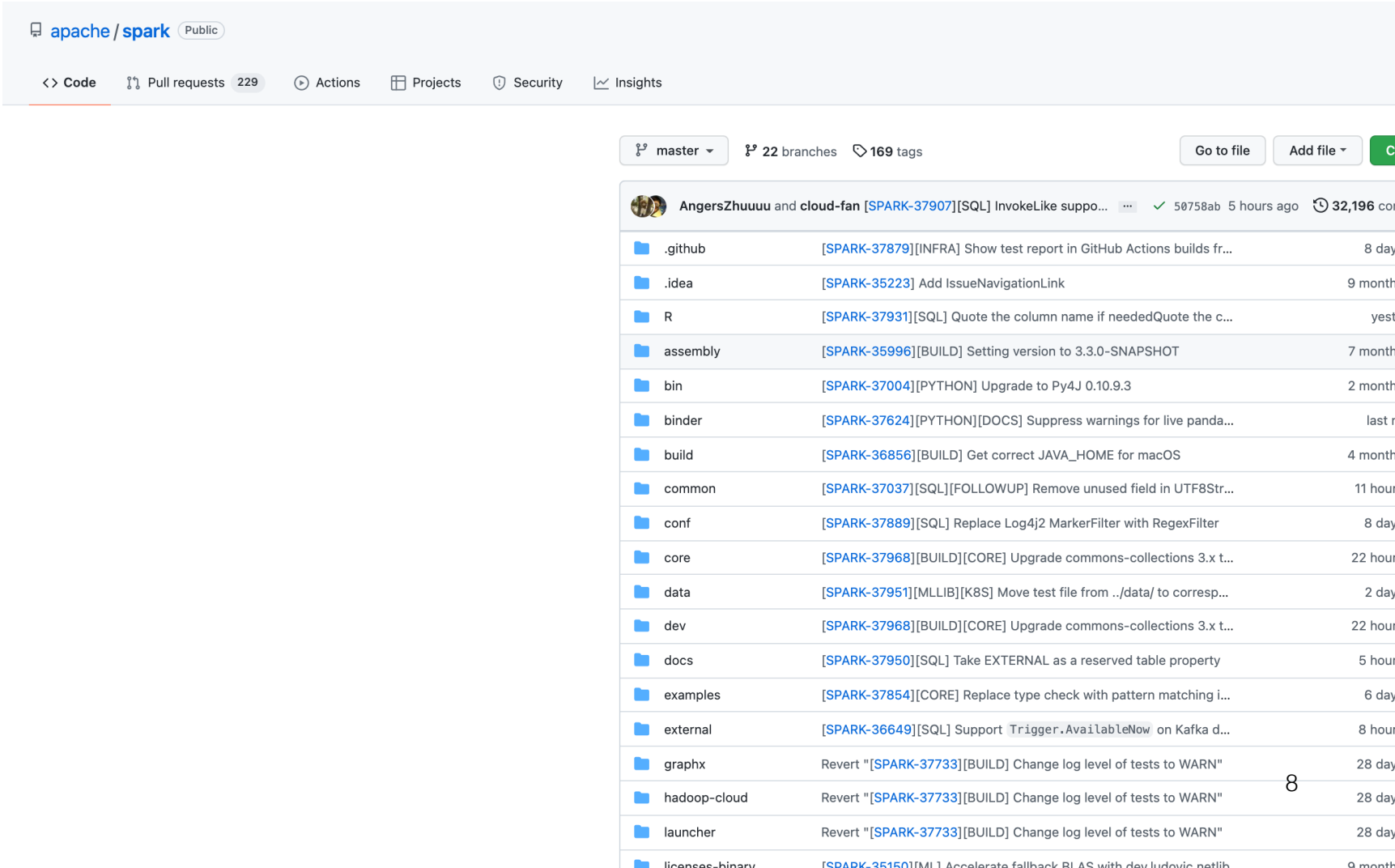
No strict syntax and structure conventions

