

Assessing Comment Quality in Object-Oriented Languages



Pooja Rani

PhD Defense

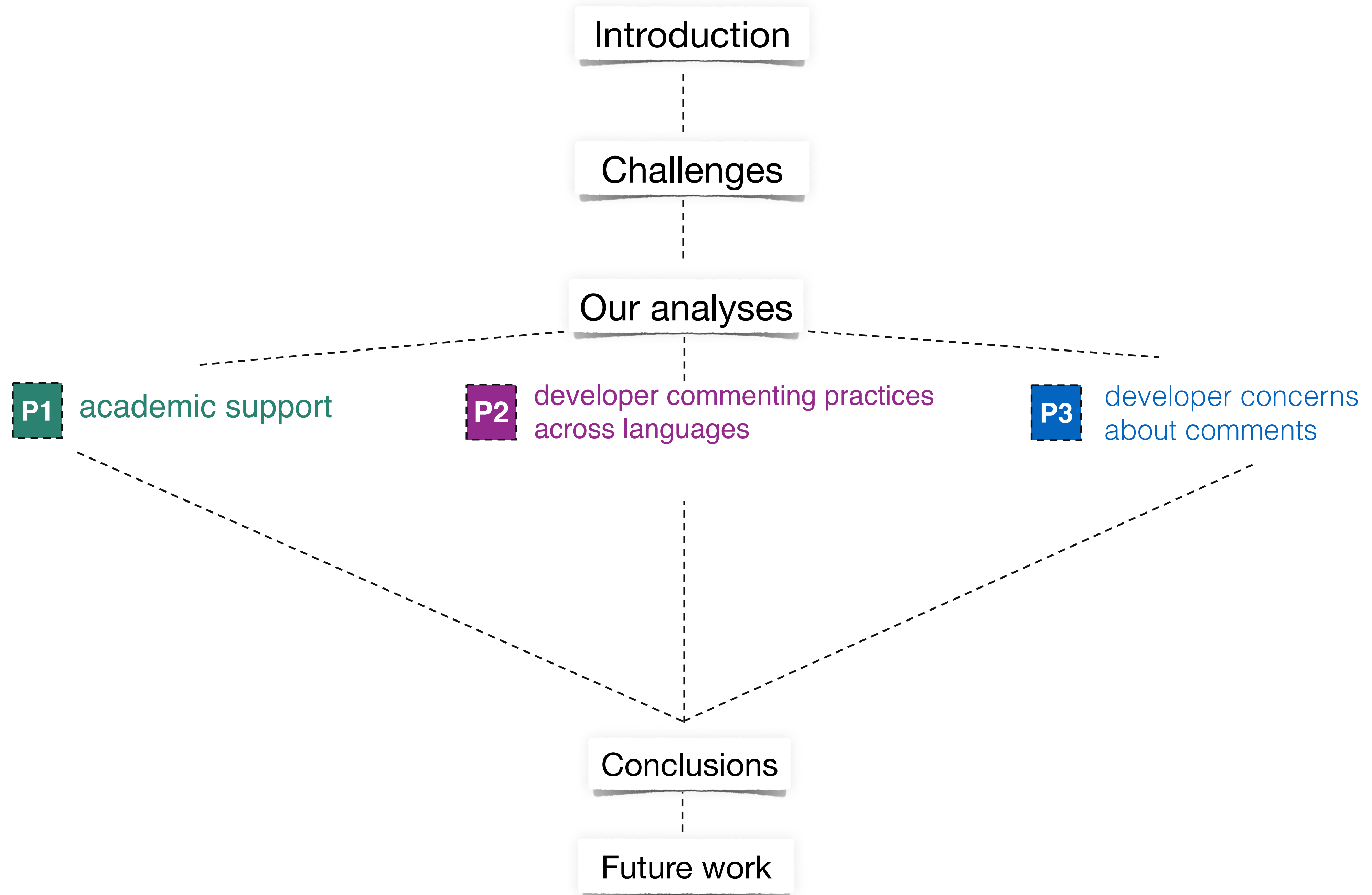
Supervisors:

Prof. Dr. Oscar Nierstrasz

Dr. Sebastiano Panichella

31 January 2022

Roadmap



Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```



Java
class comment

Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Trustworthy form of documentation

- McMillan et al. 2010

Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Trustworthy form of documentation

- McMillan et al. 2010

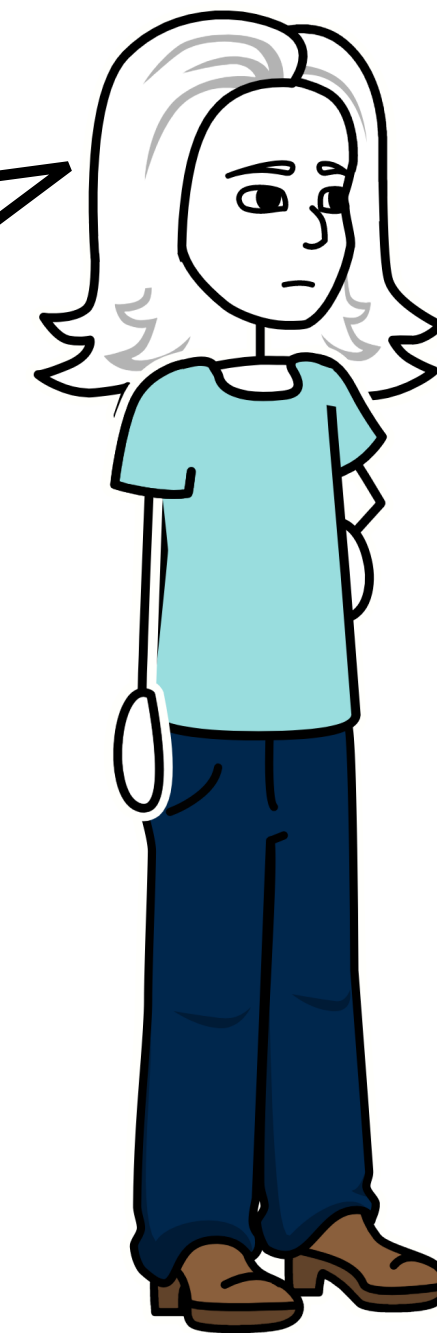
High-quality comments support developers in various activities

- Dekel et al. 2009

Code 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  
 * @see java.awt.Button  
 */  
class Window extends BaseWindow {  
    ...  
}
```

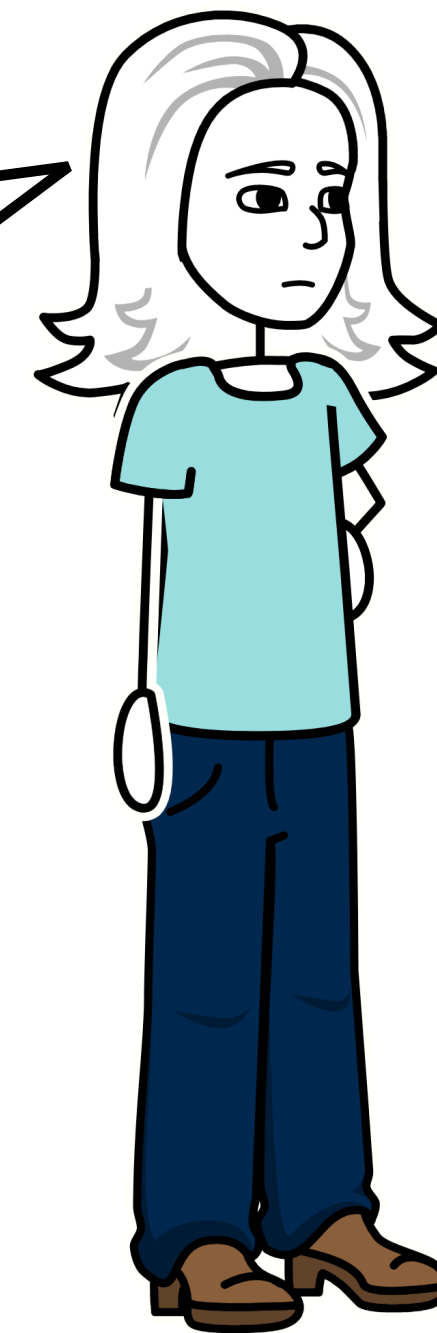
Does this
comment contain any
example?



Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Does this
comment contain any
example?

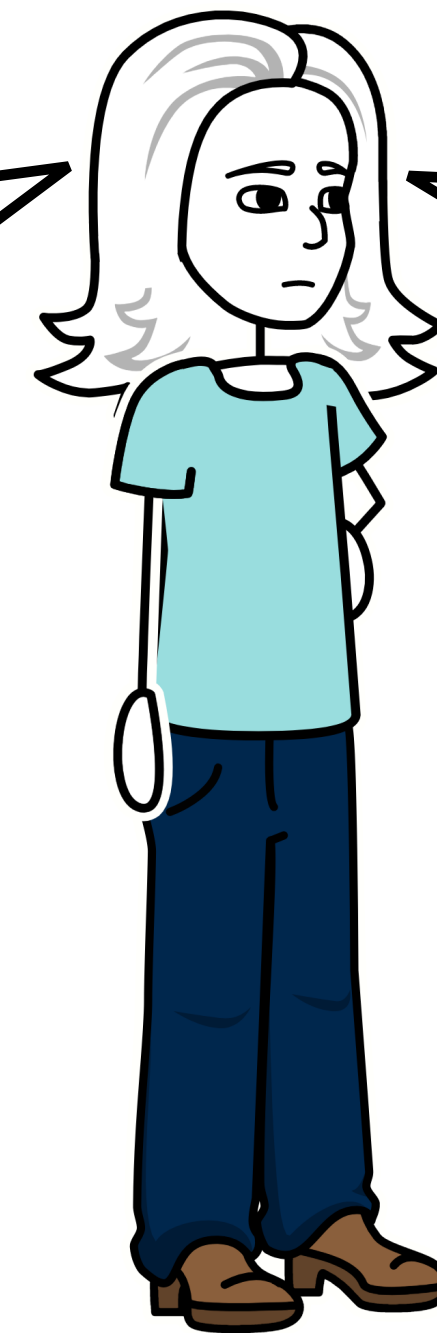


How to ensure comment quality?

```
/**
 * 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Does this comment contain any example?

Is this a high-quality comment?