Lack of quality assessment tools

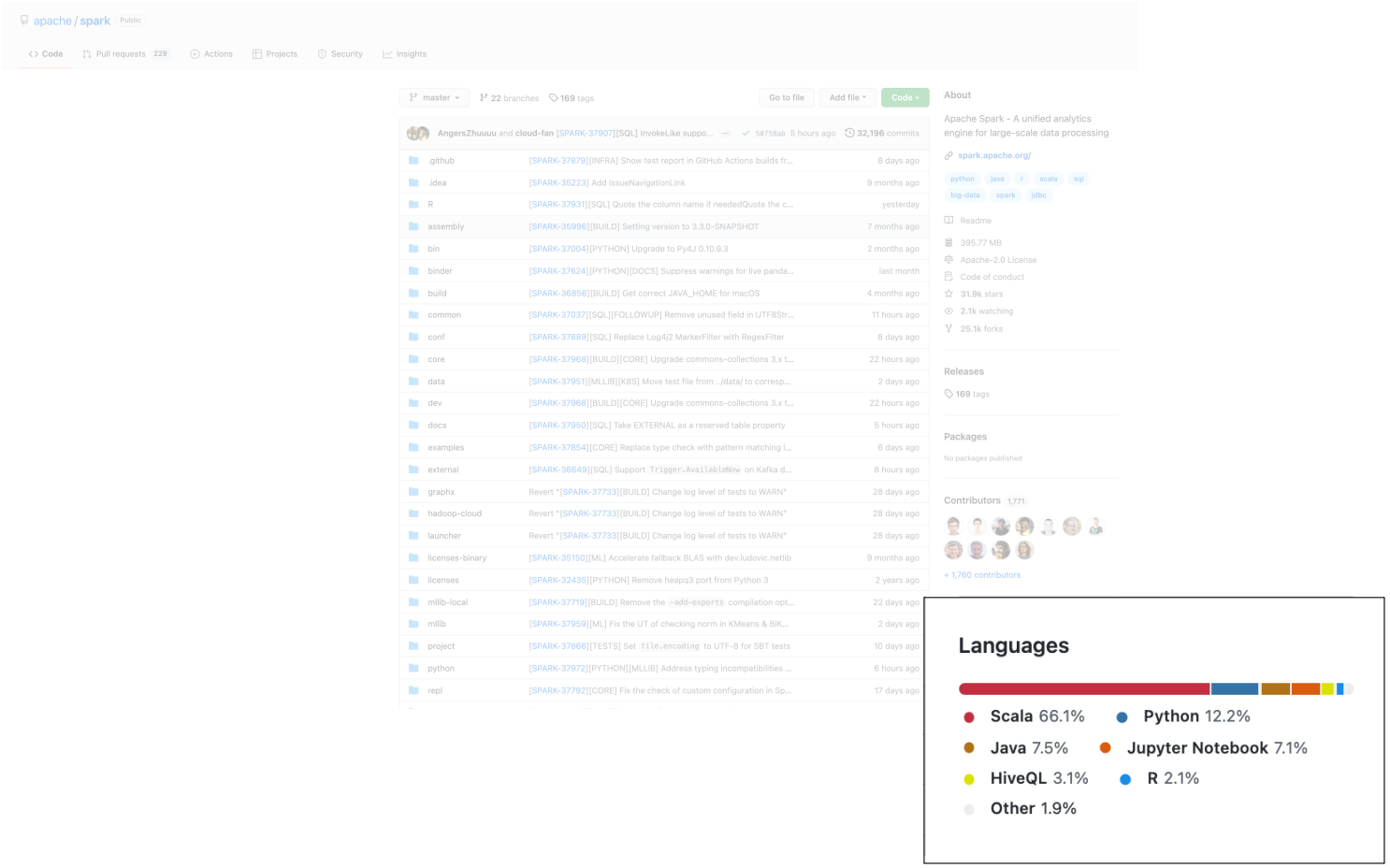


Makes information identification a non-trivial problem

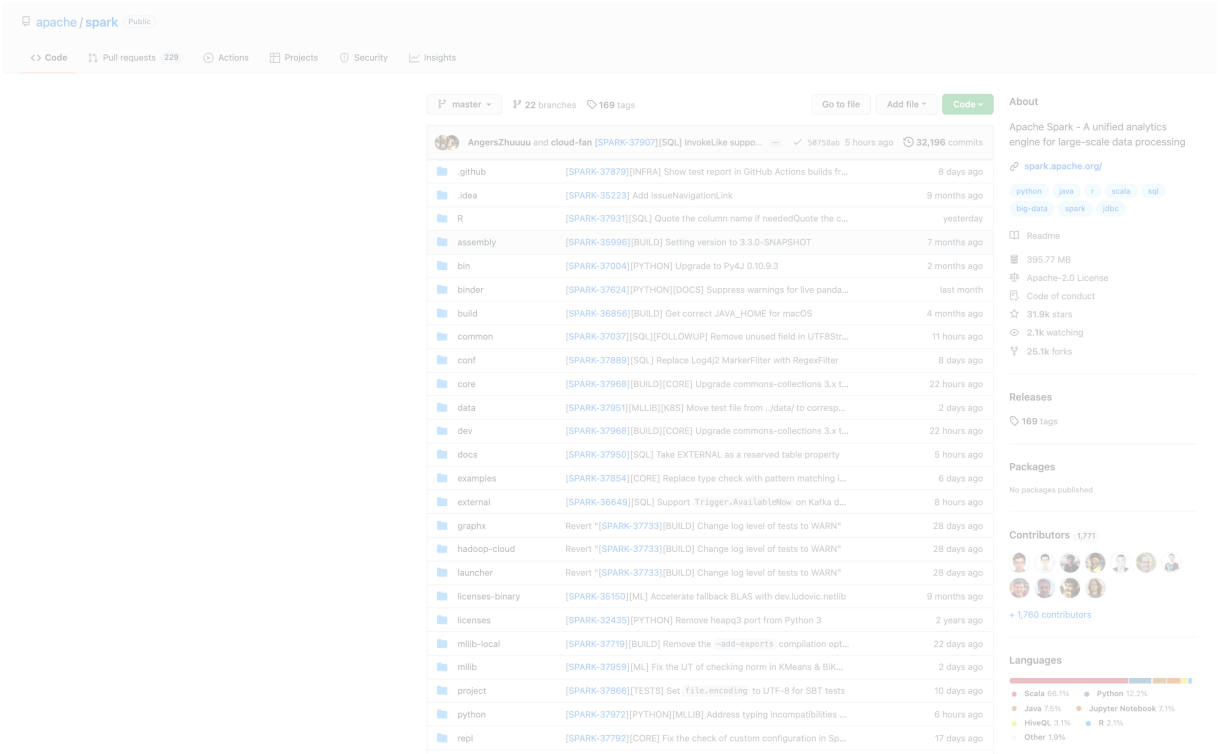
Increasing **multi-language** environments



Increasing **multi-language** environments



Increasing **multi-language** environments



97% of **open-source** projects used **two or more** programming languages

- Tomassetti et al. 2014

Each language has its **own conventions** to write comments

Given the increasing use of multi-language environments, we need a deeper understanding of **developer commenting practices** across **languages**

Given the increasing use of multi-language environments, we need a deeper understanding of **developer commenting practices** across **languages**

RQ1: What types of information are present in class comments?
To what extent do information types vary across programming languages?

Information types in comments

```
/**
* A class representing a window on the screen.
*
* For example:
* <pre>
*     Window win = new Window(parent);
*     win.show();
* </pre>
*
* @author Sami Shaio
* @version 1.13, 06/08/06
* @see java.awt.BaseWindow
*/
class Window extends BaseWindow {
  …
}
```

Summary

Information types in comments

```
/**
* A class representing a window on the screen.
*
* For example:
* <pre>
*     Window win = new Window(parent);
*     win.show();
* </pre>
*
* @author Sami Shaio
* @version 1.13, 06/08/06
* @see java.awt.BaseWindow
*/
class Window extends BaseWindow {
  …
}
```

Summary

Usage

RQ1: What types of information are present in class comments?