How to ensure comment quality?

```
/**
 * 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

adequate?

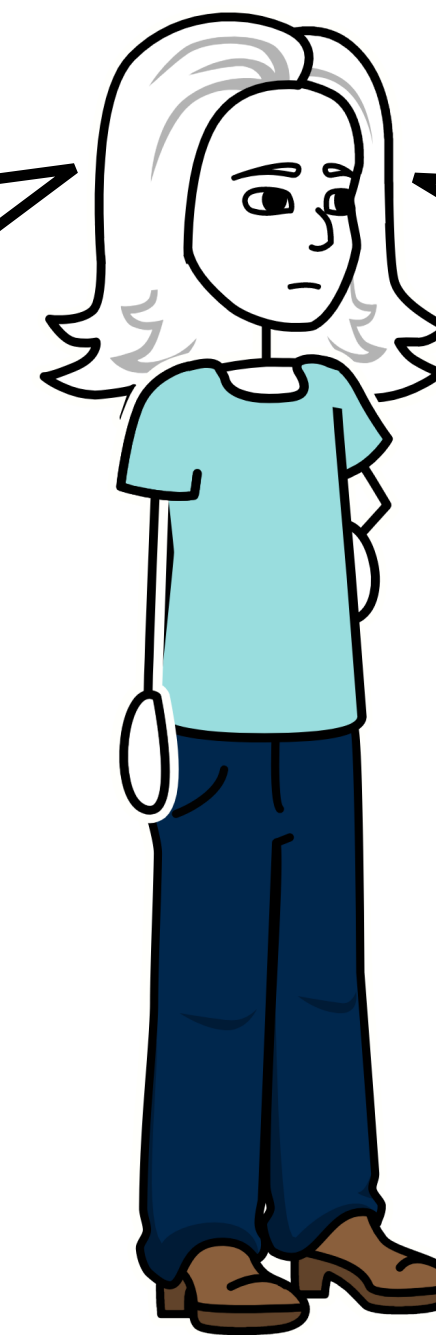
correct?

consistent?

Other quality attributes?

Does this comment contain any example?

Is this a high-quality comment?



Challenges

No standard definition of comment quality

No strict syntax and style conventions

Lack of quality assessment tools

Challenges

No standard definition of comment quality

No strict syntax and style conventions

Lack of quality assessment tools

Challenges

No standard definition of comment quality

No strict syntax and style conventions

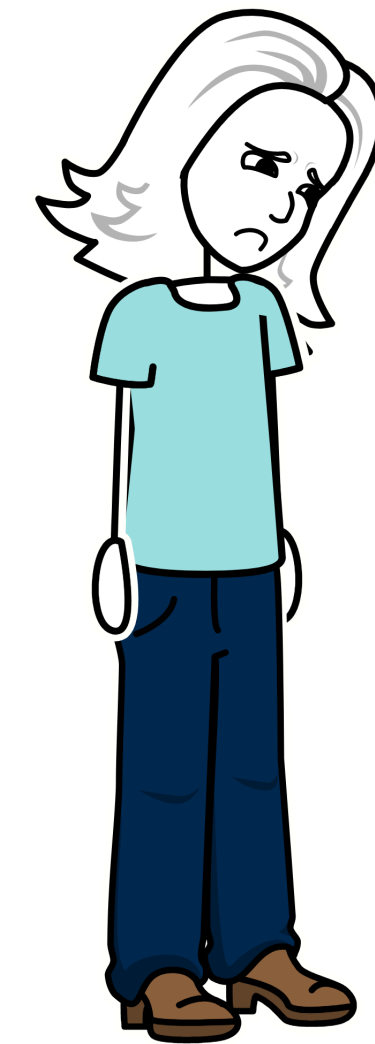
Lack of quality assessment tools

Challenges

No standard definition of comment quality

No strict syntax and style conventions

Lack of quality assessment tools



All these make **quality assessment a non-trivial problem**

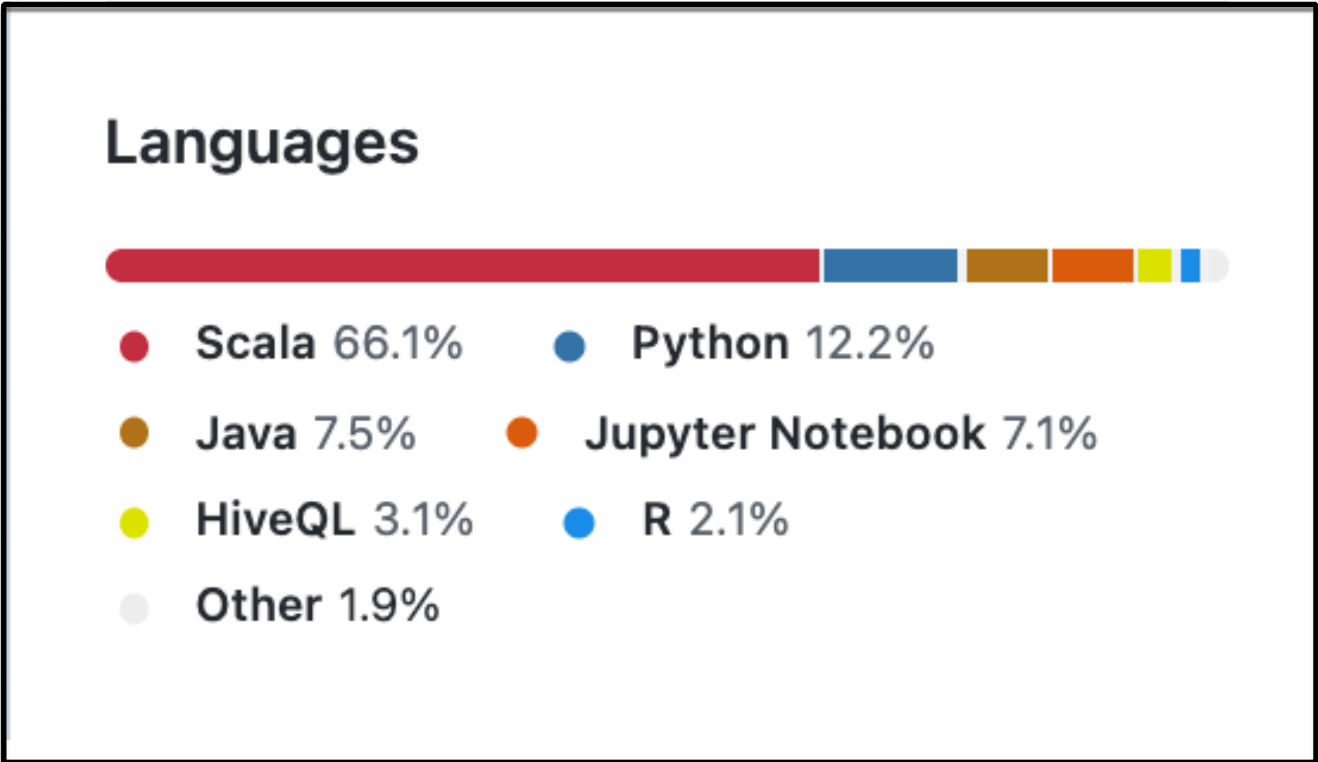
Increasing **multi-language** environments

The screenshot shows the Apache Spark GitHub repository page. At the top, the repository name "apache / spark" is displayed as "Public". Below this, navigation links for "Code", "Pull requests" (with a count of 229), "Actions", "Projects", "Security", and "Insights" are visible. The main content area shows the "master" branch selected, with 22 branches and 169 tags. A list of folders is shown, each with a corresponding pull request link and a brief description of the changes. The folders listed are: .github, .idea, R, assembly, bin, binder, build, common, conf, core, data, dev, docs, examples, external, and graphx.

Folder	Issue/PR Link	Description	Last Update
.github	[SPARK-37879][INFRA]	Show test report in GitHub Actions builds fr...	8 day
.idea	[SPARK-35223]	Add IssueNavigationLink	9 month
R	[SPARK-37931][SQL]	Quote the column name if neededQuote the c...	yest
assembly	[SPARK-35996][BUILD]	Setting version to 3.3.0-SNAPSHOT	7 month
bin	[SPARK-37004][PYTHON]	Upgrade to Py4J 0.10.9.3	2 month
binder	[SPARK-37624][PYTHON][DOCS]	Suppress warnings for live panda...	last r
build	[SPARK-36856][BUILD]	Get correct JAVA_HOME for macOS	4 month
common	[SPARK-37037][SQL][FOLLOWUP]	Remove unused field in UTF8Str...	11 hour
conf	[SPARK-37889][SQL]	Replace Log4j2 MarkerFilter with RegexFilter	8 day
core	[SPARK-37968][BUILD][CORE]	Upgrade commons-collections 3.x t...	22 hour
data	[SPARK-37951][MLLIB][K8S]	Move test file from ../data/ to corresp...	2 day
dev	[SPARK-37968][BUILD][CORE]	Upgrade commons-collections 3.x t...	22 hour
docs	[SPARK-37950][SQL]	Take EXTERNAL as a reserved table property	5 hour
examples	[SPARK-37854][CORE]	Replace type check with pattern matching i...	6 day
external	[SPARK-36649][SQL]	Support Trigger.AvailableNow on Kafka d...	8 hour
graphx	[SPARK-37733][BUILD]	Change log level of tests to WARN"	28 day

Increasing **multi-language** environments

The screenshot shows the Apache Spark GitHub repository page. At the top, it displays 'apache/spark' with a 'Public' badge. Below this are navigation links for Code, Pull requests (229), Actions, Projects, Security, and Insights. The main content area shows a list of recent commits, each with a folder icon, a commit message, and a timestamp. The folders listed include .github, .idea, R, assembly, bin, binder, build, common, conf, core, data, dev, docs, examples, external, graphx, hadoop-cloud, launcher, licenses-binary, licenses, mllib-local, mllib, project, python, and repl. To the right of the commit list, there are buttons for 'Go to file', 'Add file', and 'Code'. Below the commit list, there is an 'About' section with the text 'Apache Spark - A unified analytics engine for large-scale data processing', the website 'spark.apache.org', and various tags like 'python', 'java', 'r', 'scala', 'sql', 'big-data', 'spark', and 'jdbc'. There is also a 'Releases' section with '169 tags' and a 'Packages' section with 'No packages published'. At the bottom right, there is a 'Contributors' section with '1,771' contributors and a '+ 1,760 contributors' link.



Increasing **multi-language** environments

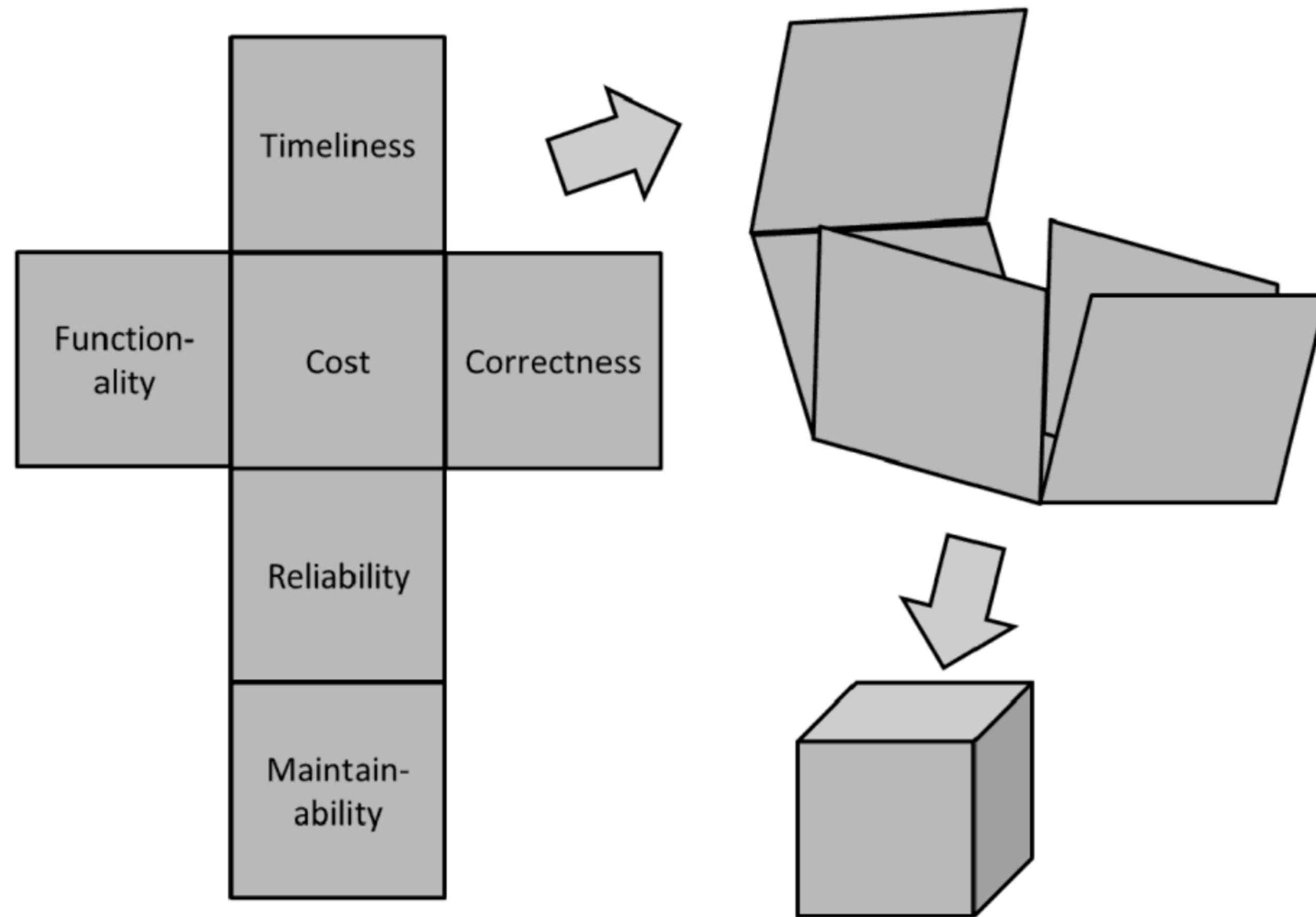
The screenshot shows the Apache Spark GitHub repository page. At the top, it displays the repository name 'apache/spark' and navigation links for Code, Pull requests (229), Actions, Projects, Security, and Insights. Below this, there are buttons for 'Go to file', 'Add file', and 'Code'. The main content area is a list of recent commits, each with a folder icon, a commit message, and a timestamp. The commits are sorted by time, with the most recent at the top. The right sidebar contains information about the repository, including the project name 'Apache Spark - A unified analytics engine for large-scale data processing', the website 'spark.apache.org', supported languages (python, java, r, scala, sql), and other details like '31.9k stars' and '25.1k forks'. At the bottom of the sidebar, there is a 'Languages' section with a horizontal bar chart showing the distribution of programming languages used in the project: Scala (66.1%), Python (12.2%), Java (7.5%), Jupyter Notebook (7.1%), HiveQL (3.1%), R (2.1%), and Other (1.9%).

Folder	Commit Message	Time Ago
.github	[SPARK-37879][INFRA] Show test report in GitHub Actions builds fr...	8 days ago
.idea	[SPARK-35223] Add IssueNavigationLink	9 months ago
R	[SPARK-37931][SQL] Quote the column name if neededQuote the c...	yesterday
assembly	[SPARK-35996][BUILD] Setting version to 3.3.0-SNAPSHOT	7 months ago
bin	[SPARK-37004][PYTHON] Upgrade to Py4J 0.10.9.3	2 months ago
binder	[SPARK-37624][PYTHON][DOCS] Suppress warnings for live panda...	last month
build	[SPARK-36856][BUILD] Get correct JAVA_HOME for macOS	4 months ago
common	[SPARK-37037][SQL][FOLLOWUP] Remove unused field in UTF8Str...	11 hours ago
conf	[SPARK-37889][SQL] Replace Log4j2 MarkerFilter with RegexFilter	8 days ago
core	[SPARK-37968][BUILD][CORE] Upgrade commons-collections 3.x t...	22 hours ago
data	[SPARK-37951][MLLIB][K8S] Move test file from ../data/ to corresp...	2 days ago
dev	[SPARK-37968][BUILD][CORE] Upgrade commons-collections 3.x t...	22 hours ago
docs	[SPARK-37950][SQL] Take EXTERNAL as a reserved table property	5 hours ago
examples	[SPARK-37854][CORE] Replace type check with pattern matching i...	6 days ago
external	[SPARK-36649][SQL] Support Trigger.AvailableNow on Kafka d...	8 hours ago
graphx	Revert "[SPARK-37733][BUILD] Change log level of tests to WARN"	28 days ago
hadoop-cloud	Revert "[SPARK-37733][BUILD] Change log level of tests to WARN"	28 days ago
launcher	Revert "[SPARK-37733][BUILD] Change log level of tests to WARN"	28 days ago
licenses-binary	[SPARK-35150][ML] Accelerate fallback BLAS with dev.ludovic.netlib	9 months ago
licenses	[SPARK-32435][PYTHON] Remove heapq3 port from Python 3	2 years ago
mlib-local	[SPARK-37719][BUILD] Remove the --add-exports compilation opt...	22 days ago
mlib	[SPARK-37959][ML] Fix the UT of checking norm in KMeans & BIK...	2 days ago
project	[SPARK-37866][TESTS] Set file.encoding to UTF-8 for SBT tests	10 days ago
python	[SPARK-37972][PYTHON][MLLIB] Address typing incompatibilities ...	6 hours ago
repl	[SPARK-37792][CORE] Fix the check of custom configuration in Sp...	17 days ago

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

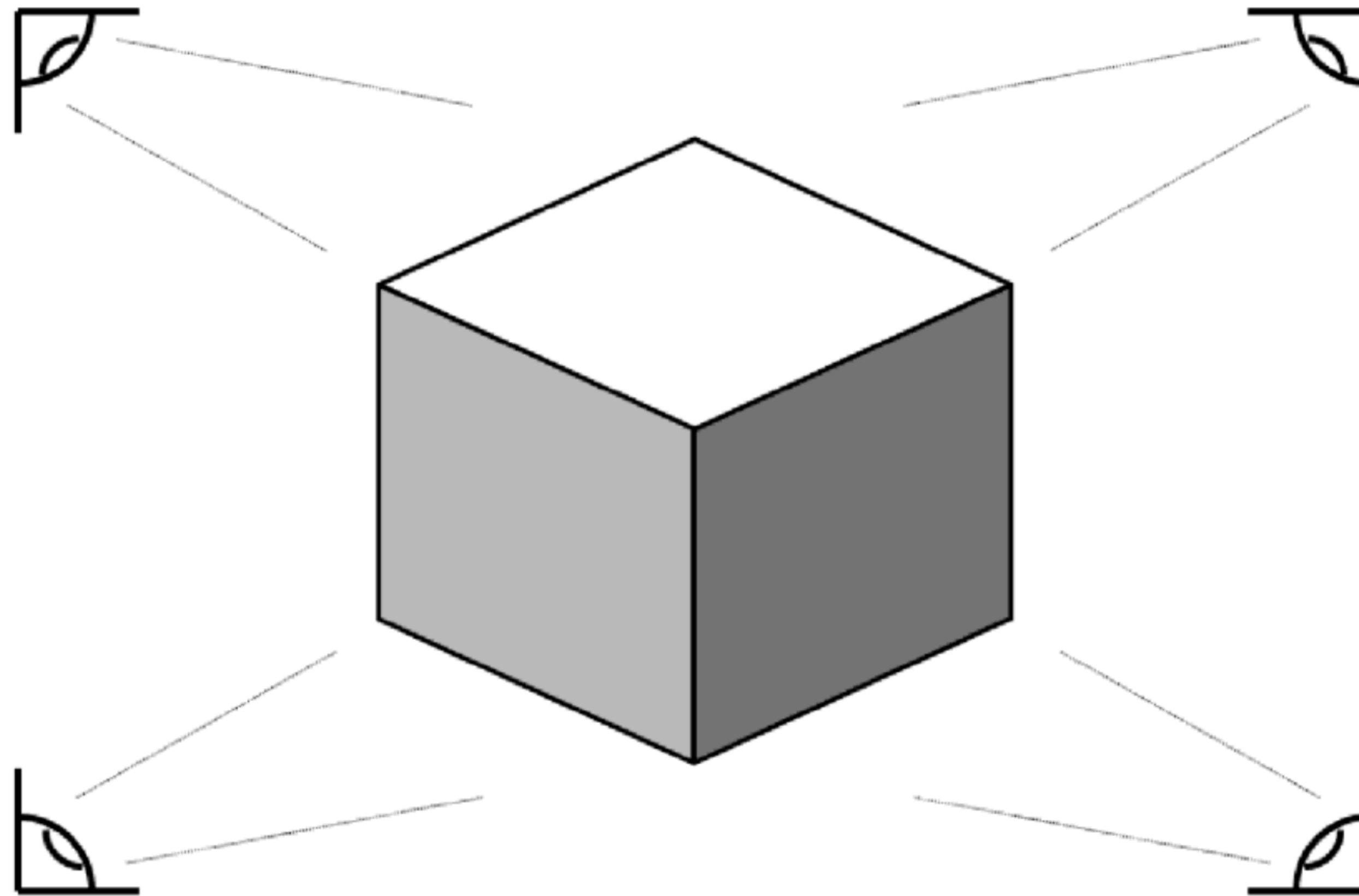
- Tomassetti et al. 2014

Quality is a multi-dimensional concept



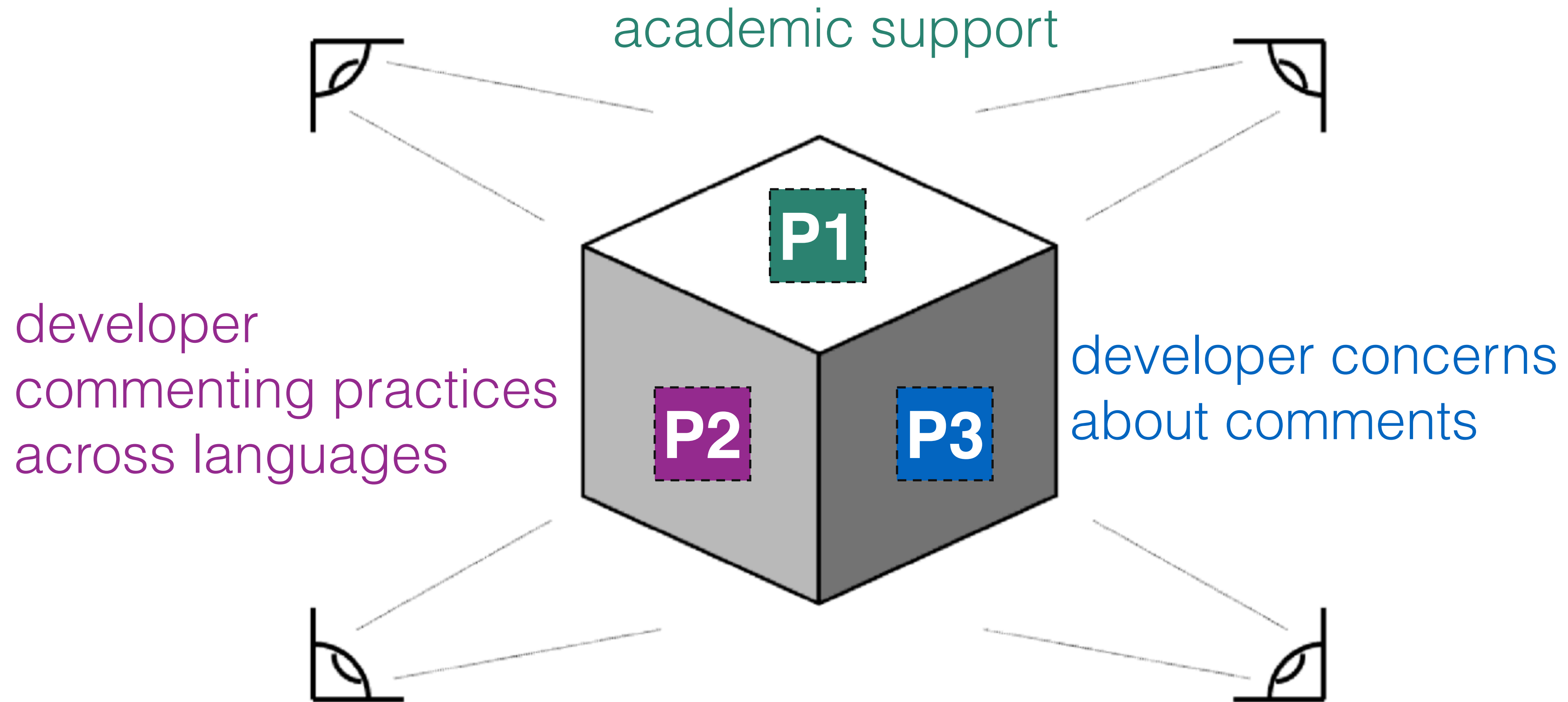
Gillies et al. 2011

It requires a multi-perspective view

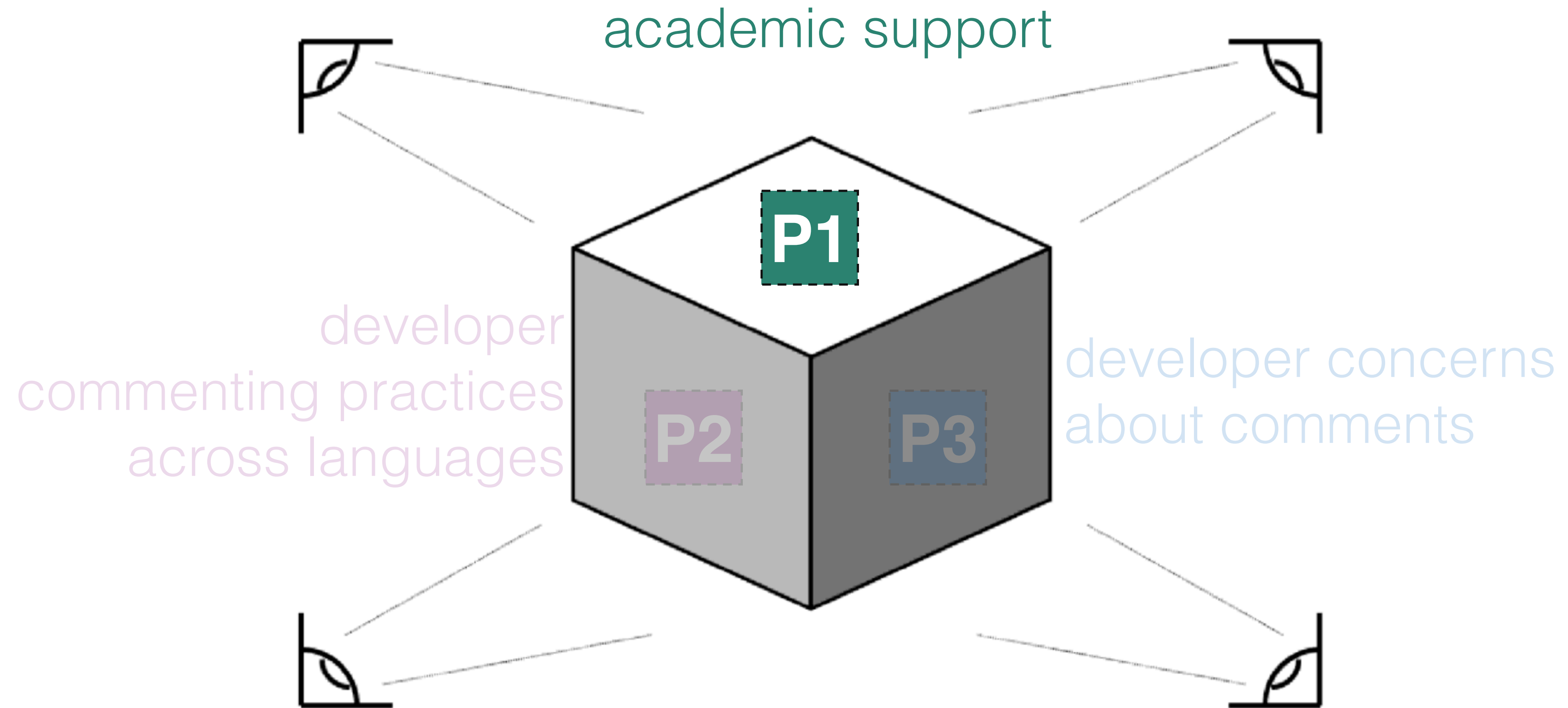


Gillies et al. 2011

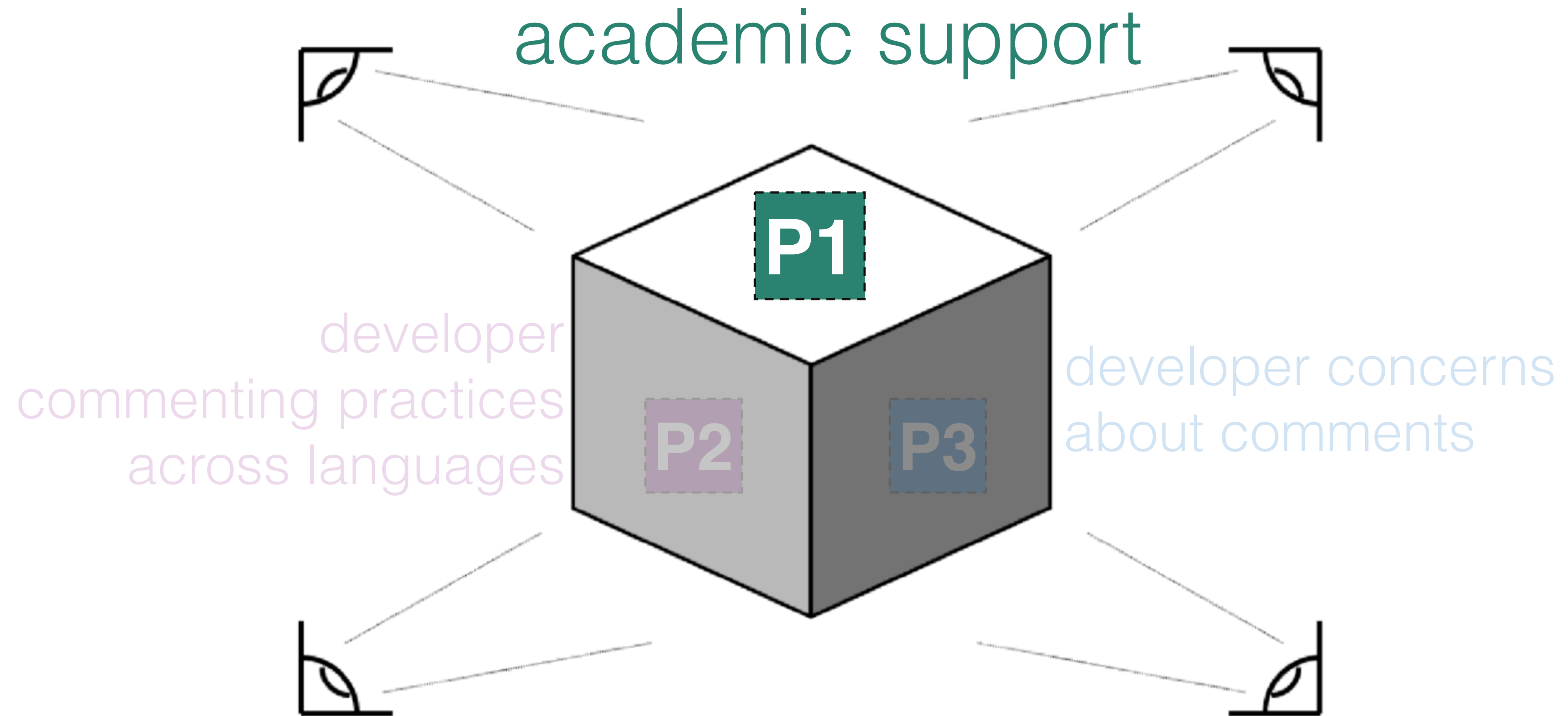
We define three perspectives



We define three perspectives



We define three perspectives



How do researchers measure comment quality?

Previous work on software documentation

All kinds of software documentation

Studies of **1971-2010**

The Journal of Systems and Software 99 (2015) 175–198

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Cost, benefits and quality of software development documentation: A systematic mapping

Junji Zhi^a, Vahid Garousi-Yusifoglu^{b,c,*}, Bo Sun^{d,e}, Golar Garousi^{c,f}, Shawn Shahnewaz^c, Guenther Ruhe^{c,d}

^a Department of Computer Science, University of Toronto, Ontario, Canada
^b System and Software Quality Engineering Research Group (SySoQual), Department of Software Engineering, Atilim University, Incek, Ankara, Turkey
^c Department of Electrical and Computer Engineering, University of Calgary, Alberta, Canada
^d Department of Computer Science, University of Calgary, Alberta, Canada
^e Solutions Inc., Calgary, Alberta, Canada
^f geoLOGIC Systems Ltd., Calgary, Alberta, Canada

ARTICLE INFO

Article history:
Received 5 September 2012
Received in revised form 28 September 2014
Accepted 28 September 2014
Available online 22 October 2014

Keywords:
Software documentation
Documentation benefit
Systematic mapping

ABSTRACT

Context: Software documentation is an integral part of any software development process. Researchers and practitioners have expressed concerns about costs, benefits and quality of software documentation in practice. On the one hand, there is a lack of a comprehensive model to evaluate the quality of documentation. On the other hand, researchers and practitioners need to assess whether documentation cost outweighs its benefit.

Objectives: In this study, we aim to summarize the existing literature and provide an overview of the field of software documentation cost, benefit and quality.

Method: We use the systematic-mapping methodology to map the existing body of knowledge related to software documentation cost, benefit and quality. To achieve our objectives, 11 Research Questions (RQ) are raised. The primary papers are carefully selected. After applying the inclusion and exclusion criteria, our study pool included a set of 69 papers from 1971 to 2011. A systematic map is developed and refined iteratively.

Results: We present the results of a systematic mapping covering different research aspects related to software documentation cost, benefit and quality (RQ 1–11). Key findings include: (1) validation research papers are dominating (27 papers), followed by solution proposals (21 papers). (2) Most papers (61 out of 69) do not mention the development life-cycle model explicitly. Agile development is only mentioned in 6 papers. (3) Most papers include only one “System under Study” (SUS) which is mostly academic prototype. The average number of participants in survey-based papers is 106, the highest one having approximately 1000 participants. (4) In terms of focus of papers, 50 papers focused on documentation quality, followed by 37 papers on benefit, and 12 papers on documentation cost. (5) The quality attributes of documentation that appear in most papers are, in order: completeness, consistency and accessibility. Additionally, improved meta-models for documentation cost, benefit and quality are also presented. Furthermore, we have created an online paper repository of the primary papers analyzed and mapped during this study.

Zhi et al. 2014

Only 10% of the studies focused on code comments

How do researchers measure comment quality?

Systematic literature review

10 years timeline **(2010-2020)**

195 software engineering venues

332 proceedings

48 relevant papers

Analyzed dimensions



Comment types

method comments, class comments

Analyzed dimensions



Comment types

method comments, class comments



Quality attributes

consistency, completeness

Analyzed dimensions



Comment types

method comments, class comments



Quality attributes

consistency, completeness



Techniques

heuristic-based, machine learning-based

Comment types

Code comments

Method comments

API documentation

License comments

Inline comments

TODO Comments

Software documentation

Deprecation Comments

Comment types

Code comments

Method comments

API documentation

License comments

Inline comments

TODO Comments

Software documentation

Deprecation Comments

52% of the studies

Comment types

- Code comments
- Method comments
- API documentation
- License comments
- Inline comments
- TODO Comments
- Software documentation
- Deprecation Comments

48% of the studies

Comment types

Code comments

Method comments

API documentation

License comments

Inline comments

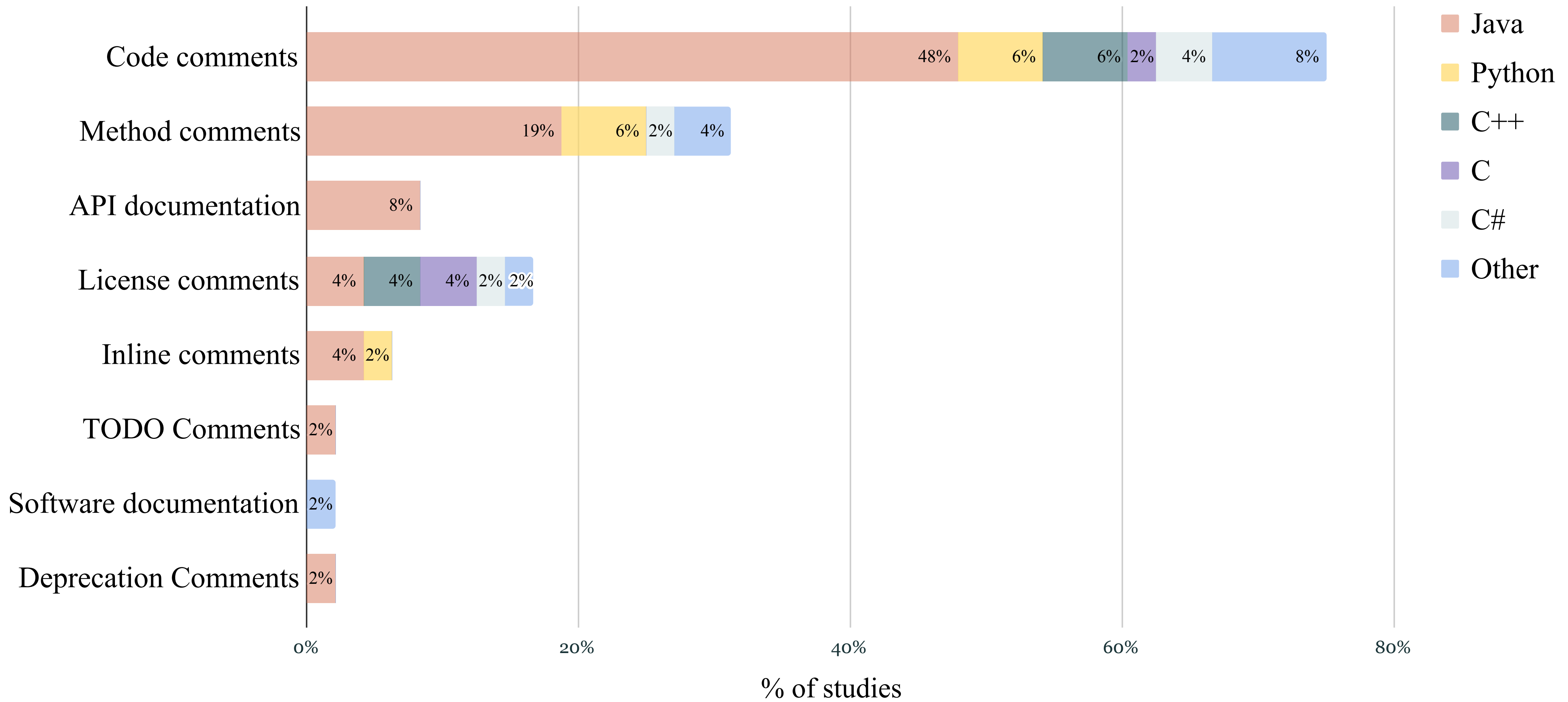
TODO Comments

Software documentation

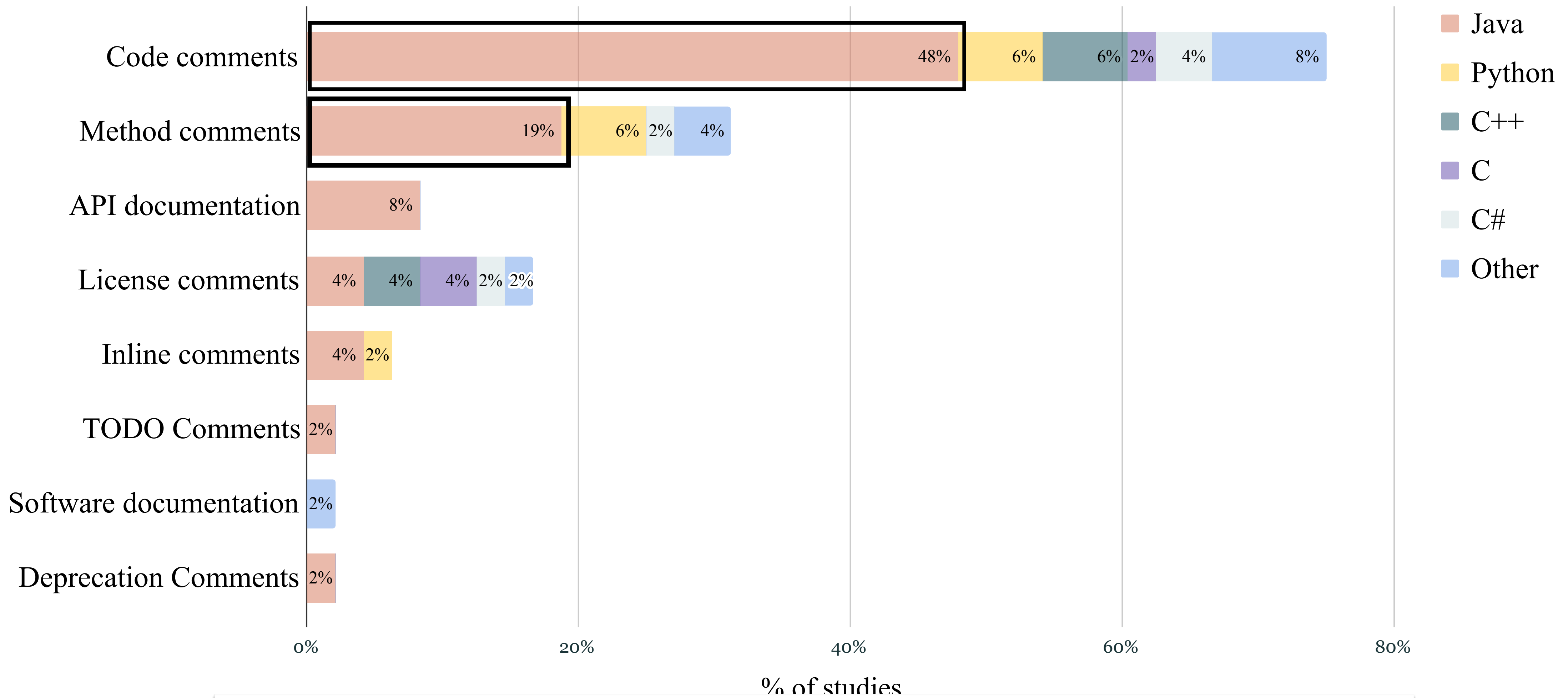
Deprecation Comments

The studies focus on specific types of comments but class comments

Comment types

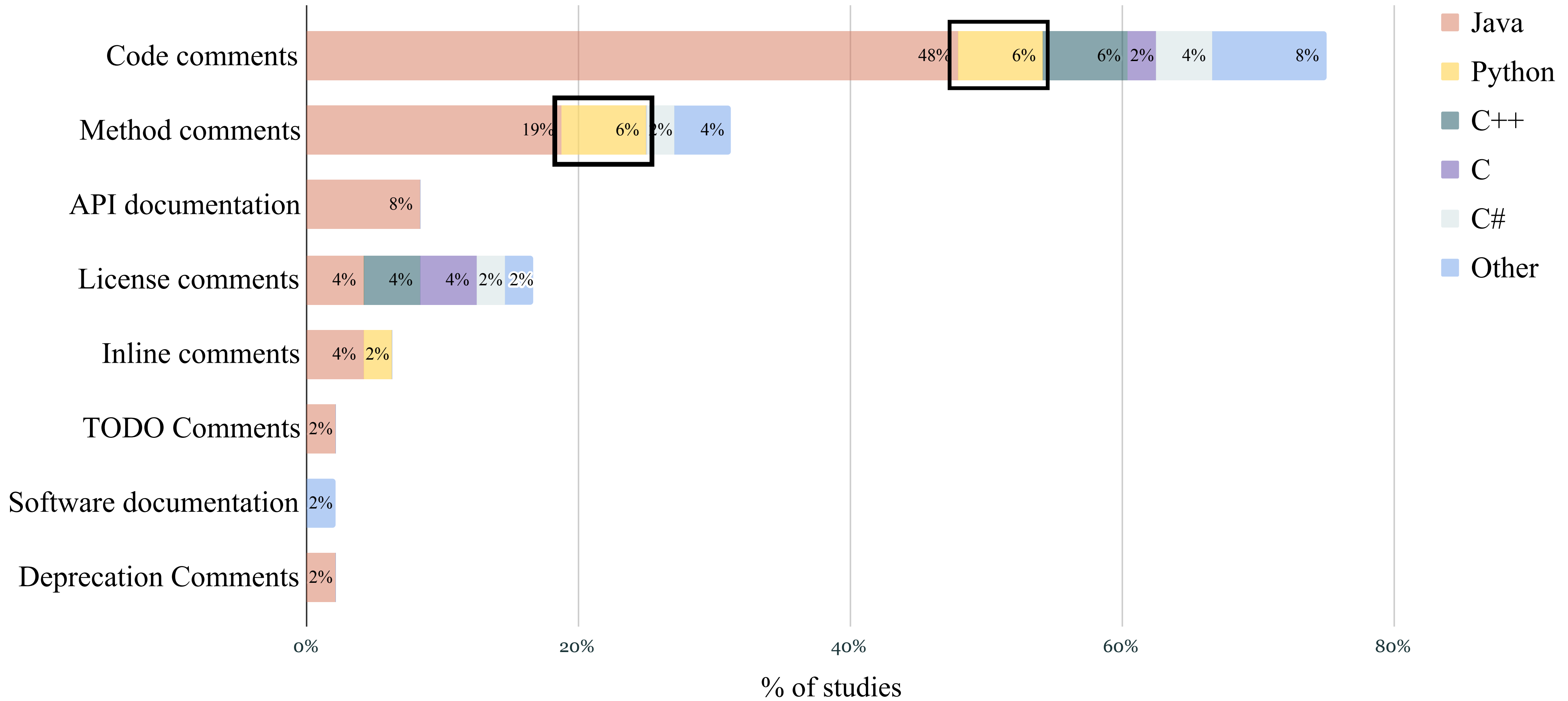


Comment types

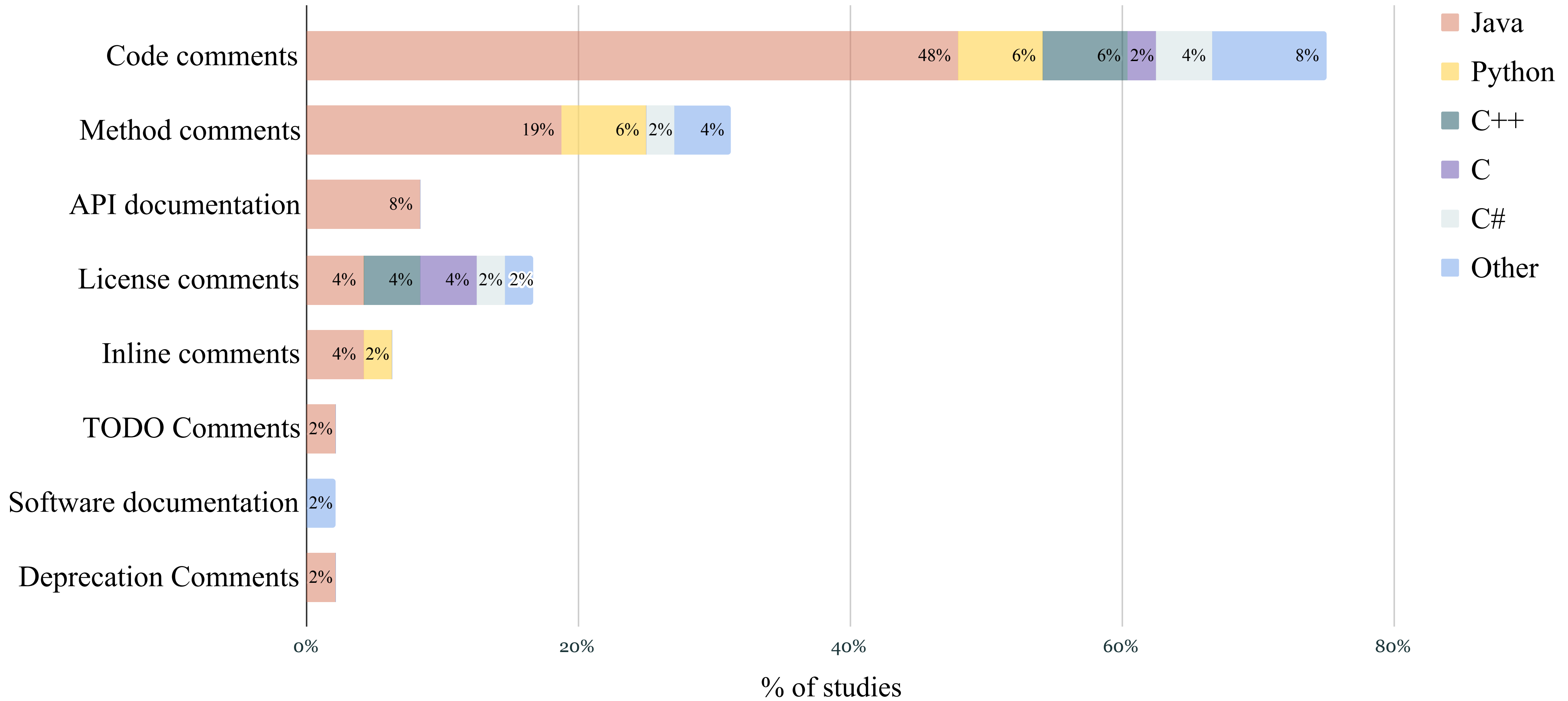


88% of the studies focus on **Java**

Comment types



Comment types





Quality attributes

21 quality attributes



Quality attributes

21 quality attributes

Quality attributes

Consistency

Completeness

Accuracy

Readability

Up-to-date-ness

Content relevance

Maintainability

Spelling and grammar



Quality attributes

Some quality attributes are frequently considered

Quality attributes

Consistency

Completeness

Accuracy

Readability

Up-to-date-ness

Content relevance

Maintainability

Spelling and grammar



Quality attributes

Some quality attributes are frequently considered

- Quality attributes
- Consistency
 - Completeness
 - Accuracy
 - Readability
 - Up-to-date-ness
 - Content relevance
 - Maintainability
 - Spelling and grammar

- Conciseness
- Usability
- Correctness
- Traceability
- Accessibility
- Coherence
- Format
- Information organization
- Understandability
- Documentation technology
- Internationalization
- Author-related



Quality attributes

Quality attributes

Consistency

Completeness

Accuracy

Readability

Up-to-date-ness

Content relevance

Maintainability

Spelling and grammar

Conciseness

Usability

Correctness

Traceability

Accessibility

Coherence

Format

Information organization

Understandability

Documentation technology

Internationalization

Author-related

Some quality attributes are rarely considered



Techniques

Quality attributes

Consistency

Completeness

Accuracy

Readability

Up-to-date-ness

Content relevance

Maintainability

Spelling and grammar

Conciseness

Usability

Correctness

Traceability

Accessibility

Coherence

Format

Information organization

Understandability

Documentation technology

Internationalization

Author-related



Techniques

Quality attributes

- Consistency
- Completeness
- Accuracy
- Readability
- Up-to-date-ness
- Content relevance
- Maintainability
- Spelling and grammar
- Conciseness
- Usability
- Correctness
- Traceability
- Accessibility
- Coherence
- Format
- Information organization
- Understandability
- Documentation technology
- Internationalization
- Author-related

Techniques

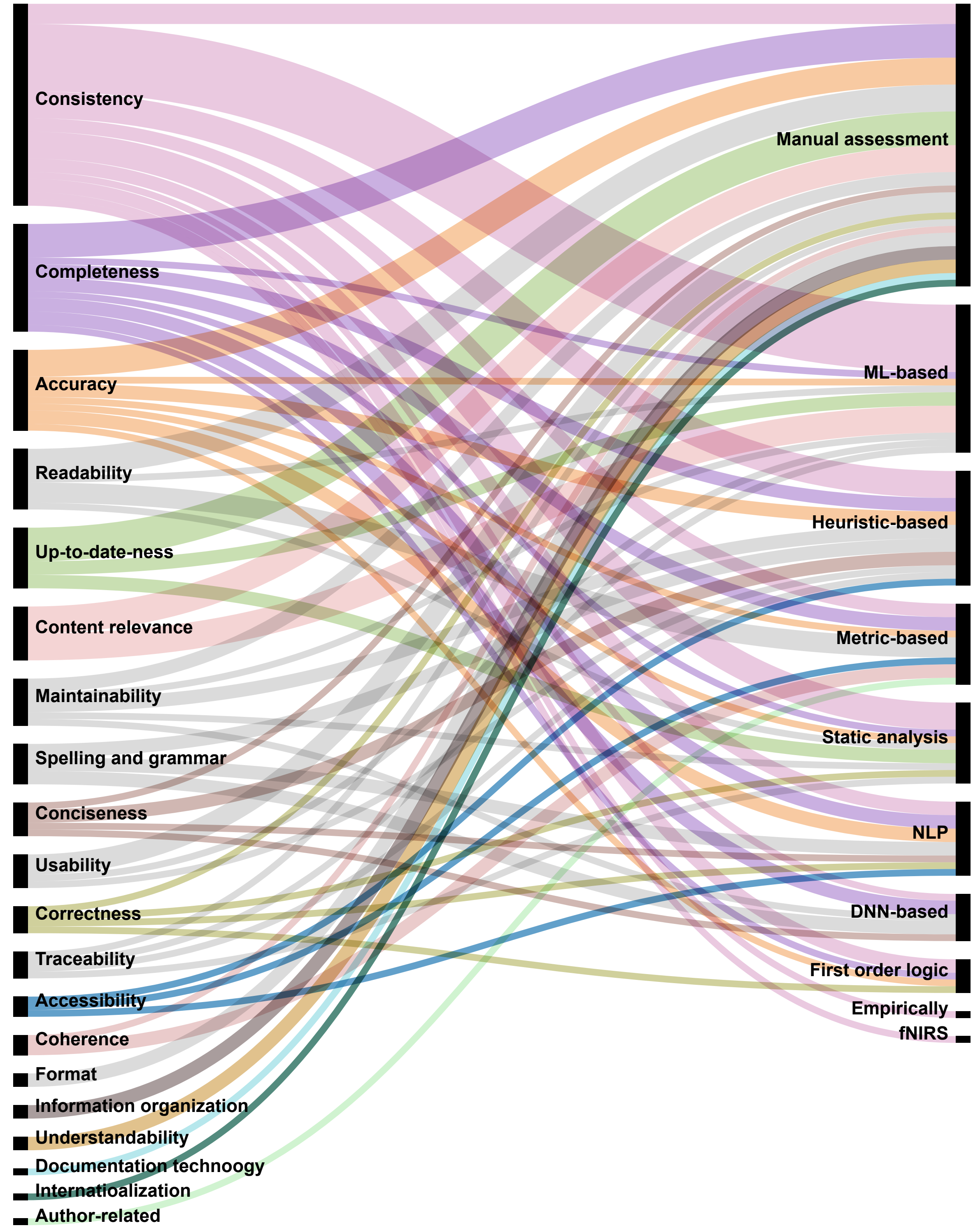
- Manual assessment
- ML-based
- Heuristic-based
- Metric-based
- Static analysis
- NLP
- DNN-based
- First order logic
- Empirically
- fNIRS



Techniques

Quality attributes

Techniques

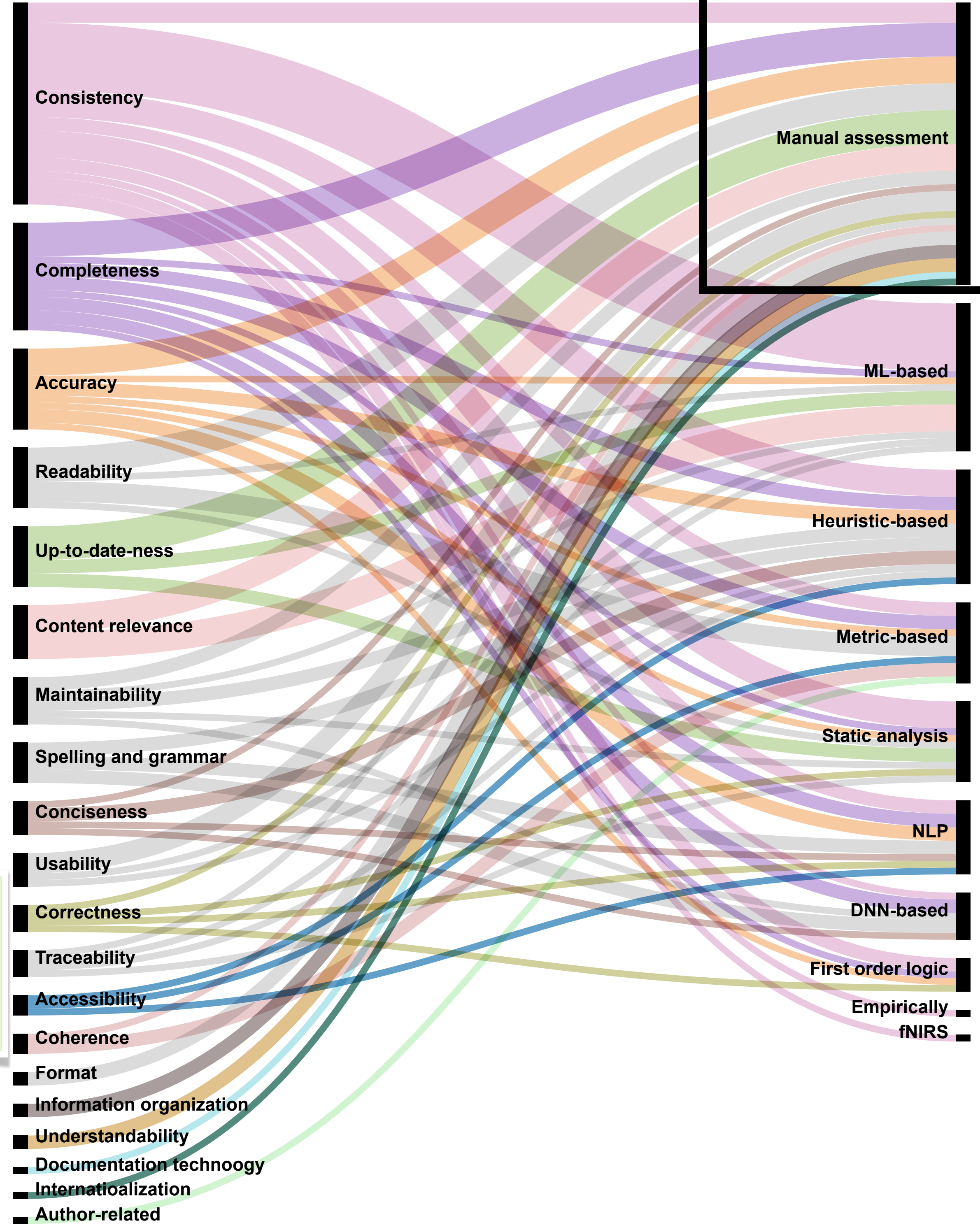




Techniques

Quality attributes

Techniques



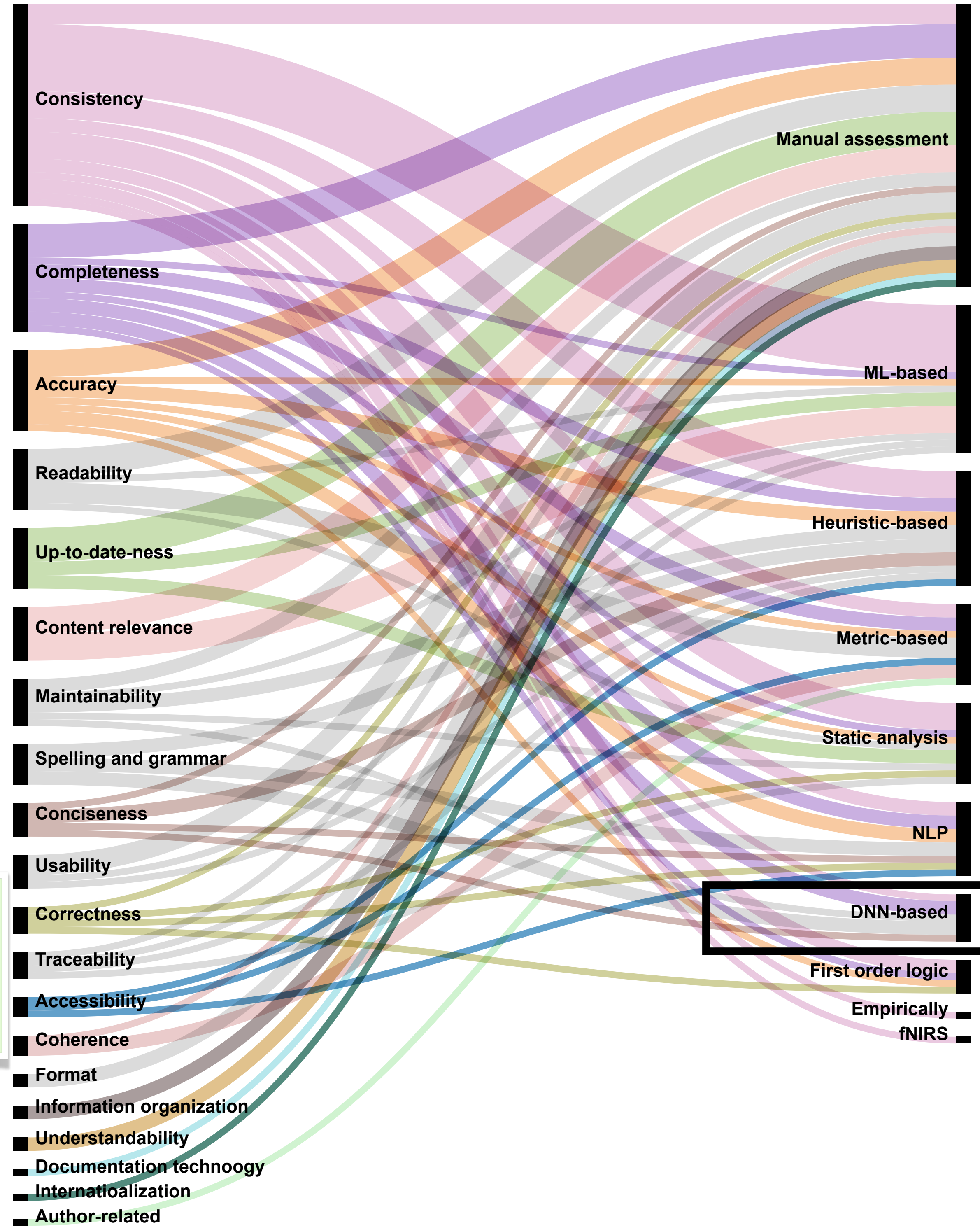
Manual assessment is still the most frequent technique to measure quality attributes



Techniques

Quality attributes

Techniques



Deep learning-based techniques have not been extensively explored for comment analysis

Take-home messages

- ☑ Class comments provide an overview of a program, but are not analyzed enough.
- ☑ Studies focus mainly on Java.
- ☑ Manual assessment is still the most frequent technique to measure various quality attributes.

Take-home messages

- ☑ Class comments provide an overview of a program, but are not analyzed enough.
- ☑ Studies focus mainly on Java.
- ☑ Manual assessment is still the most frequent technique to measure various quality attributes.

Take-home messages

- ☑ Class comments provide an overview of a program, but are not analyzed enough.
- ☑ Studies focus mainly on Java.
- ☑ Manual assessment is still the most frequent technique to measure various quality attributes.

A Decade of Code Comment Quality Assessment: A Systematic Literature Review

Pooja Rani^a, Arianna Blasi^b, Nataliia Stulova^a, Sebastiano Panichella^c, Alessandra Gorla^d, Oscar Nierstrasz^a

^aSoftware Composition Group, University of Bern, Bern, Switzerland

^bUniversità della Svizzera italiana, Lugano, Switzerland

^cZurich University of Applied Sciences, Zurich, Switzerland

^dIMDEA Software Institute, Madrid, Spain

Abstract

Code comments are important artifacts in software systems and play a paramount role in many software engineering (SE) tasks related to maintenance and program comprehension. However, while it is widely accepted that high quality matters in code comments just as it matters in source code, *assessing* comment quality in practice is still an open problem. First and foremost, there is no unique definition of quality when it comes to evaluating code comments. The few existing studies on this topic rather focus on specific attributes of quality that can be easily quantified and measured. Existing techniques and corresponding tools may also focus on comments bound to a specific programming language, and may only deal with comments with specific scopes and clear goals (e.g., Javadoc comments at the method level, or in-body comments describing TODOs to address).

In this paper, we present a Systematic Literature Review (SLR) of the last decade of research in SE to answer the following research questions: (i) What *types of comments* do researchers focus on when assessing comment quality? (ii) What *quality attributes* (QAs) do they consider? (iii) Which *tools and techniques* do they use to assess comment quality?, and (iv) How do they *evaluate* their studies on comment quality assessment in general?

Our evaluation, based on the analysis of 2353 papers and actual review of 48 relevant ones, shows that (i) most studies and techniques focus on comments in Java code, and thus may not be generalizable to other languages; and (ii) the analyzed studies focus on four main QAs of a total of 21 QAs identified in the literature, with a clear predominance of checking *consistency* between comments and the code. We observe that researchers rely on manual assessment and specific heuristics rather than the automated assessment of the comment quality attributes, with evaluations often involving surveys of students and the authors of the original studies but rarely professional developers.

Keywords: code comments, documentation quality, systematic literature review

1. Introduction

Software systems are often written in several programming languages [1], and interact with many hardware devices and software components [2, 3]. To deal with such complexity and to ease maintenance tasks, developers tend to document their

Email addresses: pooja.rani@inf.unibe.ch (Pooja Rani), arianna.blasi@usi.ch (Arianna Blasi), nataliia.stulova@inf.unibe.ch (Nataliia Stulova), panc@zhaw.ch (Sebastiano Panichella), alessandra.gorla@imdea.org (Alessandra Gorla), oscar.nierstrasz@inf.unibe.ch (Oscar Nierstrasz)

Preprint submitted to Journal of Systems and Software

software with various artifacts, such as design documents and code comments [4]. Several studies have demonstrated that *high quality* code comments can support developers in software comprehension, bug detection, and program maintenance activities [5, 6, 7]. However, code comments are typically written using natural language sentences, and their syntax is neither imposed by a programming language's grammar nor checked by its compiler. Additionally, static analysis tools and linters provide limited syntactic support to check comment quality. There-

October 8, 2021

P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, and O. Nierstrasz. **A Decade of comment quality assessment: A systematic literature review**, *Journal of Systems & Software*, 2021



^b
UNIVERSITÄT
BERN



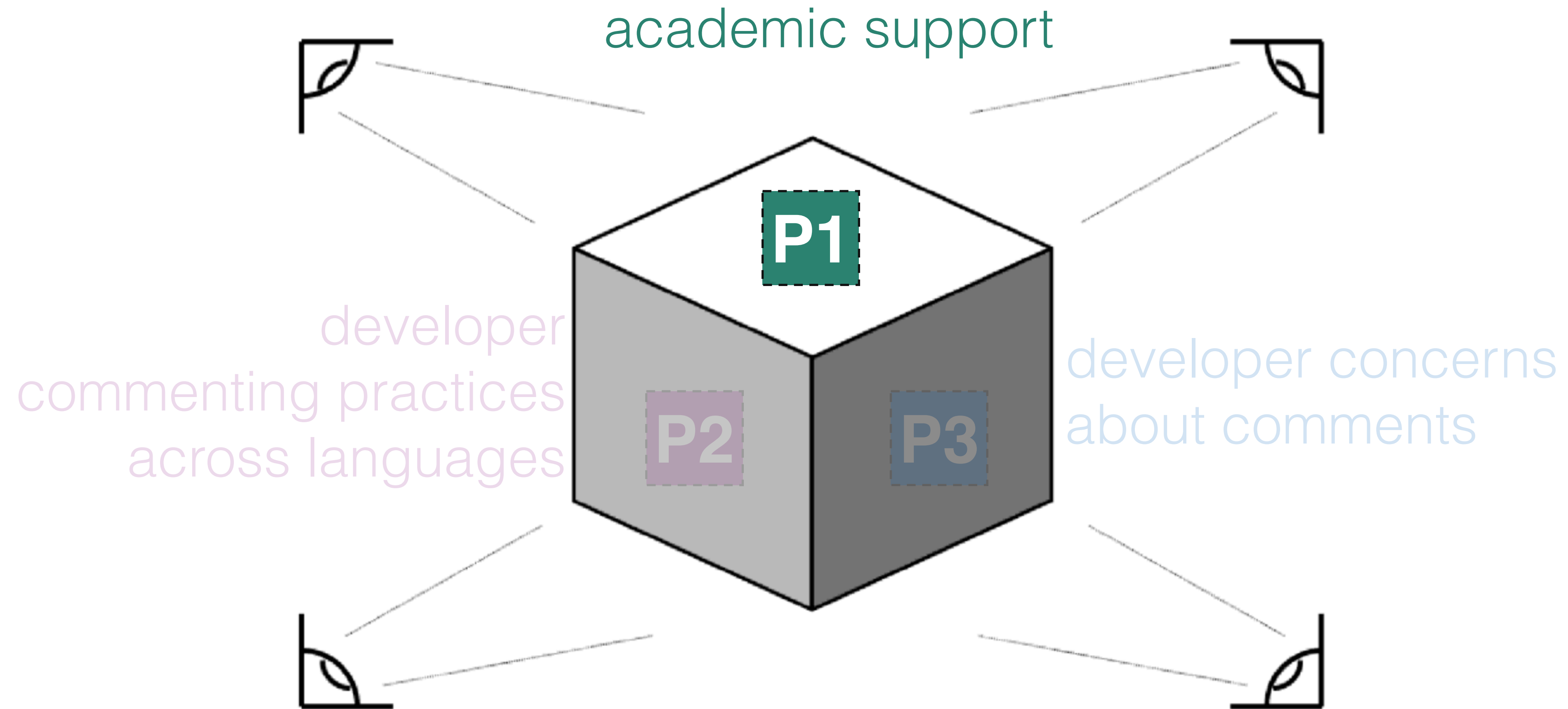
Zurich University
of Applied Sciences



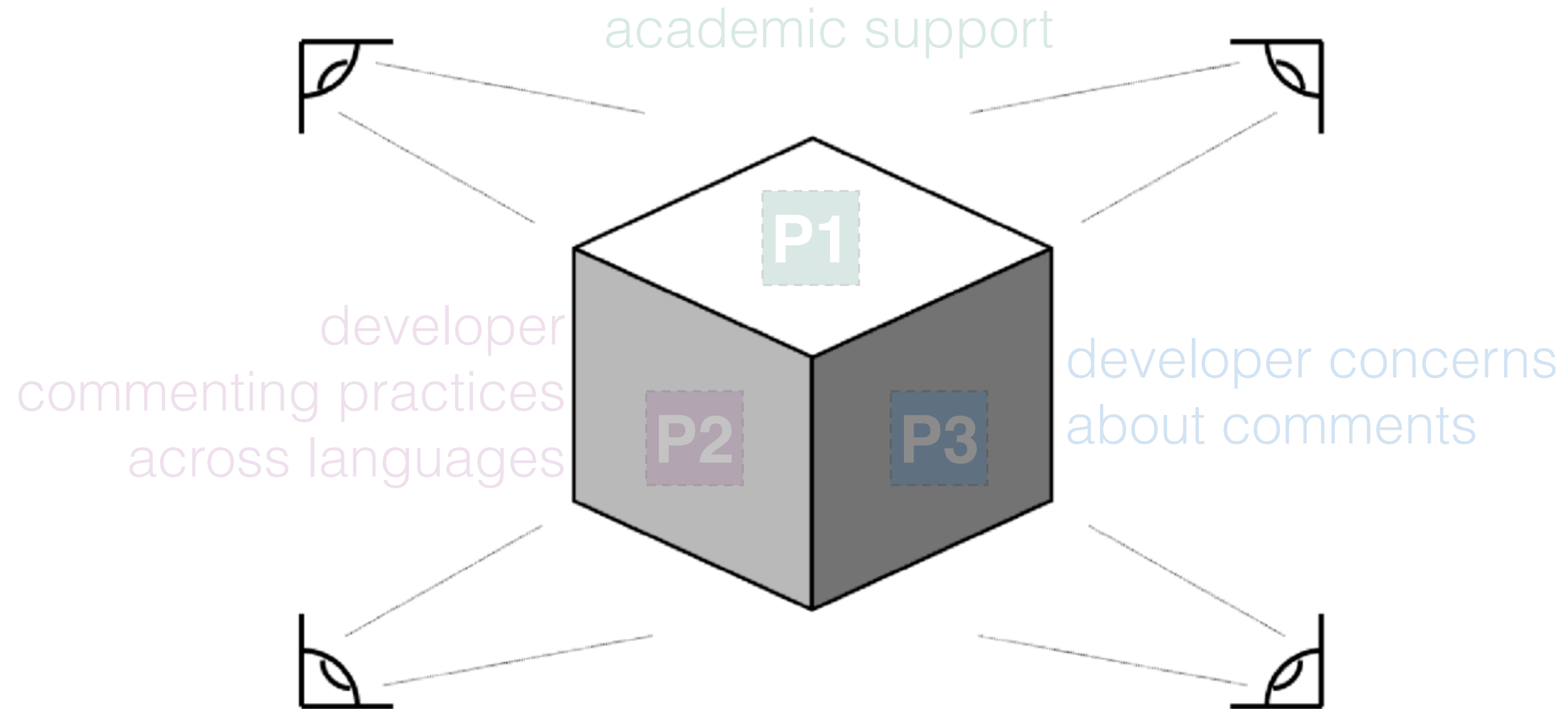
^b
UNIVERSITÄT
BERN



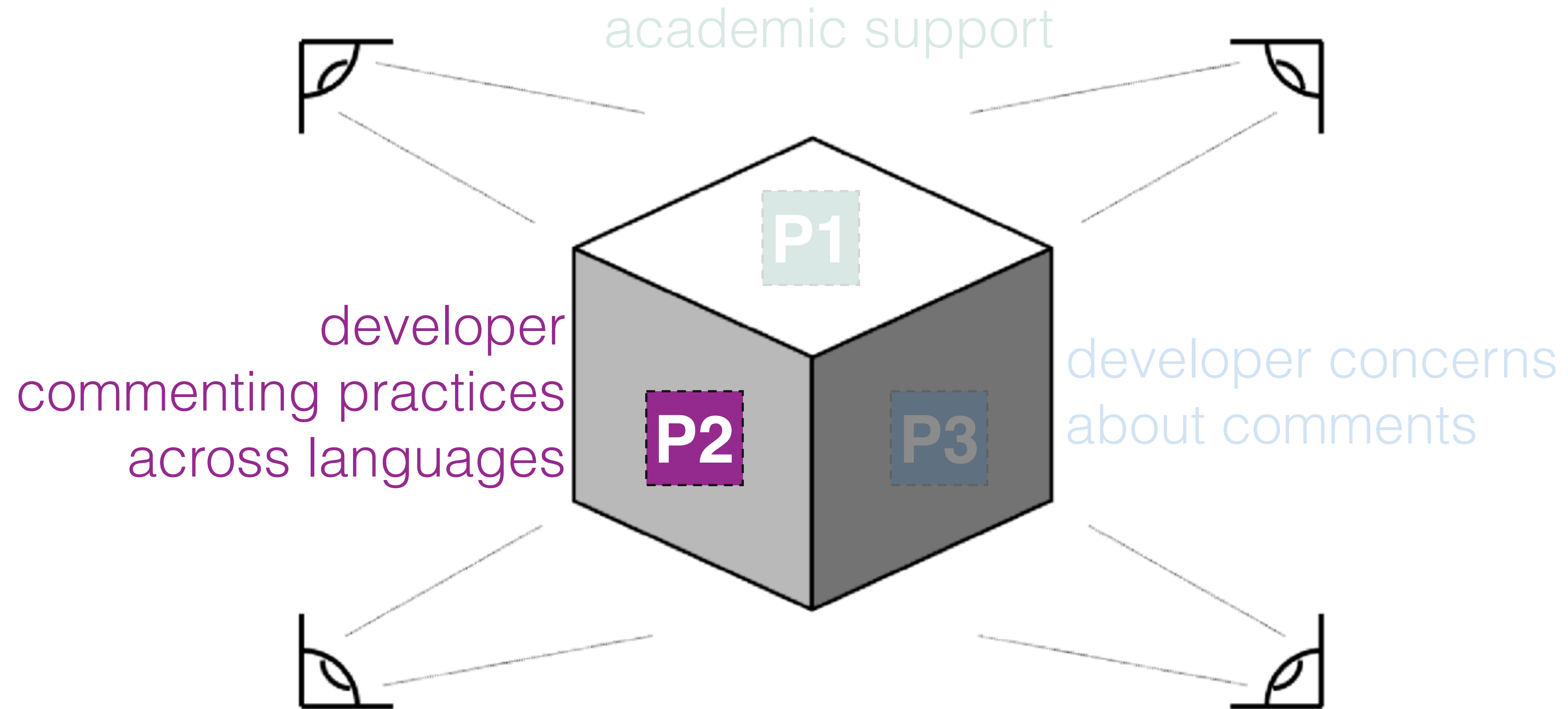
We define three perspectives



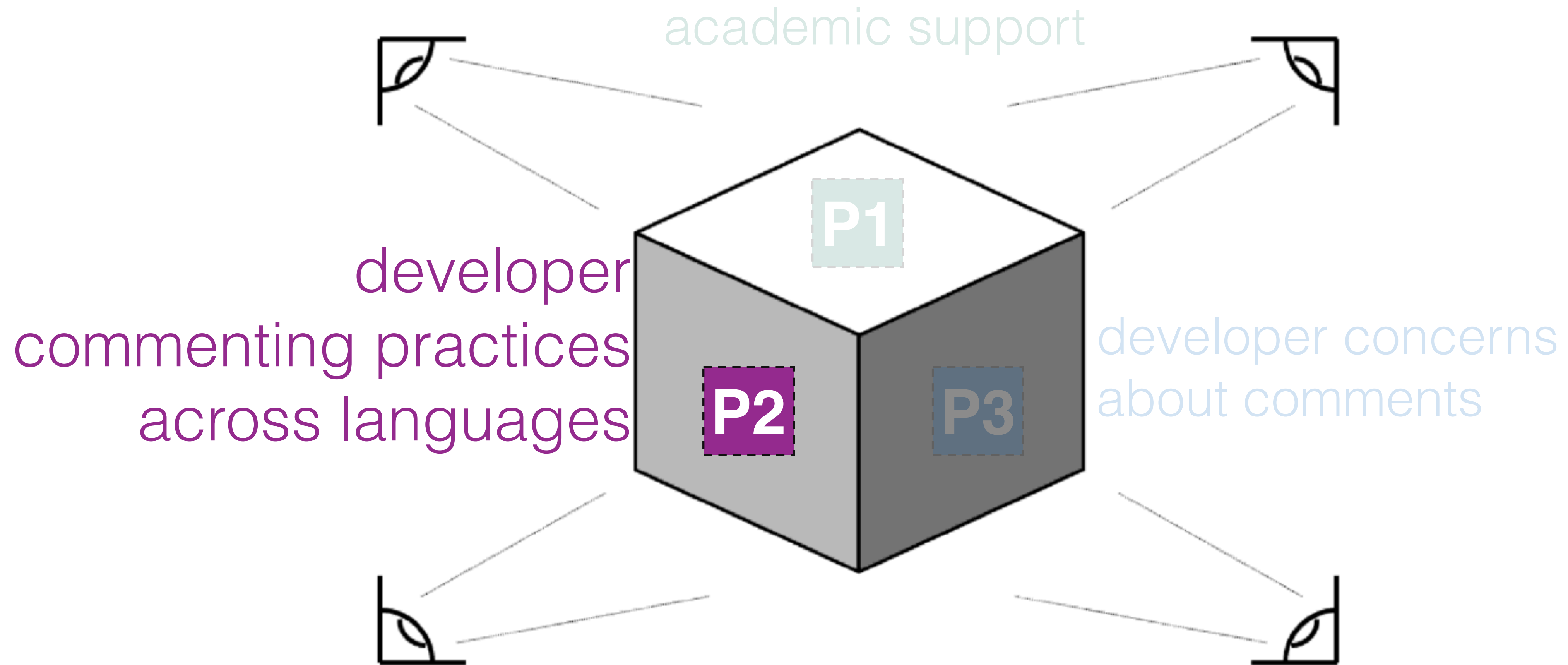
We define three perspectives



We define three perspectives



P2: class commenting practices



What information developers write in class comments across languages?

Do they follow the coding style guidelines?

What information developers write in class comments across languages?

Do they follow the coding style guidelines?

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
 * @see java.awt.Button
 */
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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Usage

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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Usage

Pointer

Comment taxonomies in Java and Python



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

Pascarella et al., 2017



Classifying Python Code Comments Based on Supervised Learning

Jingyi Zhang¹, Lei Xu^{2(✉)}, and Yanhui Li²

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, yanhui11}@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.

Zhang et al., 2018

Comment taxonomies in Java and Python



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

Pascarella et al., 2017



Classifying Python Code Comments Based on Supervised Learning

Jingyi Zhang¹, Lei Xu^{2(✉)}, and Yanhui Li²

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

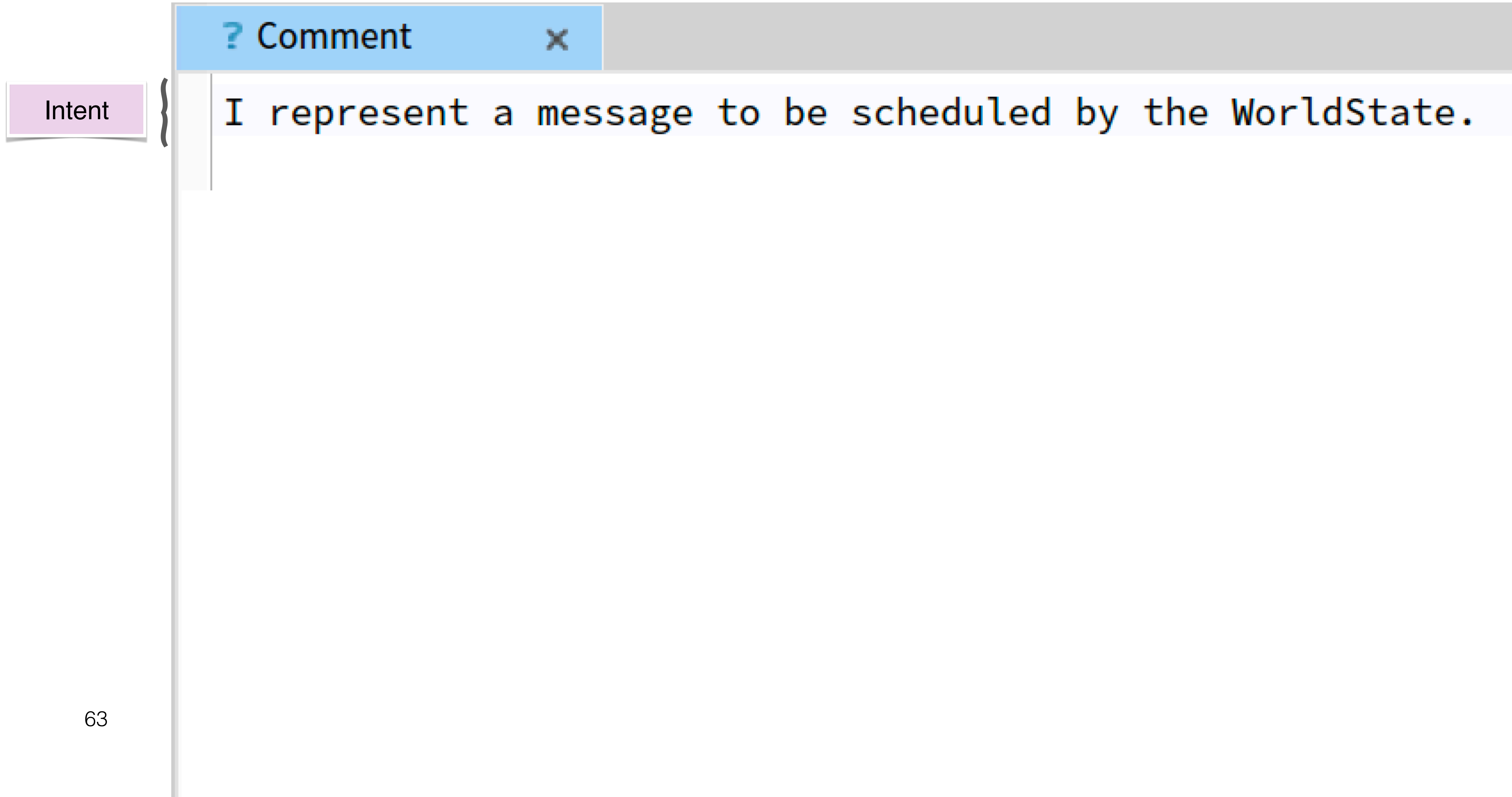
² Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, China
{xlei, yanhui11}@nju.edu.cn

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.

Zhang et al., 2018

The taxonomies do not focus specifically on class comments

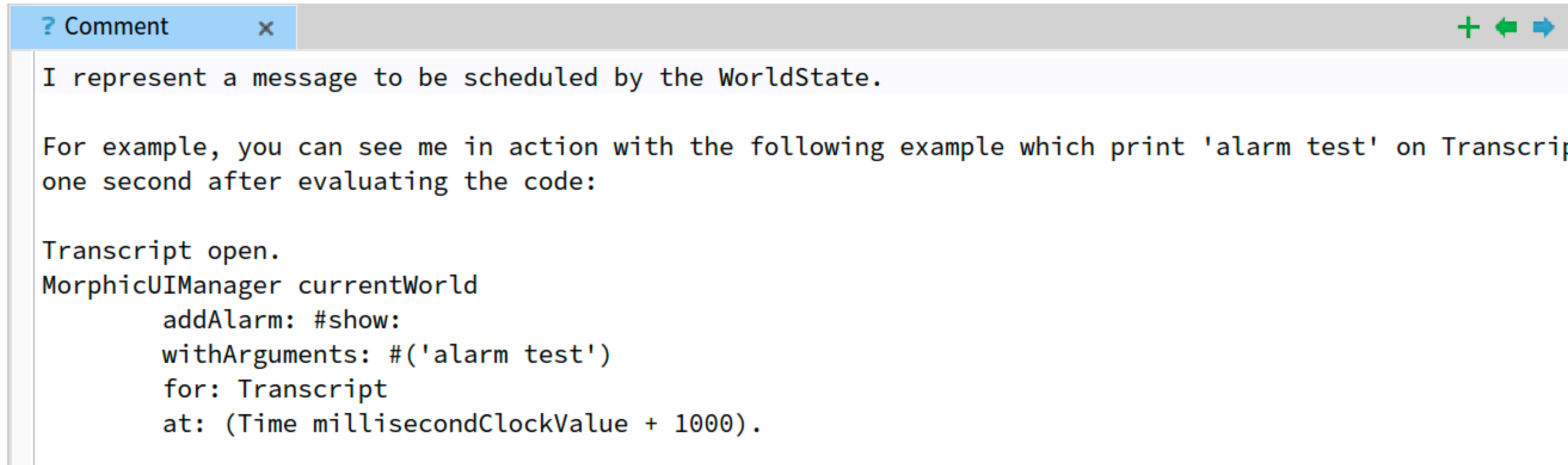
Smalltalk class comments



The image shows a screenshot of a Smalltalk IDE. On the left, a class browser pane displays the class 'Intent'. A comment window is open, showing the comment text: 'I represent a message to be scheduled by the WorldState.'

```
? Comment x  
Intent {  
I represent a message to be scheduled by the WorldState.
```

Smalltalk class comments

A screenshot of a Smalltalk comment window. The window has a title bar with a question mark icon, the text "Comment", a close button (x), and navigation icons (plus, left arrow, right arrow). The comment text is as follows:

```
I represent a message to be scheduled by the WorldState.  
  
For example, you can see me in action with the following example which print 'alarm test' on Transcript  
one second after evaluating the code:  
  
Transcript open.  
MorphicUIManager currentWorld  
    addAlarm: #show:  
    withArguments: #('alarm test')  
    for: Transcript  
    at: (Time millisecondClockValue + 1000).
```

Example

Smalltalk class comments

```
? Comment x + ← → ▾
I represent a message to be scheduled by the WorldState.

For example, you can see me in action with the following example which print 'alarm test' on Transcript
one second after evaluating the code:

Transcript open.
MorphicUIManager currentWorld
  addAlarm: #show:
  withArguments: #('alarm test')
  for: Transcript
  at: (Time millisecondClockValue + 1000).