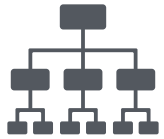
Java, Python, Smalltalk,
20 projects

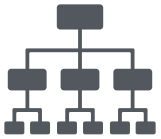
Extract class comments,
37, 446 comments

Classify sample comments,
1, 066 comments

Output: a taxonomy, and a classifier

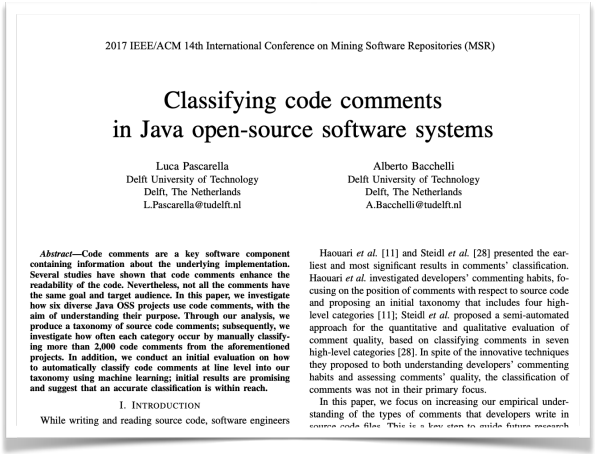
Comment taxonomies in languages





Pascarella et al., 2017

Java



2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

Classifying code comments in Java open-source software systems

Luca Pascarella
Delft University of Technology
Delft, The Netherlands
L.Pascarella@tudelft.nl

Alberto Bacchelli
Delft University of Technology
Delft, The Netherlands
A.Bacchelli@tudelft.nl

***Abstract*—Code comments are a key software component containing information about the underlying implementation. Several studies have shown that code comments enhance the readability of the code. Nevertheless, not all the comments have the same goal and target audience. In this paper, we investigate how six diverse Java OSS projects use code comments, with the aim of understanding their purpose. Through our analysis, we produce a taxonomy of source code comments; subsequently, we investigate how often each category occur by manually classifying more than 2,000 code comments from the aforementioned projects. In addition, we conduct an initial evaluation on how to automatically classify code comments at line level into our taxonomy using machine learning; initial results are promising and suggest that an accurate classification is within reach.**

I. Introduction

While writing and reading source code, software engineers

Haouari *et al.* [11] and Steidl *et al.* [28] presented the earliest and most significant results in comments' classification. Haouari *et al.* investigated developers' commenting habits, focusing on the position of comments with respect to source code and proposing an initial taxonomy that includes four high-level categories [11]; Steidl *et al.* proposed a semi-automated approach for the quantitative and qualitative evaluation of comment quality, based on classifying comments in seven high-level categories [28]. In spite of the innovative techniques they proposed to both understanding developers' commenting habits and assessing comments' quality, the classification of comments was not in their primary focus.

In this paper, we focus on increasing our empirical understanding of the types of comments that developers write in source code files. This is a key step to guide future research

Rani et al., 2021

Smalltalk

Empirical Software Engineering (2021) 26: 112
https://doi.org/10.1007/s10664-021-09981-5

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

Pooja Rani[1] · Sebastiano Panichella[2] · Manuel Leuenberger[1] · Mohammad Ghafari[3] · Oscar Nierstrasz[1]

Accepted: 25 May 2021 Published online: 18 August 2021

Abstract

Context Previous studies have characterized code comments in various programming languages, showing how 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 understanding developer practices to write comments. None of them has compared such developer practices to the standard comment guidelines to study the extent to which developers follow the guidelines.

Objective Therefore, our goal is to investigate developer commenting practices and compare them to the comment guidelines.

Method This paper reports the first empirical study investigating commenting practices in Pharo Smalltalk. First, we analyze class comment evolution over seven Pharo versions. Then, we quantitatively and qualitatively investigate the information types embedded in class comments. Finally, we study the adherence of developer commenting practices to the official *class comment template* over Pharo versions.

Results Our results 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 code to comment ratio. We furthermore found three times as many information types in class comments as those suggested by the template. However, the information types suggested by the 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.

Conclusion The results suggest the need to standardize the commenting guidelines for formatting the text, and to provide headers for the different information types to ensure a consistent style and to identify the information easily. Given the importance of high-quality code comments, we draw numerous implications for developers and researchers to improve the support for comment quality assessment tools.

Keywords Commenting practices · Class comment analysis · Comment evolution · Template analysis · Pharo · Program comprehension

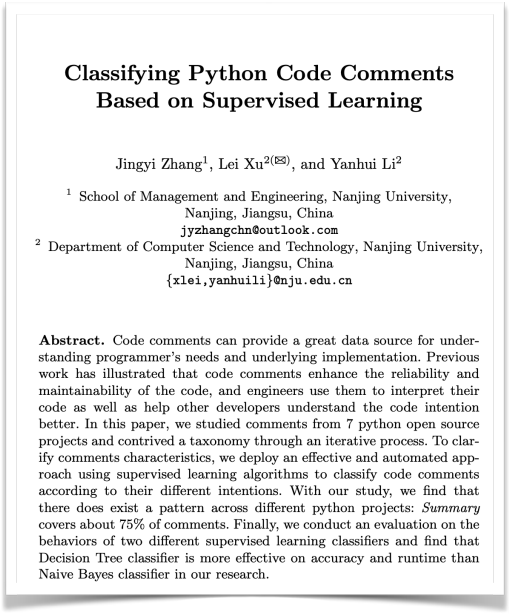
Communicated by Andrian Marcus

✉ Pooja Rani
pooja.rani@inf.unibe.ch

Extended author information available on the last page of the article

Zhang et. al., 2018

Python



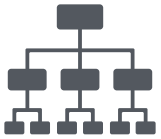
Classifying Python Code Comments Based on Supervised Learning

Jingyi Zhang[1], Lei Xu[2(✉)], and Yanhui Li[2]

[1] School of Management and Engineering, Nanjing University, Nanjing, Jiangsu, China
jyzhangchn@outlook.com

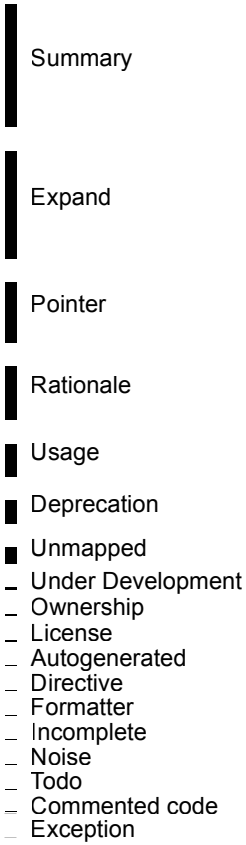
[2] Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, China
{xlei,yanhuili}@nju.edu.cn

Abstract. Code comments can provide a great data source for understanding programmer's needs and underlying implementation. Previous work has illustrated that code comments enhance the reliability and maintainability of the code, and engineers use them to interpret their code as well as help other developers understand the code intention better. In this paper, we studied comments from 7 python open source projects and contrived a taxonomy through an iterative process. To clarify comments characteristics, we deploy an effective and automated approach using supervised learning algorithms to classify code comments according to their different intentions. With our study, we find that there does exist a pattern across different python projects: *Summary* covers about 75% of comments. Finally, we conduct an evaluation on the behaviors of two different supervised learning classifiers and find that Decision Tree classifier is more effective on accuracy and runtime than Naive Bayes classifier in our research.