* Note *
Compared to doing:
[(Delay forMilliseconds: 1000) wait. Transcript show: 'alarm test'] forkAt: Processor activeProcess
priority +1.

the alarm system has several distinctions:
- Runs with the step refresh rate resolution.
- Alarms only run for the active world. (Unless a non-standard scheduler is in use)
- Alarms with the same scheduled time are guaranteed to be executed in the order they were added
```

Implementation details

Smalltalk class comments

Intent

I represent a message to be scheduled by the WorldState.

Example

For example, you can see me in action with the following example which print 'alarm test' on Transcript one second after evaluating the code:

```
Transcript open.  
MorphicUIManager currentWorld  
  addAlarm: #show:  
  withArguments: #('alarm test')  
  for: Transcript  
  at: (Time millisecondClockValue + 1000).
```

* Note *

Compared to doing:

```
[(Delay forMilliseconds: 1000) wait. Transcript show: 'alarm test'] forkAt: Processor activeProcess  
priority +1.
```

Implementation
details

the alarm system has several distinctions:

- Runs with the step refresh rate resolution.
- Alarms only run for the active world. (Unless a non-standard scheduler is in use)
- Alarms with the same scheduled time are guaranteed to be executed in the order they were added

Unlike Java and Python, there is **no comment taxonomy**

Information types in Smalltalk

Pharo 7 core

Smalltalk

6,324 class comments

363 sample comments

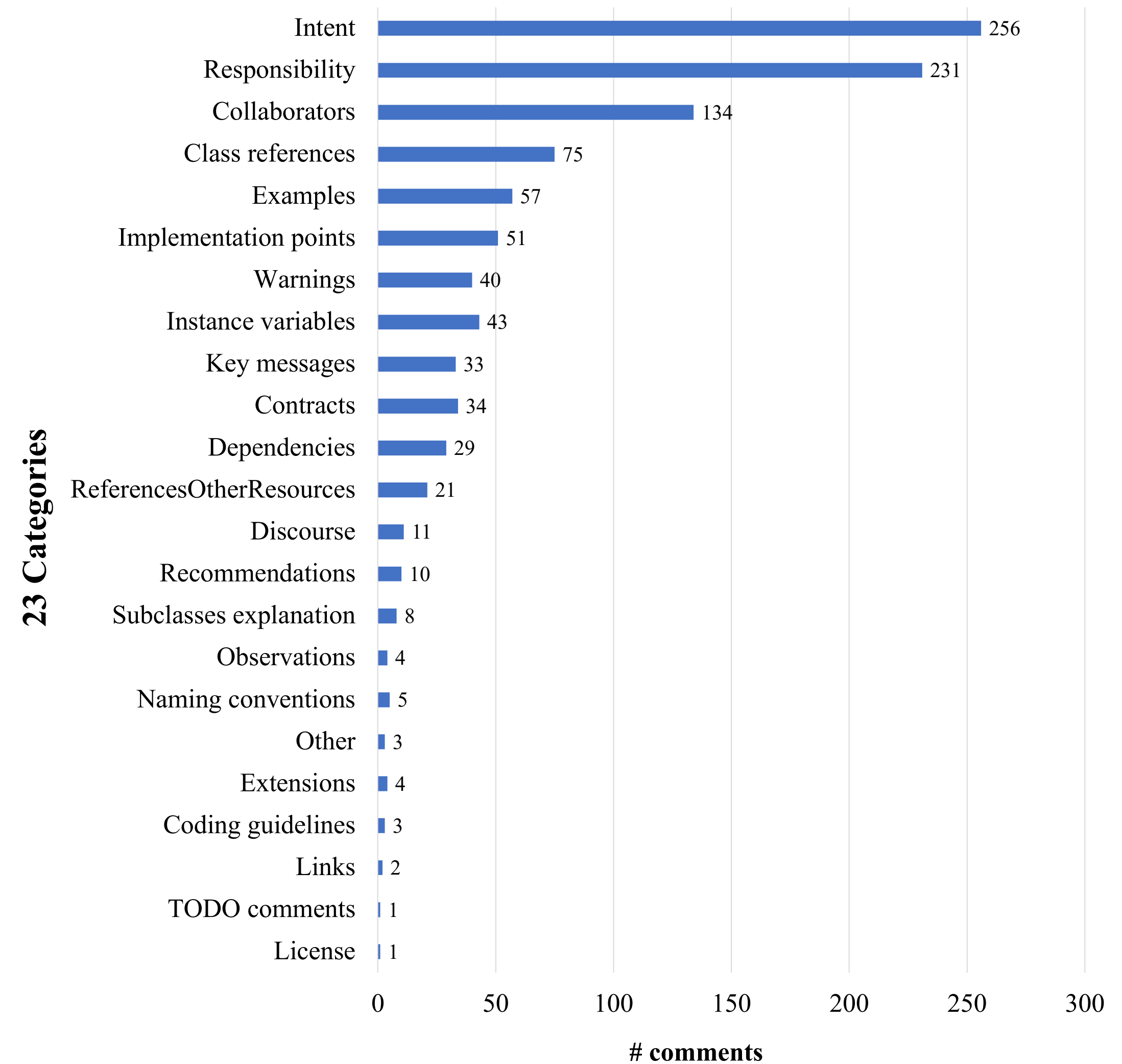
Information types in Smalltalk

Pharo 7 core

Smalltalk

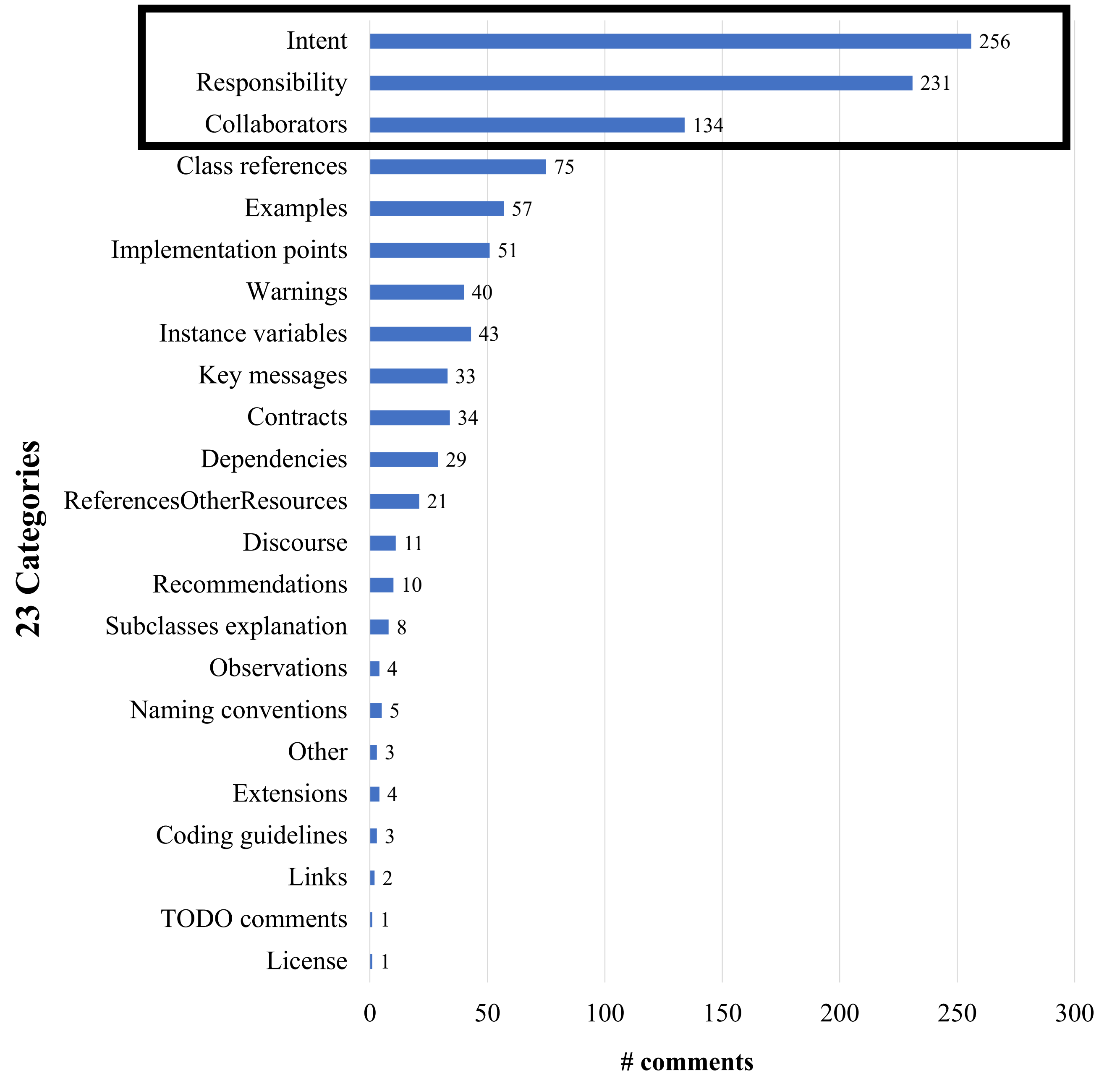
6,324 class comments

363 sample comments



Information types in Smalltalk

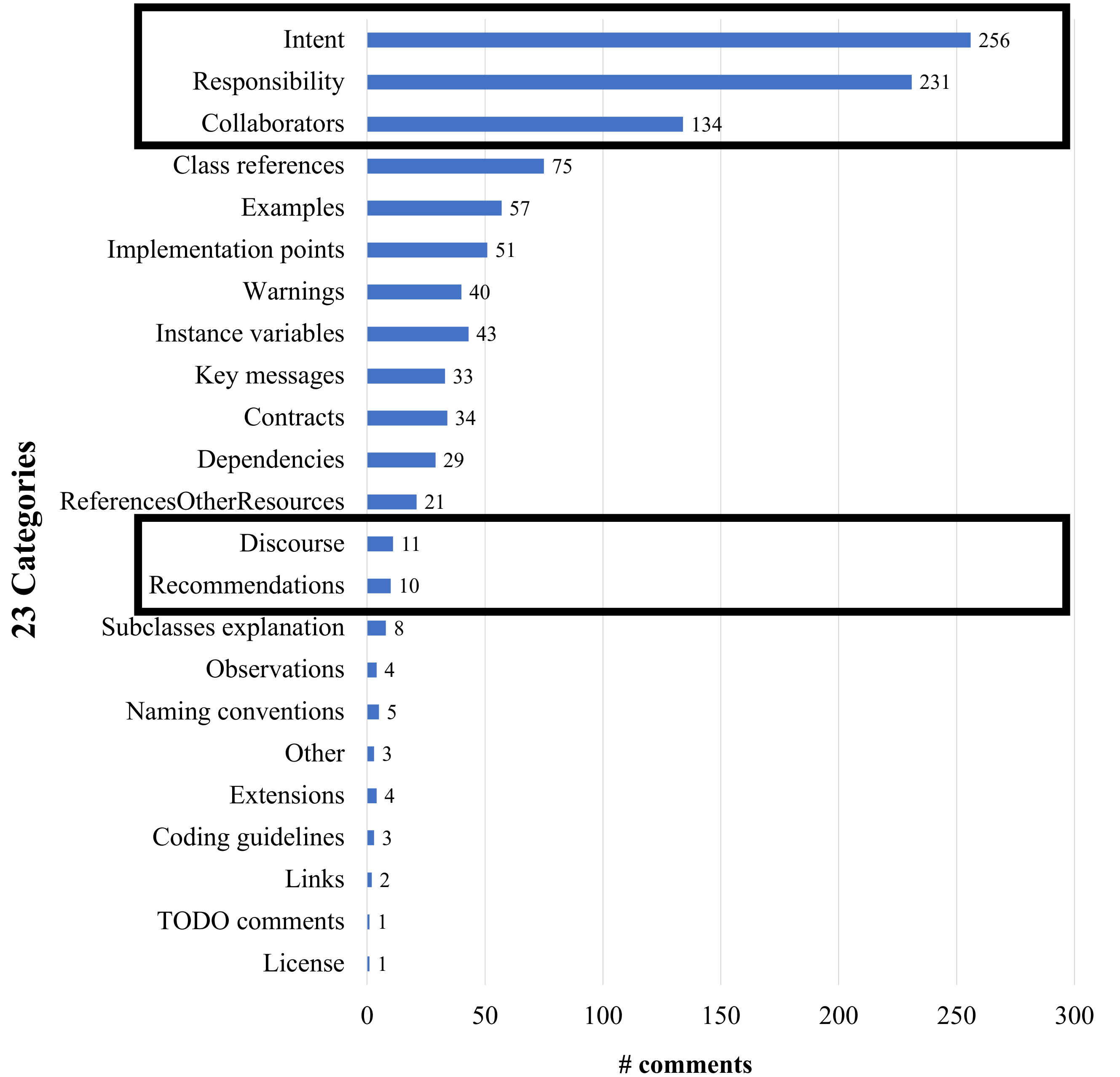
High-level overview



Information types in Smalltalk

High-level overview

Conversation exchanged



*We mapped first the existing taxonomies of **Java**, **Python**, and **Smalltalk** and then adapted them to **class** comments.*

Information types across languages

20 open-source projects

Java, Python, Smalltalk

37,446 class comments

1,066 sample comments

Information types across languages

Java

- Summary
- Expand
- Pointer
- Rationale
- Usage
- Deprecation
-
-
-

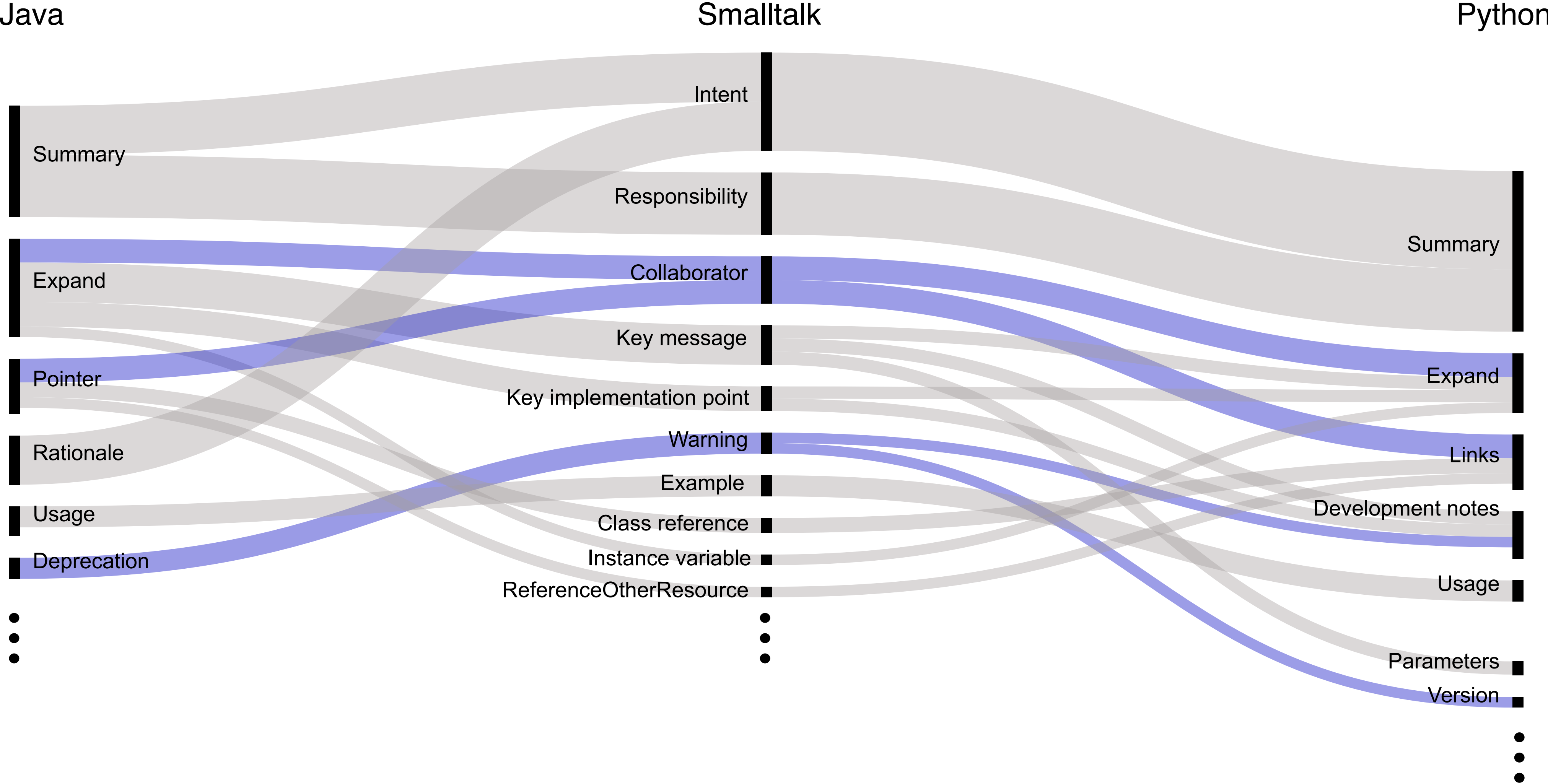
Smalltalk

- Intent ■
- Responsibility ■
- Collaborator ■
- Key message ■
- Key implementation point ■
- Examples ■
- Class reference ■
-
-
-

Python

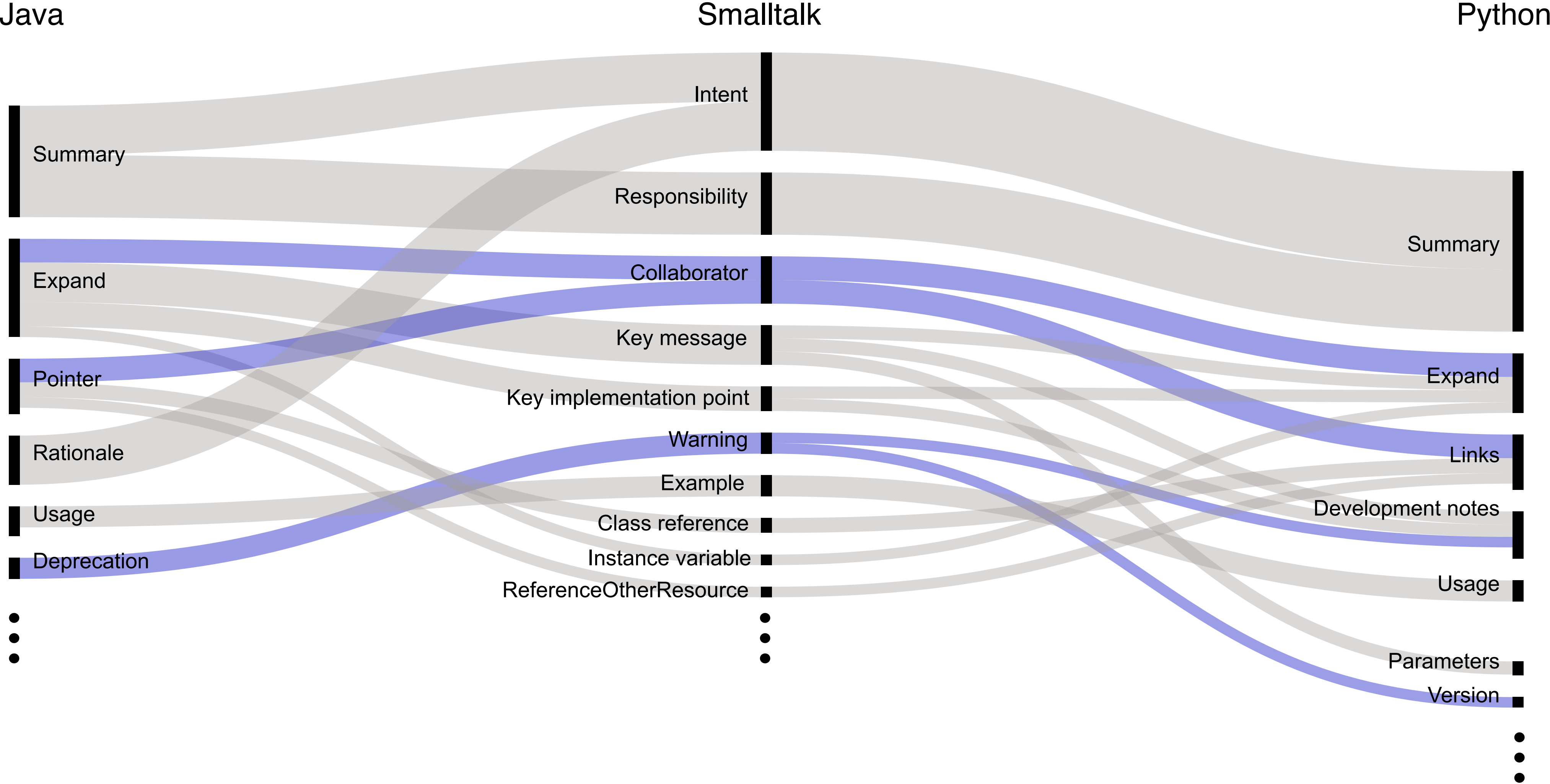
- Summary ■
- Expand ■
- Links ■
- Development notes ■
- Usage ■
-
-
-

Information types across languages



Developers write similar kinds of information in class comments across languages

Information types across languages



There are also other language-specific information types

Automatic identification of 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Automatic identification of 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Recurrent natural language patterns exist in various information types

Automatic identification of 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Represents [something]

[verb]s [noun]

Recurrent natural language patterns exist in various information types

Automatic identification of 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Represents [something]

[verb]s [noun]

Pointer

Recurrent natural language patterns exist in various information types

Automatic identification of 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Represents [something]

[verb]s [noun]

Pointer

Sees [something]

Recurrent natural language patterns exist in various information types

Automatic identification of 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Summary

Represents [something]

[verb]s [noun]

Pointer

Sees [something]

How to extract such patterns?

Intention mining in informal software documents

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

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

Development Emails Content Analyzer: Intention Mining in Developer Discussions

Andrea Di Sorbo*, Sebastiano Panichella[†], Corrado A. Visaggio*,
Massimiliano Di Penta*, Gerardo Canfora* and Harald C. Gall[†]
*University of Sannio, Benevento, Italy
[†]University of Zurich, Switzerland
disorbo@unisannio.it, panichella@ifi.uzh.ch, {visaggio,dipenta,canfora}@unisannio.it, gall@ifi.uzh.ch

Abstract—Written development communication (e.g. mailing lists, issue trackers) constitutes a precious source of information to build recommenders for software engineers, for example aimed at suggesting experts, or at redocumenting existing source code. In this paper we propose a novel, semi-supervised approach named DECA (Development Emails Content Analyzer) that uses Natural Language Parsing to classify the content of development emails according to their purpose (e.g. feature request, opinion asking, problem discovery, solution proposal, information giving etc), identifying email elements that can be used for specific tasks. A study based on data from Qt and Ubuntu, highlights a high precision (90%) and recall (70%) of DECA in classifying email content, outperforming traditional machine learning strategies. Moreover, we successfully used DECA for re-documenting source code of Eclipse and Lucene, improving the recall, while keeping high precision, of a previous approach based on ad-hoc heuristics.

Keywords—Unstructured Data Mining, Natural Language Processing, Empirical Study

I. INTRODUCTION

In many open sources and industrial projects, developers make an intense usage of written communication channels, such as mailing lists, issue trackers and chats [44]. Although voice communication still remains something unavoidable [1], [37], such channels ease the communication of developers

For example, an issue report may relate to a feature request, a bug, or just to a project management discussion. For example, Herzig *et al.* [30] and Antoniol *et al.* [2] found that over 30% of all issue reports are misclassified (i.e., rather than referring to a code fix, they resulted in a new feature, an update of documentation, or an internal refactoring). Hence, relying on such data to build fault prediction or localization approaches might result in incorrect results. Kochhar *et al.* [35] shed light on the need for additional cleaning steps to be performed on issue reports for improving bug localization tasks. This, for example, may involve a re-classification of issue reports.

On a different side, certain recommender may require to mine specific portions of a written communication, for example to identify questions being asked by developers [29] or to mine descriptions about certain methods [5], [45]. Also, sometimes an email or a discussion is too long and this does not help a developer who get lost in unnecessary details. To cope with this issue, previous literature proposed approaches aimed at generating summaries of emails [36], [46], [48] and bug reports [47]. However, none of the aforementioned approaches is able to classify paragraphs contained in developers' communication according to the developers' intent, in order to only focus on paragraphs useful for a specific purposes (e.g. fixing bugs, add new features, improve existing features etc.).

Di Sorbo et al., 2015

Intention mining in informal software documents

- 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

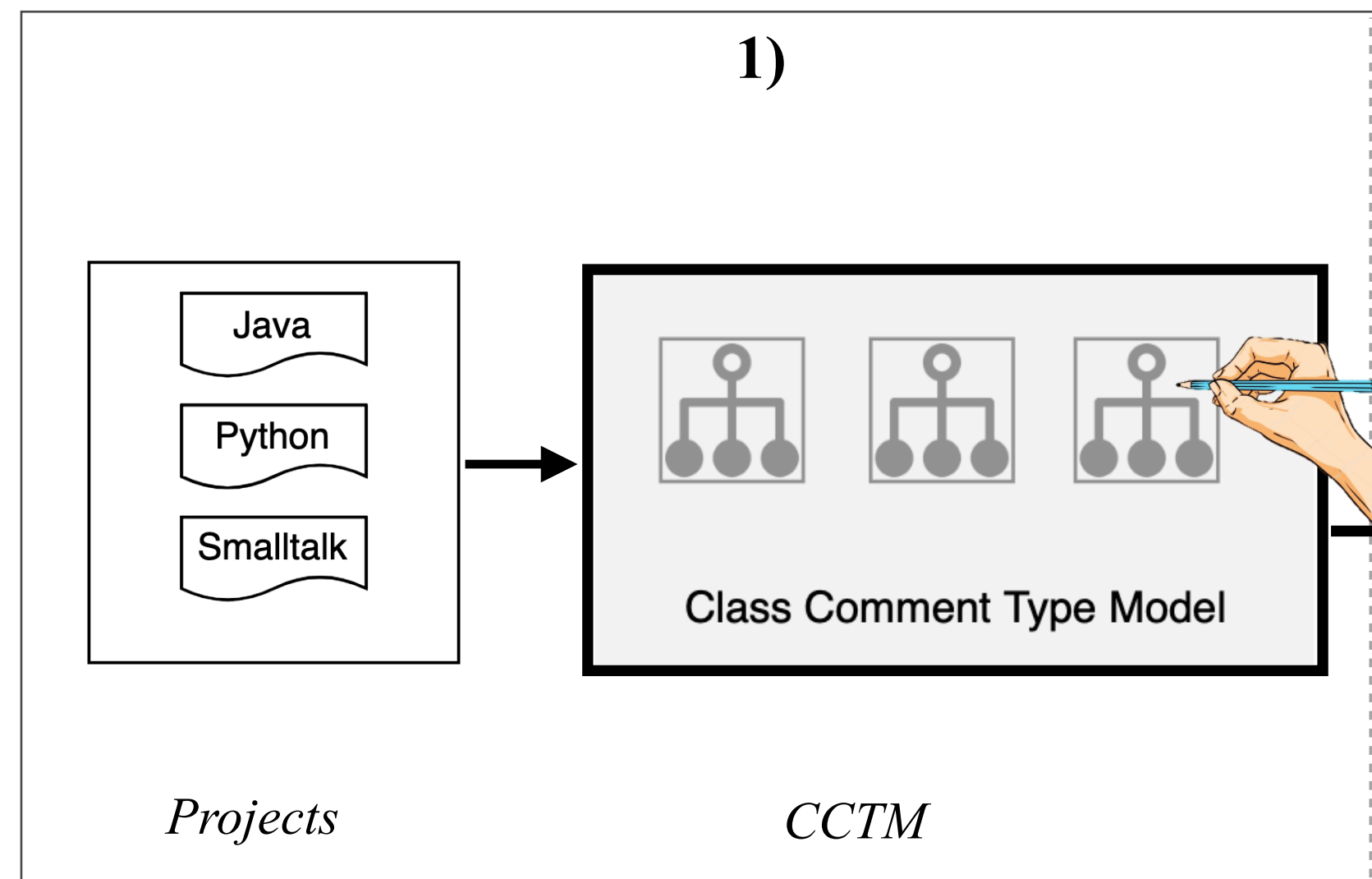
Software developers intensively rely on of software repositories [1], [9], [29] and written communication channels [7], [24] for exchanging relevant information about the ongoing development tasks and the status of overall project progress.

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.,

Automatic identification of information types

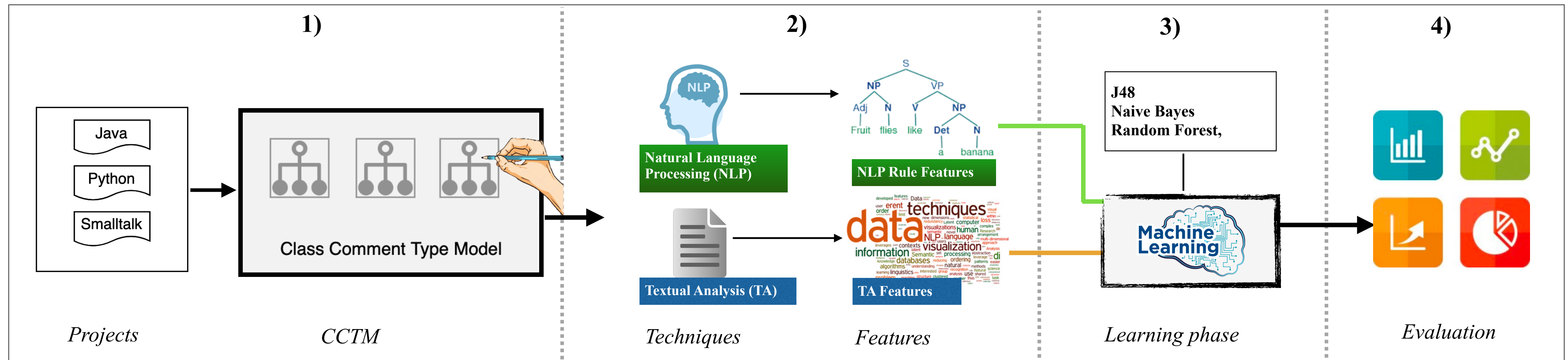


Ground truth: 1,066
classified comments

Features: recurrent NL
patterns + text features

Supervised ML
algorithms

Automatic identification of information types

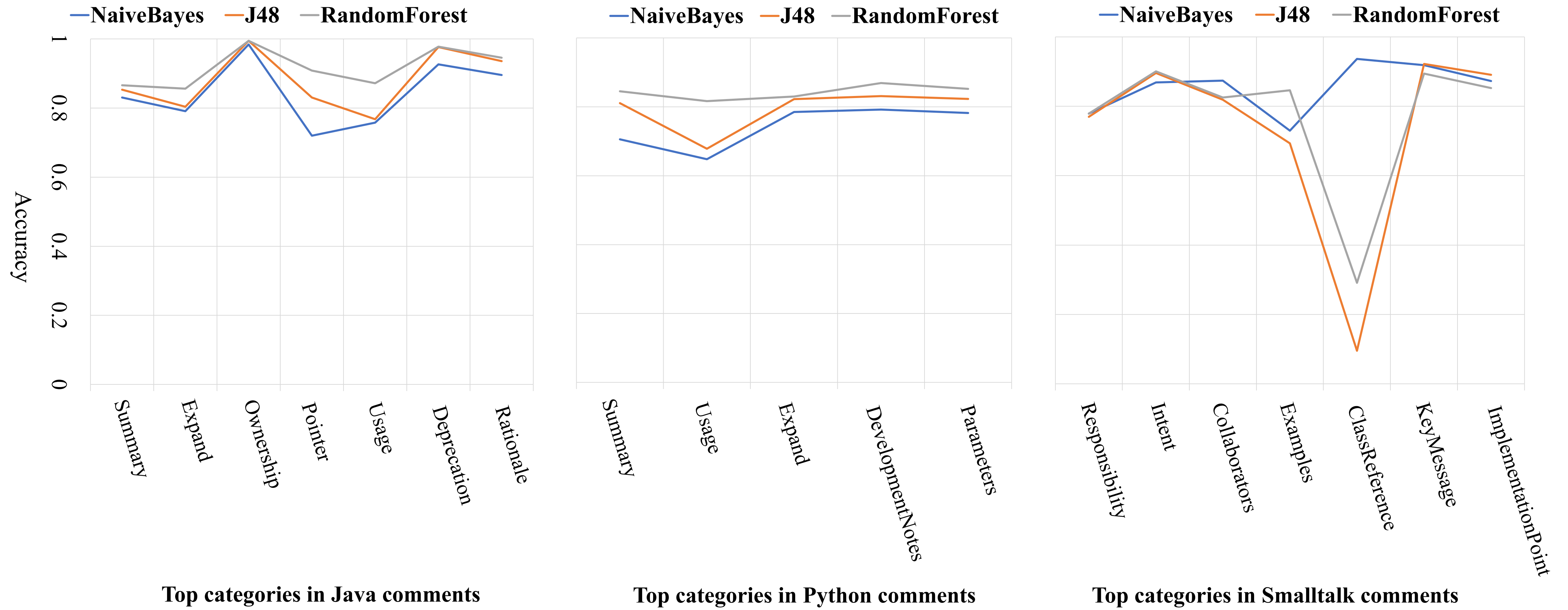


Ground truth: 1,066
classified comments

Features: recurrent NL
patterns + text features

Supervised ML
algorithms

Results



Random Forest technique classifies comments better

Take-home messages

- ☑ Class comments contain high-level design to low-level implementation details.
- ☑ Using recurrent NL patterns as features improves the classification.
- ☑ Some information types need more advanced techniques to accurately identify.

Take-home messages

- ☑ Class comments contain high-level design to low-level implementation details.
- ☑ Using recurrent NL patterns as features improves the classification.
- ☑ Some information types need more advanced techniques to accurately identify.

Take-home messages

- ☑ Class comments contain high-level design to low-level implementation details.
- ☑ Using recurrent NL patterns as features improves the classification.
- ☑ Some information types need more advanced techniques to accurately identify.



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

Pooja Rani¹ · Sebastiano Panichella² · Manuel Leuenberger¹ · Mohammad Ghafari³ · Oscar Nierstrasz¹

Accepted: 25 May 2021 | Published online: 18 August 2021
© The Author(s) 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.

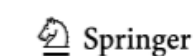
Conclusion The results suggest the need to standardize the commenting guidelines for formatting the text, and to provide headers for the different information types to ensure a consistent style and to identify the information easily. Given the importance of high-quality code comments, we draw numerous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



P. Rani, S. Panichella, M. Leuenberger, M. Ghafari, and O. Nierstrasz. **What do class comments tell us? An investigation of comment evolution and practices in Pharo Smalltalk**, *Empirical Software Engineering*, 2021

Zurich University
of Applied Sciences

zhaw

u^b

UNIVERSITÄT
BERN

THE UNIVERSITY OF
AUCKLAND
NEW ZEALAND

u^b

UNIVERSITÄT
BERN





Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



How to identify class comment types? A multi-language approach for class comment 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



ARTICLE INFO

Article history:
Received 16 December 2020
Received in revised form 5 June 2021
Accepted 9 July 2021
Available online 19 July 2021

Keywords:
Natural language processing technique
Code comment analysis
Software documentation

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 Smalltalk. 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 Smalltalk programming languages. We believe this work can help in monitoring and assessing the quality and evolution of code comments in different programming languages, and thus support maintenance and evolution tasks.

© 2021 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

Software maintenance and evolution tasks require developers to perform program comprehension activities (Fjeldstad and Hamlen, 1983; Haiduc et al., 2010). To understand a software system, developers usually refer to the software documentation of the system (Bavota et al., 2013; de Souza et al., 2005). Previous studies have demonstrated that developers trust code comments more than other forms of documentation when they try to answer program comprehension questions (Maalej et al., 2014; Woodfield et al., 1981; de Souza et al., 2005). 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 (Müller and Fritz,

2013). In particular, well-documented code simplifies software maintenance activities, but many programmers often overlook or delay code commenting tasks (Curiel and Collet, 2013).

Class comments play an important role in obtaining a high-level overview of the classes in object-oriented languages (Cline, 2015). In particular, when applying code changes, developers using object-oriented programming languages can inspect class comments to achieve most or the majority of the high-level insights about the software system design, which is critical for program comprehension activities (Khamis et al., 2010; Nurvitadhi et al., 2003; Steidl et al., 2013). Class comments contain various types of information related to the usage of a class or its implementation (Haouari et al., 2011), which can be useful for other developers performing program comprehension (Woodfield et al., 1981) and software maintenance tasks (de Souza et al., 2005). Unfortunately, (i) different object-oriented programming languages adopt language-specific code commenting notations and guidelines (Farooq et al., 2015), (ii) they embed different kinds of information in the comments (Scowen and Wichmann,

* Corresponding author.
E-mail addresses: pooja.rani@inf.unibe.ch (P. Rani), panc@zhaw.ch (S. Panichella), manuel.leuenberger@inf.unibe.ch (M. Leuenberger), disorbo@unisannio.it (A. Di Sorbo), oscar.nierstrasz@inf.unibe.ch (O. Nierstrasz).

<https://doi.org/10.1016/j.jss.2021.111047>
0164-1212/© 2021 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

P. Rani, S. Panichella, M. Leuenberger, A. Sorbo, and O. Nierstrasz. **How to identify class comment types? A multi-language approach for class comment classification**, *Journal of Systems & Software*, 2021

Zurich University of Applied Sciences



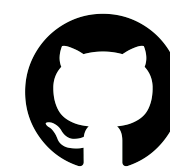
u^b

UNIVERSITÄT BERN



u^b

UNIVERSITÄT BERN



What information developers write in class comments across languages?

Do they follow the coding style guidelines?

Coding style guidelines


Each community has its own guidelines



Coding style guidelines

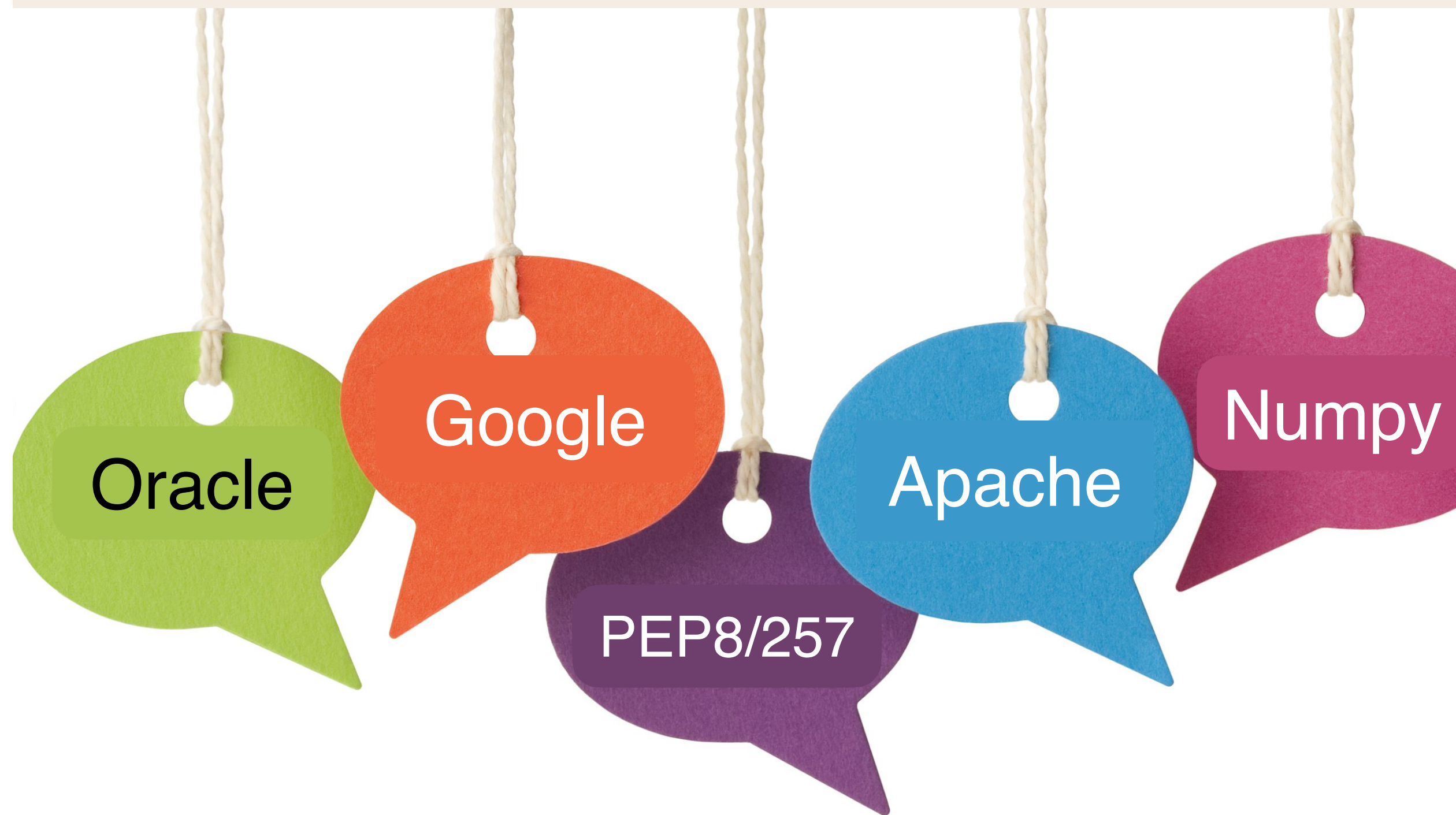
Each community has its **own** guidelines



- 
- First sentence should summarise the class
 - Use phrases instead of complete sentences
 - The `@deprecated` description should tell what to use as a replacement

Coding style guidelines

Each community has its own guidelines



94 rules


13 rules

29 rules

222 rules

76 rules

Comment related rules are hard to locate

- 
- First sentence should summarise the class
 - Use phrases instead of complete sentences
 - The @deprecated description should tell what to use as a replacement

Do developers follow comment conventions?

```
/**
 * 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```



- First sentence should summarise the class
- Use phrases instead of complete sentences
- The @deprecated description should tell what to use as a replacement

Do developers follow comment conventions?

```
/**
 * 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Followed



- ✓ First sentence should summarise the class

Do developers follow comment conventions?

```
/**
 * 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

Not Followed



- ✓ First sentence should summarise the class
- ✗ Use phrases instead of complete sentences