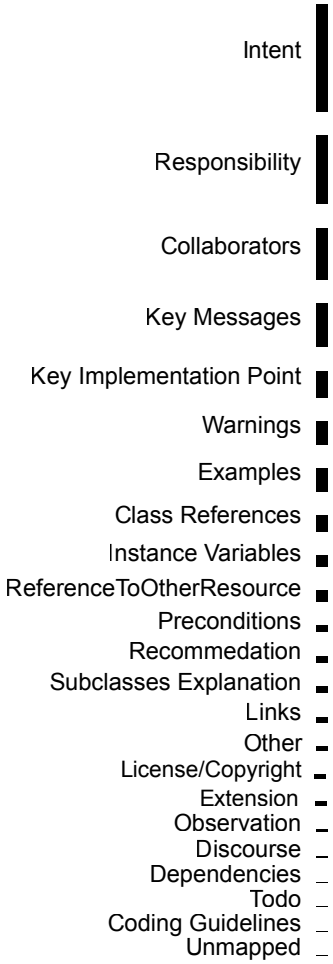
Pascarella et al., 2017

Java



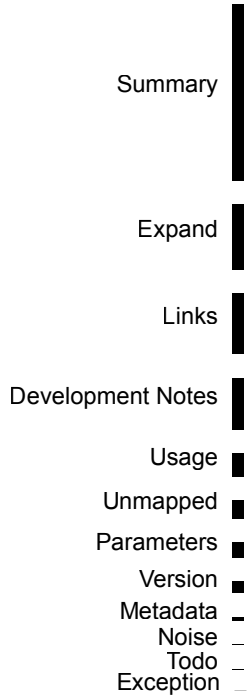
Rani et al., 2021

Smalltalk

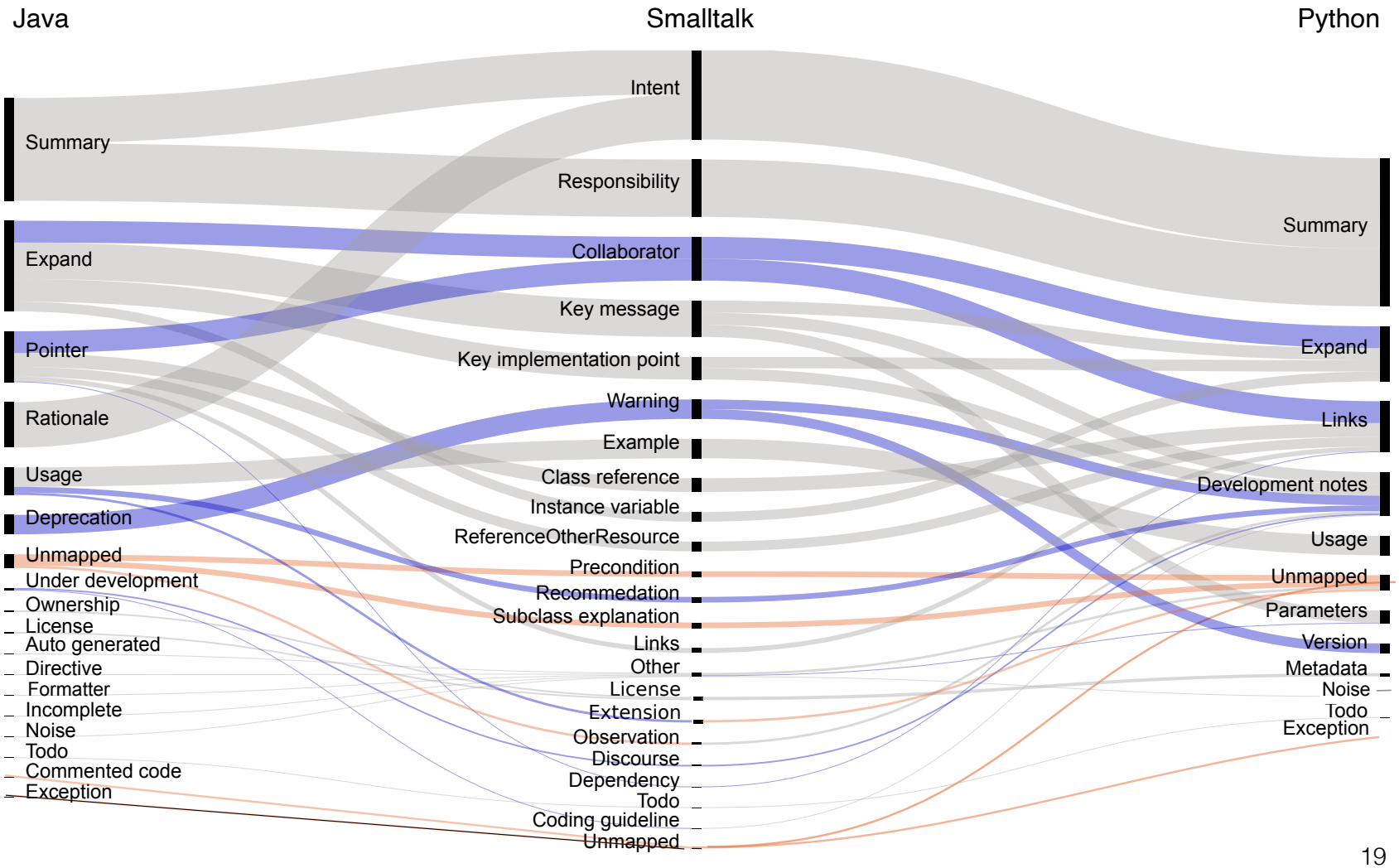


Zhang et. al., 2018

Python



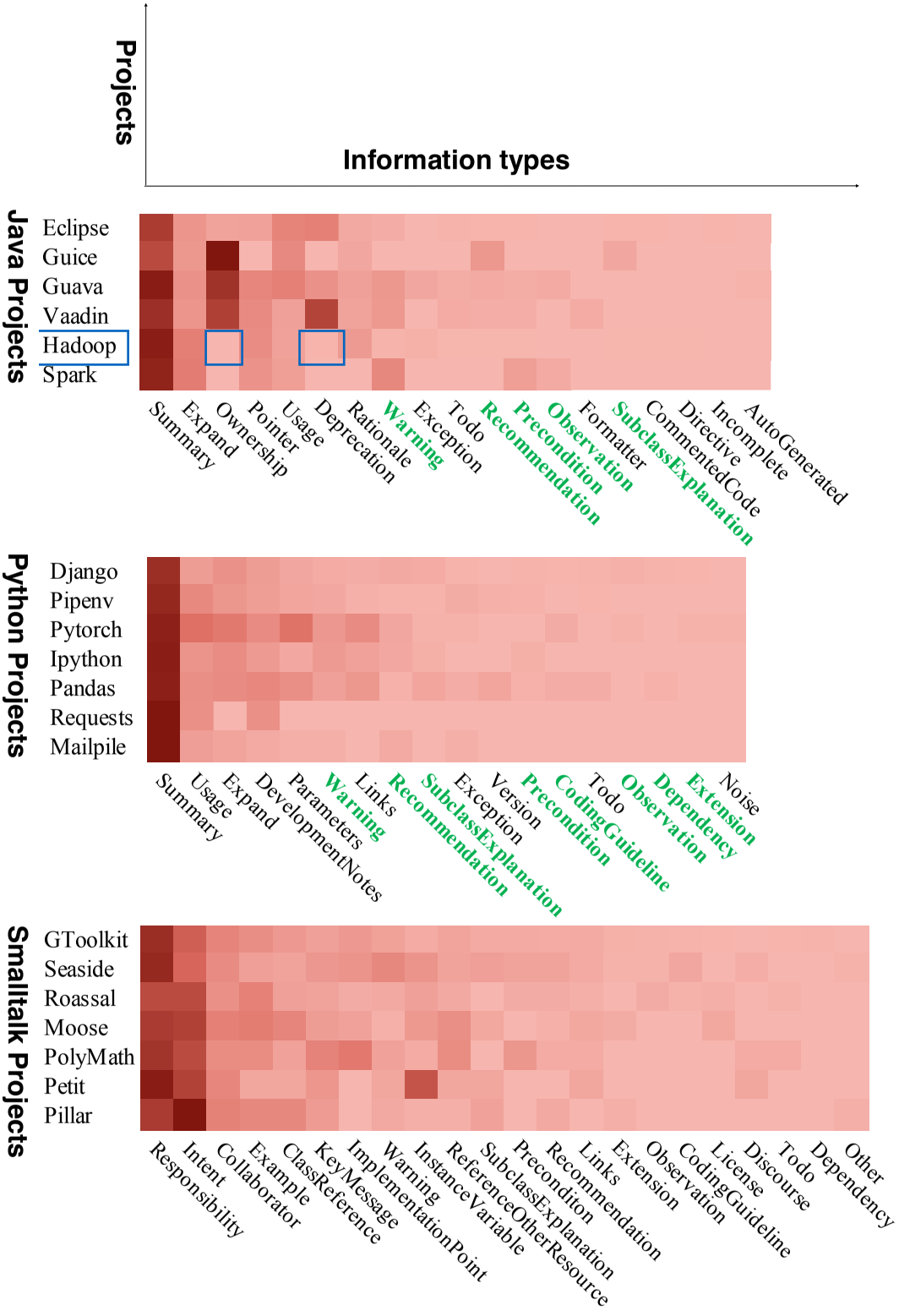
Pascarella et al., 2017 Rani et al., 2021 Zhang et. al., 2018



CCTM (Class Comment Types Model)

Information types
Projects
AutoGenerated
Incomplete
Directive
CommentedCode
SubclassExplanation
Formatter
Observation
Precondition
Recommendation
Todo
Exception
Warning
Rationale
Deprecation
Usage
Pointer
Ownership
Expand
Summary
Eclipse
Guice
Guava
Vaadin
Hadoop
Spark
Java Projects
Noise
Extension
Dependency
Observation
Todo
CodingGuideline
Precondition
Version
Exception
SubclassExplanation
Recommendation
Links
Warning
Parameters
DevelopmentNotes
Expand
Usage
Summary
Django
Pipenv
Pytorch
Ipython
Pandas
Requests
Mailpile
Python Projects
Other
Dependency
Todo
Discourse
License
CodingGuideline
Observation
Extension
Links
Recommendation
Preconditon
SubclassExplanation
ReferenceOtherResource
InstanceVariable
Warning
ImplementationPoint
KeyMessage
ClassReference
Example
Collaborator
Intent
Responsibility
GToolkit
Seaside
Roassal
Moose
PolyMath
Petit
Pillar
Smalltalk Projects

Information types
Projects
AutoGenerated
Incomplete
Directive
CommentedCode
SubclassExplanation
Formatter
Observation
Precondition
Recommendation
Todo
Exception
Warning
Rationale
Deprecation
Usage
Pointer
Ownership
Expand
Summary
Eclipse
Guice
Guava
Vaadin
Hadoop
Spark
Java Projects
Noise
Extension
Dependency
Observation
Todo
CodingGuideline
Precondition
Version
Exception
SubclassExplanation
Recommendation
Links
Warning
Parameters
DevelopmentNotes
Expand
Usage
Summary
Django
Pipenv
Pytorch
Ipython
Pandas
Requests
Mailpile
Python Projects
Other
Dependency
Todo
Discourse
License
CodingGuideline
Observation
Extension
Links
Recommendation
Precondiiton
SubclassExplanation
ReferenceOtherResource
InstanceVariable
Warning
ImplementationPoint
KeyMessage
ClassReference
Example
Collaborator
Intent
Responsibility
GToolkit
Seaside
Roassal
Moose
PolyMath
Petit
Pillar
Smalltalk Projects


Projects
Information types
Java Projects
Eclipse
Guice
Guava
Vaadin
Hadoop
Spark
Summary
Expand
Ownership
Pointer
Usage
Deprecation
Rationale
Warning
Exception
Todo
Recommendation
Precondition
Observation
Formatter
SubclassExplanation
CommentedCode
Directive
Incomplete
AutoGenerated
Python Projects
Django
Pipenv
Pytorch
Ipython
Pandas
Requests
Mailpile
Summary
Usage
Expand
DevelopmentNotes
Parameters
Warning
Links
RecommendationSubclassExplanation
Exception
Version
PreconditionCodingGuideline
Todo
Observation
DependencyExtension
Noise
Smalltalk Projects
GToolkit
Seaside
Roassal
Moose
PolyMath
Petit
Pillar
Responsibility
Intent
Collaborator
Example
ClassReference
KeyMessage
ImplementationPoint
Warning
InstanceVariable
ReferenceOtherResource
SubclassExplanation
Precondition
Recommendation
Links
Extension
Observation
CodingGuideline
License
Discourse
Todo
Dependency
Other

Given the increasing use of multi-language environment, we need a deeper understanding of **developer commenting practices** across **languages**

RQ1: What types of information are present in class comments? To what extent do information types vary across programming languages?