Do developers follow comment conventions?

```
/**
 * 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

The rule is not applicable



- ✓ First sentence should summarise the class
- ✗ Use phrases instead of complete sentences
- The @deprecated description should tell what to use as a replacement

Do developers follow the style guidelines?

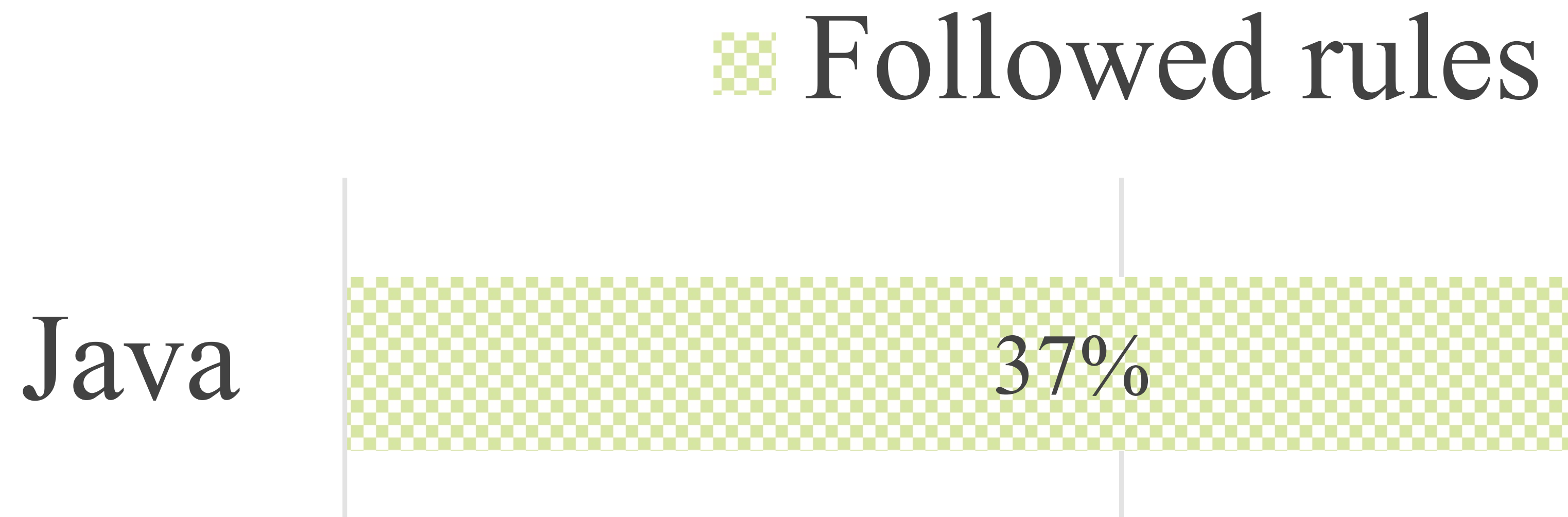
20 open-source projects

Java, Python, Smalltalk

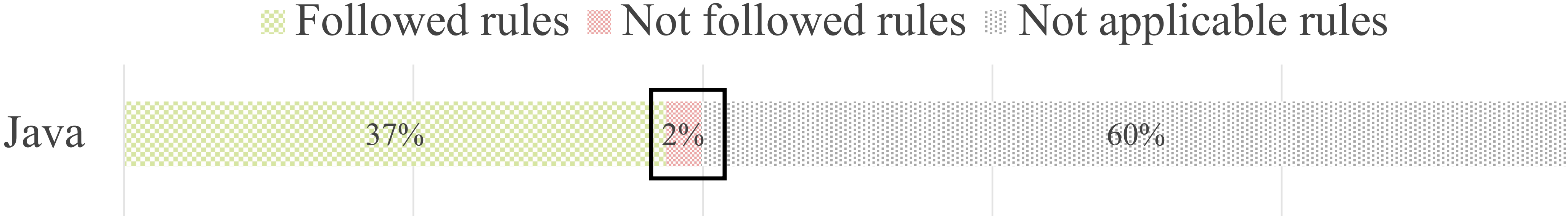
1,245 class comment conventions

1,066 sample class comments

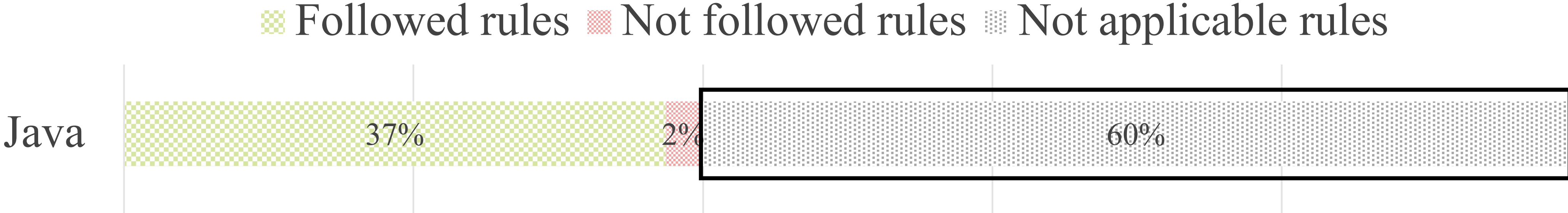
Do developers follow comment conventions?



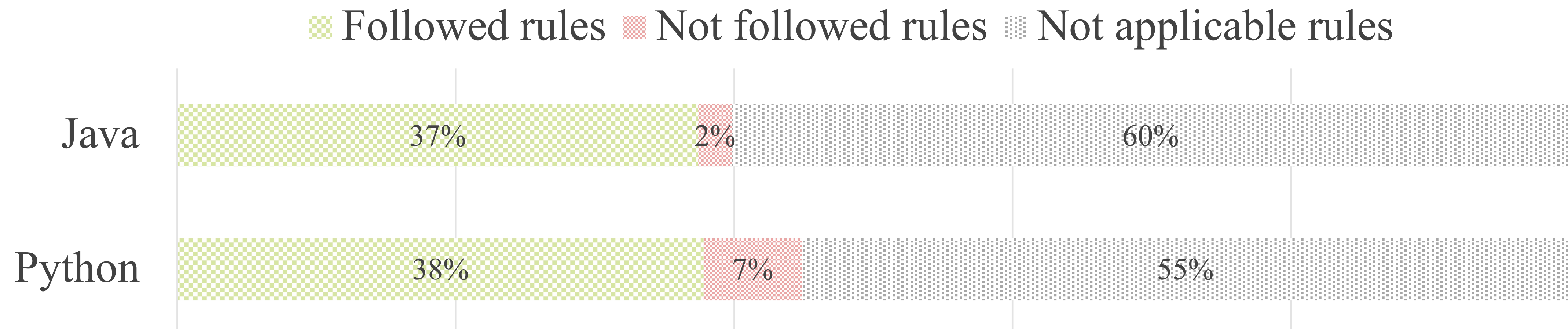
Do developers follow comment conventions?



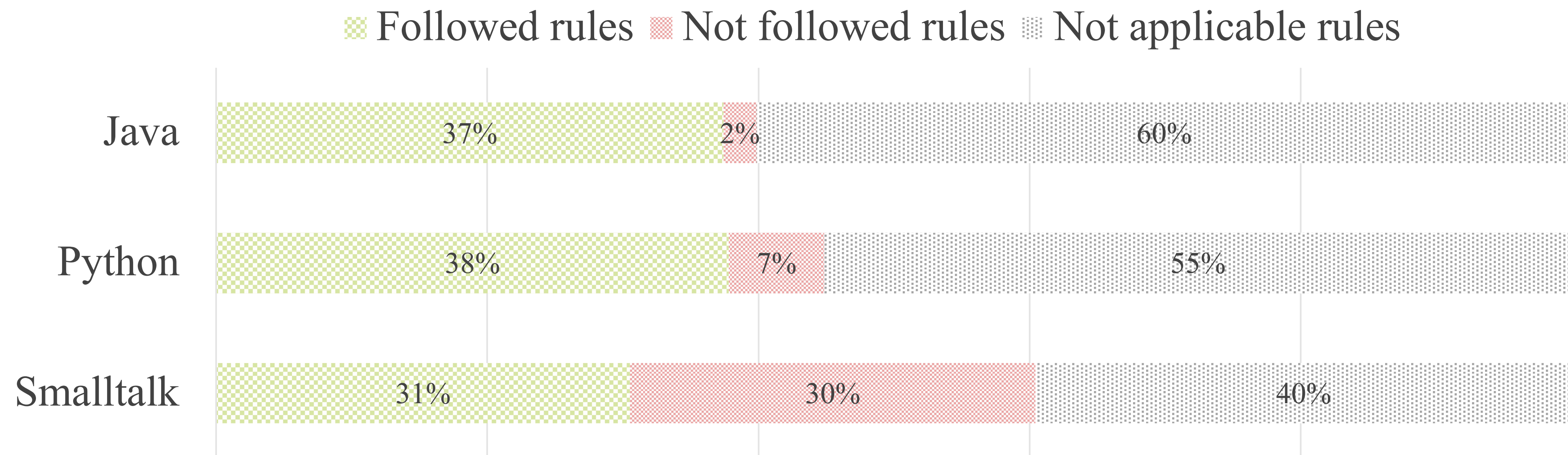
Do developers follow comment conventions?



Do developers follow comment conventions?



Do developers follow comment conventions?



Developers do not always adopt conventions (not applicable rules)

Java developers violate rules less often than Python and Smalltalk

Do developers follow comment conventions?



Do developers follow comment conventions?



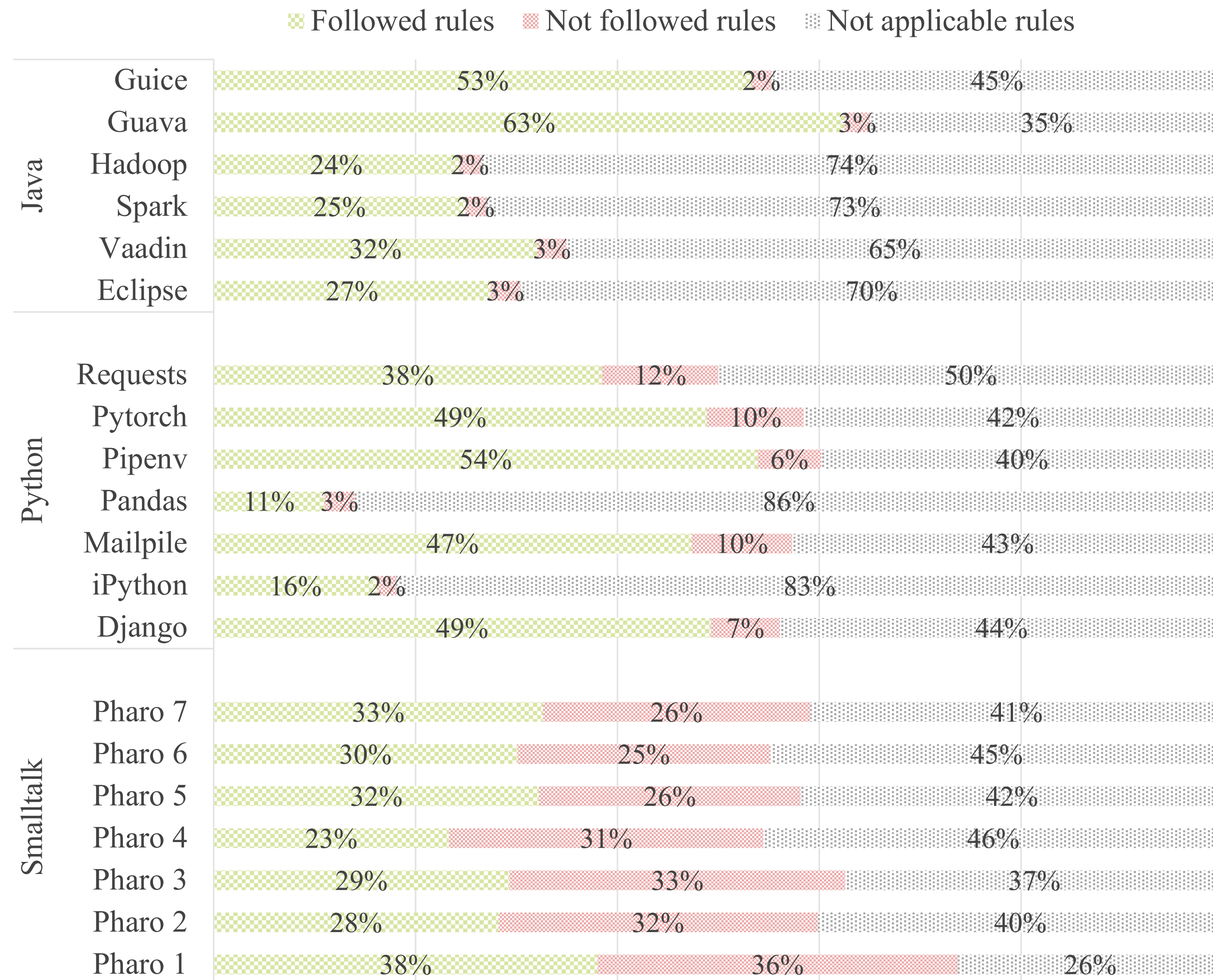
Do developers follow comment conventions?



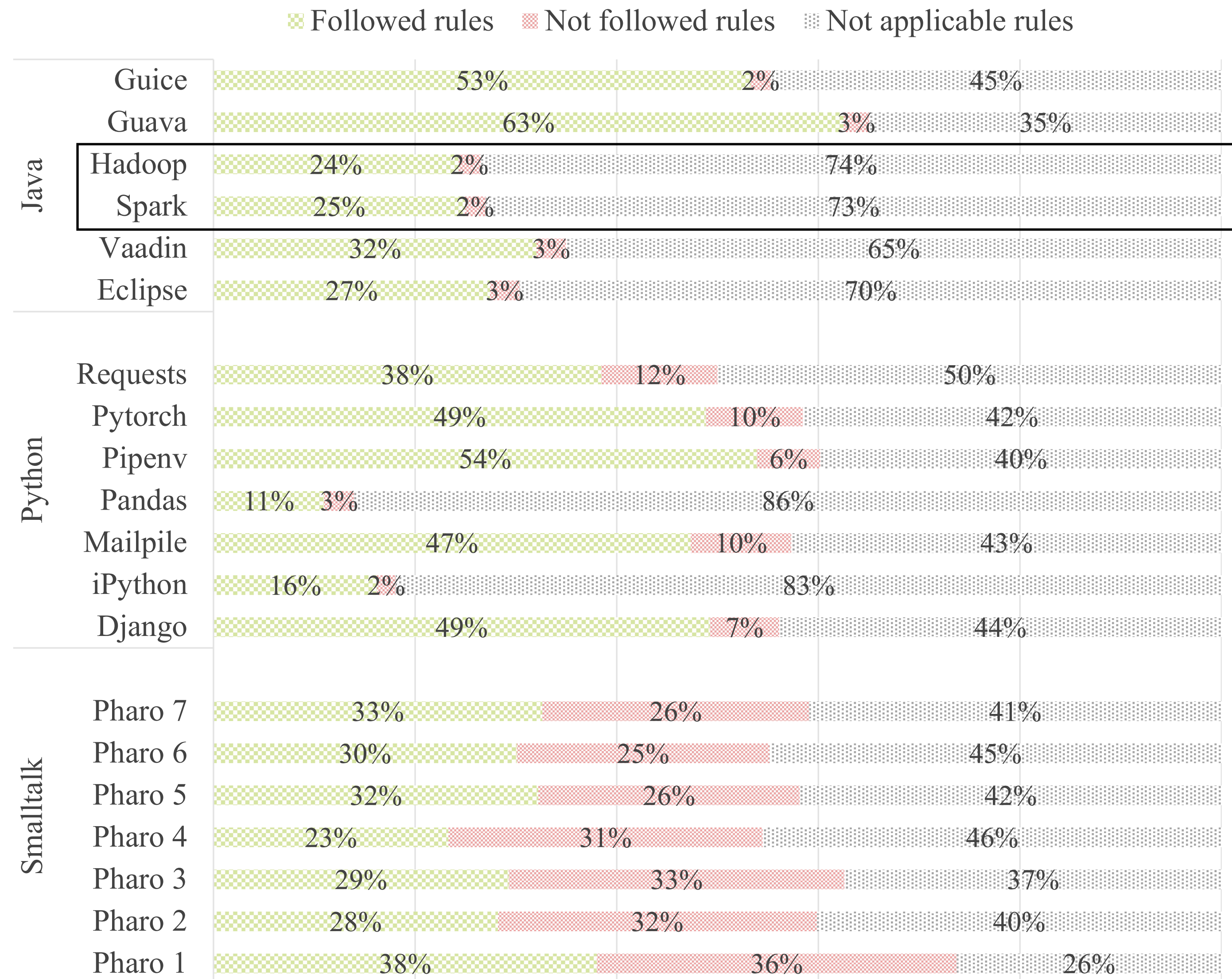
Do developers follow comment conventions?



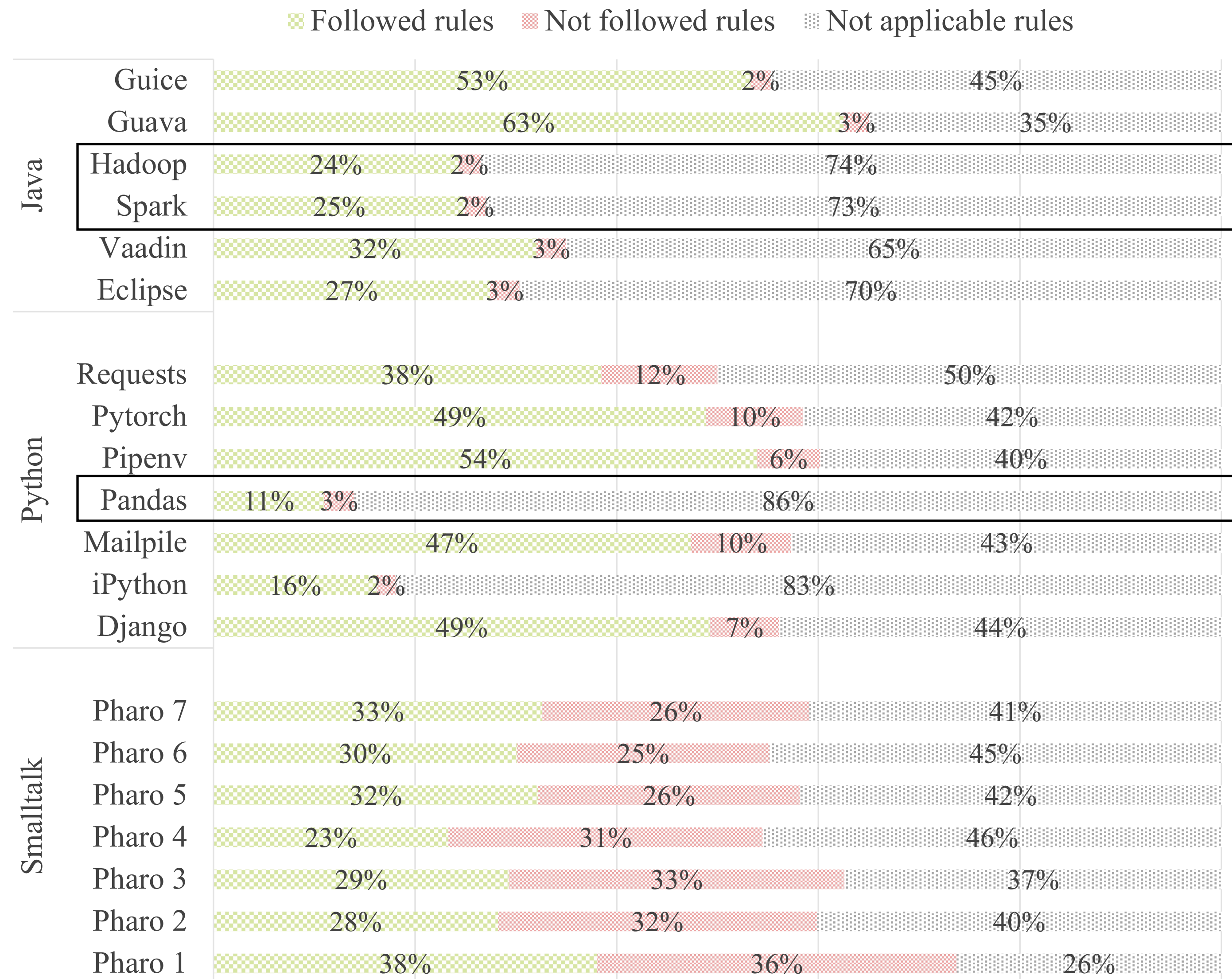
Do developers follow comment conventions?



Do developers follow comment conventions?



Do developers follow comment conventions?



Do Comments follow Commenting Conventions? A Case Study in Java and Python

Pooja Rani*, Suada Abukar*, Nataliia Stulova*, Alexandre Bergel[†], Oscar Nierstrasz*

*Software Composition Group, University of Bern, Bern, Switzerland

[†]Department of Computer Science (DCC), University of Chile, Santiago, Chile

scg.unibe.ch/staff

Abstract—Assessing code comment quality is known to be a difficult problem. A number of coding style guidelines have been created with the aim to encourage writing of informative, readable, and consistent comments. However, it is not clear from the research to date which specific aspects of comments the guidelines cover (e.g., syntax, content, structure). Furthermore, the extent to which developers follow these guidelines while writing code comments is unknown.

We analyze various style guidelines in Java and Python and uncover that the majority of them address more the content aspect of the comments rather than syntax or formatting. However, when considering the different types of information developers embed in comments and the concerns they raise on various online platforms about the commenting practices, existing comment conventions are not yet specified clearly enough, nor do they adequately cover important concerns. We find that developers of both languages follow the writing style and content-related comment conventions more often than syntax and structure types of conventions. Our results highlight the mismatch between developer commenting practices and style guidelines, and provide several focal points for the design and improvement of comment quality checking tools.

Index Terms—Comment analysis, Software documentation, Coding Style Guidelines, Coding Standards

I. INTRODUCTION

Developers use several kinds of software documentation, including design documents, wikis, and code comments, to understand and maintain programs. Studies show that developers trust code comments more than other forms of documentation [1]. As code comments are usually written in a semi-structured manner using natural language sentences, and they are not checked by the compiler, developers have the freedom to write comments in various ways [2], [3], [4].

To encourage developers to write consistent, readable, and informative code comments, programming language communities and several large organizations, such as Google and Apache, provide coding style guidelines that also suggest comment-related conventions [5], [6], [7]. These conventions cover various aspects of comments, such as syntactic, stylistic, or content-related aspects. For instance, “Use 3rd person (descriptive), not 2nd person (prescriptive)” is an example of a stylistic comment convention for Java documentation comments [5]. However, to what extent these aspects are covered within different style guidelines and languages is not known. Therefore, we formulate the question: **RQ₁**: Which

type of comment conventions are suggested by various style guidelines?

As high-quality comments support developers in understanding and maintaining their programs, it is essential to ensure the adherence of their comments to the style guidelines to evaluate the overall comment quality. Rani *et al.* have investigated class comments of Smalltalk and their adherence to the commenting conventions provided by a default template [8]. They found that Smalltalk developers follow writing style and content-related comment conventions more than 50% of the time, but they use inconsistent structure and formatting of comment content. As Java and Python are among the most popular languages in use, several research works have focused on studying comments in Java and Python ([3], [4]), some especially focusing on class comments [9]. However, it remains largely unknown whether Java and Python developers adhere to the commenting conventions suggested by the style guidelines or not. To obtain this understanding, we formulate another research question: **RQ₂**: To what extent do developers follow commenting conventions in writing code comments in Java and Python?

Our initial results show that the majority of style guidelines propose more content-related conventions than other types of conventions, but compared to the different types of content developers actually embed in comments ([3], [4], [9]), and the concerns they raise on online platforms (e.g., StackOverflow or Quora) regarding comment conventions [10], it is clear that existing conventions are neither adequate, nor precise enough. On the other hand, these style guidelines often include conventions that are not relevant or applicable in many cases, leading developers to ignore them.

When the conventions are applicable, developers often follow the writing style and content conventions (80% of comments), but violate structure conventions in Java and Python class comments (nearly 30% of comments), confirming the previous results for Smalltalk by Rani *et al.* [8]. Although the project-specific guidelines provide very few additional class comment conventions, these conventions are followed more often compared to the conventions suggested by the standard guidelines both in Java and Python class comments. The data related to RQ₁ and RQ₂ is given in the replication package.¹

¹<https://doi.org/10.5281/zenodo.5296443>

P. Rani, S. Abukar, N. Stulova, A. Bergel, and O. Nierstrasz. **Do comments follow commenting conventions? A case study in Java and Python**, In Proceedings of 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), 2021

113 Replication Package on 

<https://doi.org/10.5281/zenodo.5296443>

Video on YouTube 

https://youtu.be/mX_9XxQTSxQ

u^b

UNIVERSITÄT
BERN

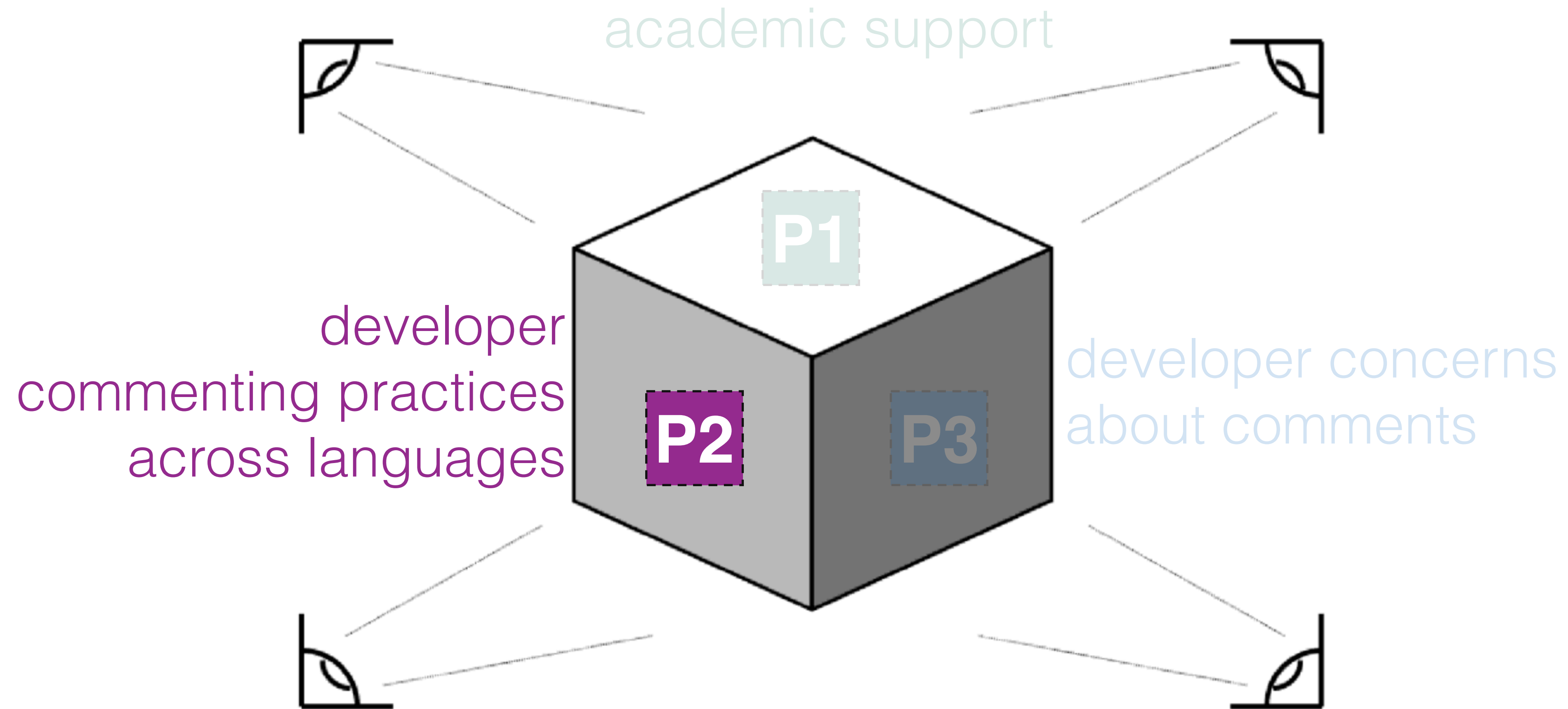


u^b

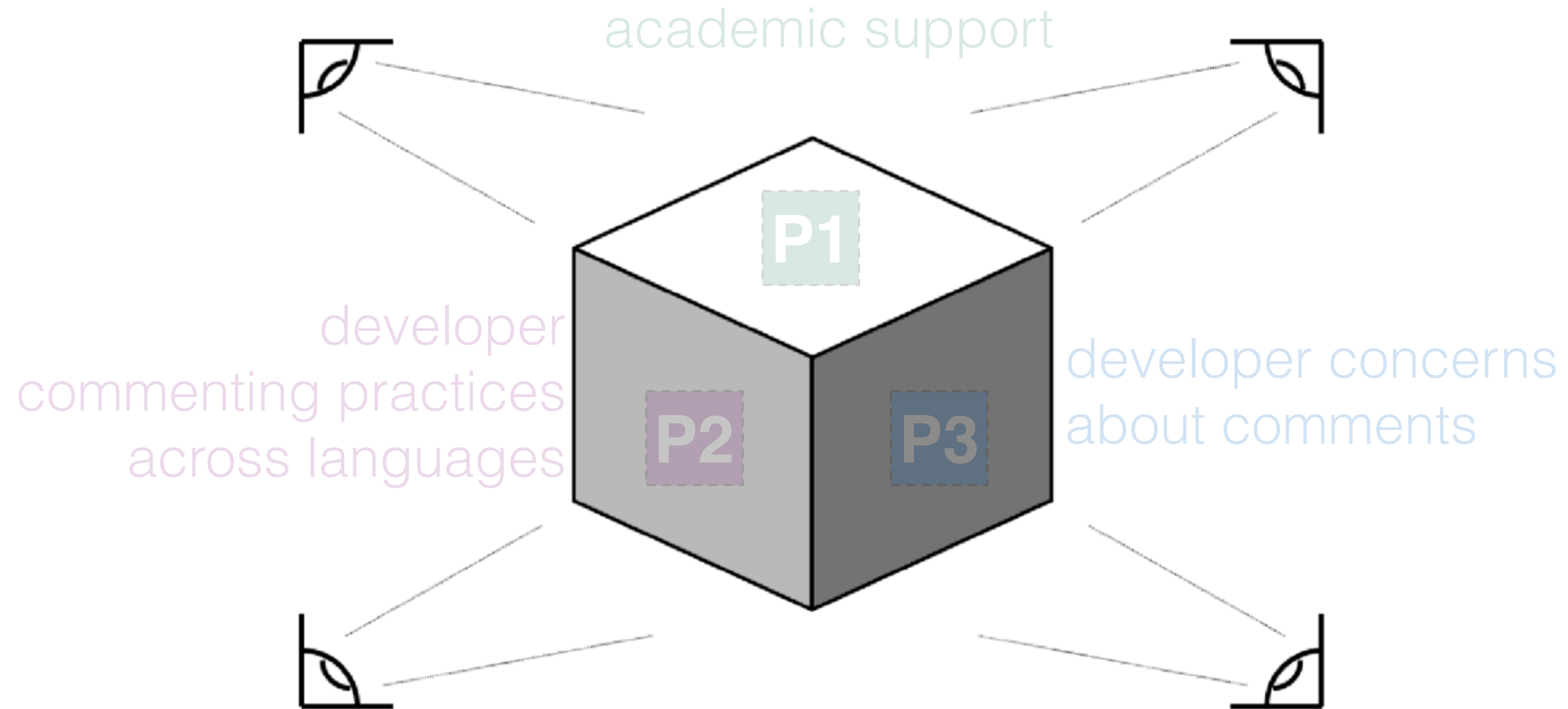
UNIVERSITÄT
BERN



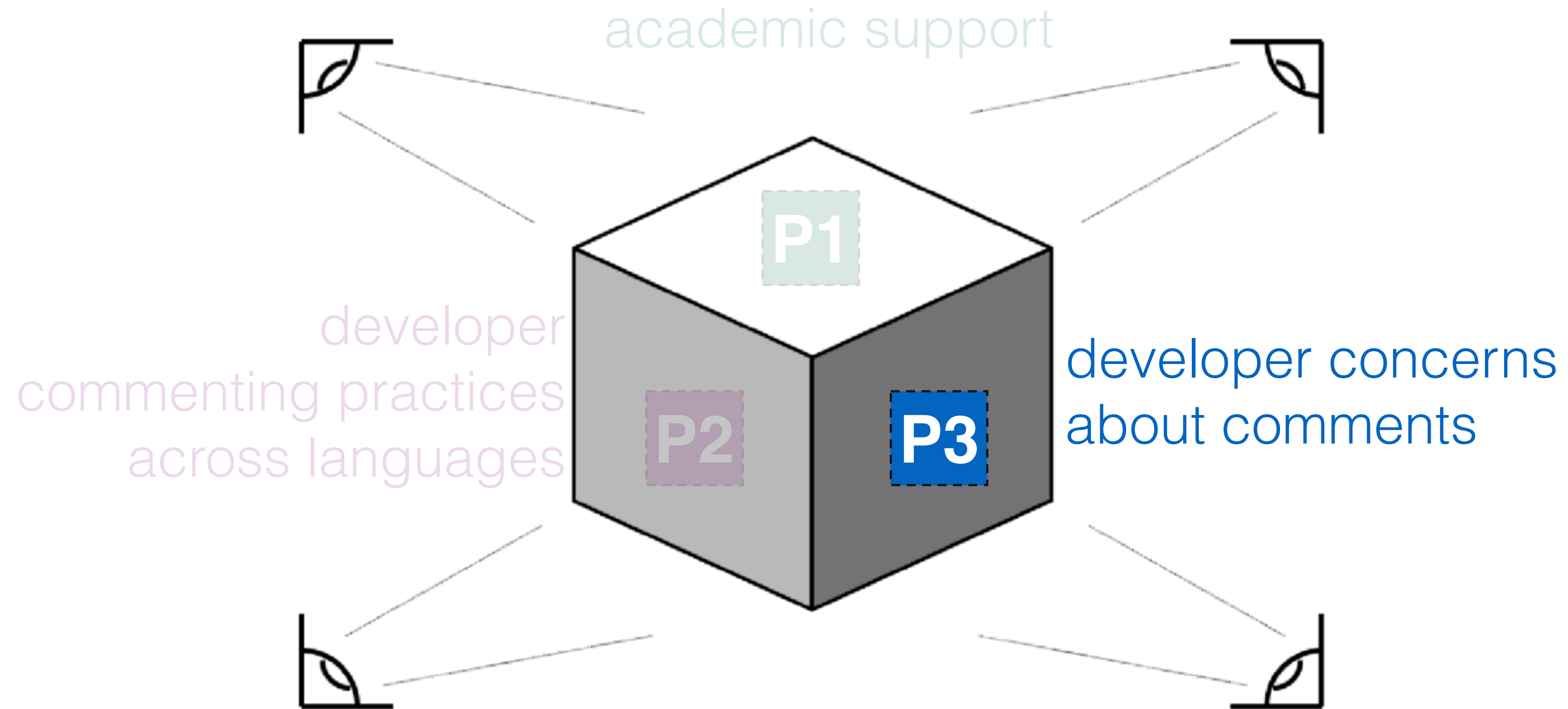
We define three perspectives



We define three perspectives



P3: Questions developer ask



Developer concerns about comments

Various syntax and structure conventions

Availability of multiple style guidelines

Lack of tools to verify all aspects of comments

Developer concerns about comments

Various syntax and structure conventions

Availability of multiple style guidelines

Lack of tools to verify all aspects of comments

Developer concerns about comments

Various syntax and structure conventions

Availability of multiple style guidelines

Lack of tools to verify all aspects of comments

Developer concerns about comments

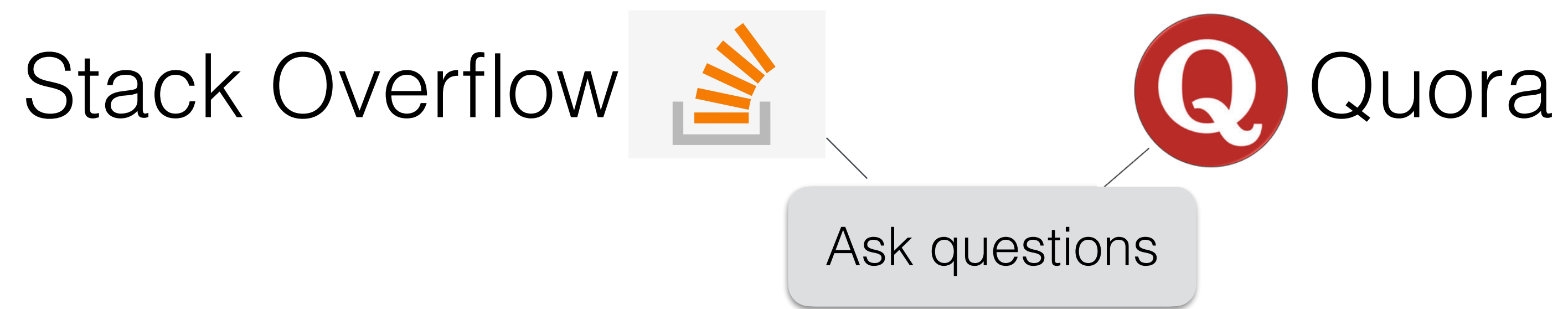
Various syntax and structure conventions

Availability of multiple style guidelines

Lack of tools to verify all aspects of comments

Confusion among developers on what convention to adopt and when

Ask questions on various online platforms



Developer concerns about comments

Stack Overflow, Quora

23,631 comment related posts

1,400 sample posts

Automated analysis (LDA topic modelling)

Manual analysis

LDA Topic modelling

10 topics

#	Topic Name
1	Syntax and Format
2	IDEs & Editors
3	R Documentation
4	Code Conventions
5	Developing frameworks for thread commenting

LDA Topic modelling

#	Topic Name
1	Syntax and Format
2	IDEs & Editors
3	R Documentation
4	Code Conventions
5	Developing frameworks for thread commenting
6	Open-source software
7	Documentation generation
8	Thread comments in web-sites
9	Naming conventions & data types
10	Seeking documentation & learning language

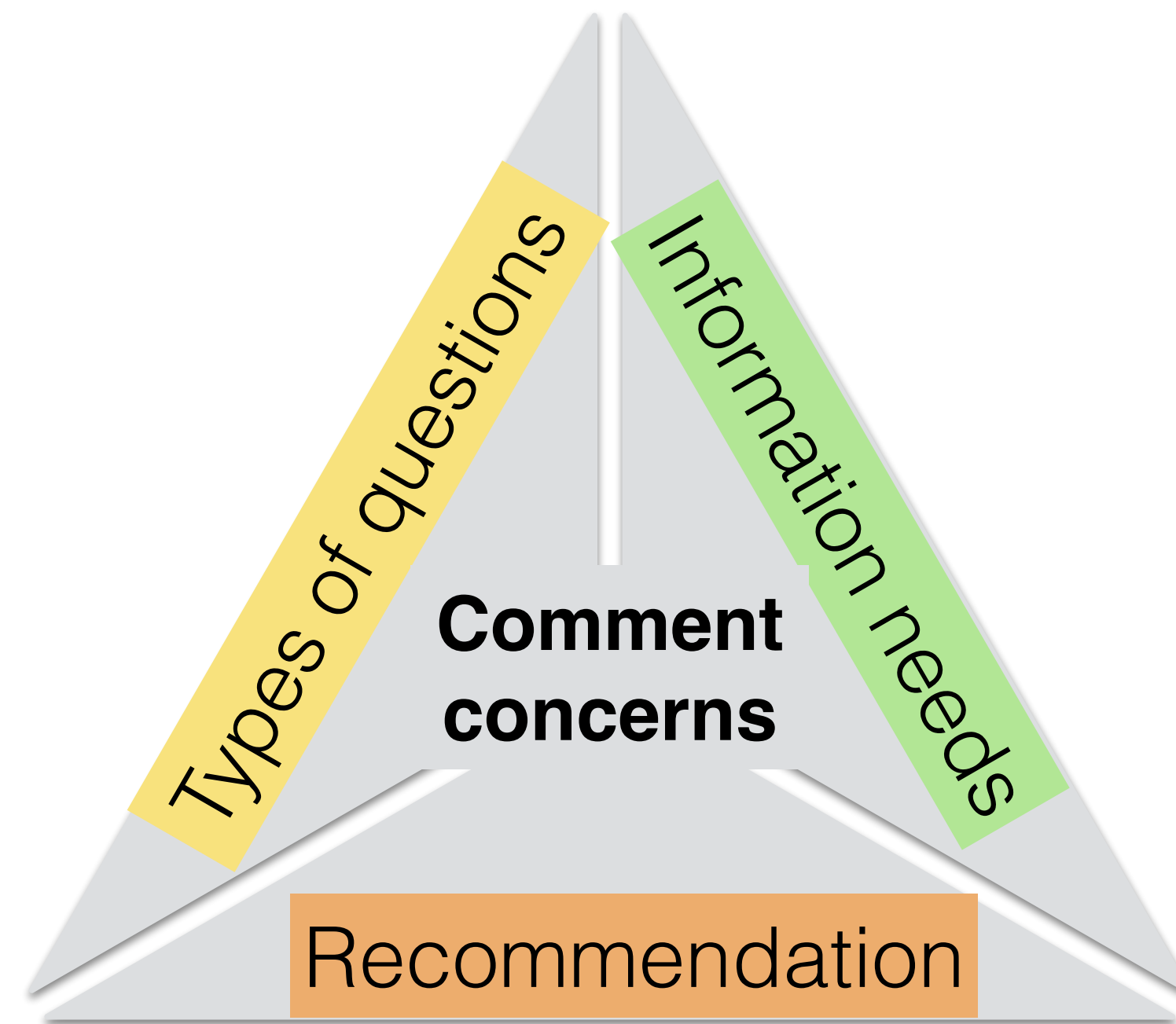
Expected topics like **Documentation Generation** or **Syntax and Format** were successfully identified by LDA.

LDA Topic modelling

#	Topic Name
1	Syntax and Format
2	IDEs & Editors
3	R Documentation
4	Code Conventions
5	Developing frameworks for thread commenting
6	Open-source software
7	Documentation generation
8	Thread comments in web-sites
9	Naming conventions & data types
10	Seeking documentation & learning language

Comment term is used in various contexts and environments

Manual analysis



127

Should JavaDoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times

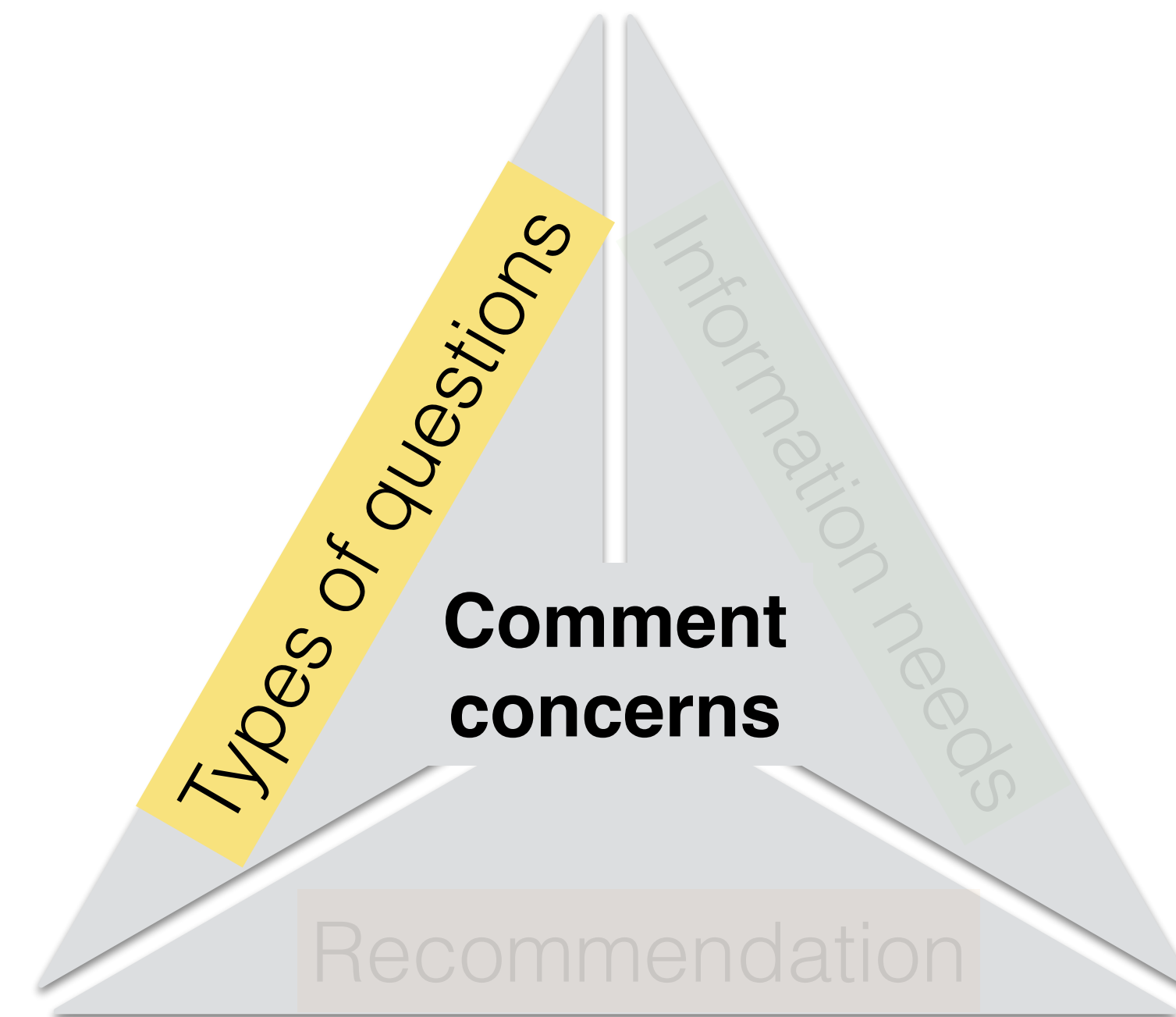
128

stackoverflow Products

Home
PUBLIC
Questions

Should JavaDoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times



Should JavaDoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times

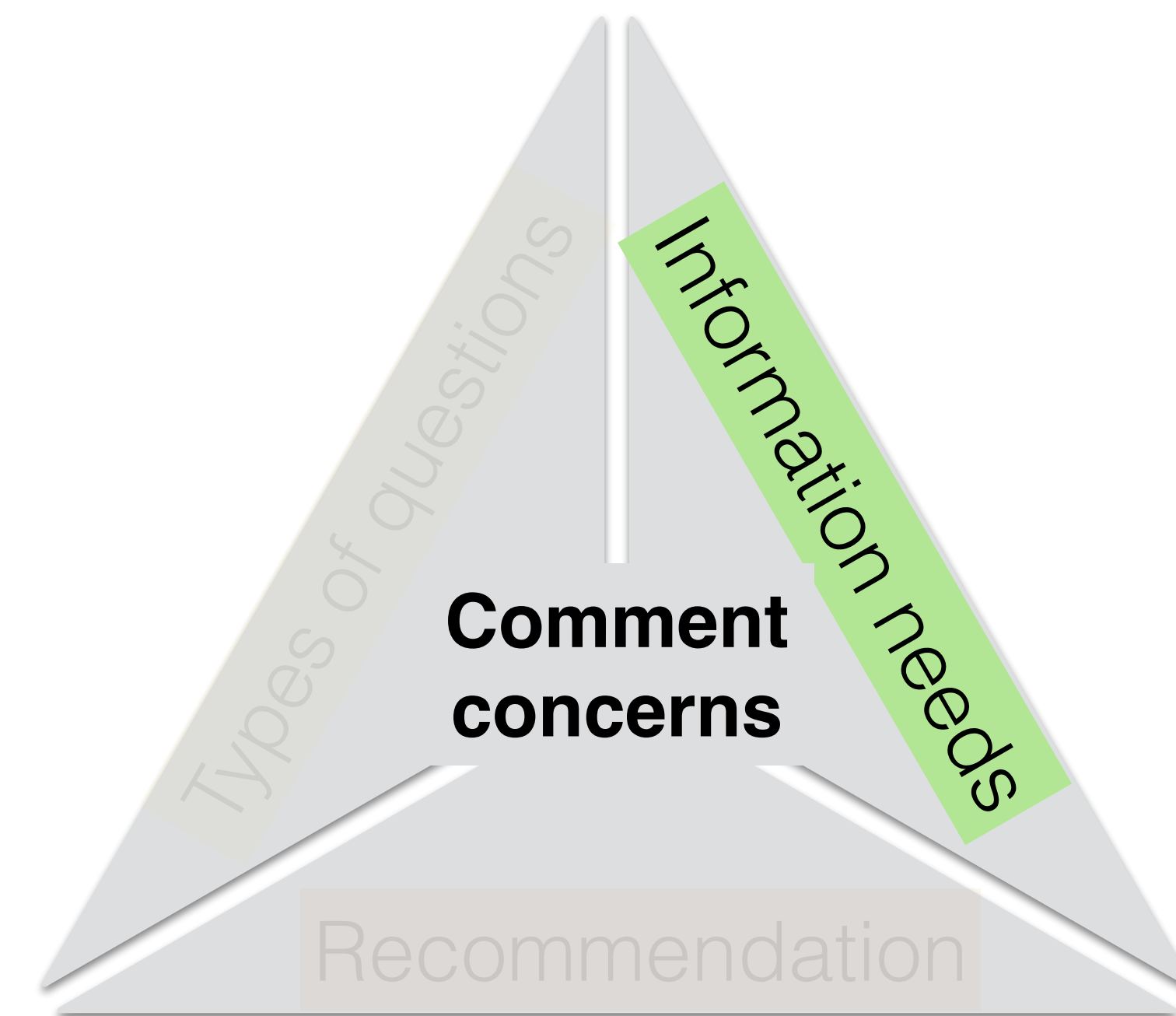
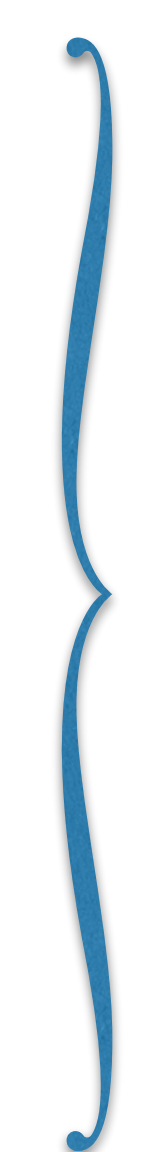
4 What is the recommended place to put JavaDoc for a method with an annotation? Before or after the annotation?

```
@Test
/**
 * My doc
 */
public void testMyTest(){
}
```

OR

```
/**
 * My doc
 */
@Test
public void testMyTest(){
}
```

java coding-style annotations javadoc



Should JavaDoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times

4 What is the recommended place to put JavaDoc for a method with an annotation? Before or after the annotation?