RQ2: Can machine learning be used to automatically identify class comment types according to CCTM?

```
/**
* A class representing a window on the screen.
*
* For example:
* <pre>
*     Window win = new Window(parent);
*     win.show();
* </pre>
*
* @author Sami Shaio
* @version 1.13, 06/08/06
* @see java.awt.BaseWindow
*/
class Window extends BaseWindow {
  …
}
```

Summary

Class represents [something]

[verb]s [noun]

Recurrent natural language patterns exist in various information types

```
/**
* A class representing a window on the screen.
*
* For example:
* <pre>
*     Window win = new Window(parent);
*     win.show();
* </pre>
*
* @author Sami Shaio
* @version 1.13, 06/08/06
* @see java.awt.BaseWindow
*/
class Window extends BaseWindow {
   …
}
```

Summary

Class represents [something]

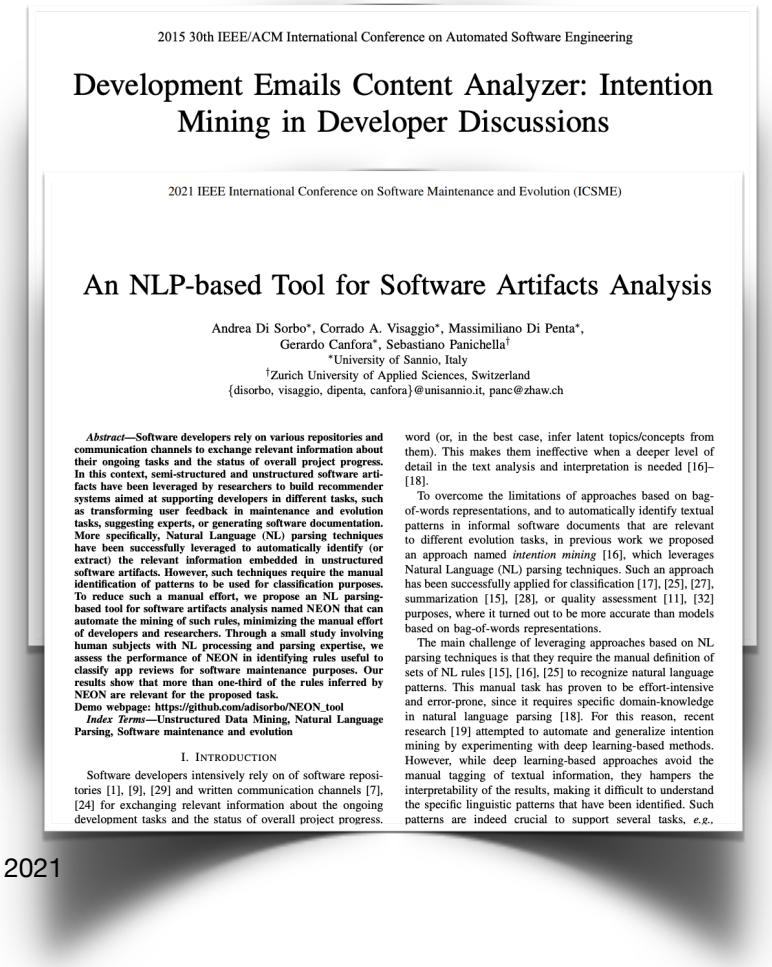
[verb]s [noun]

How do we extract such patterns?

Extract **patterns**

- To automatically identify textual patterns in informal software documents, **intention mining** can be used.

- Di Sorbo et al., developed a tool, **NEON**, to detect natural language patterns.



2015 30th IEEE/ACM International Conference on Automated Software Engineering

Development Emails Content Analyzer: Intention Mining in Developer Discussions

2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)

An NLP-based Tool for Software Artifacts Analysis

Andrea Di Sorbo*, Corrado A. Visaggio*, Massimiliano Di Penta*,
Gerardo Canfora*, Sebastiano Panichella†
*University of Sannio, Italy
†Zurich University of Applied Sciences, Switzerland
{disorbo, visaggio, dipenta, canfora}@unisannio.it, panc@zhaw.ch

***Abstract*—Software developers rely on various repositories and communication channels to exchange relevant information about their ongoing tasks and the status of overall project progress. In this context, semi-structured and unstructured software artifacts have been leveraged by researchers to build recommender systems aimed at supporting developers in different tasks, such as transforming user feedback in maintenance and evolution tasks, suggesting experts, or generating software documentation. More specifically, Natural Language (NL) parsing techniques have been successfully leveraged to automatically identify (or extract) the relevant information embedded in unstructured software artifacts. However, such techniques require the manual identification of patterns to be used for classification purposes. To reduce such a manual effort, we propose an NL parsing-based tool for software artifacts analysis named NEON that can automate the mining of such rules, minimizing the manual effort of developers and researchers. Through a small study involving human subjects with NL processing and parsing expertise, we assess the performance of NEON in identifying rules useful to classify app reviews for software maintenance purposes. Our results show that more than one-third of the rules inferred by NEON are relevant for the proposed task.**
Demo webpage: https://github.com/adisorbo/NEON_tool
***Index Terms*—Unstructured Data Mining, Natural Language Parsing, Software maintenance and evolution**

I. Introduction

Software developers intensively rely on of software repositories [1], [9], [29] and written communication channels [7], [24] for exchanging relevant information about the ongoing development tasks and the status of overall project progress. word (or, in the best case, infer latent topics/concepts from them). This makes them ineffective when a deeper level of detail in the text analysis and interpretation is needed [16]–[18].

To overcome the limitations of approaches based on bag-of-words representations, and to automatically identify textual patterns in informal software documents that are relevant to different evolution tasks, in previous work we proposed an approach named *intention mining* [16], which leverages Natural Language (NL) parsing techniques. Such an approach has been successfully applied for classification [17], [25], [27], summarization [15], [28], or quality assessment [11], [32] purposes, where it turned out to be more accurate than models based on bag-of-words representations.

The main challenge of leveraging approaches based on NL parsing techniques is that they require the manual definition of sets of NL rules [15], [16], [25] to recognize natural language patterns. This manual task has proven to be effort-intensive and error-prone, since it requires specific domain-knowledge in natural language parsing [18]. For this reason, recent research [19] attempted to automate and generalize intention mining by experimenting with deep learning-based methods. However, while deep learning-based approaches avoid the manual tagging of textual information, they hampers the interpretability of the results, making it difficult to understand the specific linguistic patterns that have been identified. Such patterns are indeed crucial to support several tasks, *e.g.*,

 Di Sorbo et. al., *An NLP-based Tool for Software Artifact Analysis*. ICSME 2021

Example patterns from summary

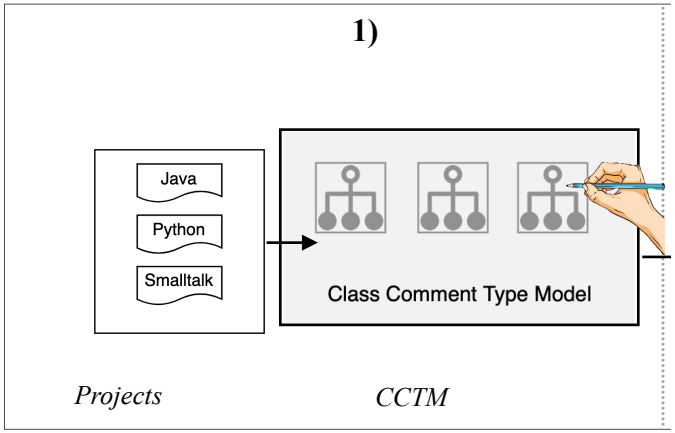
```
/**
*A class representing a window on the
 screen.
*For example:
*<pre>
*Window win = new Window(parent);
*win.show();
*</pre>
*
*@author Sami Shaio
*@version 1.13, 06/08/06
*@see java.awt.BaseWindow
*/

class Window extends BaseWindow{
..
}
```

Summary

Class represents [something]

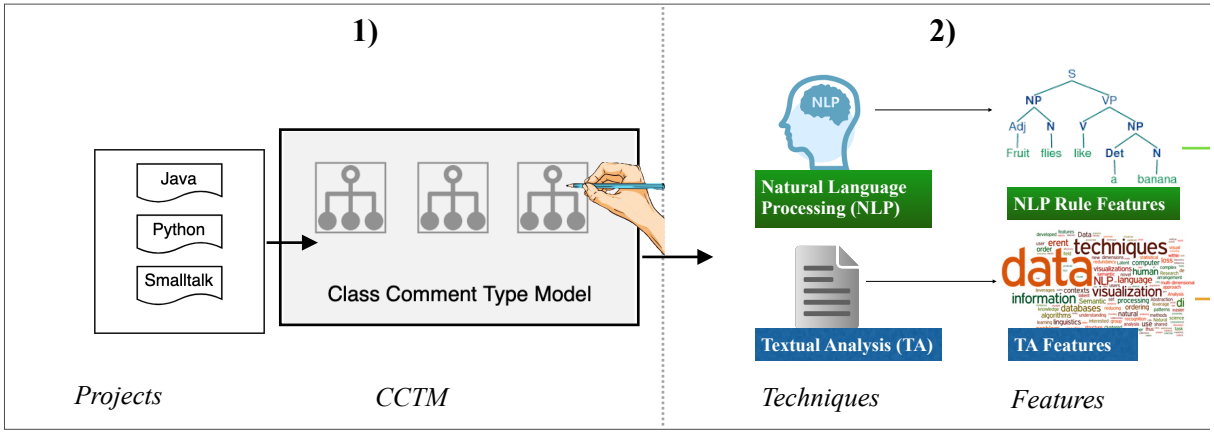
```
<NLP_heuristic>
    <sentence type="declarative"/>
    <type>nsubj/dobj</type>
    <text>Class represents [something].</text>
    <conditions>
        <condition>nsubj.governor="represent"</condition>
        <condition>nsubj.dependent="class"</condition>
        <condition>nsubj.governor=dobj.governor</condition>
    </conditions>
    <sentence_class>summary</sentence_class>
</NLP_heuristic>
```

Automatic identification of information types



Ground truth: 1,066 classified comments

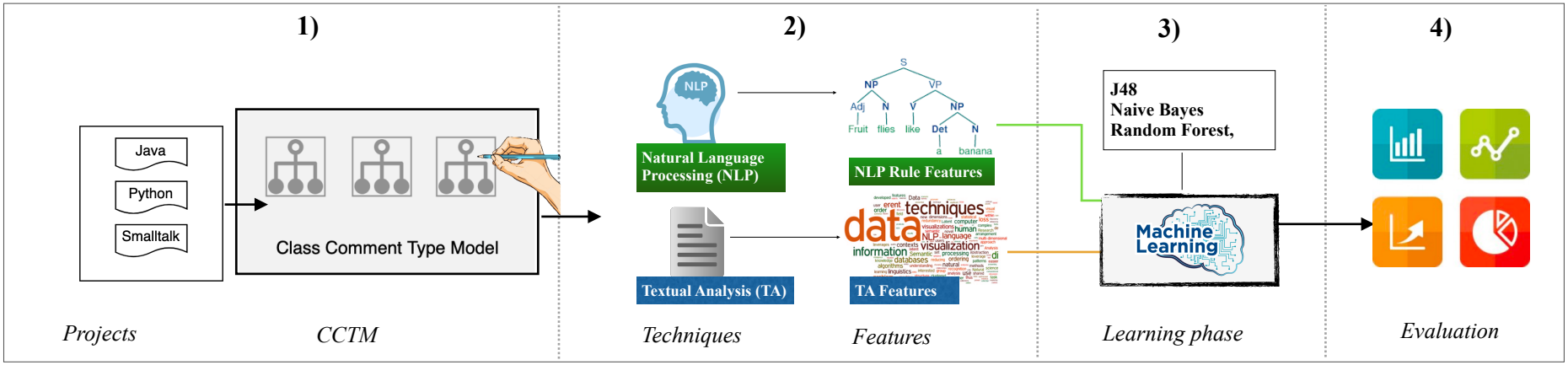
Automatic identification of information types