```

@Test
/**
 * My doc
 */
public void testMyTest(){
}

```

OR

```

/**
 * My doc
 */
@Test
public void testMyTest(){
}

```

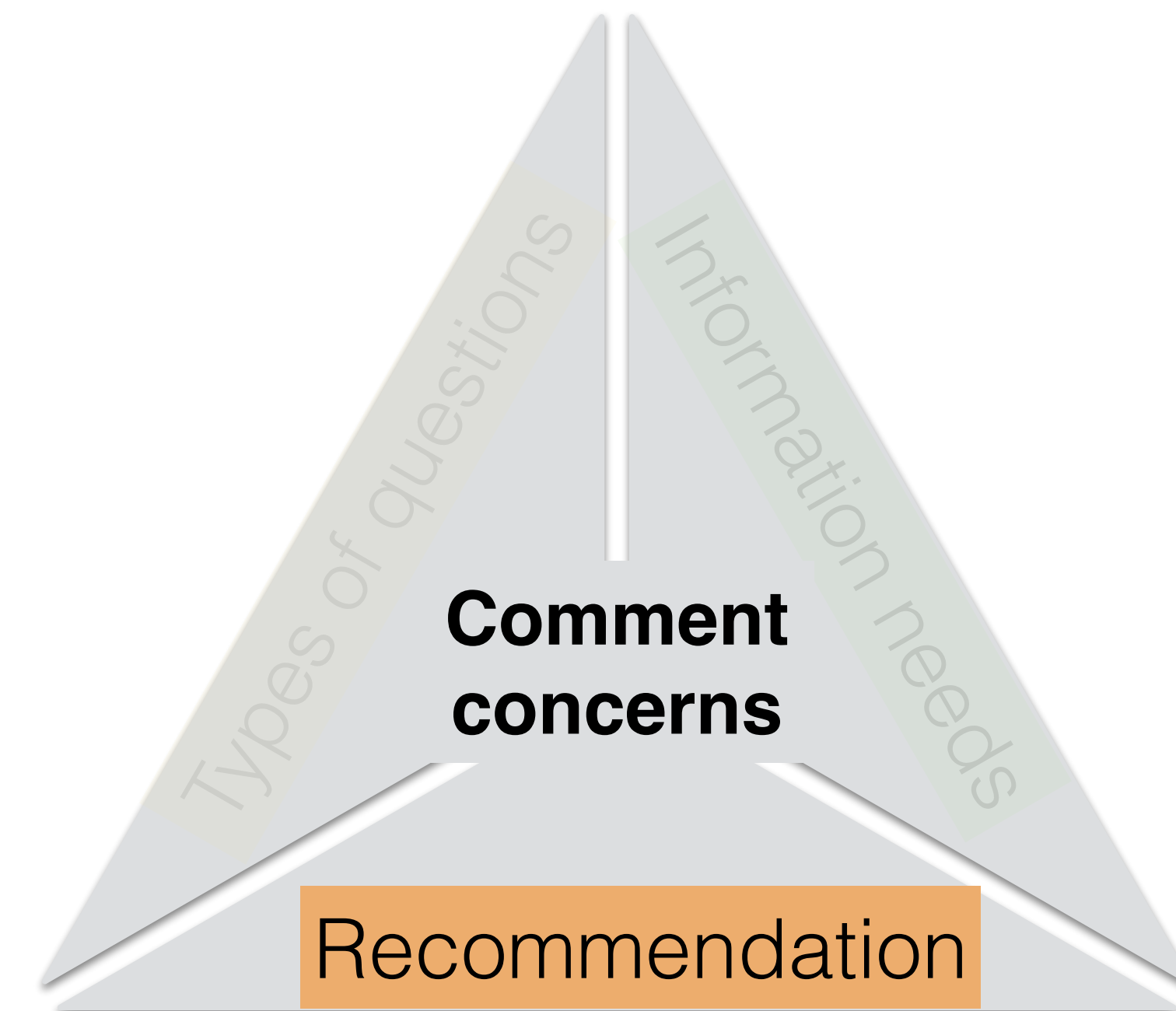
java coding-style annotations javadoc

7 I don't think it matters but second format is better. annotations are part of the code and play crucial role per their usage pattern. Better to keep all code related entries together.

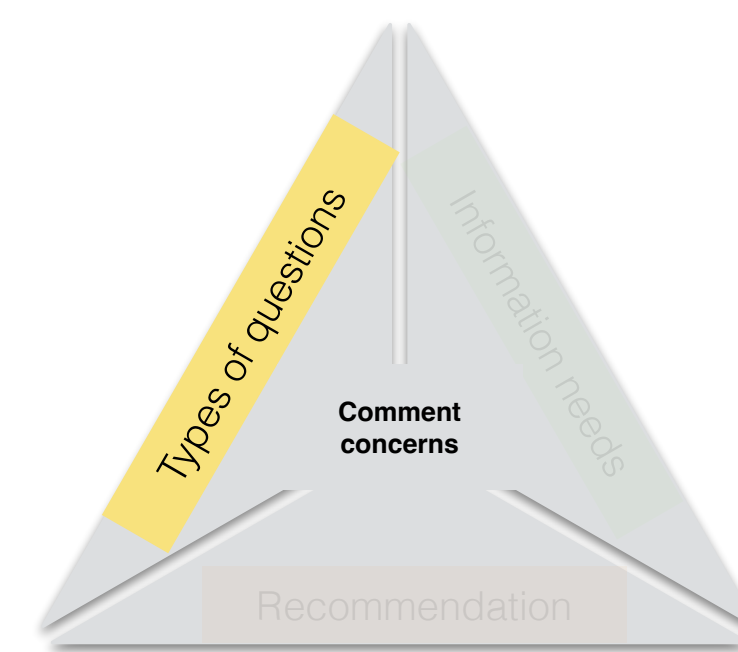
Share Edit Follow

answered Nov 14 '12 at 18:12

Yogendra Singh 32.8k 6 60 71



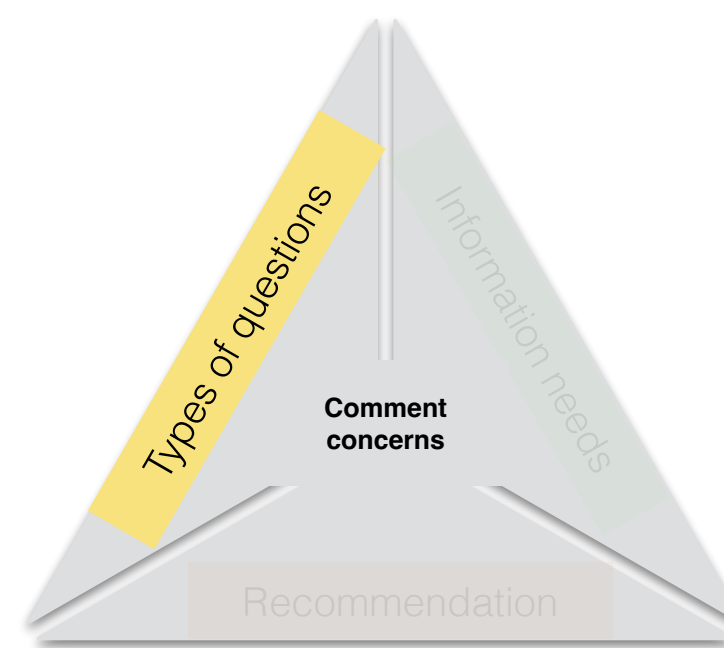
Developer concerns about comments



Developer concerns about comments

Types of questions

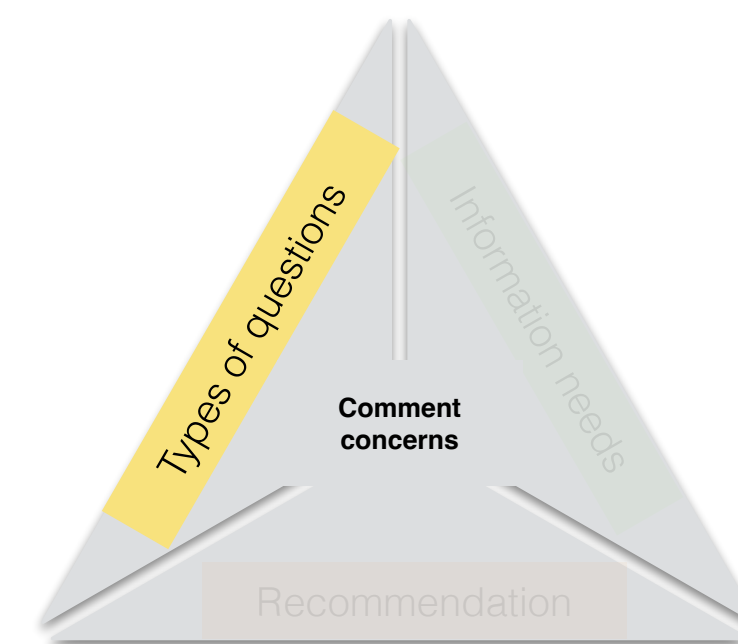
- Implementation strategy
- Best practice
- Background information
- Limitation and possibility
- Implementation problem
- Opinion
- Error



Developer concerns about comments



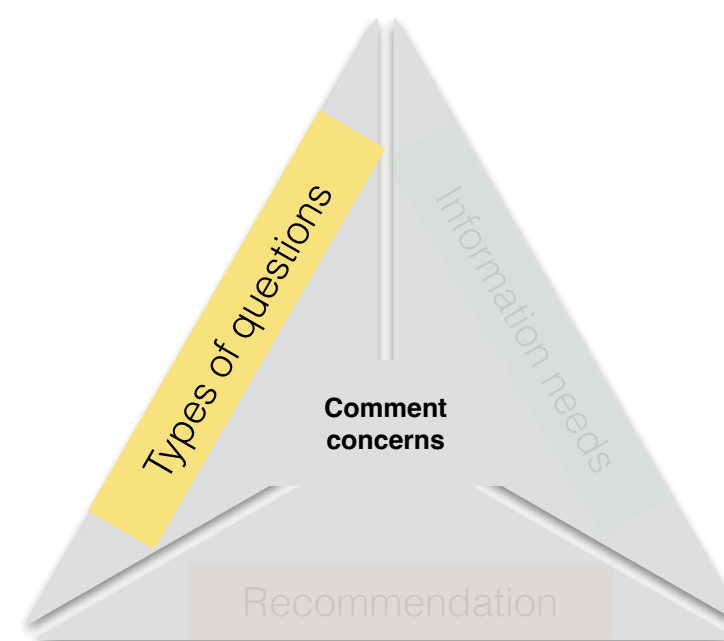
how to write/implement comments



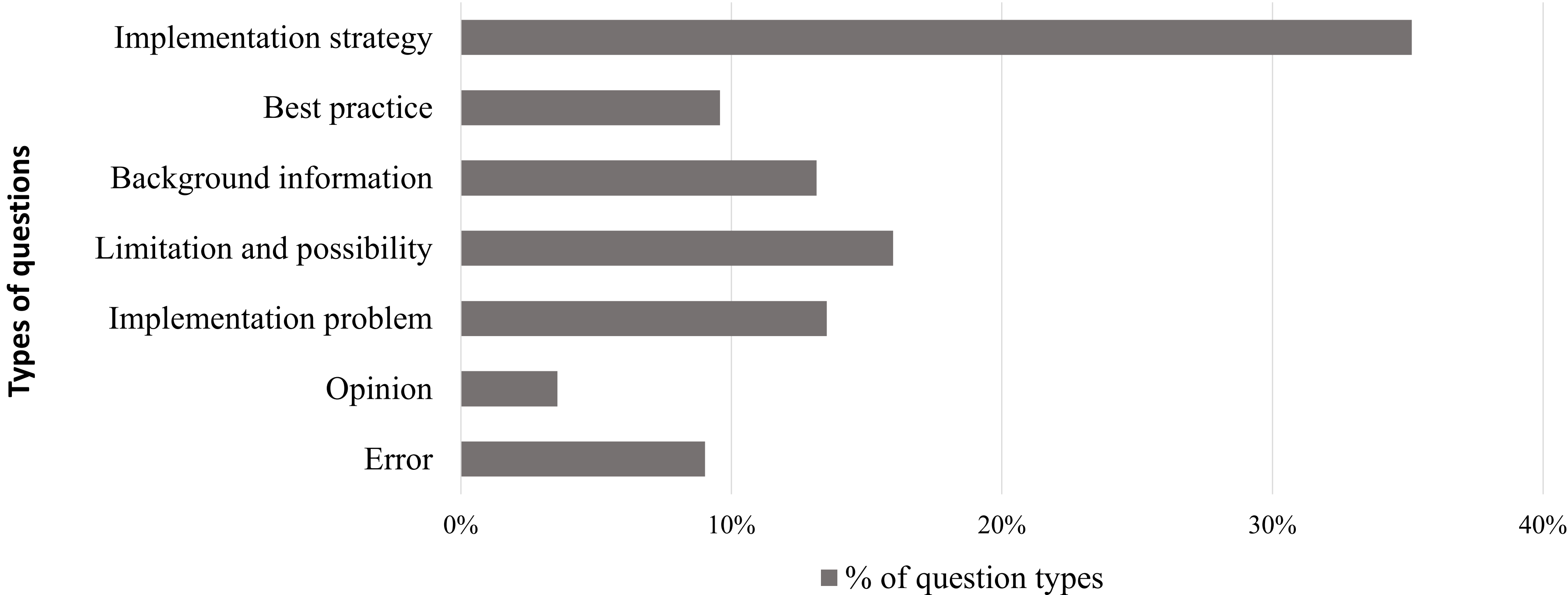
Developer concerns about comments

- Types of questions
- Implementation strategy
 - Best practice
 - Background information
 - Limitation and possibility**
 - Implementation problem
 - Opinion
 - Error

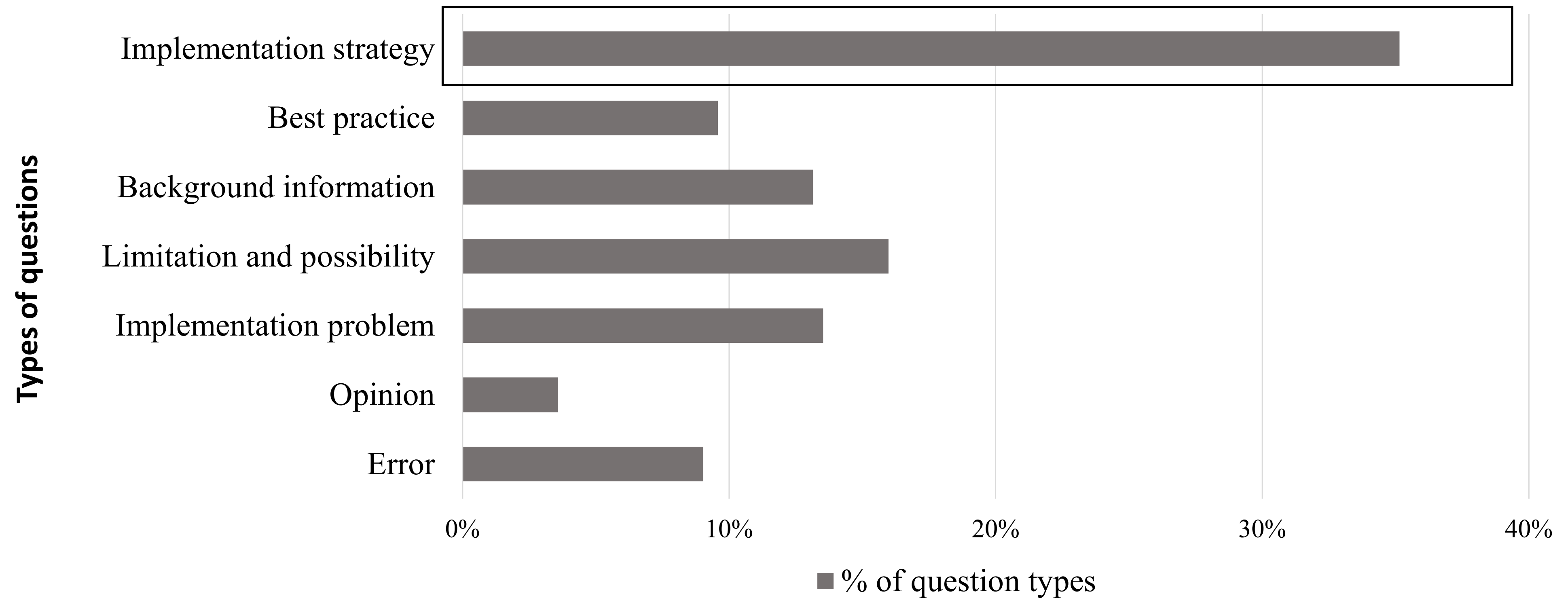
Is it possible to do this?



Developer concerns about comments

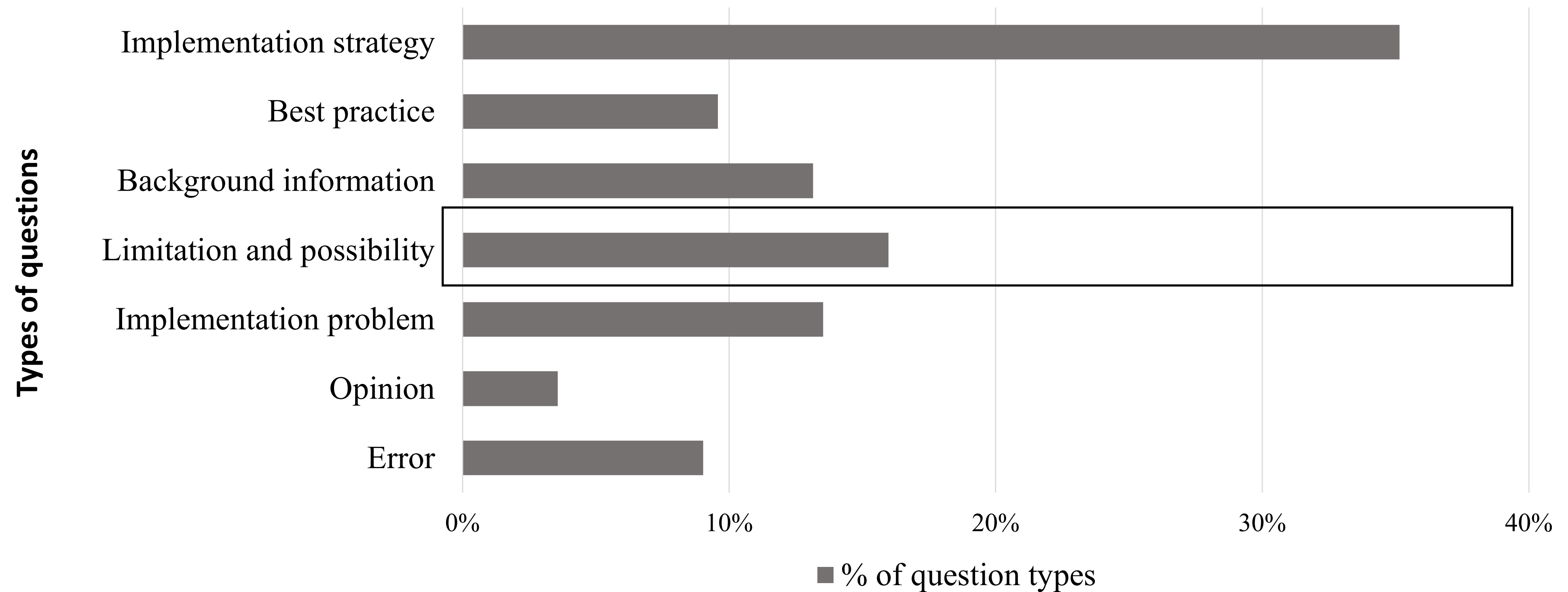


Developer concerns about comments



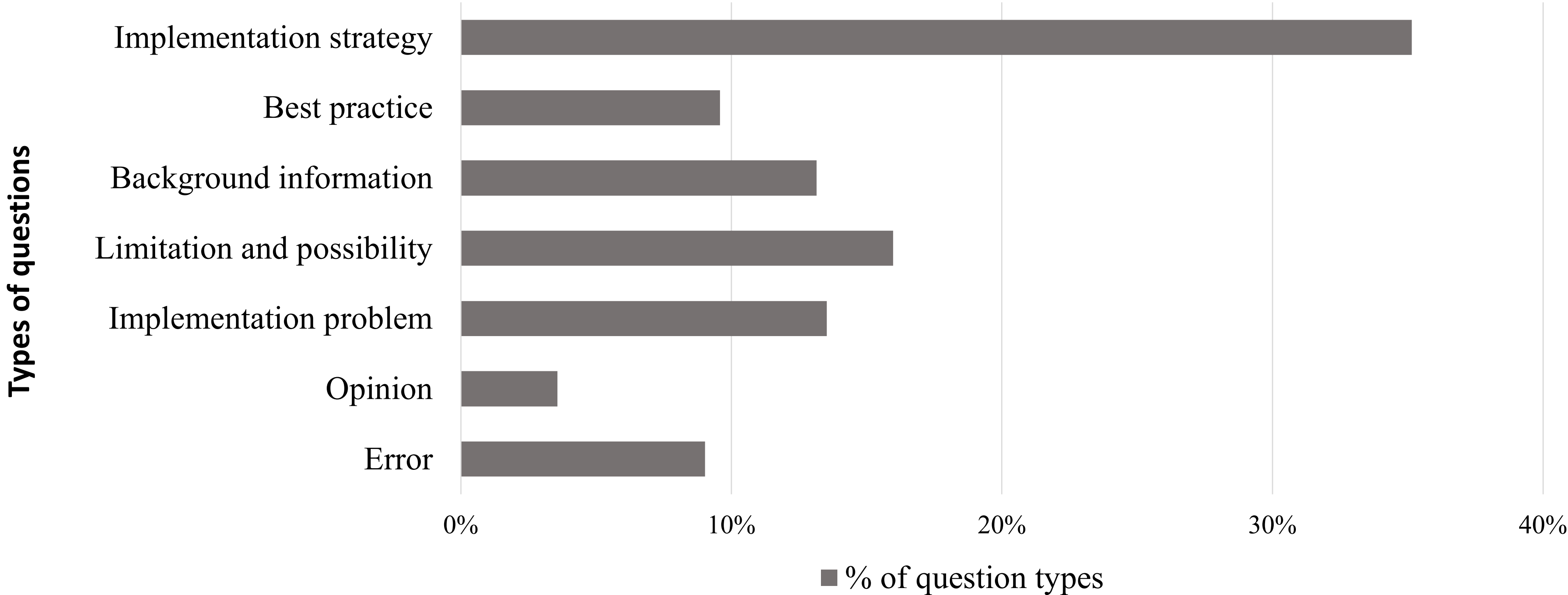
Developers ask frequently “**how to** write comments?”

Developer concerns about comments

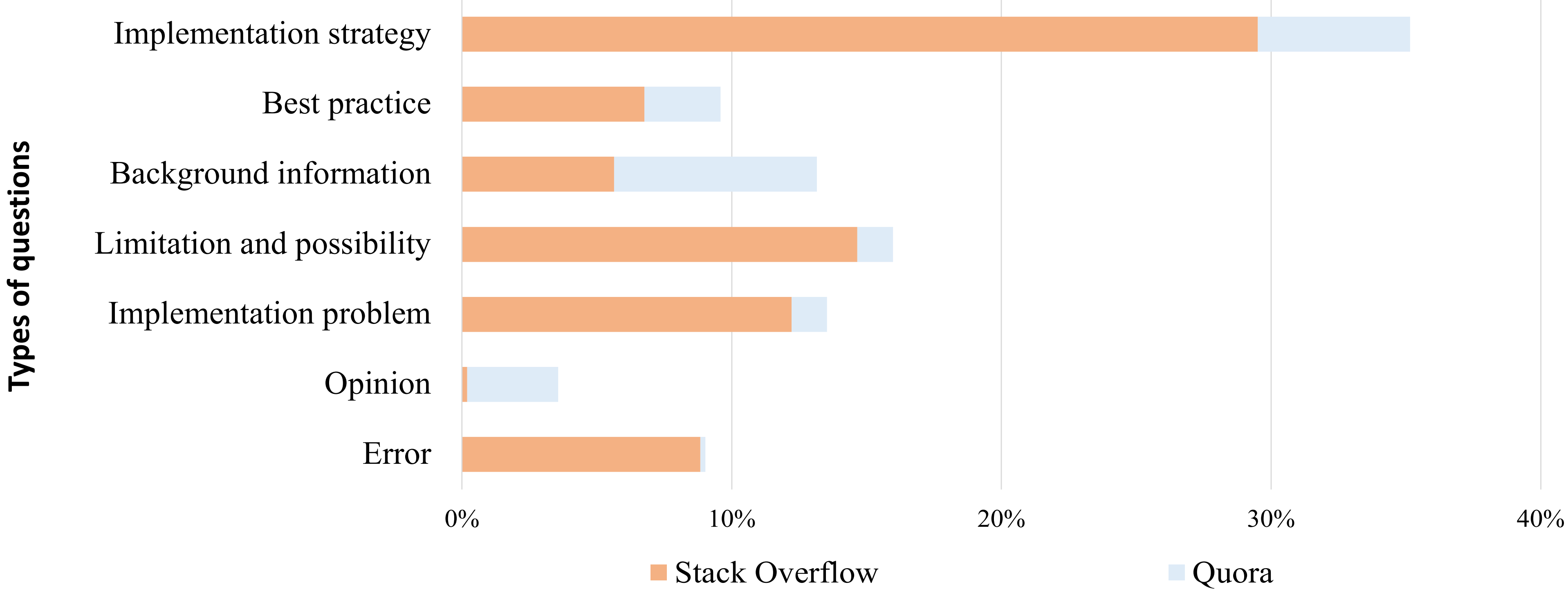


Followed by **“is it possible”** questions

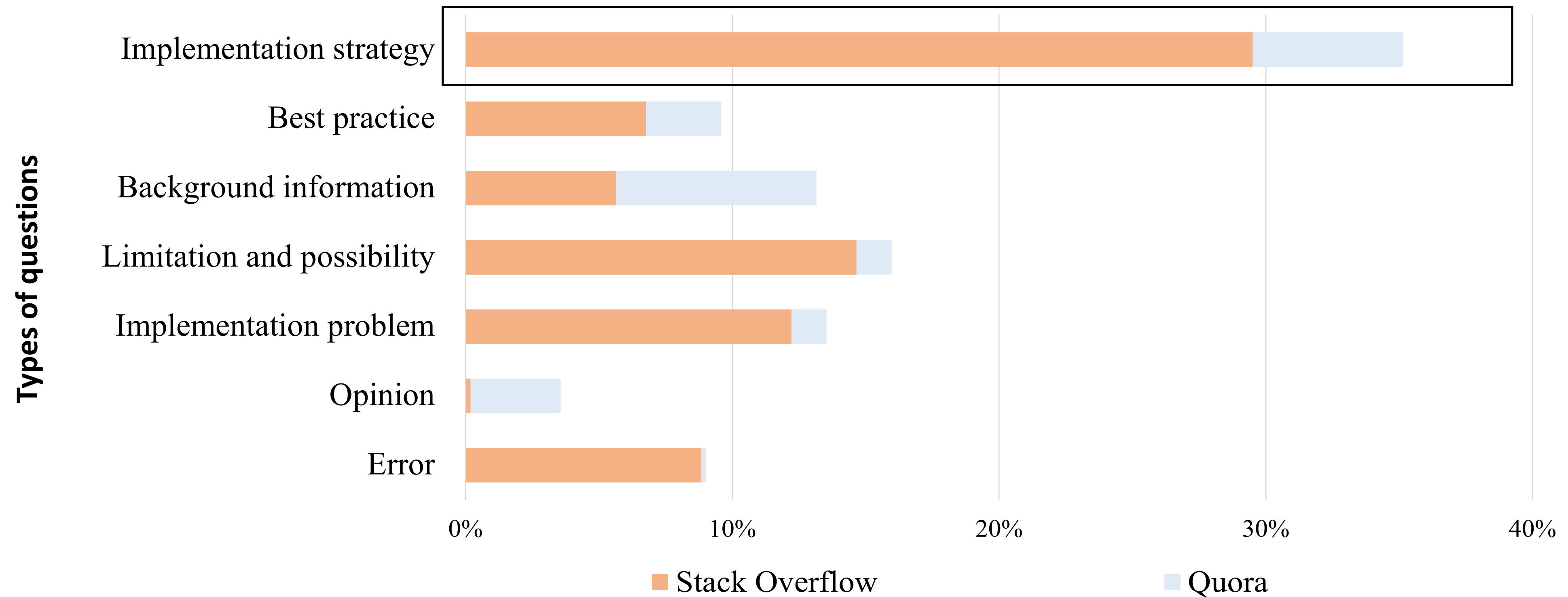
Developer concerns about comments



Developer concerns about comments

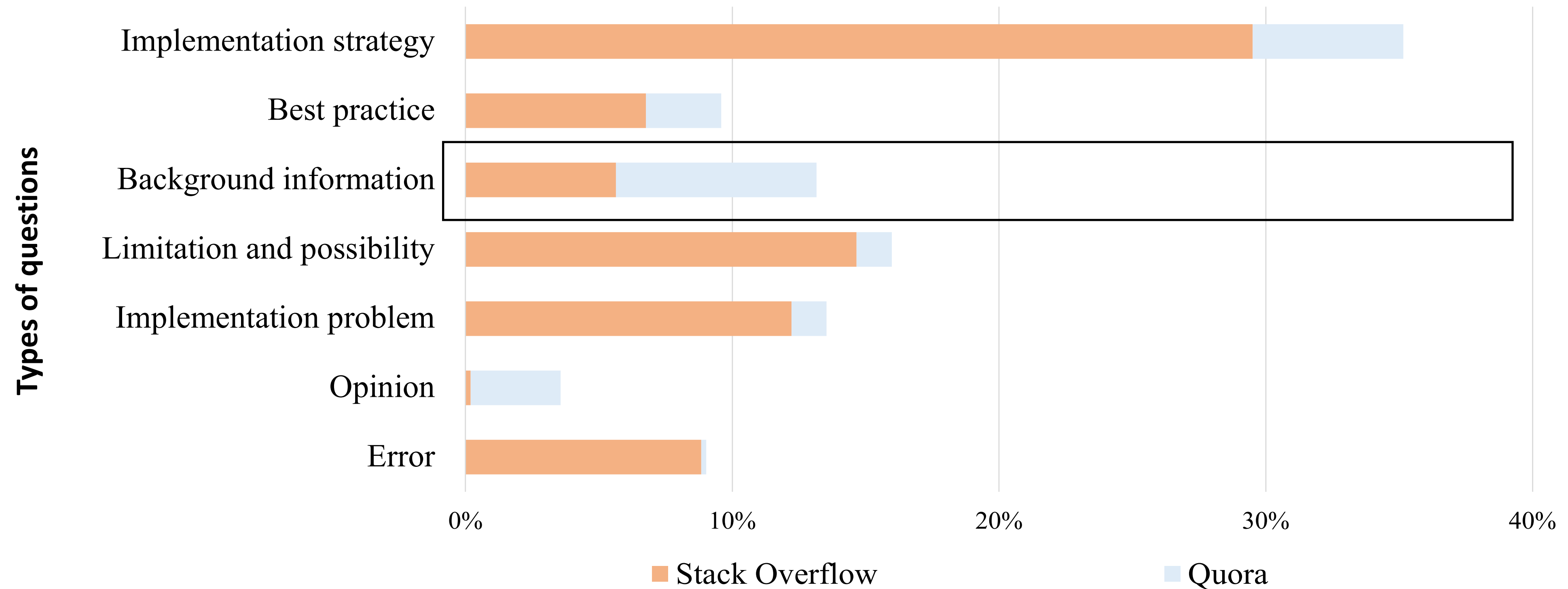


Developer concerns about comments



Developers prefer different sources to know different aspects of comments

Developer concerns about comments



Developers prefer different sources to know different aspects of comments

Syntax and Format

Project documentation

8.2%

Function documentation

23.5%

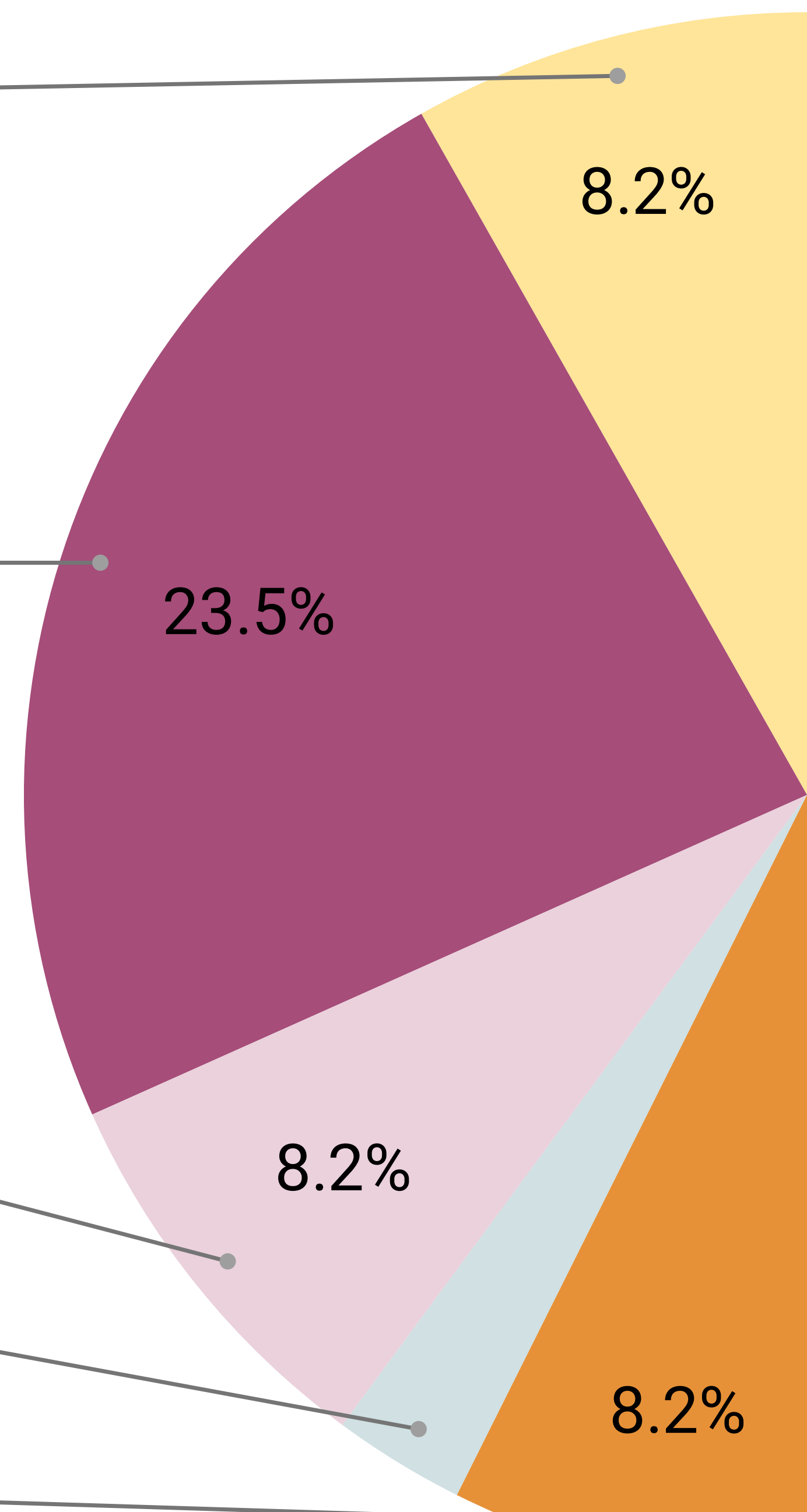
Class documentation

8.2%

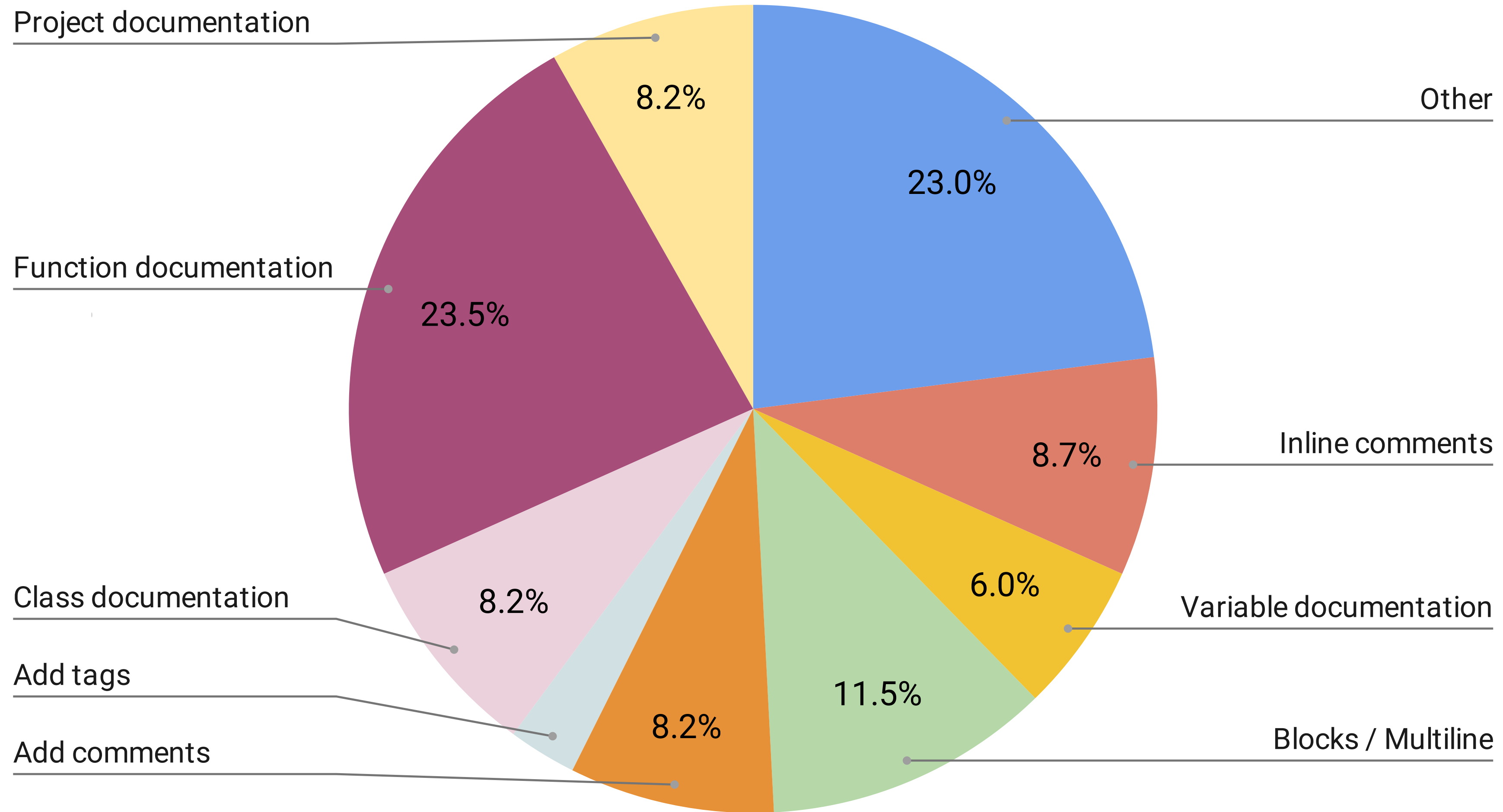
Add tags

Add comments

8.2%

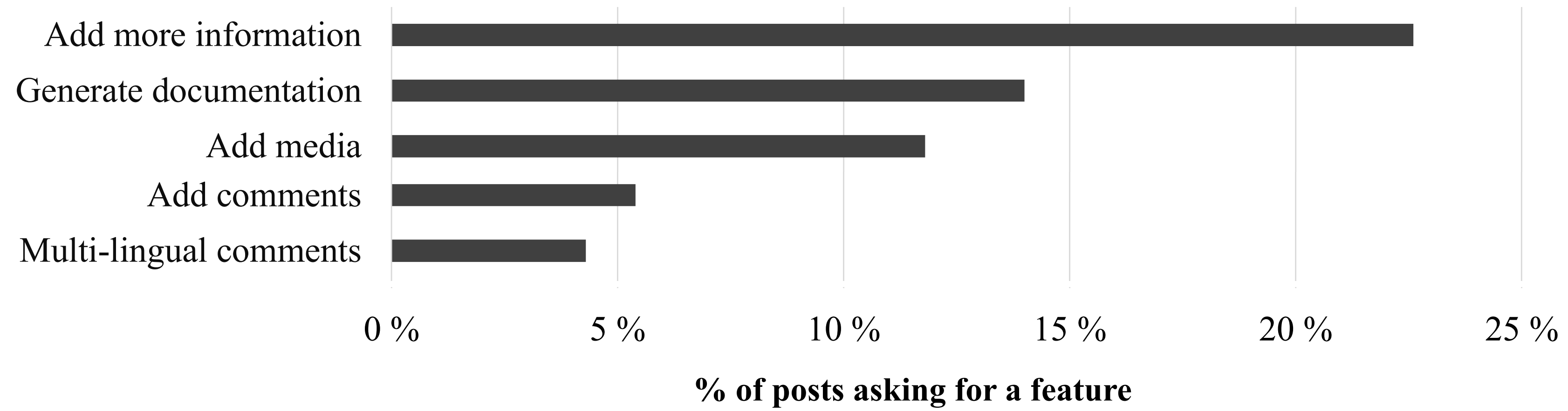


Syntax and Format

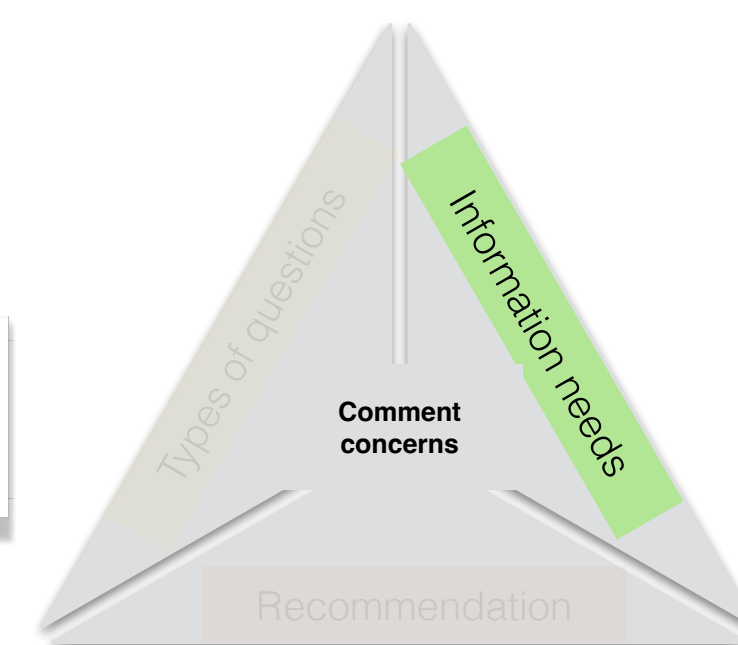


Developers want to improve **Function documentation**

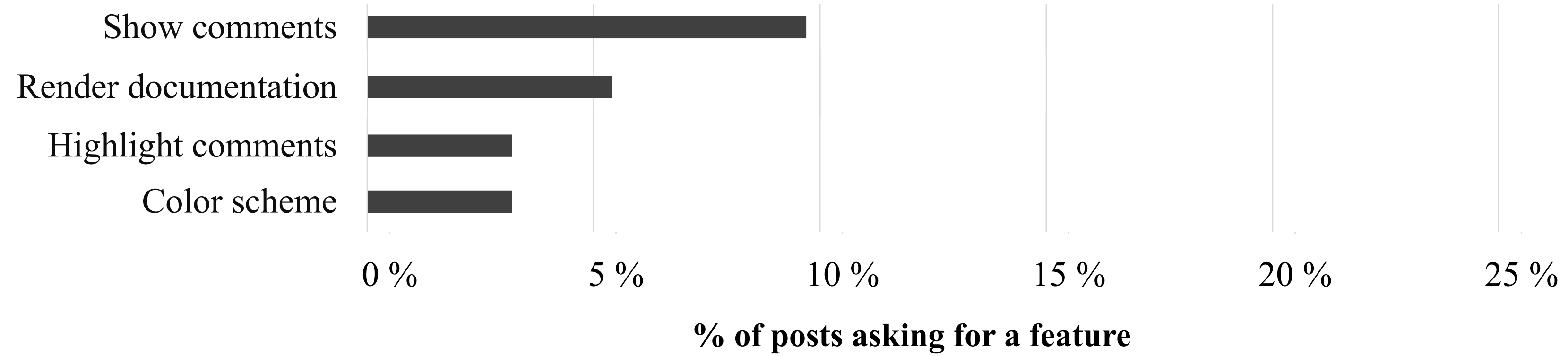
Features asked



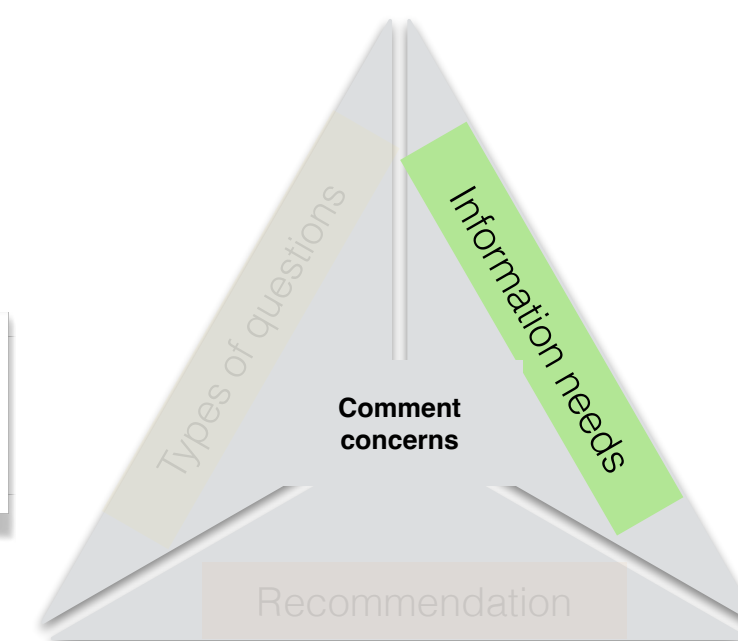
Adding information in comments



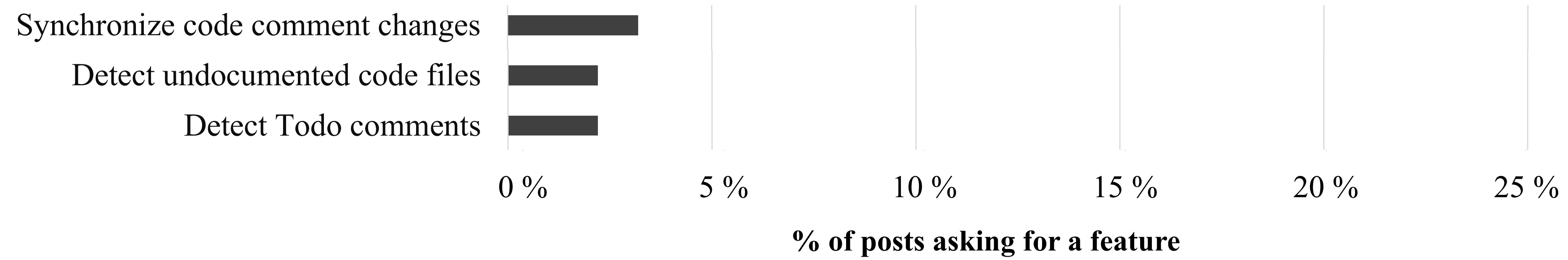
Features asked



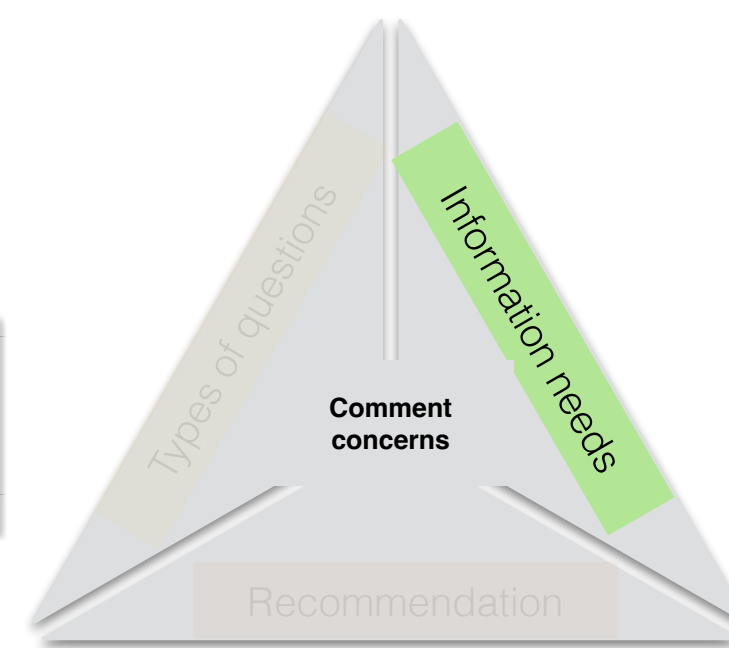
Visualizing comments or part of comments



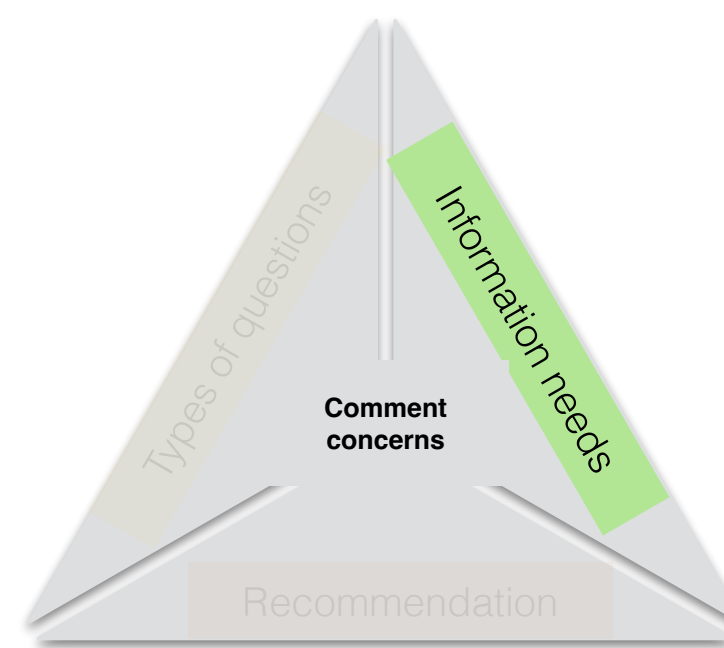
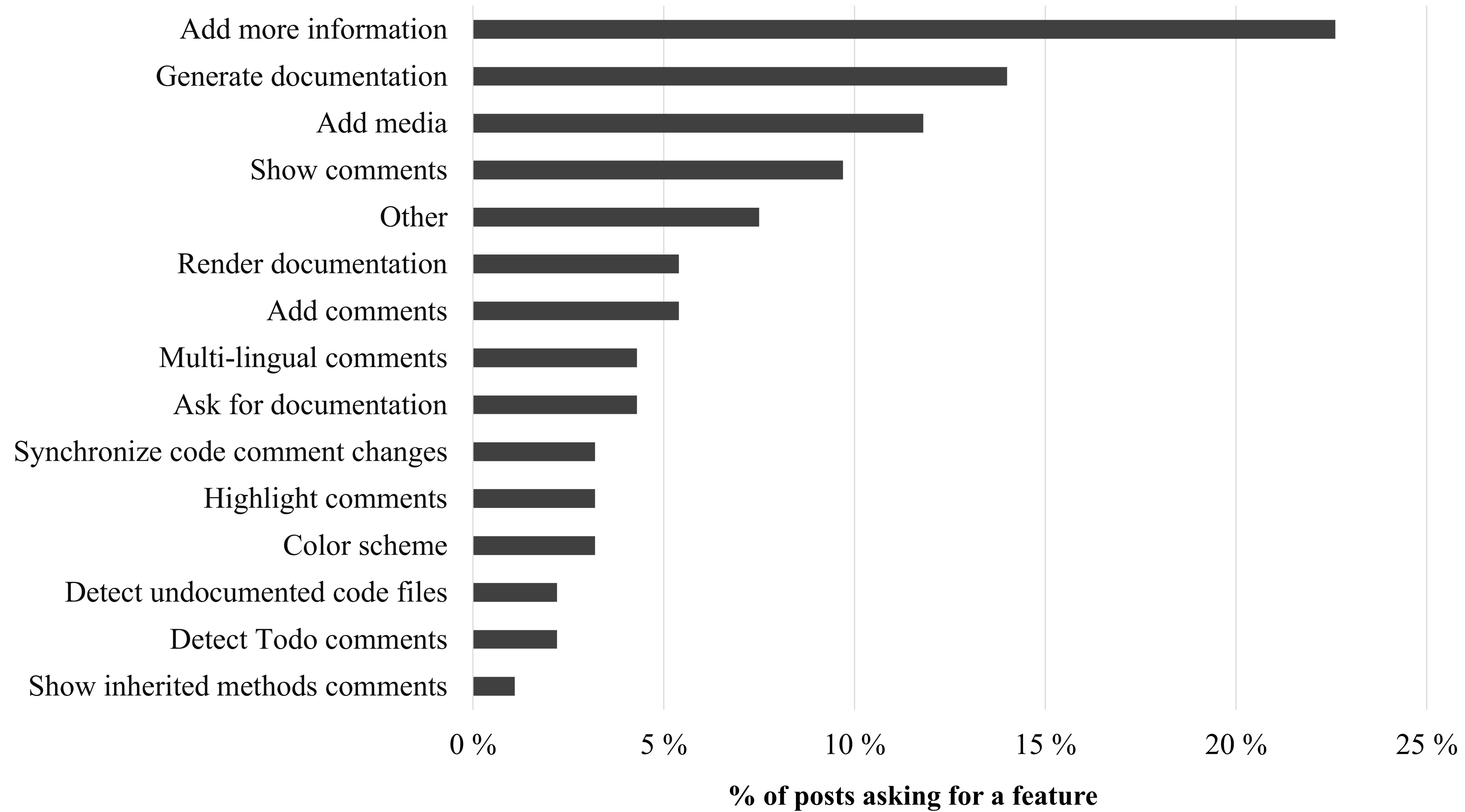
Features asked



Detecting inconsistent, incomplete, or missing comments



Features asked



Take-home messages

- Prefer different sources to know different aspects.
- Confusion about how to write comments and use various comment tools.
- Interest in embedding various information in comments but lack conventions and tools to do it automatically.

Take-home messages

- ☑ Prefer different sources to know different aspects.
- ☑ Confusion about how to write comments and use various comment tools.
- ☑ Interest in embedding various information in comments but lack conventions and tools to do it automatically.

Take-home messages

- ☑ Prefer different sources to know different aspects.
- ☑ Confusion about how to write comments and use various comment tools.
- ☑ Interest in embedding various information in comments but lack conventions and tools to do it automatically.

What Do Developers Discuss about Code Comments?

Pooja Rani*, Mathias Birrer*, Sebastiano Panichella[†], Mohammad Ghafari[‡], Oscar Nierstrasz*

*Software Composition Group, University of Bern
Bern, Switzerland
scg.unibe.ch/staff

[†] Zurich University of Applied Science (ZHAW), Switzerland
panc@zhaw.ch

[‡] University of Auckland, New Zealand
m.ghafari@auckland.ac.nz

Abstract—Code comments are important for program comprehension, development, and maintenance tasks. Given the varying standards for code comments, and their unstructured or semi-structured nature, developers get easily confused (especially novice developers) about which convention(s) to follow, or what tools to use while writing code documentation. Thus, they post related questions on external online sources to seek better commenting practices. In this paper, we analyze code comment discussions on online sources such as Stack Overflow (SO) and Quora to shed some light on the questions developers ask about commenting practices. We apply Latent Dirichlet Allocation (LDA) to identify emerging topics concerning code comments. Then we manually analyze a statistically significant sample set of posts to derive a taxonomy that provides an overview of the developer questions about commenting practices.

Our results highlight that on SO nearly 40% of the questions mention how to write or process comments in documentation tools and environments, and nearly 20% of the questions are about potential limitations and possibilities of documentation tools to add automatically and consistently more information in comments. On the other hand, on Quora, developer questions focus more on background information (35% of the questions) or asking opinions (16% of the questions) about code comments. We found that (i) not all aspects of comments are covered in coding style guidelines, e.g., how to add a specific type of information, (ii) developers need support in learning the syntax and format conventions to add various types of information in comments, and (iii) developers are interested in various automated strategies for comments such as detection of bad comments, or verify comment style automatically, but lack tool support to do that.

Index Terms—Mining online sources, Stack Overflow, Quora, Code Comment analysis, Software documentation

I. INTRODUCTION

Recent studies provide evidence that developers consider code comments to be the most important type of documentation for understanding code [1]. Code comments are written using natural language sentences, and their syntax is neither imposed by a programming language's grammar nor checked by its compiler. Consequently, developers follow various conventions in writing code comments [2]. These conventions

vary across development environments as developers embed different kinds of information in different environments [3], [4], [5]. This makes it hard to write, evaluate, and maintain the quality of comments (especially for new developers) as the software evolves [6], [7].

To help developers in writing readable, consistent, and maintainable comments, programming language communities, and large organizations, such as Google and Apache Software Foundation provide coding style guidelines that also include comment conventions [8], [9], [10], [11]. However, the availability of multiple syntactic alternatives, the freedom to adopt personalized style guidelines,¹ and the lack of tools for assessing comments, make developers confused about which commenting practice to adopt [6], or how to use a tool to write and verify comments.

To resolve potential confusion, and to learn best commenting practices, developers post questions on various Q&A forums. Stack Overflow (SO) is one of the most popular Q&A forums, enabling developers to ask questions to experts and other developers.² Barua *et al.* determined the relative popularity of a topic across all SO posts and discovered the “coding style” topic as the most popular [12]. Similarly, Quora³ is another widely adopted by developers to discuss software development aspects [13]. However, what specific problems developers report about code comments such as do they face challenges due to multiple writing conventions or development environments, or which commenting conventions experts recommend to them on these sources, is unknown.

Therefore, we analyze commenting practices discussions on SO and Quora, to shed light on these concerns. Particularly, we formulate the following research questions:

- 1) **RQ₁**: *What high-level topics do developers discuss about code comments?* Our interest is to identify high-level

¹<https://github.com/povilasb/style-guides/blob/master/cpp.rst>, accessed on Jun, 2021

²<https://www.stackoverflow.com>

³<https://www.quora.com>

P. Rani, M. Birrer, S Panichella, M Ghafari, and O Nierstrasz.
What do developers discuss about code comments?, In
Proceedings of 21st International Working Conference on
Source Code Analysis and Manipulation (SCAM), 2021



Zurich University
of Applied Sciences



Makar: A Framework for Multi-source Studies based on Unstructured Data

Mathias Birrer*, Pooja Rani*, Sebastiano Panichella†, Oscar Nierstrasz*

*Software Composition Group, University of Bern
Bern, Switzerland
scg.unibe.ch/staff

† Zurich University of Applied Science (ZHAW)
panc@zhaw.ch

Abstract—To perform various development and maintenance tasks, developers frequently seek information on various sources such as mailing lists, Stack Overflow (SO), and Quora. Researchers analyze these sources to understand developer information needs in these tasks. However, extracting and preprocessing unstructured data from various sources, building and maintaining a reusable dataset is often a time-consuming and iterative process. Additionally, the lack of tools for automating this data analysis process complicates the task to reproduce previous results or datasets.

To address these concerns we propose *Makar*, which provides various data extraction and preprocessing methods to support researchers in conducting reproducible multi-source studies. To evaluate Makar, we conduct a case study that analyzes code comment related discussions from SO, Quora, and mailing lists. Our results show that Makar is helpful for preparing reproducible datasets from multiple sources with little effort, and for identifying the relevant data to answer specific research questions in a shorter time compared to state-of-the-art tools, which is of critical importance for studies based on unstructured data. Tool webpage: <https://github.com/maethub/makar>

Index Terms—Mining developer sources, Code comments, Stack Overflow, Mailing lists

I. INTRODUCTION

As a software system continues to evolve, it becomes bigger and more complex, and developers need various kinds of information to perform activities such as adding features, or performing corrective maintenance [1]. Developers typically seek information on internal (available within IDE) or external sources such as Q&A forums,¹ Github² to satisfy their information needs as shown in Figure 1 [2].

To support developers in various activities and understand their information needs, researchers have analyzed these external sources such as Github, CVS, mailing lists, and CQA sites [3] (see Figure 1). However, extracting and preprocessing unstructured data from these sources, and maintaining the data due

¹We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project “Agile Software Assistance” (SNSF project No. 200020-181973, Feb. 1, 2019 - April 30, 2022).
²[www.stackoverflow.com](https://github.com/)
³<https://github.com/>

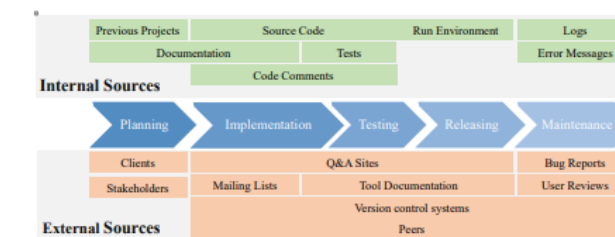


Fig. 1. Developers seek various sources during software development

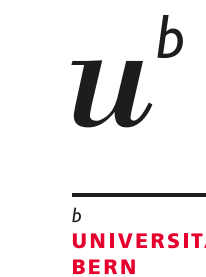
to lack of automated techniques pose various challenges in conducting reproducible studies [4], [5], [3]. To gain a deeper understanding of these challenges, we surveyed the literature that focuses on studying developers information needs from different external sources (see section II).

Prior works have raised and identified the crucial factors affecting the reproducibility of the mining studies such as data retrieval methodology, data processing steps, or dataset availability [6], [5], [4]. Chen *et al.* pointed out that 50% of articles do not report whether word stemming, a common text preprocessing step, is used or not [4]. Amann *et al.* pointed out that only 29% of the mining studies made their dataset available [5]. As a consequence, more tools and techniques are required to enable the preprocessing and analysis of multi-source studies to facilitate their replicability.

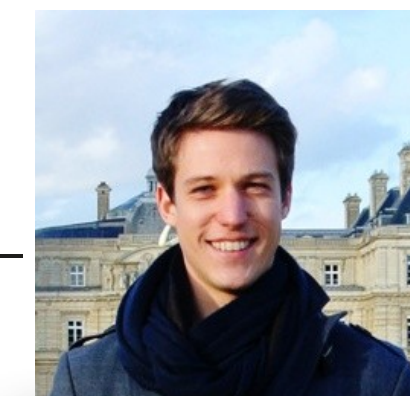
To address these concerns, we propose *Makar*, a tool that leverages popular data retrieval, processing, and handling techniques to support researchers in conducting reproducible studies. We establish its requirements based on the surveyed studies. To evaluate Makar, we conduct a case study that analyzes code comment related discussions from SO, Quora, and mailing lists. Thus the contribution of this paper is three-fold:

- We present the challenges researchers face in mining and analyzing the unstructured data from the external sources.
- We present Makar, a tool to support researchers in conducting multi-source and reproducible empirical studies.
- We report the state-of-art tools comparison to Makar.

P. Rani, M. Birrer, S. Panichella, and O. Nierstrasz. **Makar: A Framework for Multi-source Studies based on Unstructured Data**, In proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021



Zurich University of Applied Sciences

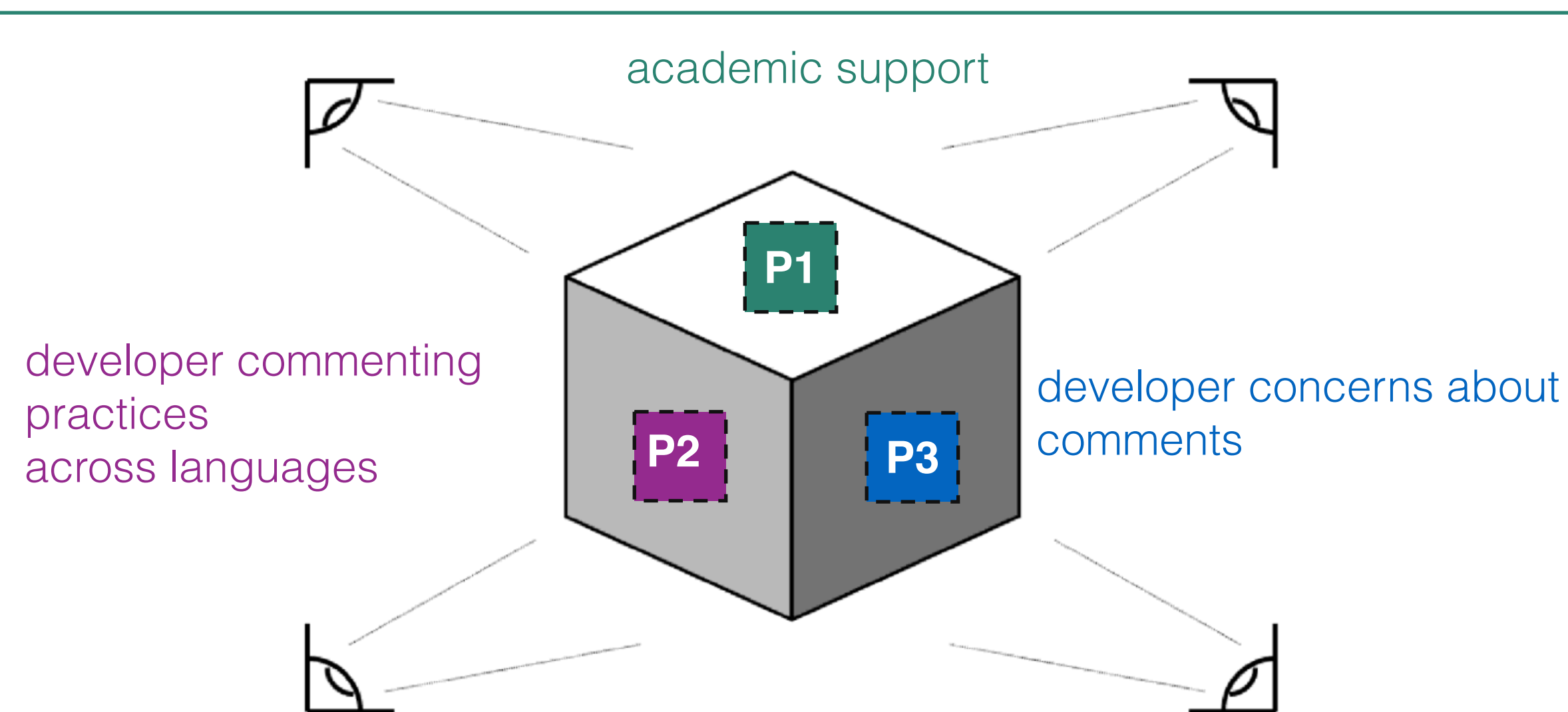


Hosted on

<https://github.com/maethub/makar>

Conclusions

- ❖ Studies focus on Java
- ❖ 21 quality attributes for comments
- ❖ Many attributes are assessed manually



- ❖ 16 different types of information in class comments
- ❖ Recurrent patterns help in identifying information
- ❖ Developers do not adopt various conventions

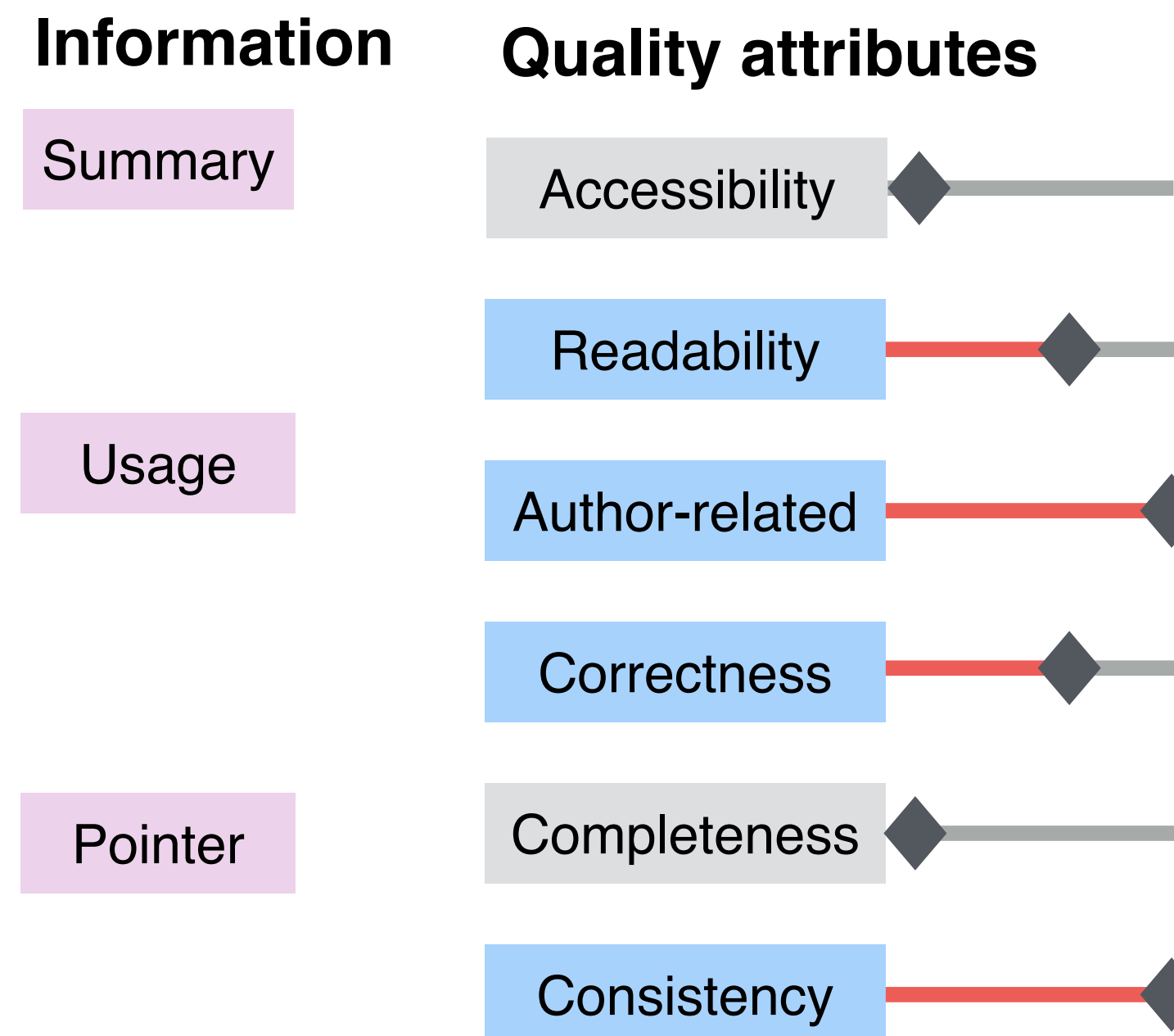
- ❖ Add more information to comments
- ❖ Visualize comments
- ❖ Detect inconsistent, incomplete, missing comments

Future work

- 📌 Assessing specific required information from comments

Assessing specific required information

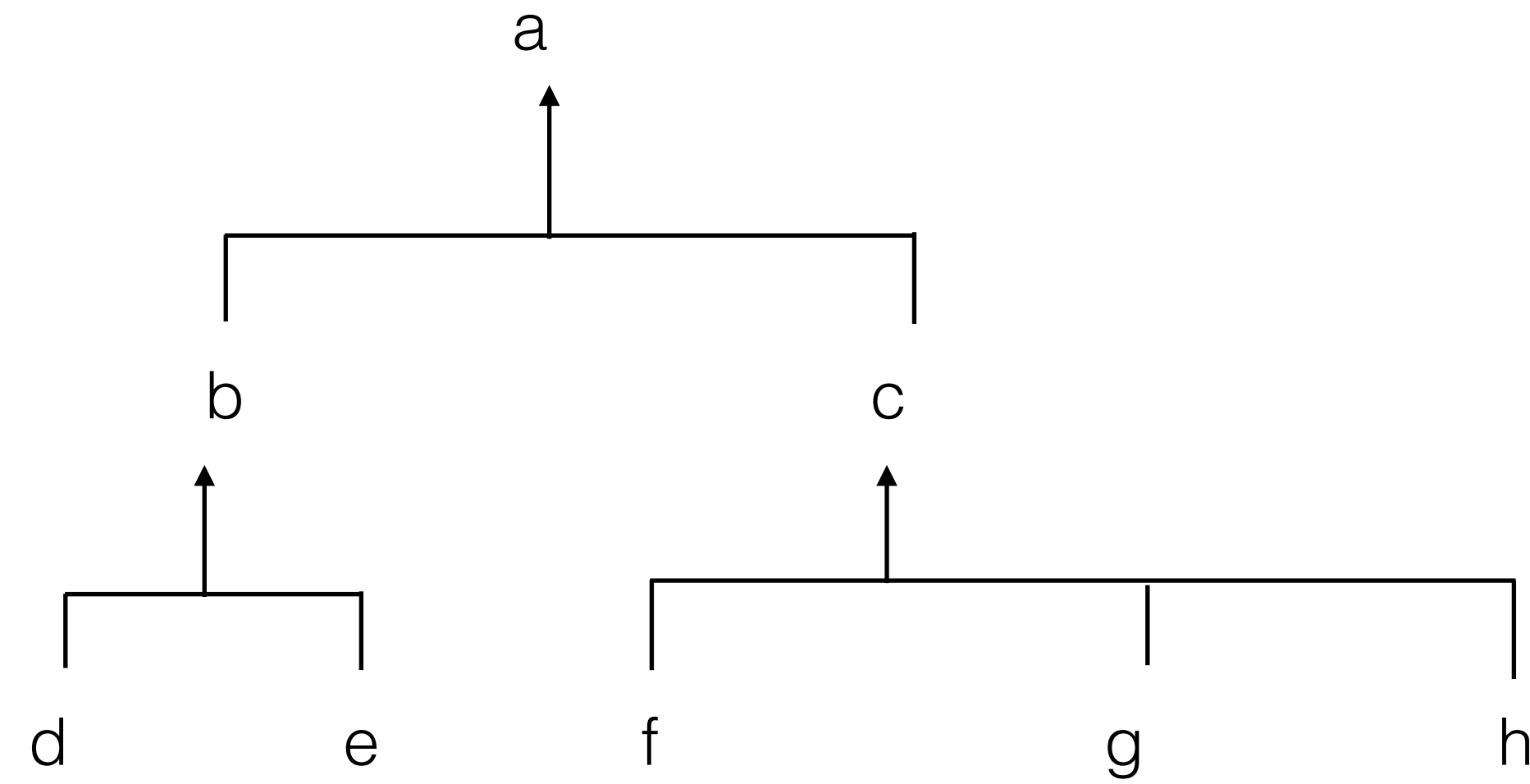
```
/**
 * A class representing a window on the screen.
 *
 * For example:
 * <pre>
 *   Window win = new Window(parent);
 *   win.show();
 * </pre>
 *
 * @author Sami Shaio
 * @version 1.13, 06/08/06
 * @see java.awt.BaseWindow
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}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