Ground truth: 1,066 classified comments

Features: recurrent NL patterns + text features

Automatic identification of information types

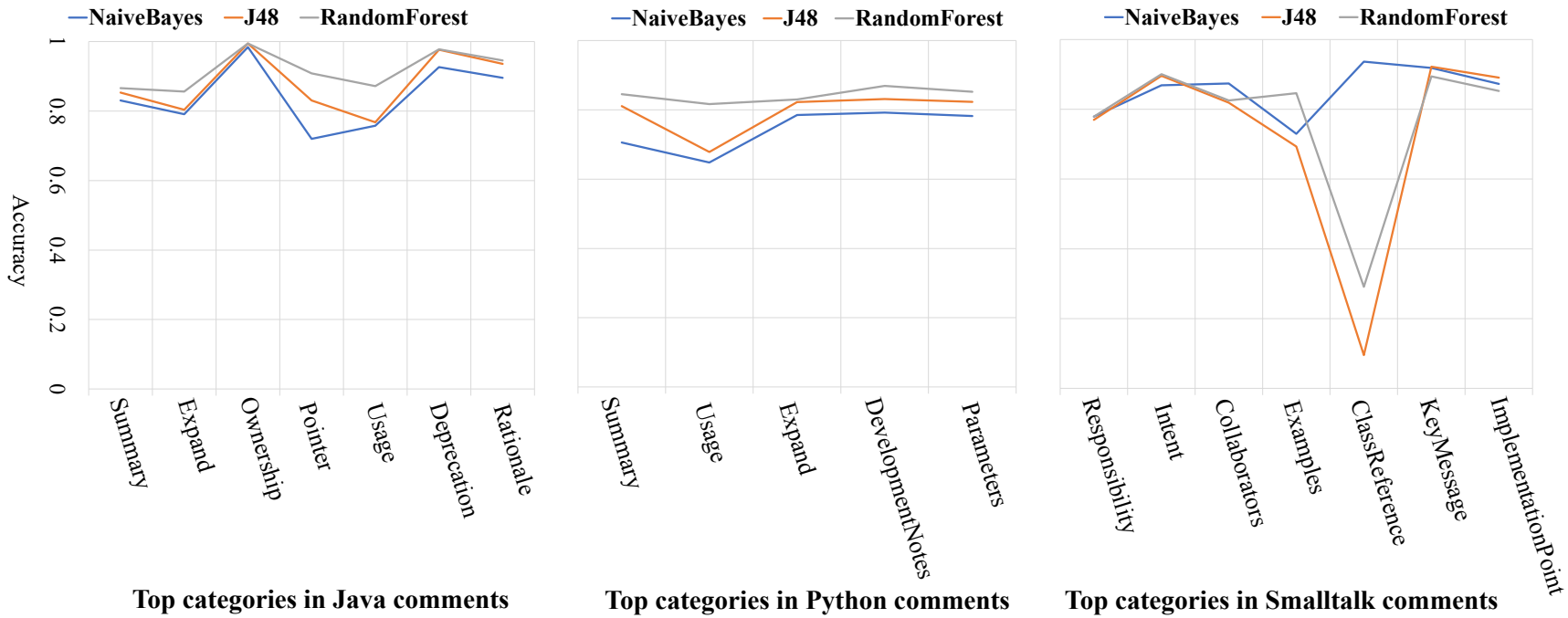


Ground truth: 1,066 classified comments

Features: recurrent NL patterns + text features

Supervised ML algorithms

Results



Random Forest technique classifies comments better

The ultimate goal of automatically assessing comments is still far away…

Future work

Which information types do developers find important?

How do various information types support developers?

What quality attributes are important for comments?

An IDE plugin to support automatic assessment of comments.

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

Paper

https://www.sciencedirect.com/science/article/pii/S0164121221001448

Replication Package on GitHub

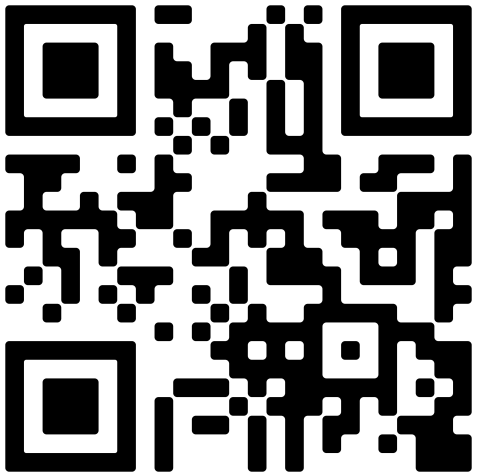
https://github.com/poojaruhal/RP-class-comment-classification.

YouTube

https://www.youtube.com/watch?v=_auMqCsxg0s

https://twitter.com/poojaruhal

http://scg.unibe.ch/staff/Pooja-Rani



Summary

Challenges

No standard definition of comments

No strict syntax and structure conventions

Lack of quality assessment tools

Makes information identification a non-trivial problem

7

RQ1: What types of information are present in class comments?

Java, Python, Smalltalk,
20 projects

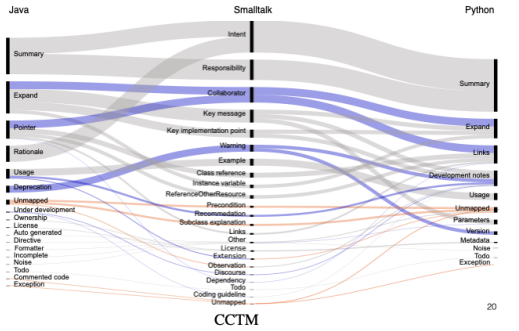
Extract class comments,
37, 446 comments

Classify sample comments,
1, 066 comments

Output: a taxonomy, and a classifier

16

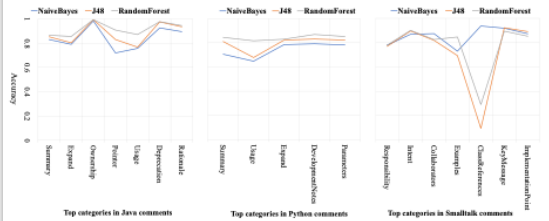
Pascarella et al., 2017 Rani et al., 2021 Zhang et. al., 2018



CCTM

20

Automatically identify an information type



Random Forest technique classifies comments better

20