Future work

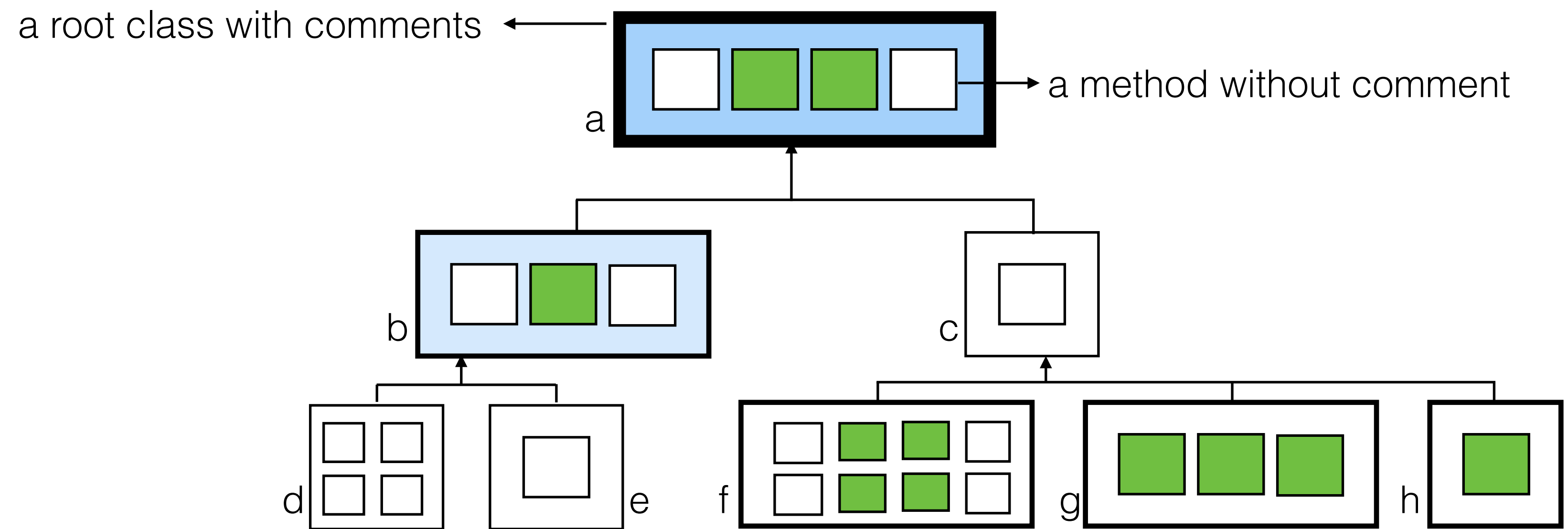
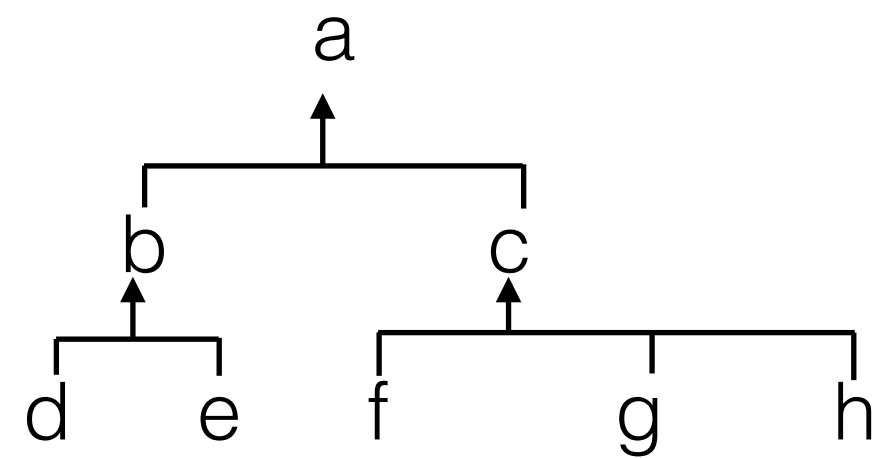
- 📌 Assessing specific required information from comments
- 📌 Visualizing documentation effort

Visualizing documentation effort



Visualizing a project hierarchy for documentation effort

Visualizing documentation effort



Visualizing a project hierarchy for documentation effort

Future work

- 📌 Assessing specific required information from comments
- 📌 Visualizing documentation effort
- 📌 Analyzing support of documentation tools to comments

Future work

- 📌 Assessing specific required information from comments
- 📌 Visualizing documentation effort
- 📌 Analyzing support of documentation tools to comments
- 📌 Speculative Analysis of comment quality

Future work

- 📌 Assessing specific required information from comments
- 📌 Visualizing commenting effort
- 📌 Analyzing support of documentation tools to comments
- 📌 Speculative Analysis of comment quality

Summary

Code 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
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    --
}
    
```


Trustworthy form of documentation

High-quality comments support developers

7

Challenges

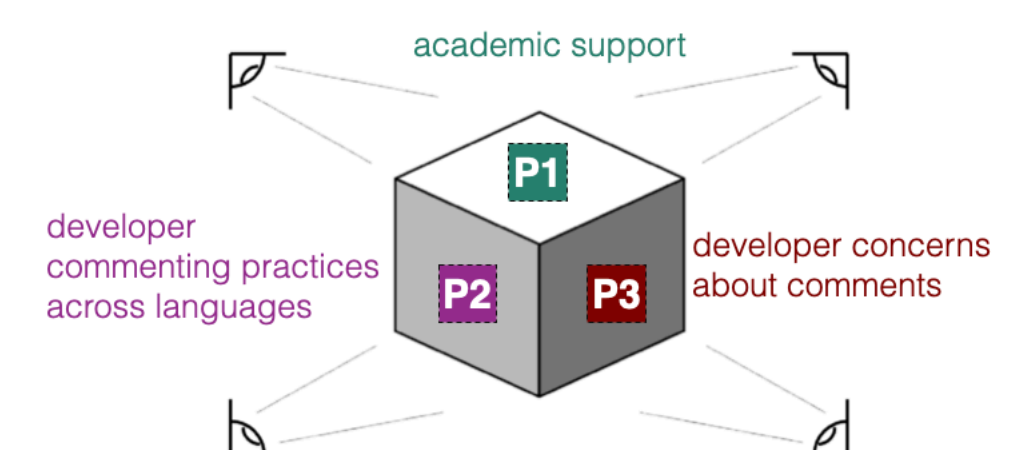
- No standard definition of comment quality
- No strict syntax and structure conventions
- Lack of quality assessment tools



All these makes **quality assessment a non-trivial problem**

14

We define three perspectives

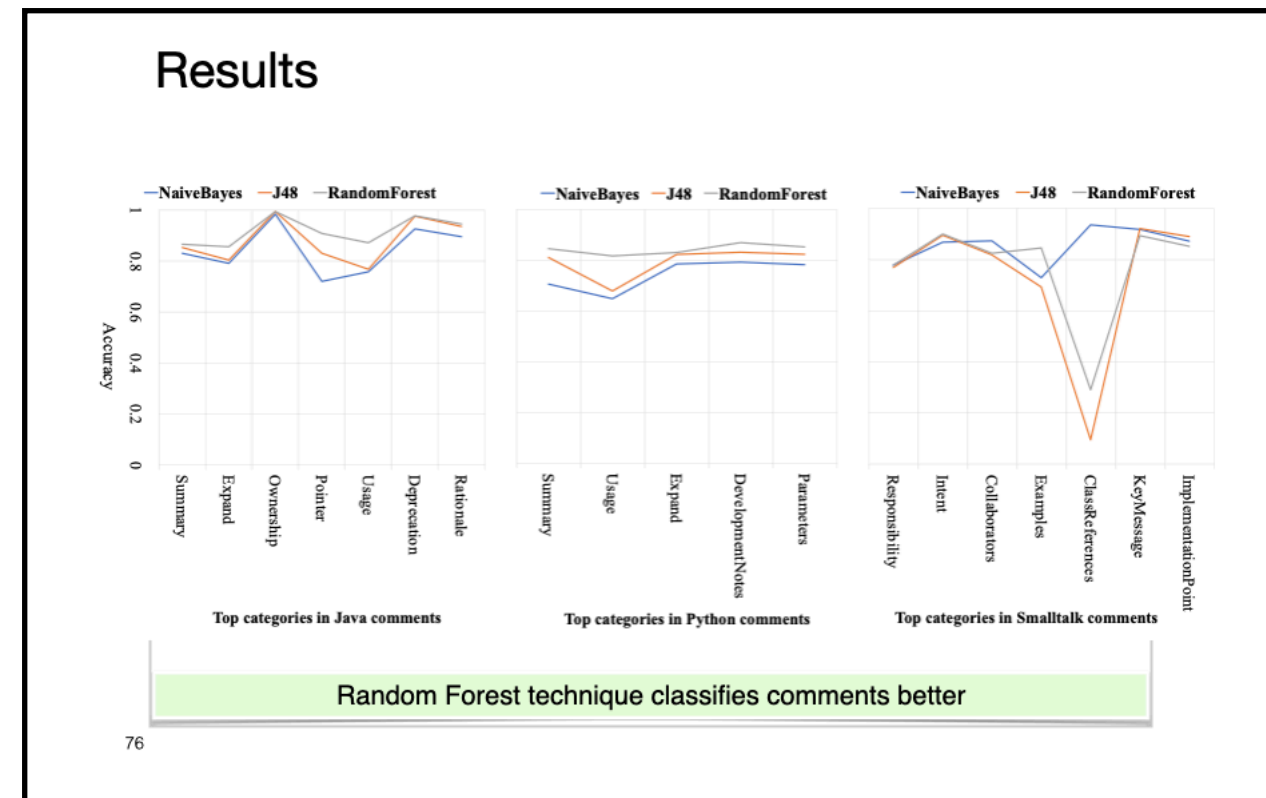
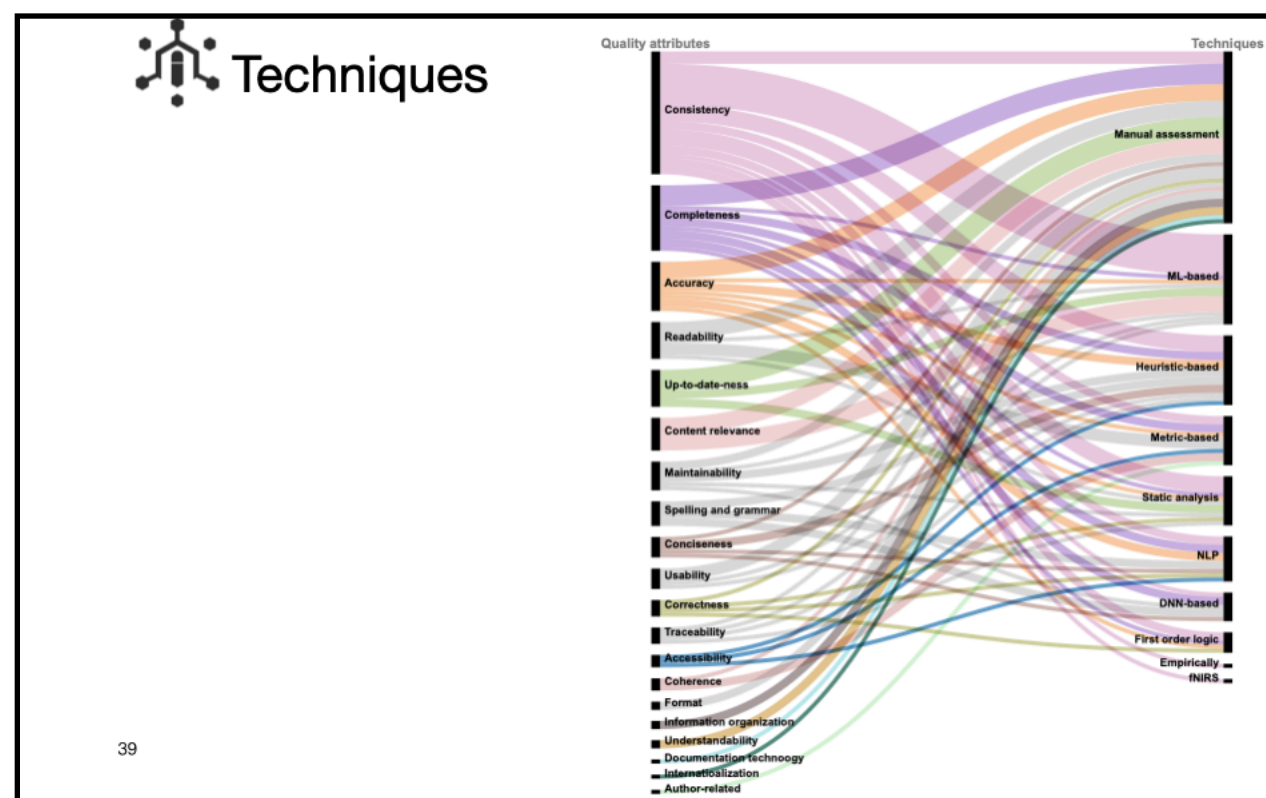


academic support

developer commenting practices across languages

developer concerns about comments

19



Future work

- Assessing specific required information from comments
- Visualizing commenting effort
- Anlyzing support of documentation tools to comments
- Speculative Analysis of comment quality

161

Thank you

–Pooja Rani

Backup slides

Python class comments

```
class OneHotCategorical(Distribution):
    r"""
    Creates a one-hot categorical distribution parameterized by :attr:`probs` or
    :attr:`logits`.

    Samples are one-hot coded vectors of size ``probs.size(-1)``.

    .. note:: The `probs` argument must be non-negative, finite and have a non-zero sum,
               and it will be normalized to sum to 1 along the last dimension. :attr:`probs`
               will return this normalized value.
               The `logits` argument will be interpreted as unnormalized log probabilities
               and can therefore be any real number. It will likewise be normalized so that
               the resulting probabilities sum to 1 along the last dimension. :attr:`logits`
               will return this normalized value.

    See also: :func:`torch.distributions.Categorical` for specifications of
    :attr:`probs` and :attr:`logits`.

    Example::

        >>> m = OneHotCategorical(torch.tensor([ 0.25, 0.25, 0.25, 0.25 ]))
        >>> m.sample() # equal probability of 0, 1, 2, 3
        tensor([ 0.,  0.,  0.,  1.])

    Args:
        probs (Tensor): event probabilities
        logits (Tensor): event log probabilities (unnormalized)
    """
```

} Summary

} Expand

} Development notes,
Warnings

} Links

} Usage

} Parameters

Comments of multi-languages

```
/**
 * 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, 06/0
 * @see java.awt.BaseW
 */
class Window extends
..
}
```

```
class OneHotCategorical(Distribution):
r"""
Creates a one-hot categorical distribution parameterized by :attr:`probs` or
:attr:`logits`.

Samples are one-hot coded vectors of size ``probs.size(-1)``.

.. note:: The `probs` argument must be non-negative, finite and have a non-zero sum.

See also:
:attr:`pr
Example:
>>> m
>>> m
tensc
Args:
probs
logit
"""
```

? Comment x

I represent a message to be scheduled by the WorldState.

For example, you can see me in action with the following example which print 'alarm test' on Transcript one second after evaluating the code:

```
Transcript open.
MorphicUIManager currentWorld
  addAlarm: #show:
  withArguments: #('alarm test')
  for: Transcript
  at: (Time millisecondClockValue + 1000).
```

*** Note ***
Compared to doing:
[(Delay forMilliseconds: 1000) wait. Transcript show: 'alarm test'] forkAt: Processor activeProcess priority +1.

the alarm system has several distinctions:

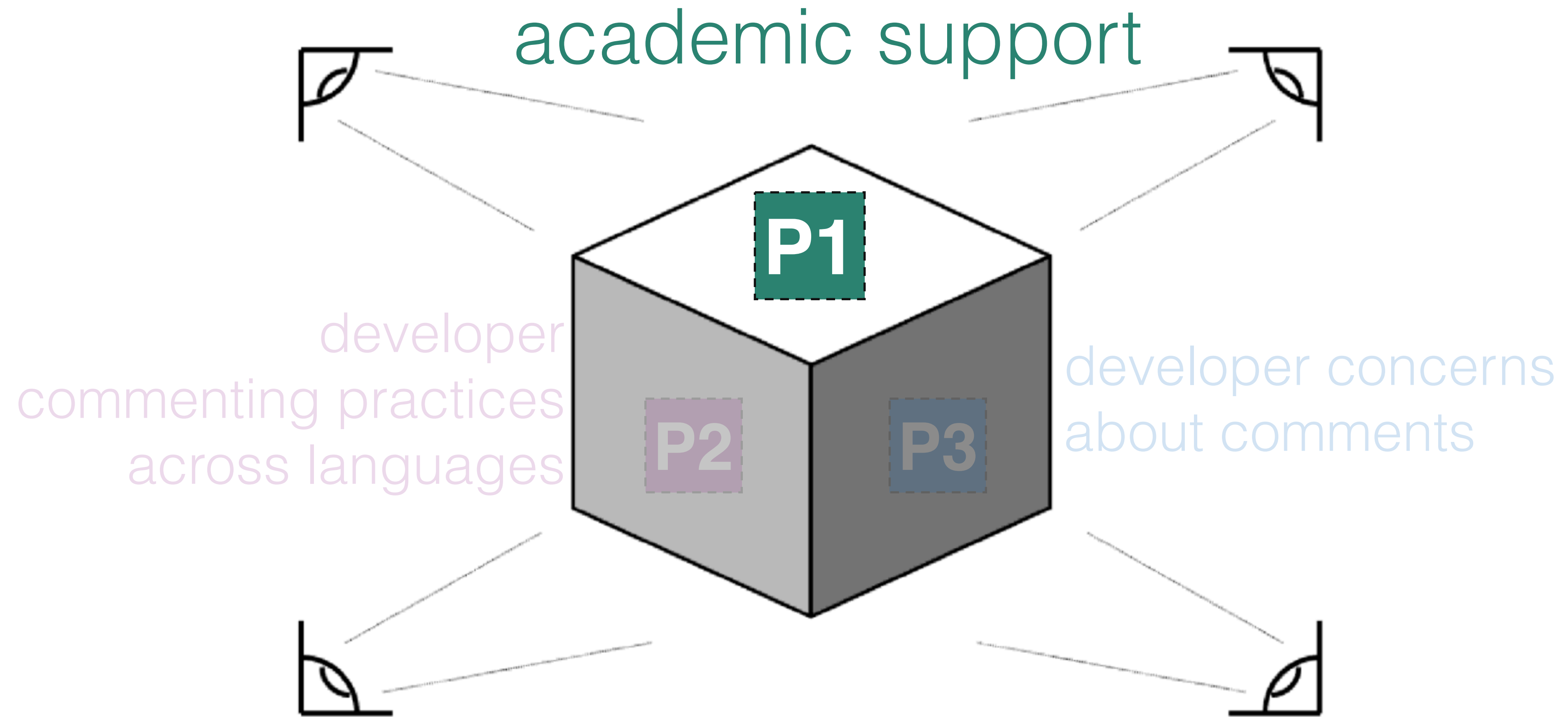
- Runs with the step refresh rate resolution.
- Alarms only run for the active world. (Unless a non-standard scheduler is in use)
- Alarms with the same scheduled time are guaranteed to be executed in the order they were added

Smalltalk class comment

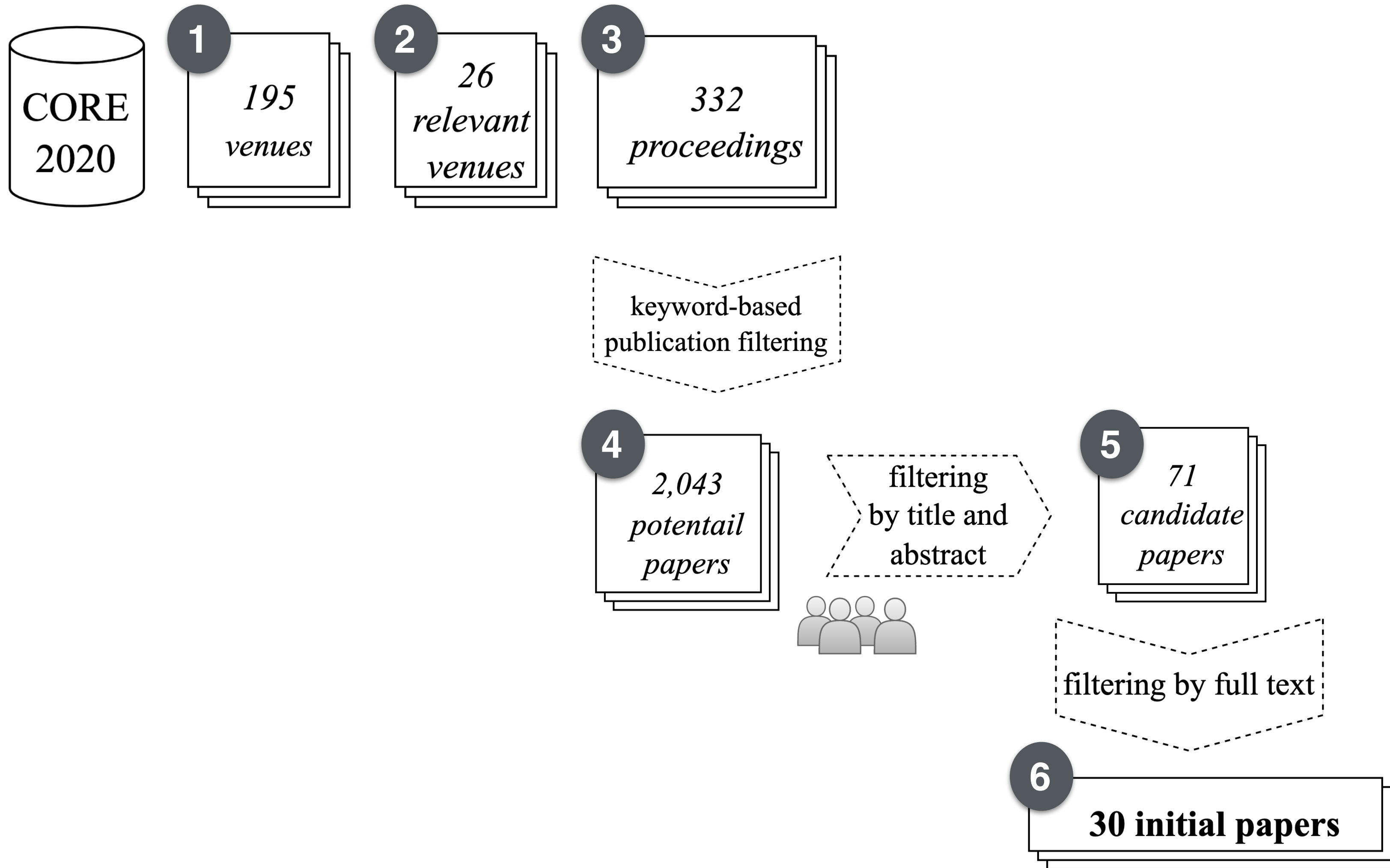
Thesis statement

“Understanding the specification of high-quality comments to build effective assessment tools requires **a multi-perspective** view of the comments. The view can be approached by analysing **(P1) the academic support** for comment quality assessment, **(P2) developer commenting practices** across languages, and **(P3) their concerns about comments.**”

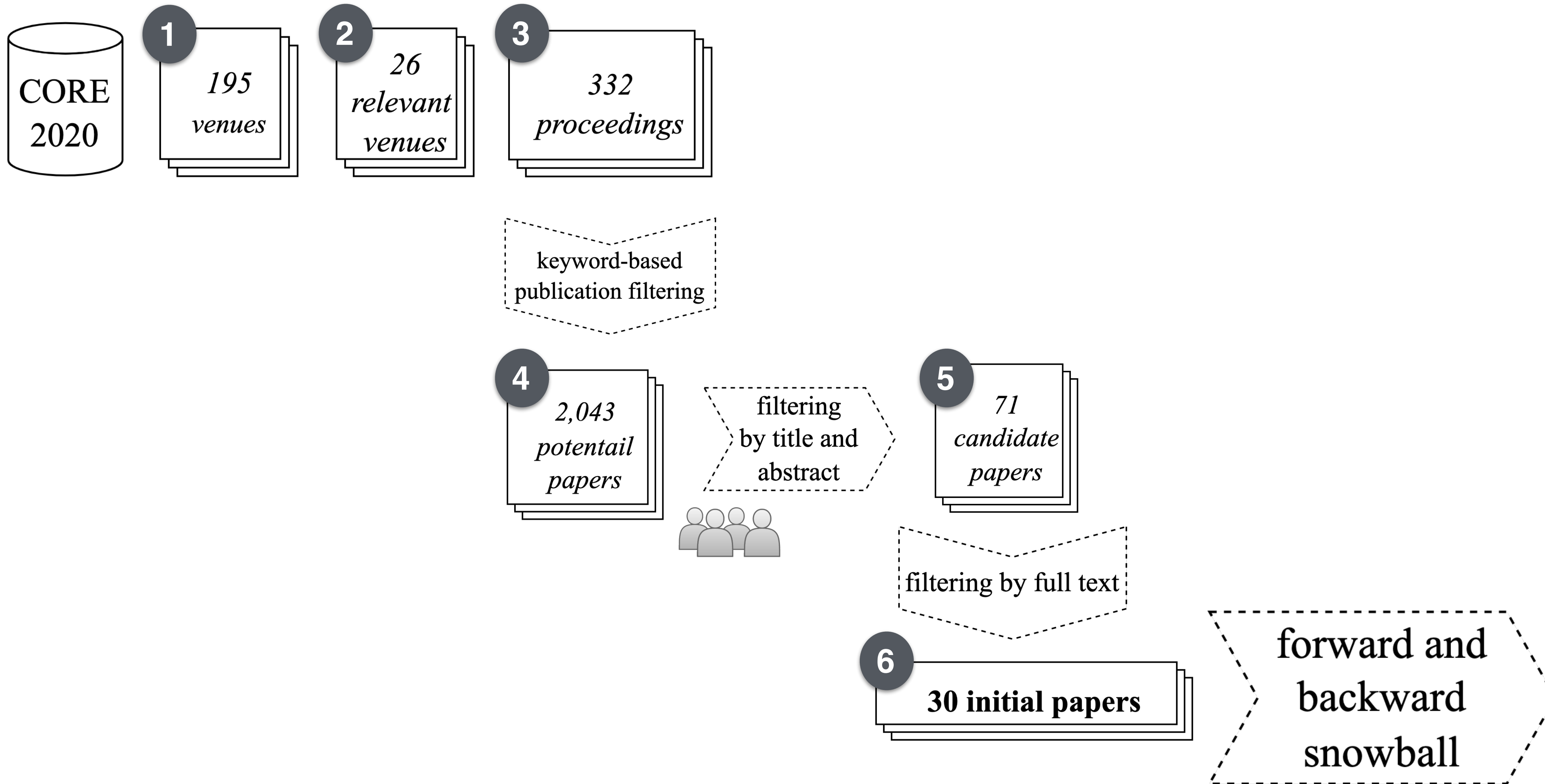
P1: academic support



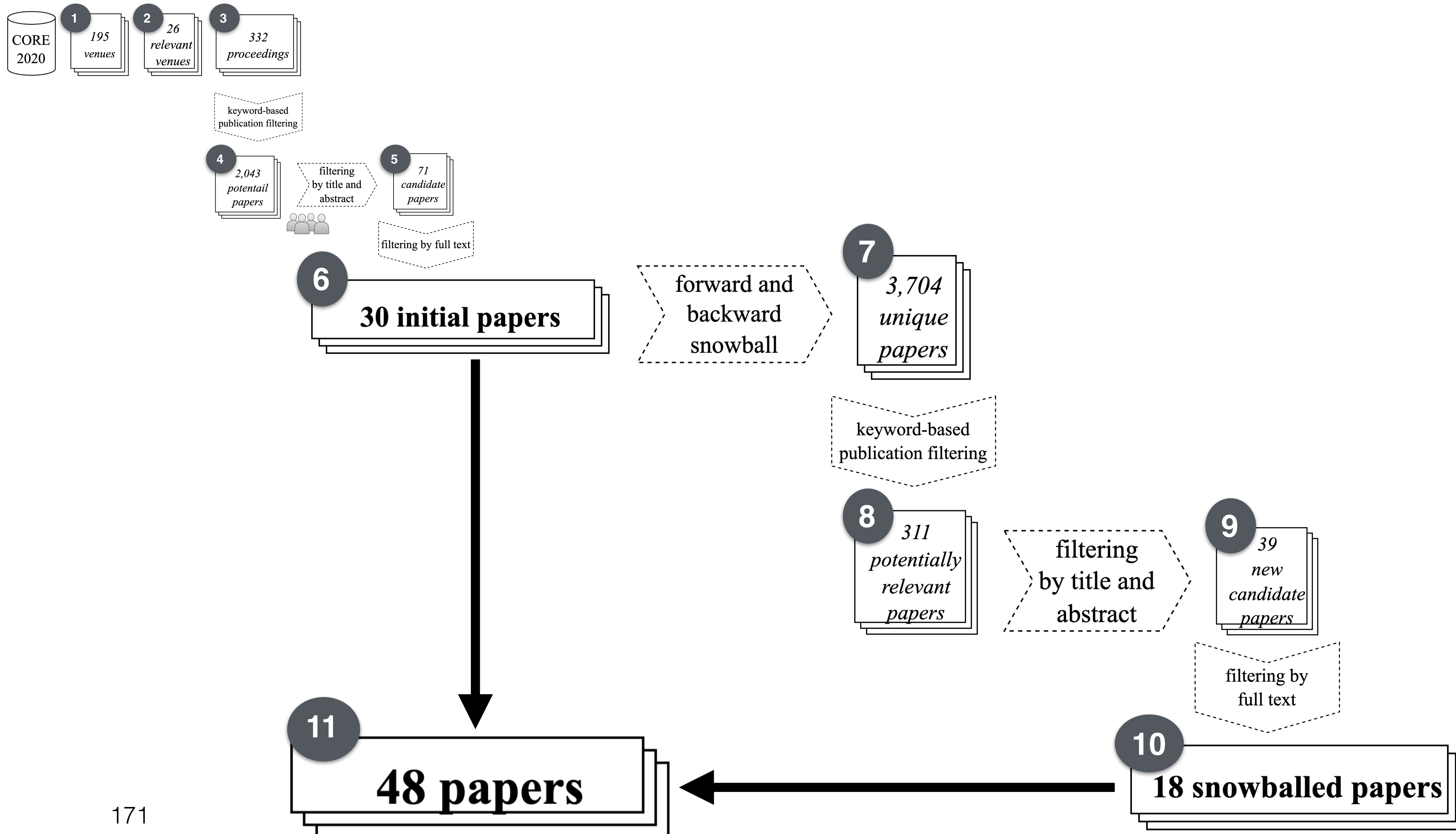
Methodology

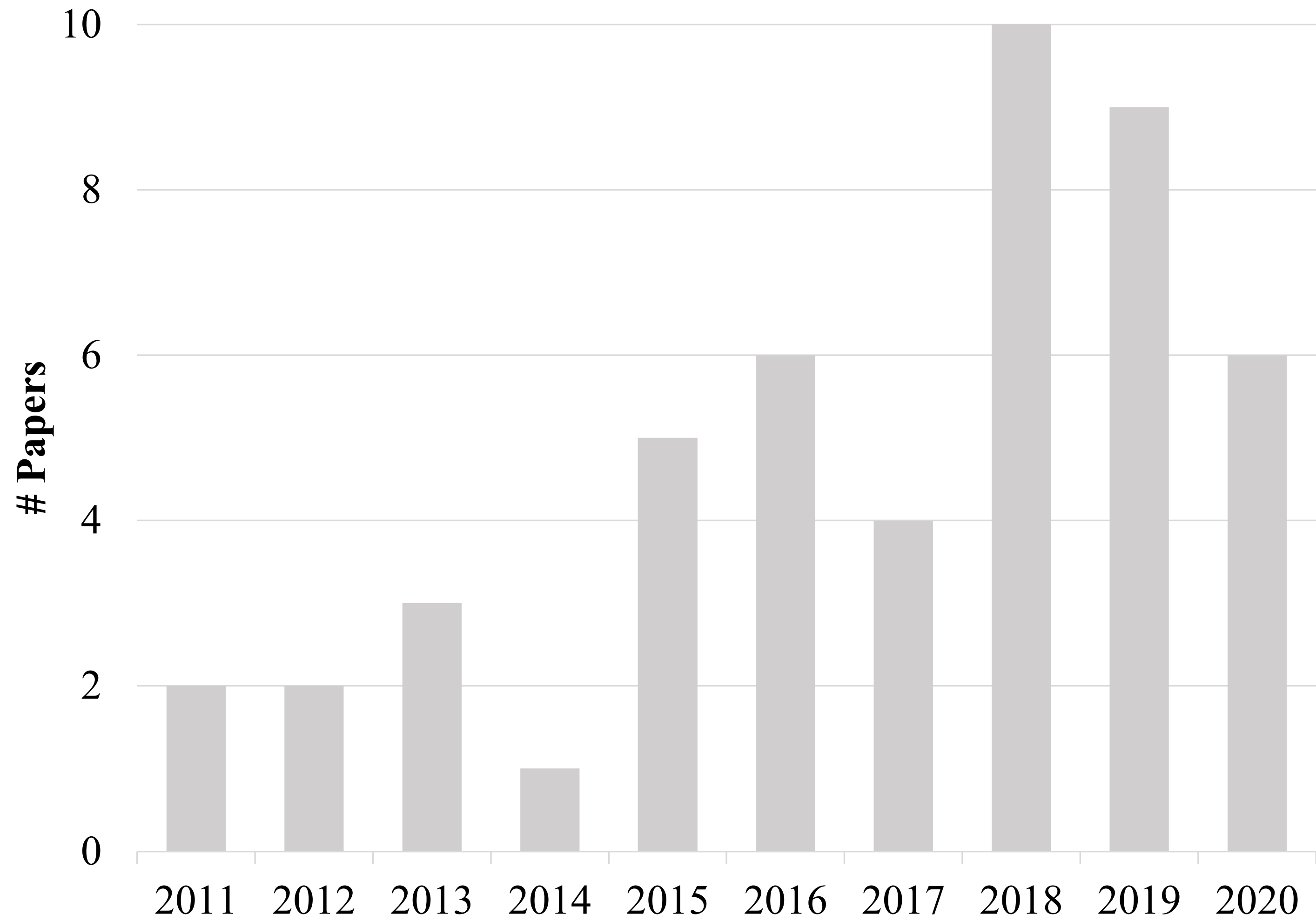


Methodology



Methodology





48 papers over years

Dimensions analyzed



Comment types

method comments, inline comments



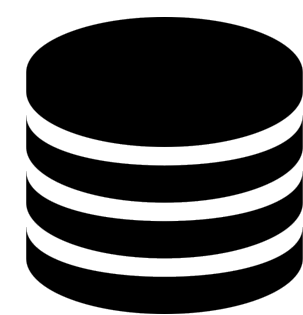
Quality attributes

consistency, completeness



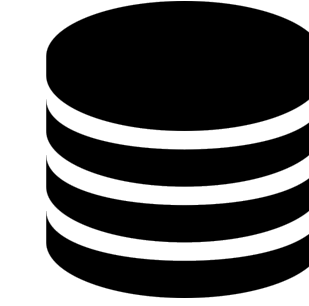
Techniques

heuristic-based, machine learning-based

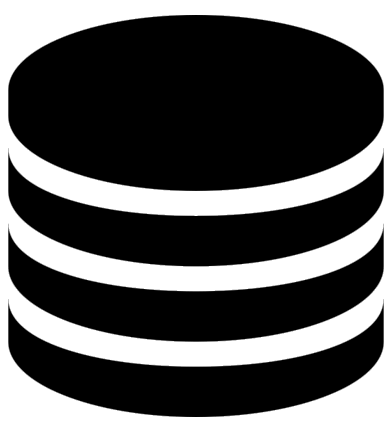


Data availability

tool, dataset

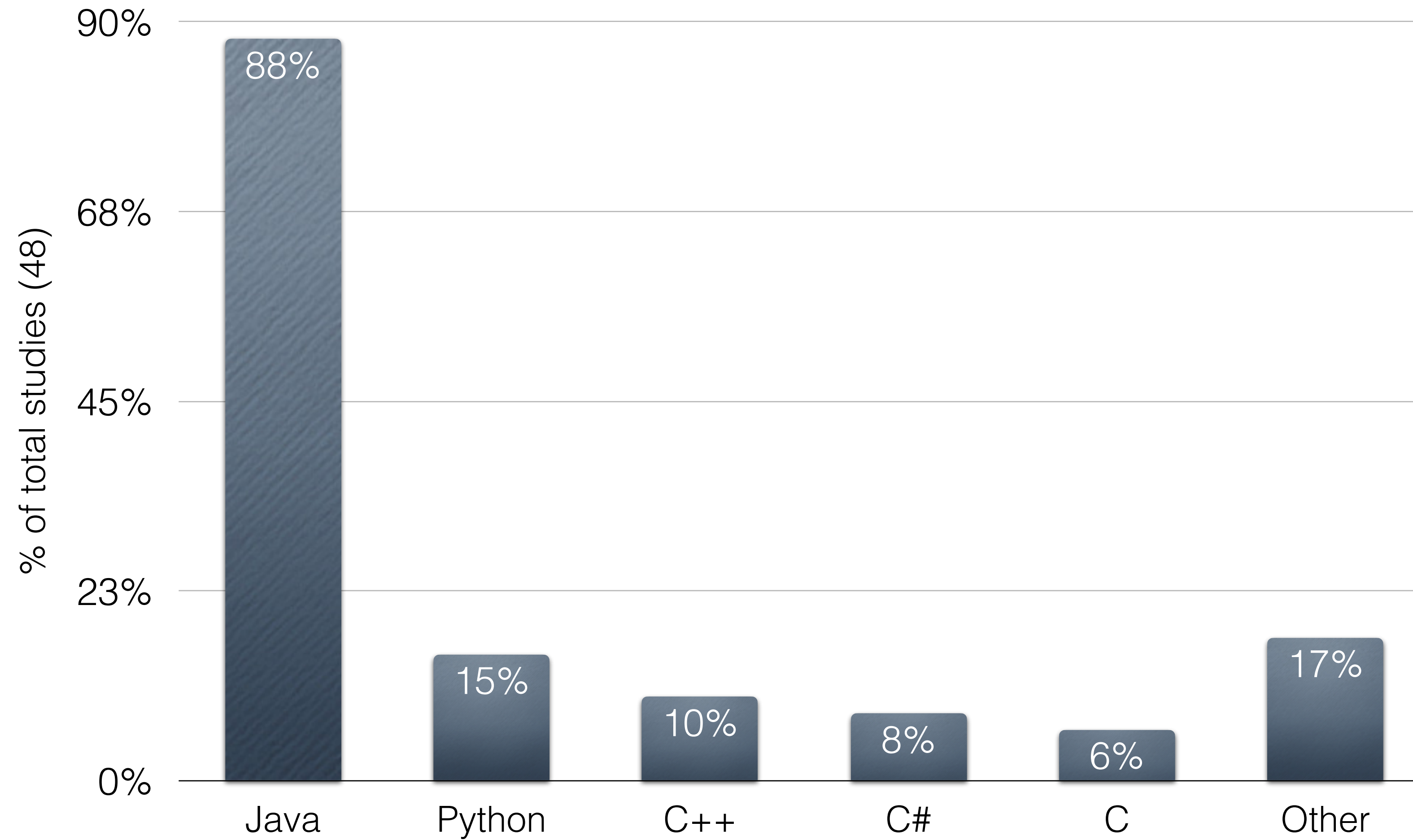


Data availability

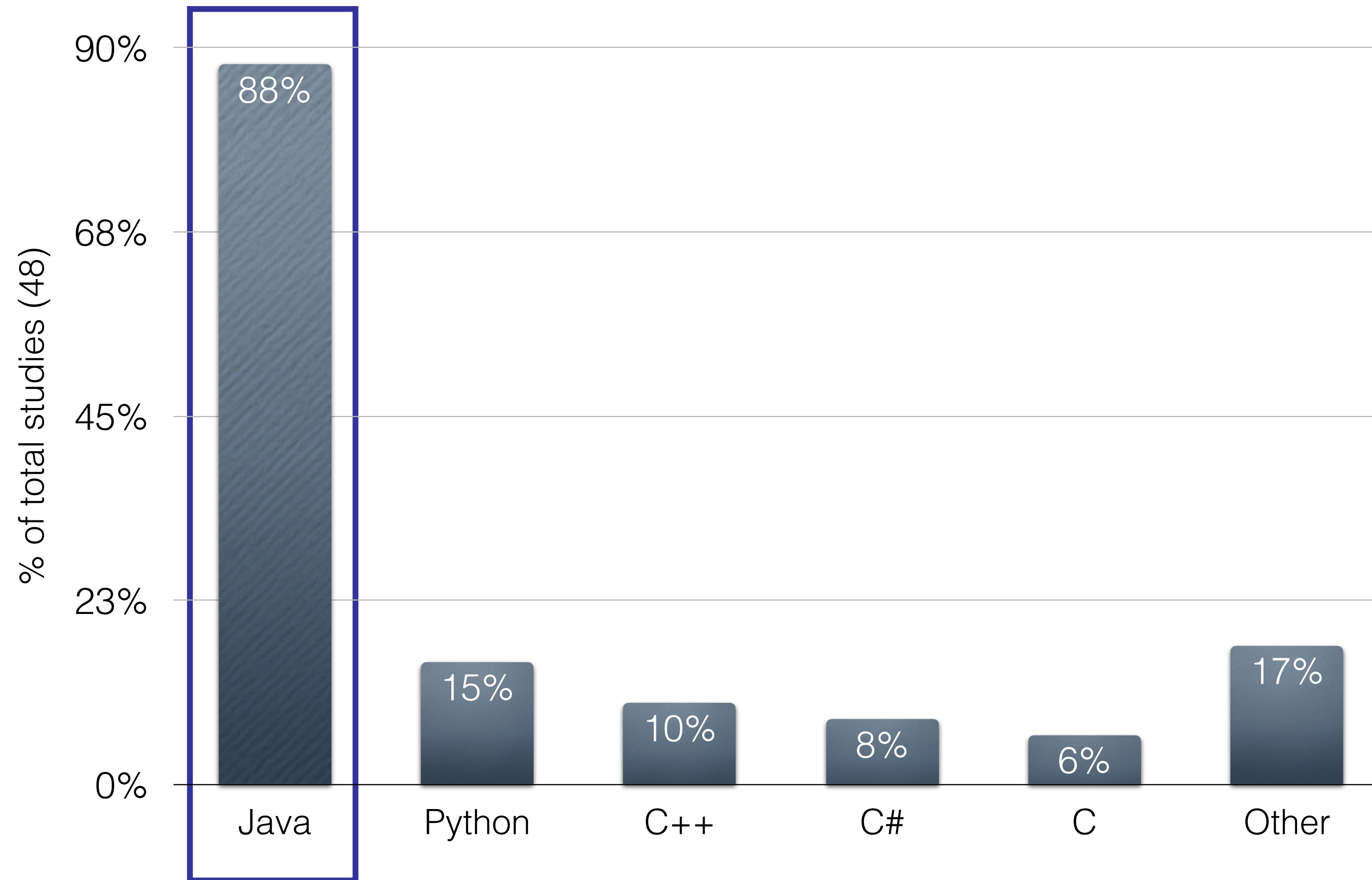


Nearly 50% of the studies still lack in the replicability dimension, as their respective dataset or tool is often not publicly accessible.

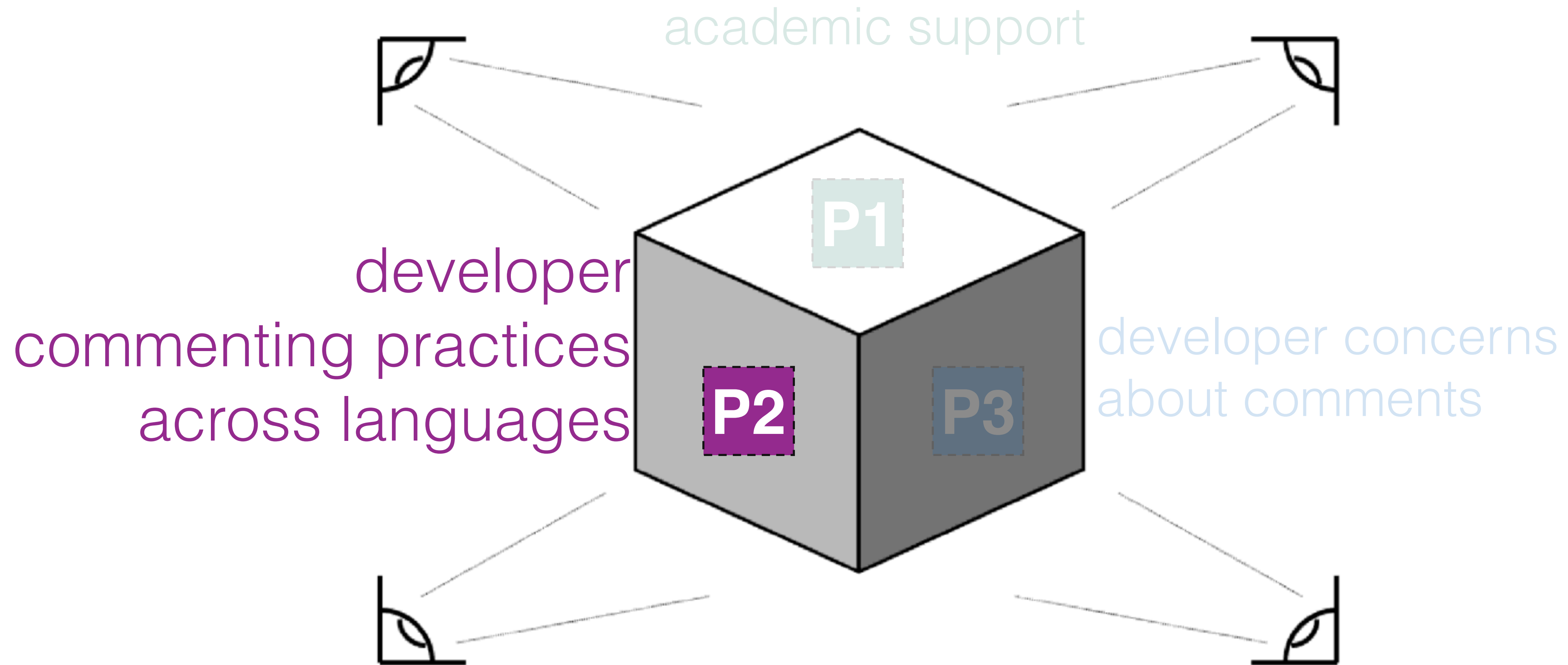
Comments analyzed of languages



Comments analyzed of languages



P2: class commenting practices



Comment taxonomies in Java and Python

Java

Python

Summary

Expand

Pointer

Rationale

Usage

Deprecation



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

Pascarella et al., 2017

Classifying Python Code Comments Based on Supervised Learning

Jingyi Zhang¹, Lei Xu^{2(✉)}, and Yanhui Li²

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, yanhui11}@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.

Zhang et al., 2018

Summary

Expand

Links

Development notes

Usage



17 types of information in code comments

11 types of information in code comments

Java

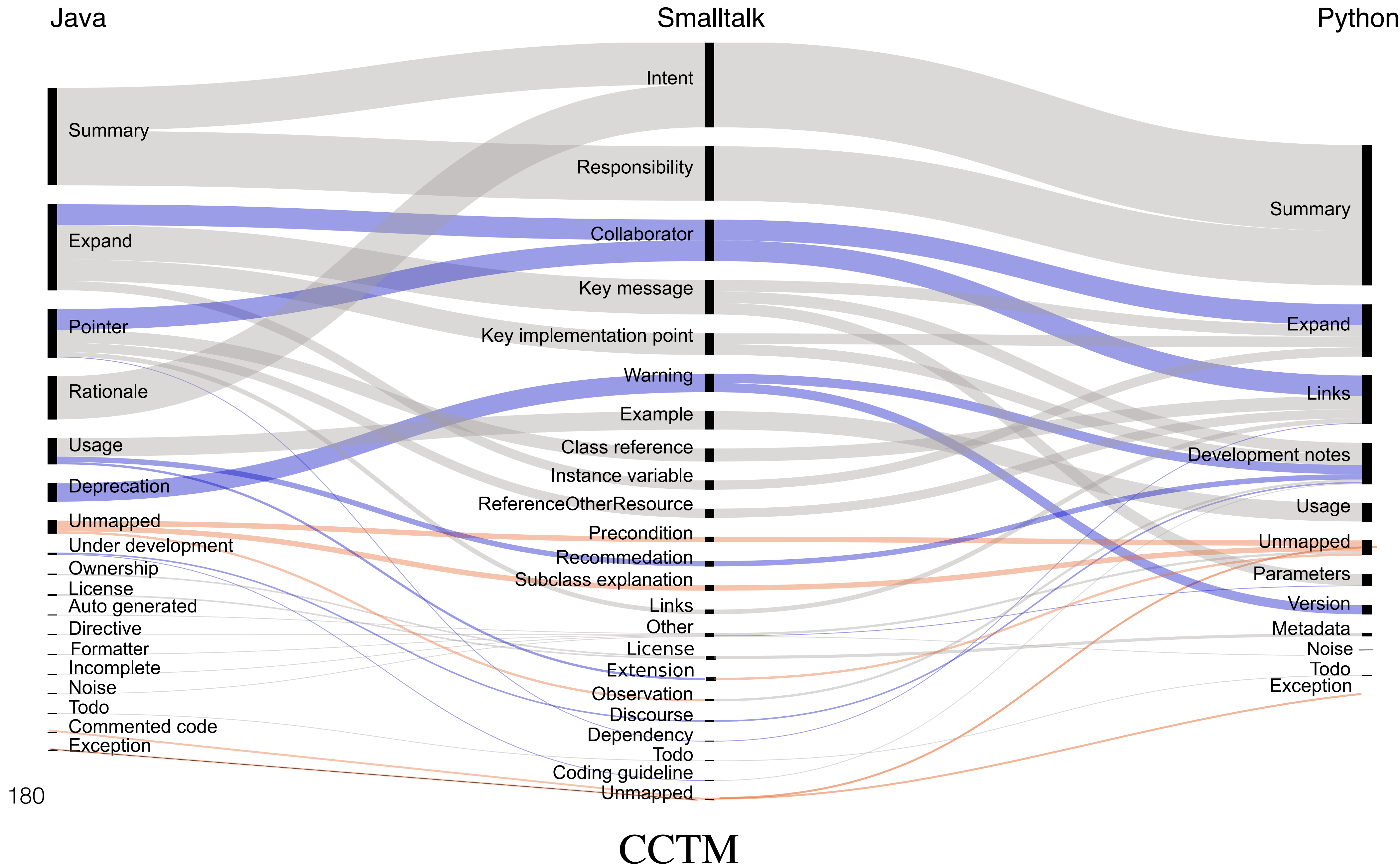
- █ Summary
- █ Expand
- █ Pointer
- █ Rationale
- █ Usage
- █ Deprecation
- █ Unmapped
- Under Development
- Ownership
- License
- Autogenerated
- Directive
- Formatter
- Incomplete
- Noise
- Todo
- Commented code
- Exception

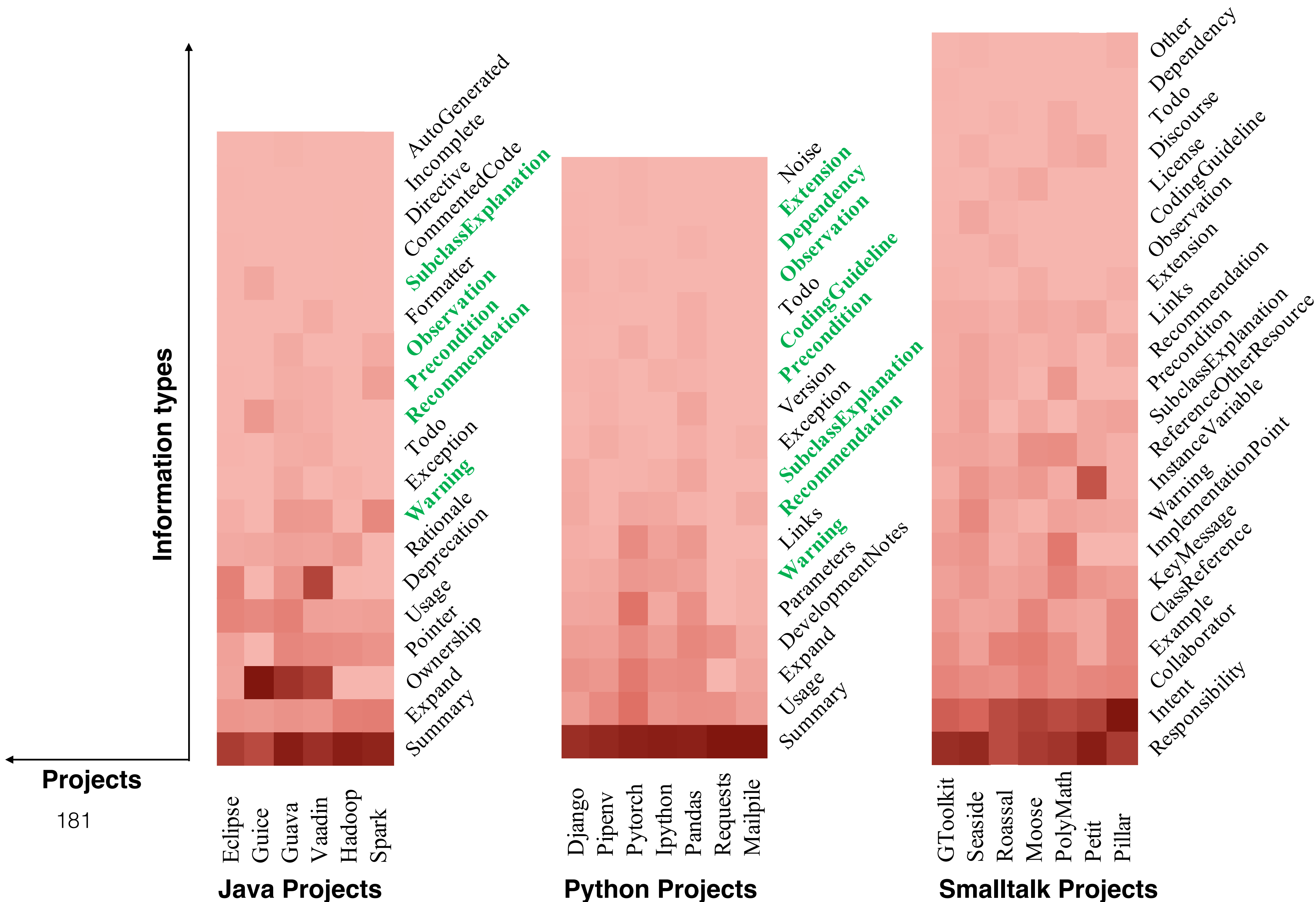
Smalltalk

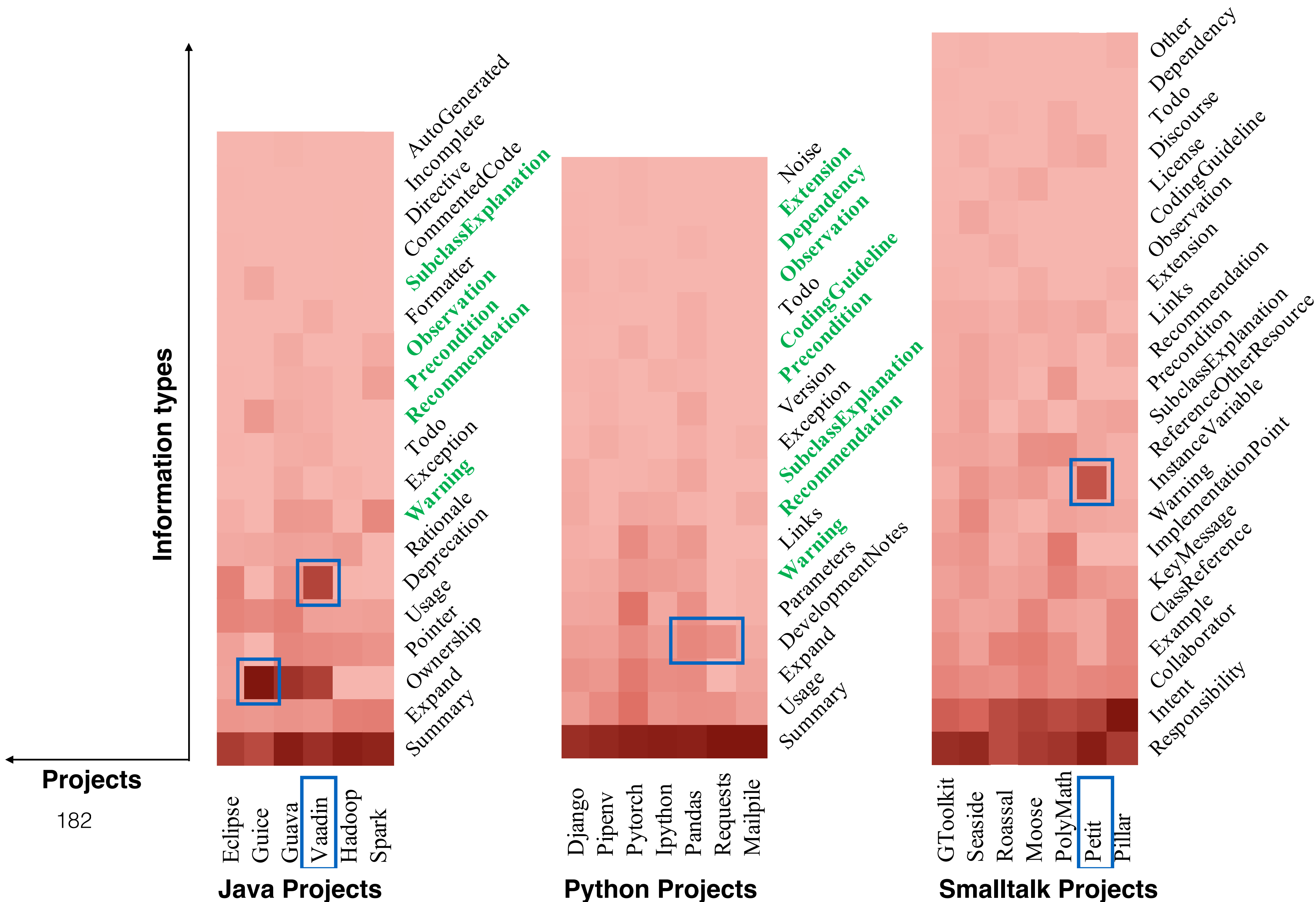
- █ Intent
- █ Responsibility
- █ Collaborators
- █ Key Messages
- █ Key Implementation Point
- █ Warnings
- █ Examples
- █ Class References
- █ Instance Variables
- █ ReferenceToOtherResource
- █ Preconditions
- █ Recommendation
- █ Subclasses Explanation
- █ Links
- █ Other
- █ License/Copyright
- █ Extension
- █ Observation
- █ Discourse
- █ Dependencies
- █ Todo
- █ Coding Guidelines
- █ Unmapped

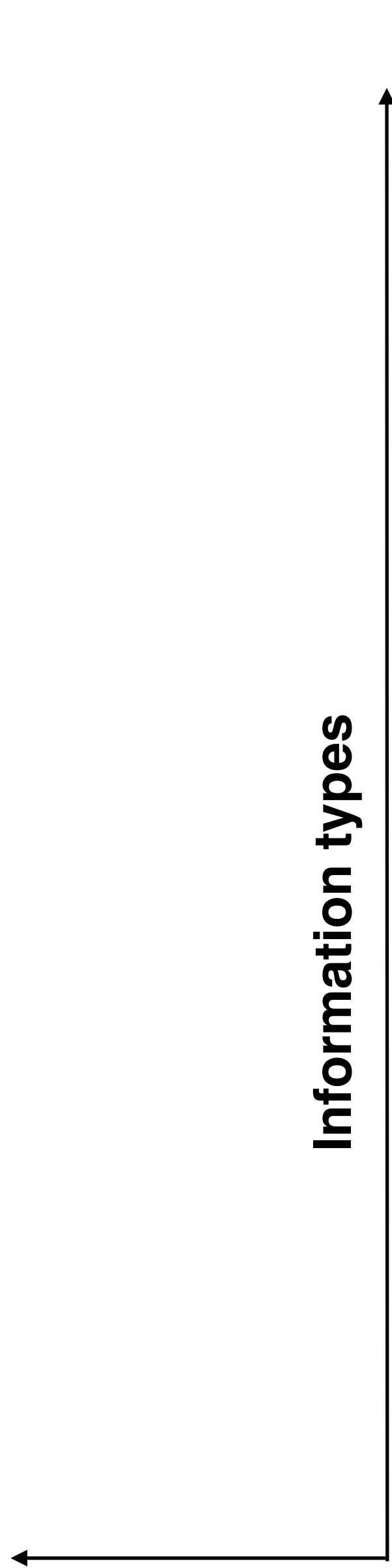
Python

- █ Summary
- █ Expand
- █ Links
- █ Development Notes
- █ Usage
- █ Unmapped
- █ Parameters
- █ Version
- █ Metadata
- █ Noise
- █ Todo
- █ Exception



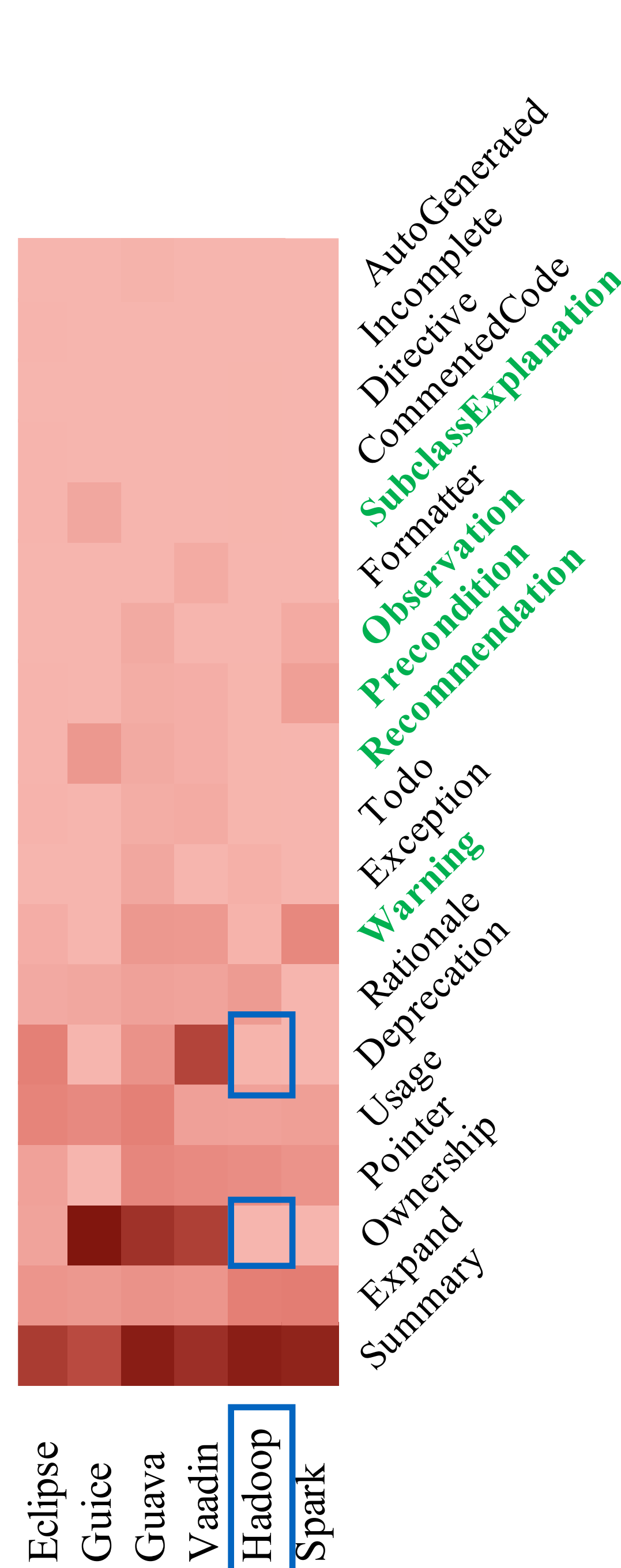






Projects

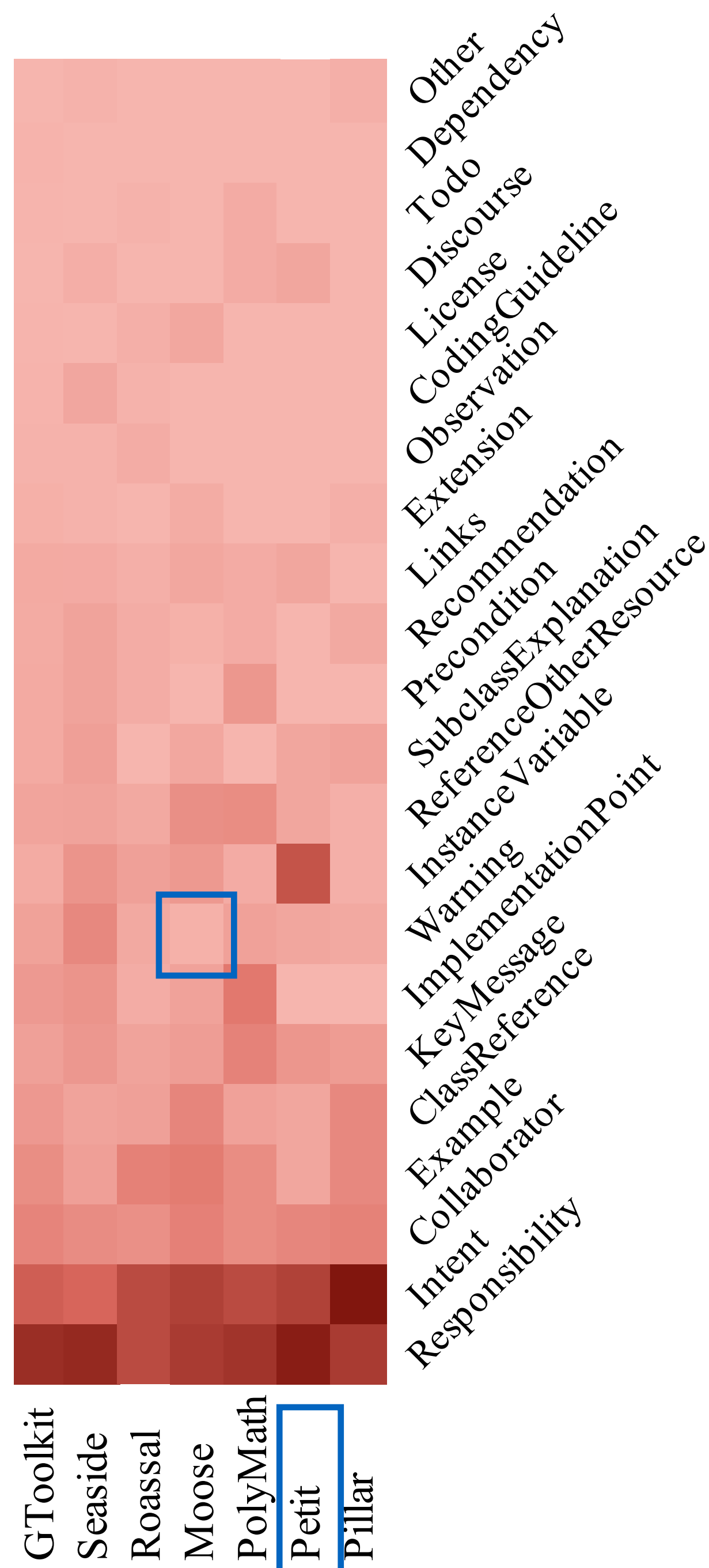
183



Java Projects



Python Projects



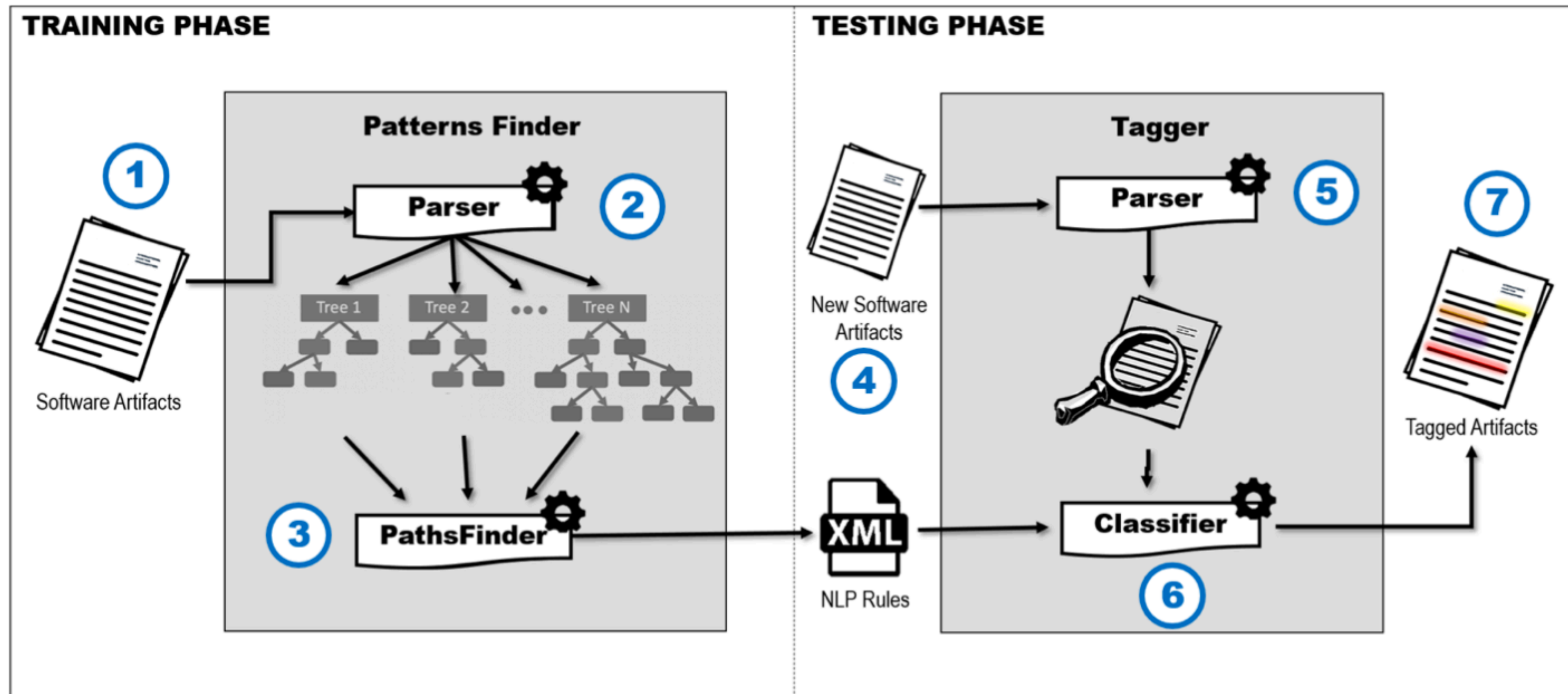
Smalltalk Projects

- AutoGenerated
- Incomplete
- Directive
- CommentedCode
- SubclassExplanation
- Formatter
- Observation
- Precondition
- Recommendation
- Todo
- Exception
- Warning
- Rationale
- Deprecation
- Usage
- Pointer
- Ownership
- Expand
- Summary

- Noise
- Extension
- Dependency
- Observation
- Todo
- CodingGuideline
- Precondition
- Version
- Exception
- SubclassExplanation
- Recommendation
- Links
- Warning
- Parameters
- DevelopmentNotes
- Expand
- Usage
- Summary

- Other
- Dependency
- Todo
- Discourse
- License
- CodingGuideline
- Observation
- Extension
- Links
- Recommendation
- Precondition
- SubclassExplanation
- ReferenceOtherResource
- InstanceVariable
- Warning
- ImplementationPoint
- KeyMessage
- ClassReference
- Example
- Collaborator
- Intent
- Responsibility

NEON to extract patterns



Training: software artifact (classified comments of each category in our case) is inspected to identify recurrent NL patterns.

Testing: the inferred rules are leverage to recognise the information of interest in a different corpus.

Example patterns from 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

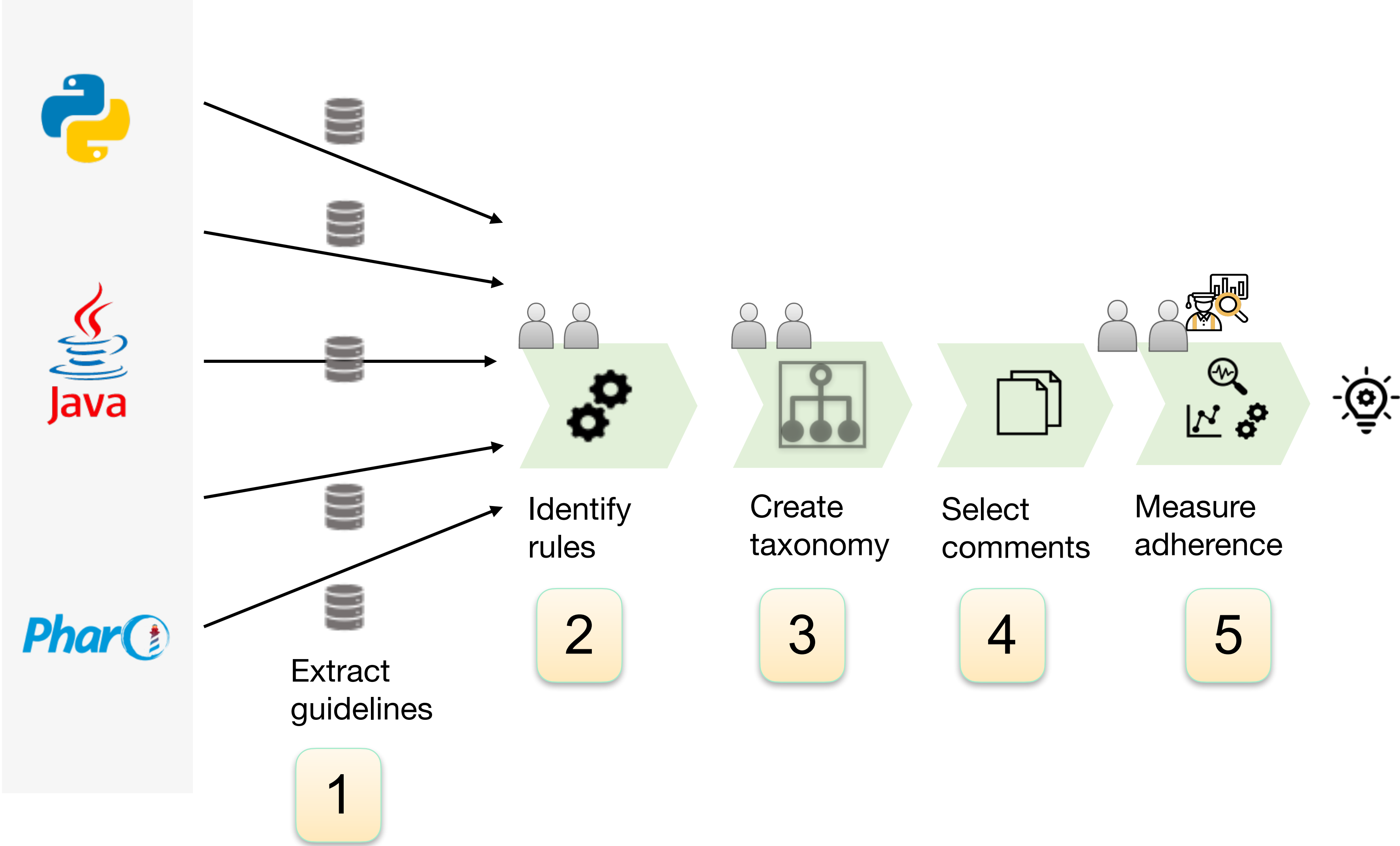
Class represents [something]

```
<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>
```

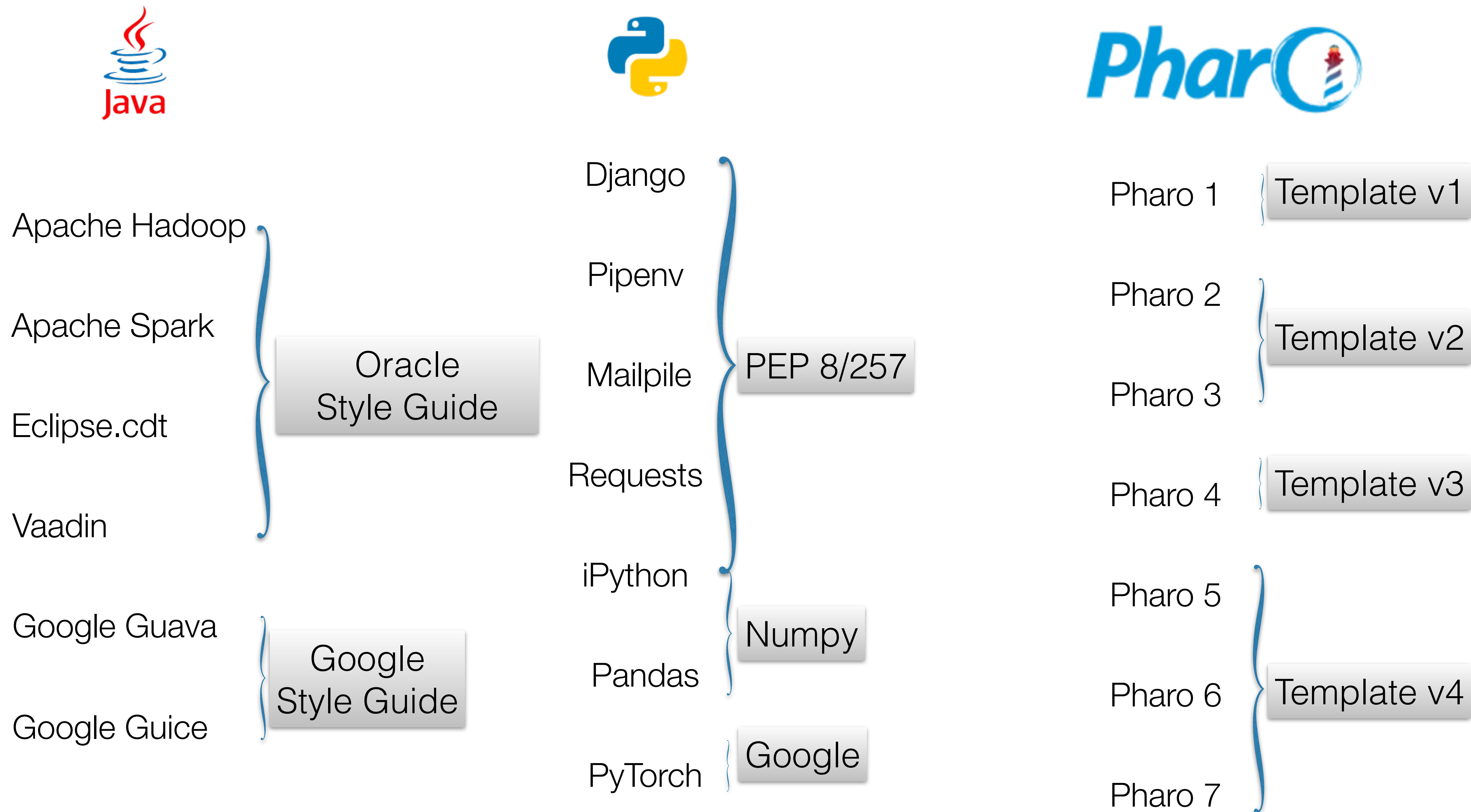
What information developers write in class comments across languages?

Do they follow the coding style guidelines?

Methodology



1 Extract guidelines



Pharo 7 Template

```
! Comment x + ← → ▾
Please comment me using the following template inspired by Class Responsibility Collaborator (CRC)
design:

For the Class part: State a one line summary. For example, "I represent a paragraph of text".

For the Responsibility part: Three sentences about my main responsibilities - what I do, what I know.

For the Collaborators Part: State my main collaborators and one line about how I interact with them.

Public API and Key Messages

- message one
- message two
- (for bonus points) how to create instances.

    One simple example is simply gorgeous.

Internal Representation and Key Implementation Points.

    Instance Variables
environmentDictionaries: <Object>

Implementation Points
```

2 Identify rules

Multi-line Docstrings

Multi-line docstrings ¹ consists of a summary line just like a one-line docstrings, ² followed by a blank line, ³ followed by a more elaborate description. The summary line may be used by automatic indexing tools; ⁴ it is important that fits on one line and is separated from the rest of the docstring by a blank line. ⁵ The summary line may be on the same line as the opening quotes or on the next line. ⁶ The entire doctoring is intended the same as the quotes at its first time.

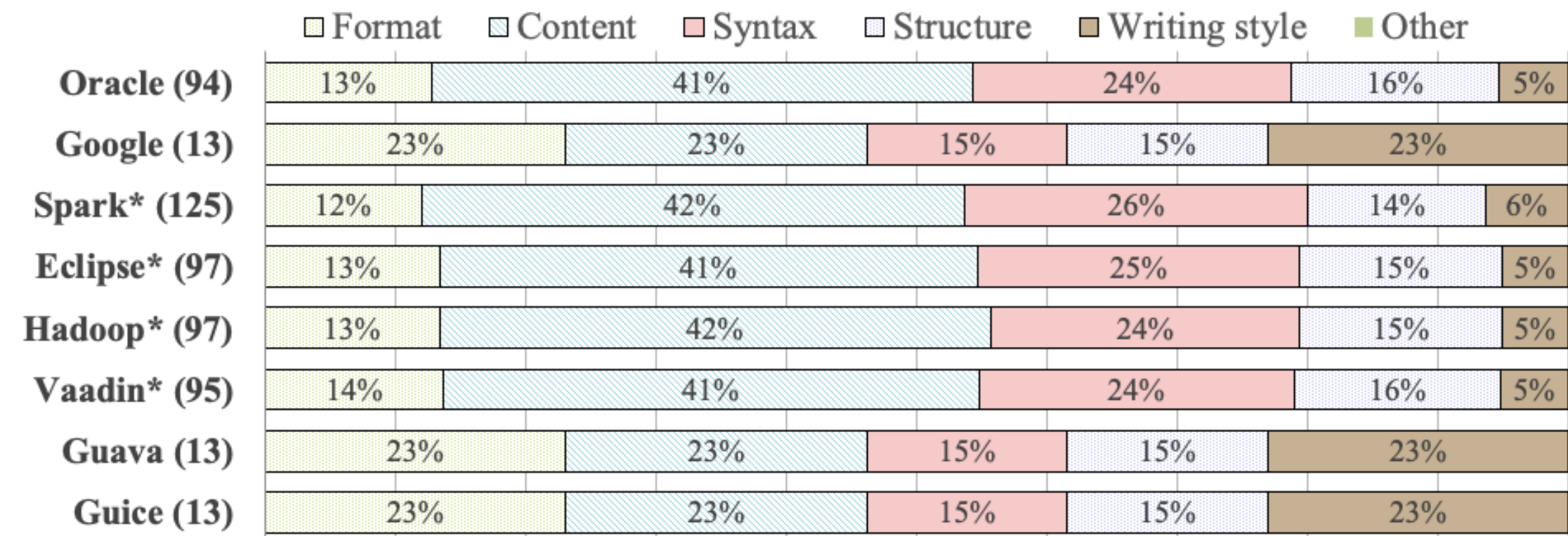
2 Create taxonomy

Multi-line Docstrings

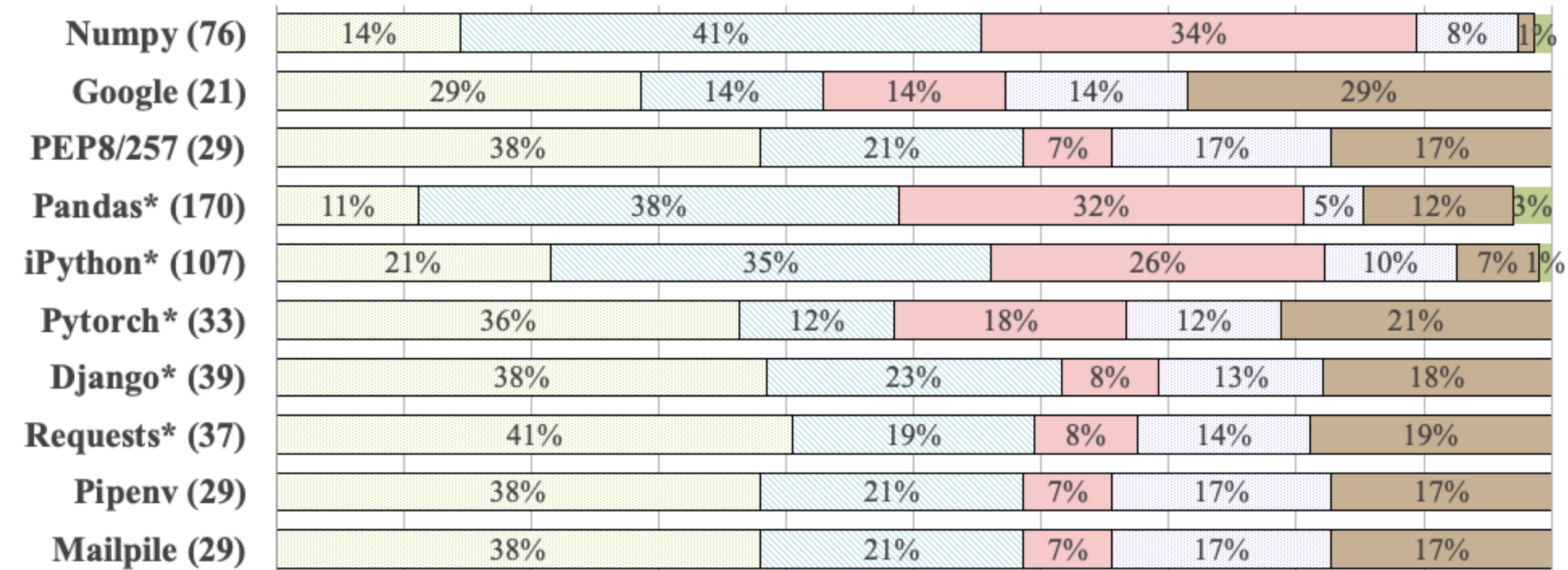
Content

Format

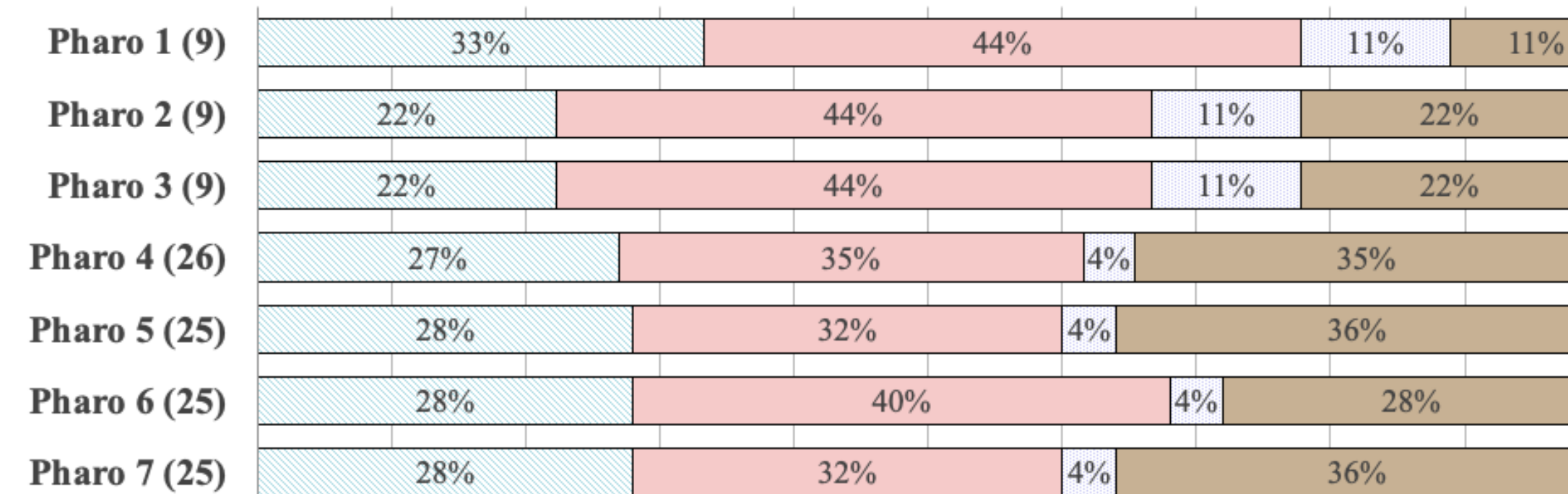
Multi-line docstrings consists of a summary line just like a one-line docstrings, followed by a blank line, followed by a more elaborate description. The summary line may be used by automatic indexing tools; it is important that fits on one line and is separated from the rest of the docstring by a blank line. The summary line may be on the same line as the opening quotes or on the next line. The entire doctoring is intended the same as the quotes at its first time.



Percentage of class comment conventions in Java



Percentage of class comment conventions in Python



Percentage of class comment conventions in Smalltalk

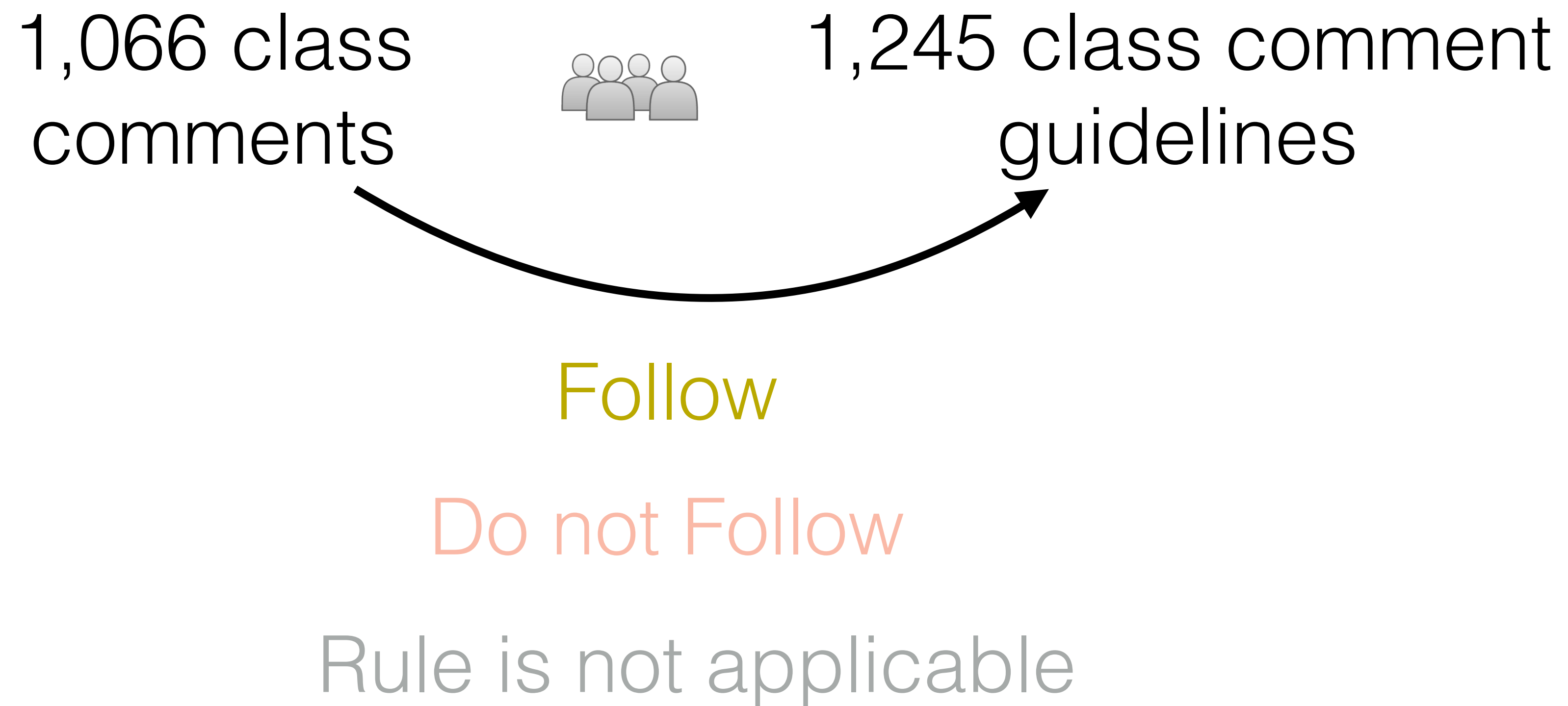
547 rules

570 rules

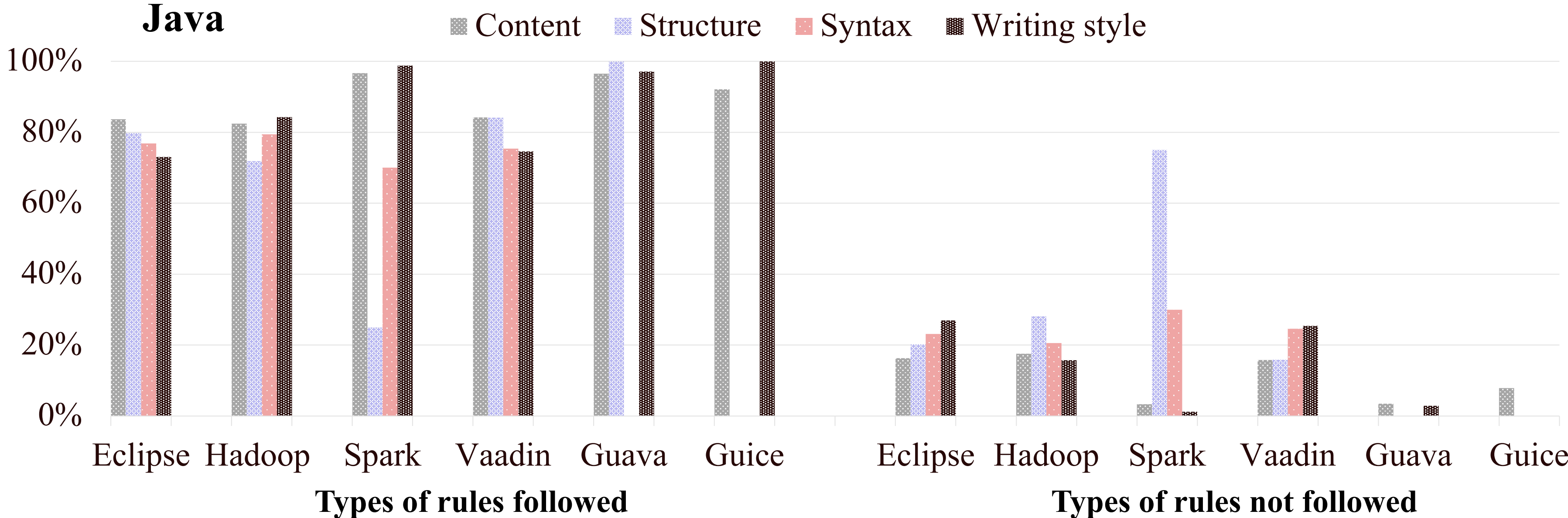
128 rules

Content rules are more prevalent in style guidelines but hard to locate comments or part of comments

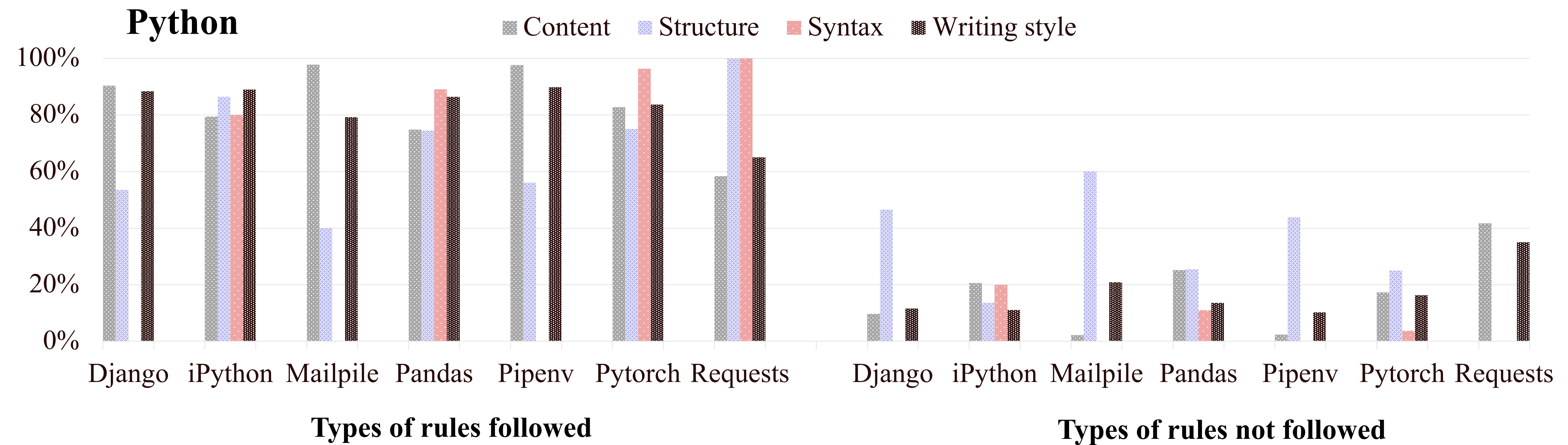
5 Measure adherence



Do developers follow conventions?

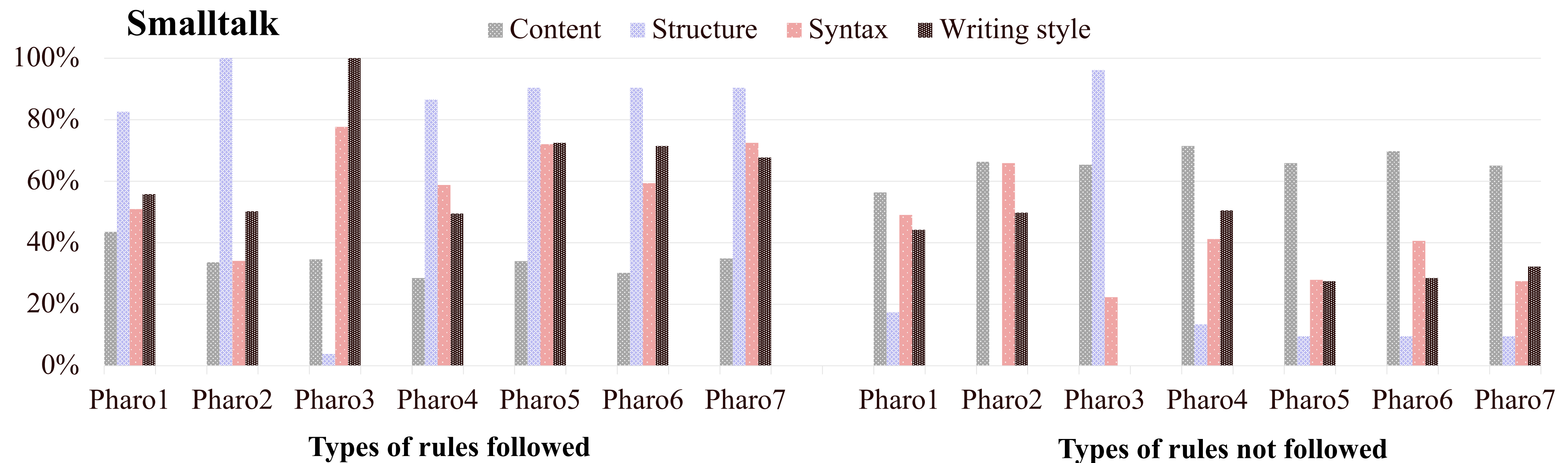


Do developers follow conventions?



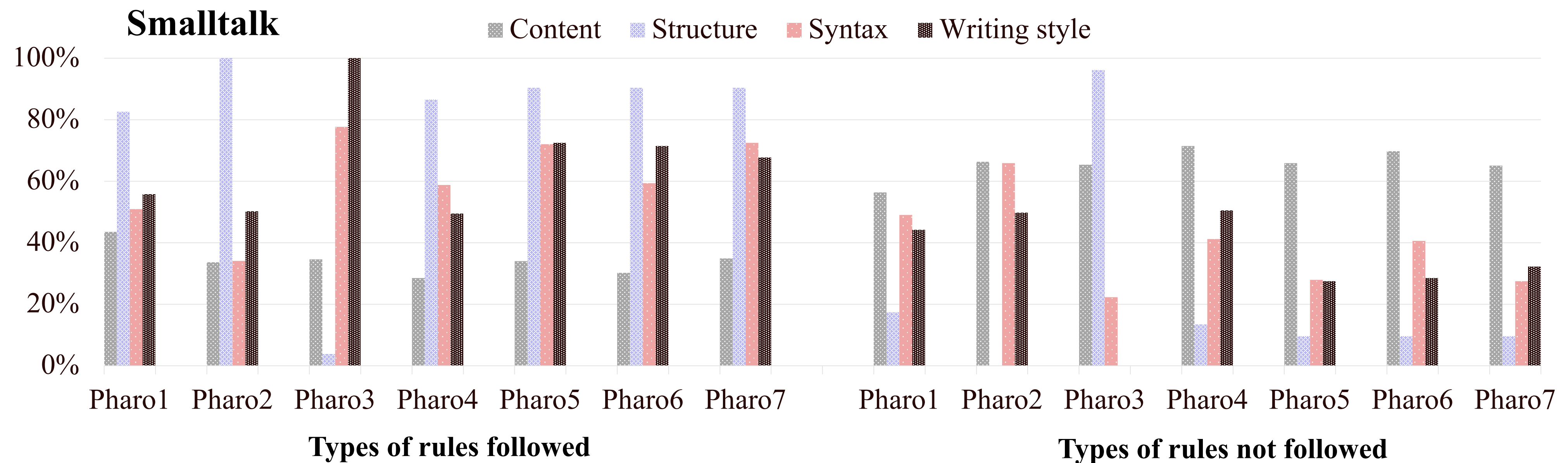
In Java and Python, comment follow **content** and **writing style** conventions.

Do developers follow conventions?



In Smalltalk, comments follow **structure** and **writing style** conventions

Do developers follow conventions?



In Java and Python, every third comment violate **structure** conventions.
In Smalltalk, every third comment violate **content** conventions

Future work

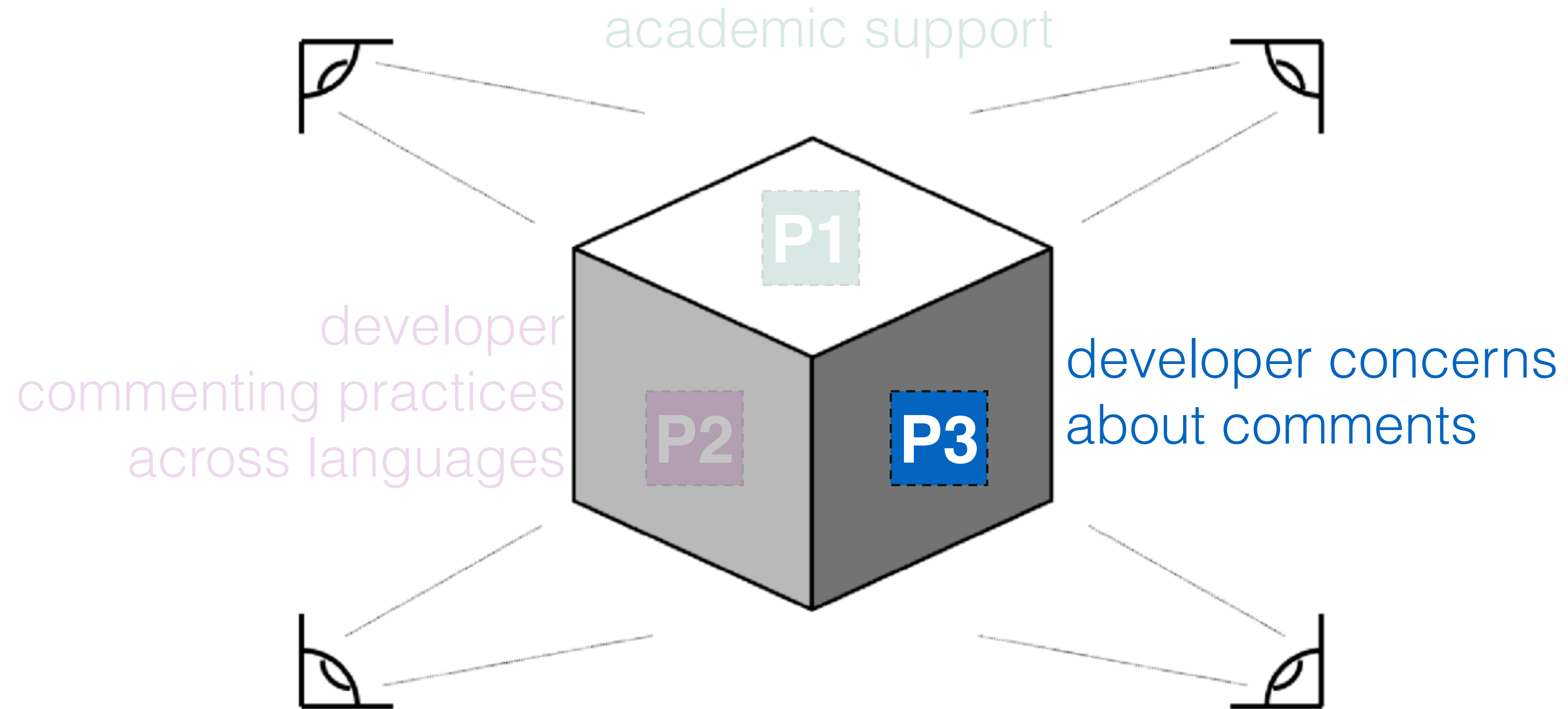
Verify other types of comments (Method, inline comments)

Verify comments of other languages (C++, JavaScript)

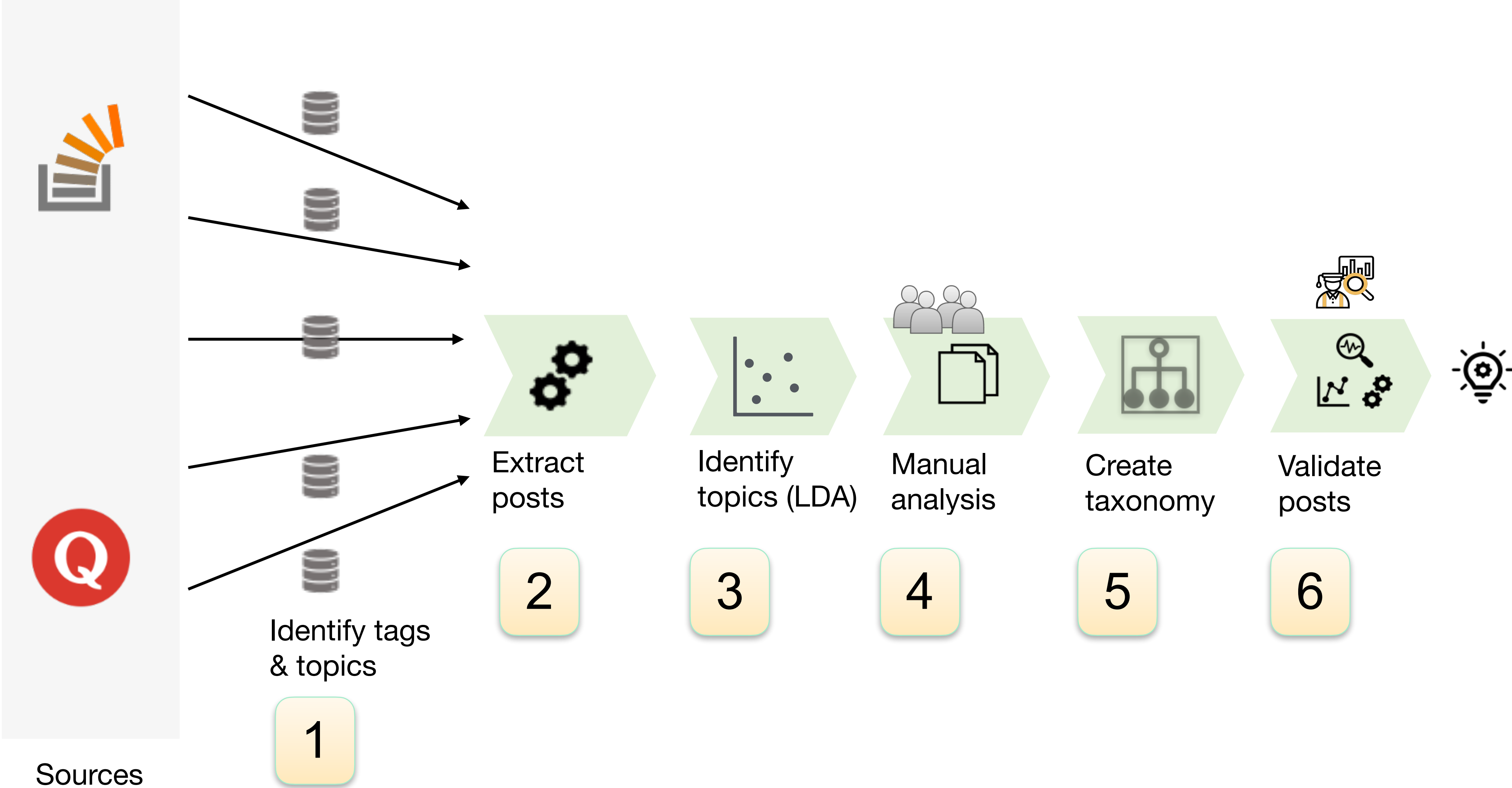
Develop tools to validate comments against the guidelines

Improve comment quality assessment

P3: Questions developer ask



Developer concerns about comments



2 Extract posts

Stack Overflow

19, 700

Quora

3, 671

3

LDA Topic modelling

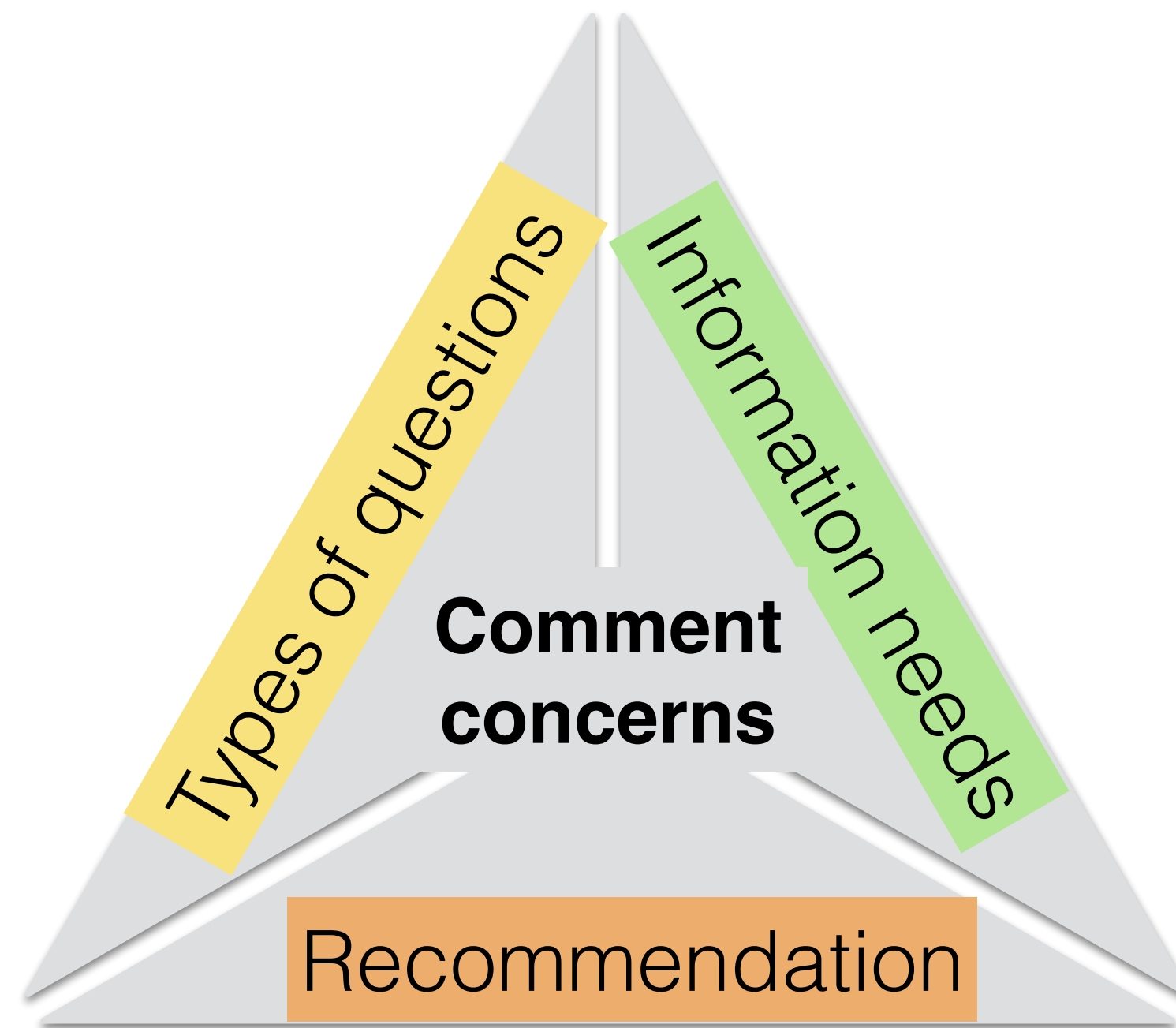
LDA Technical Details

- Stack overflow posts: 19, 705
- MALLET
- Topics $k = 10$
- Hyperparameters
 - $\alpha = 5$
 - $\beta = 0.01$

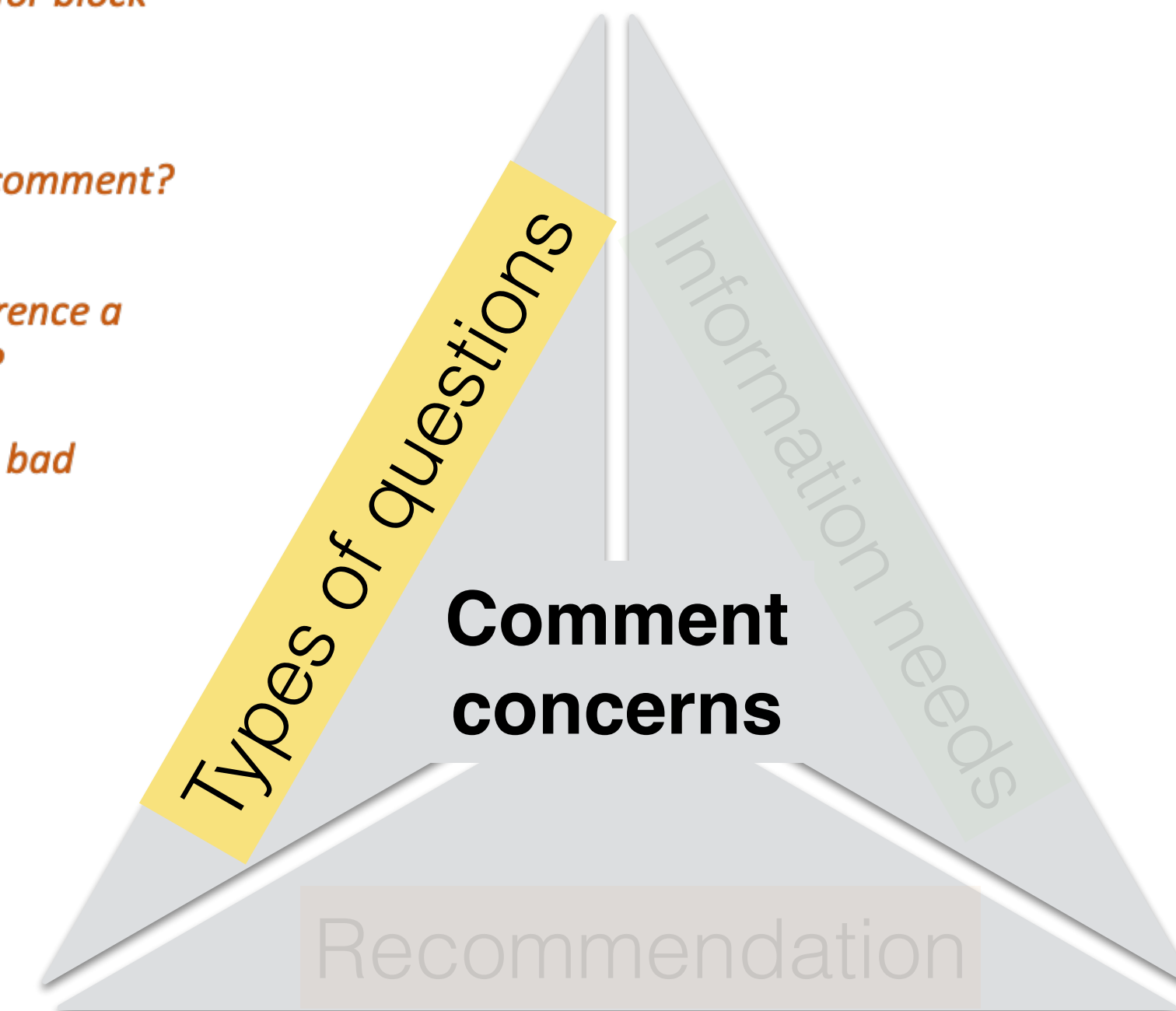
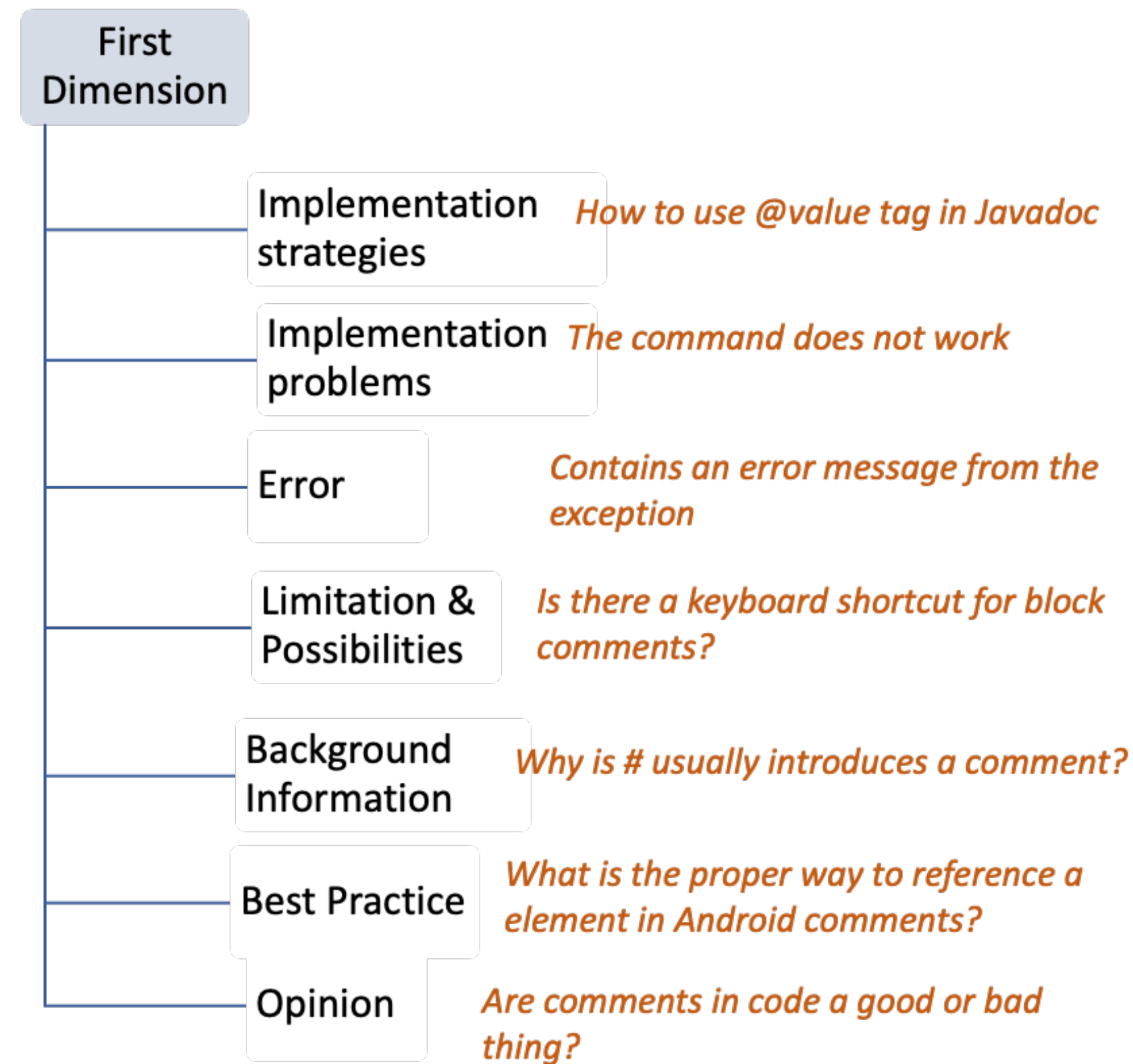
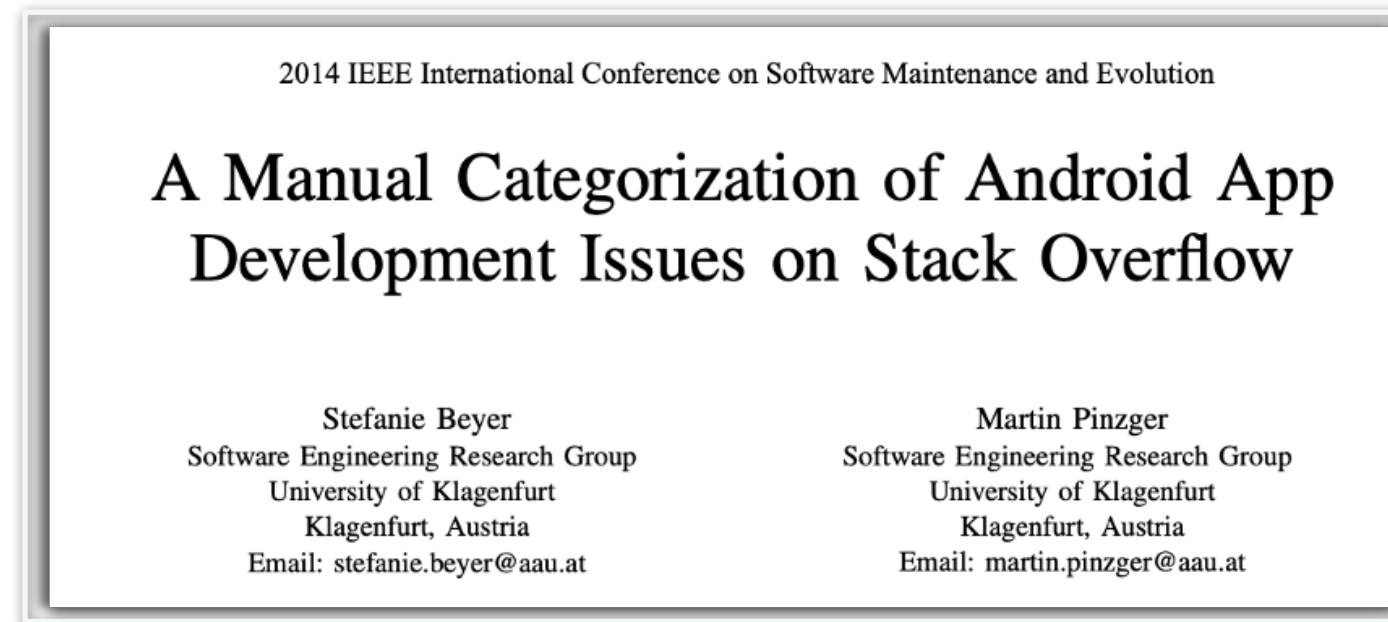
3 LDA Topic modelling

#	Topic Name
1	Syntax & Format
2	IDEs & Editors
3	R Documentation
4	Code Conventions
5	Developing frameworks for thread commenting
6	Open-source software
7	Documentation generation
8	Thread comments in web-sites
9	Naming conventions & data types
10	Seeking documentation & learning language

4 Manual analysis



5 Taxonomy

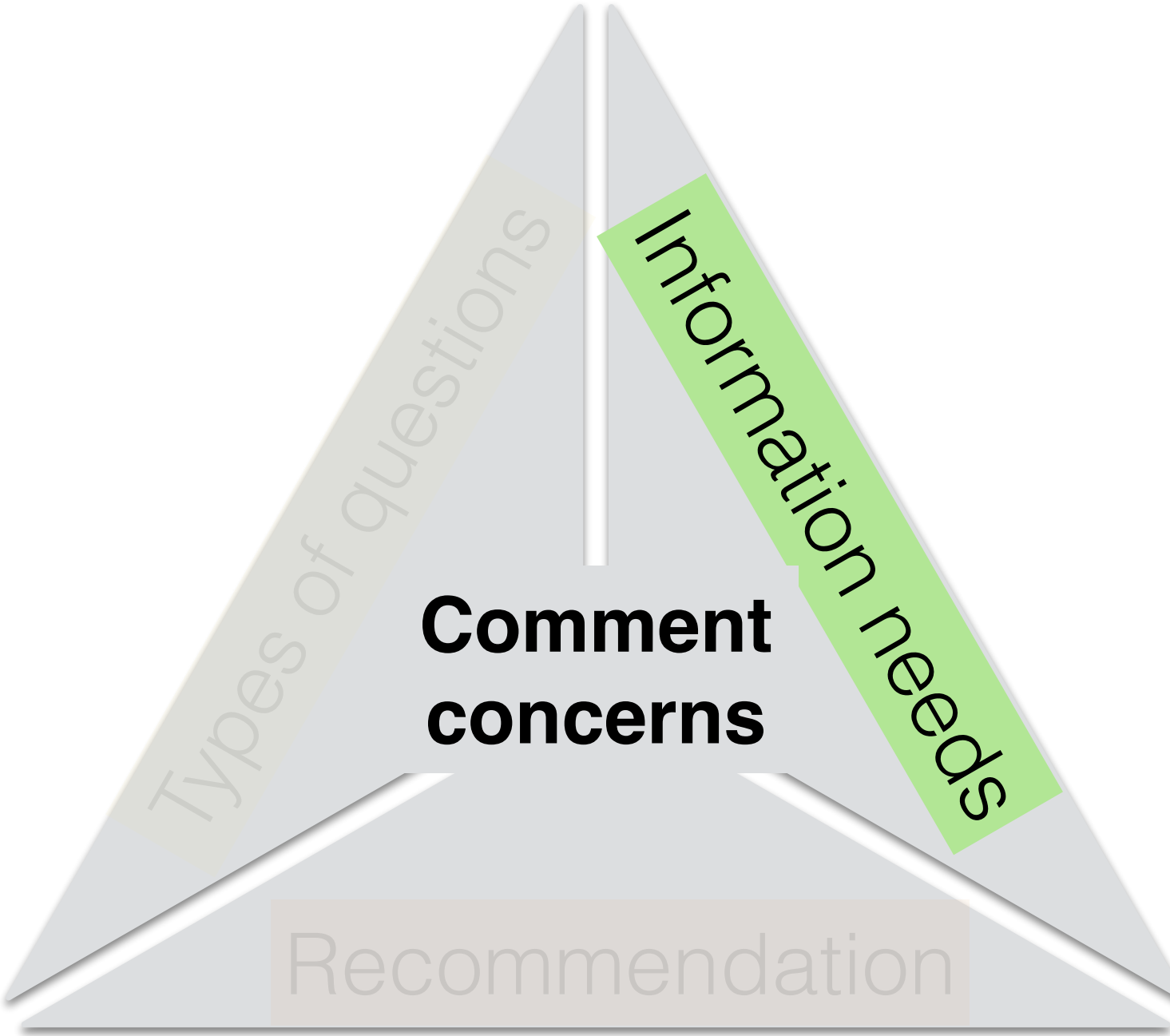
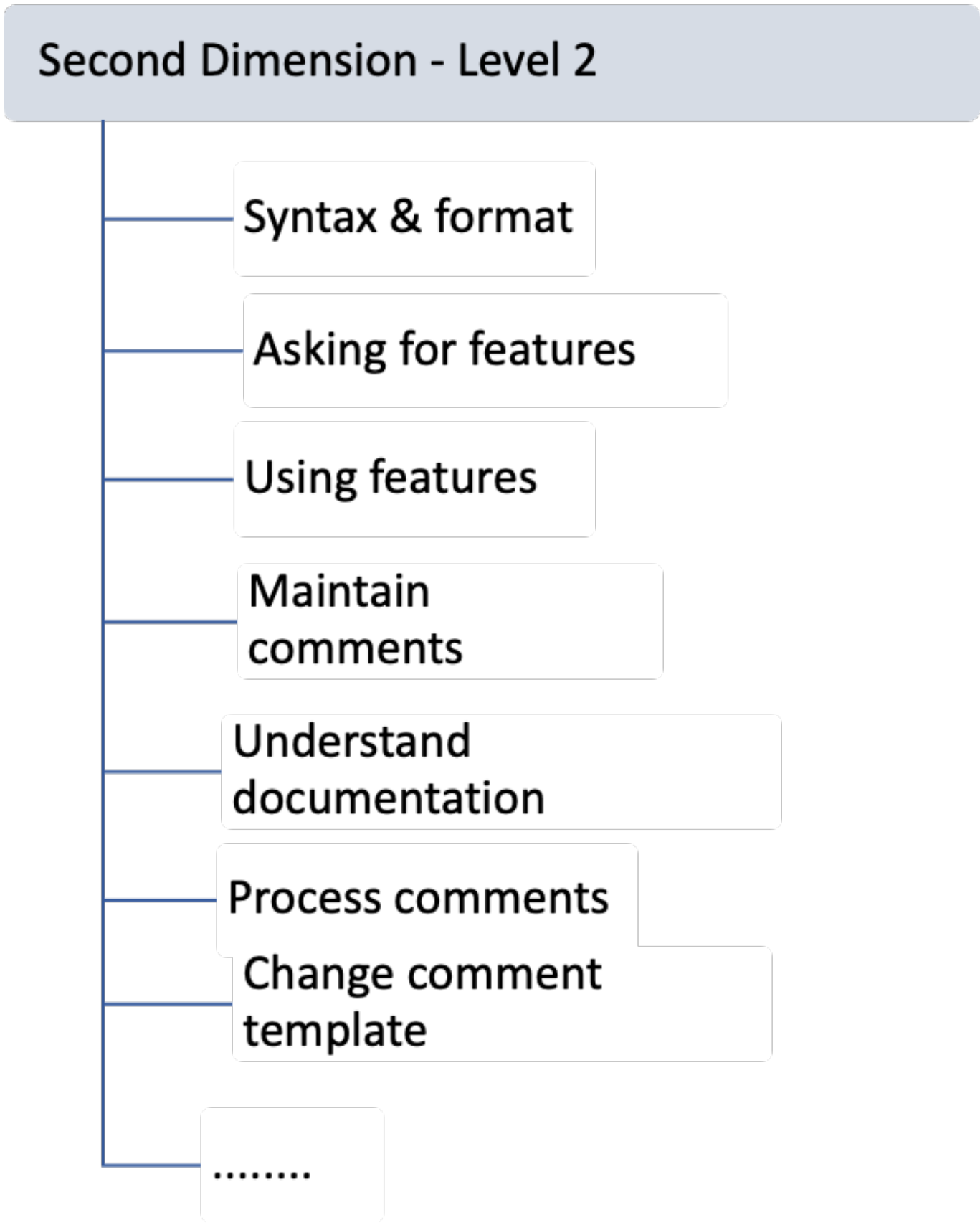
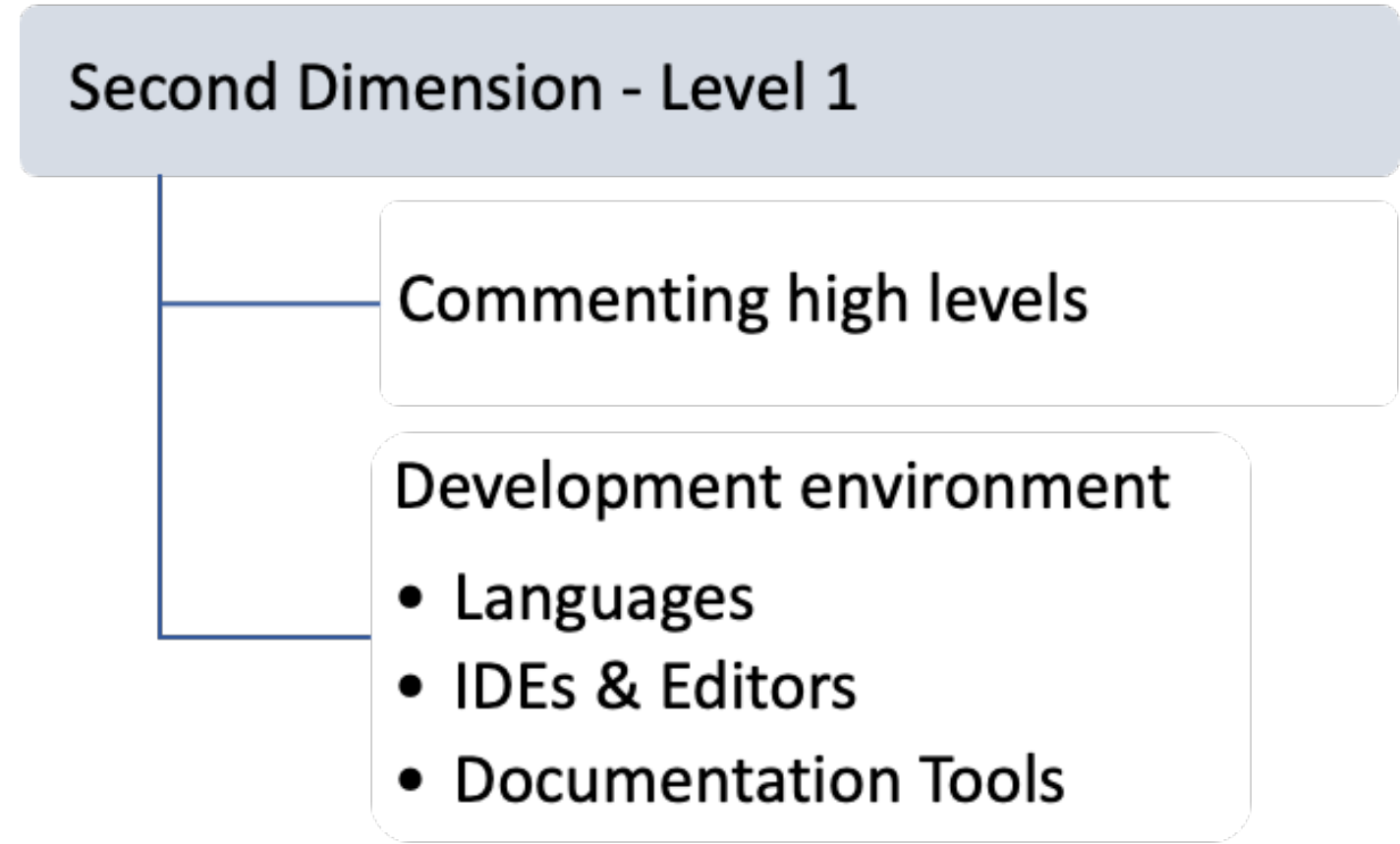


5 Taxonomy

2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

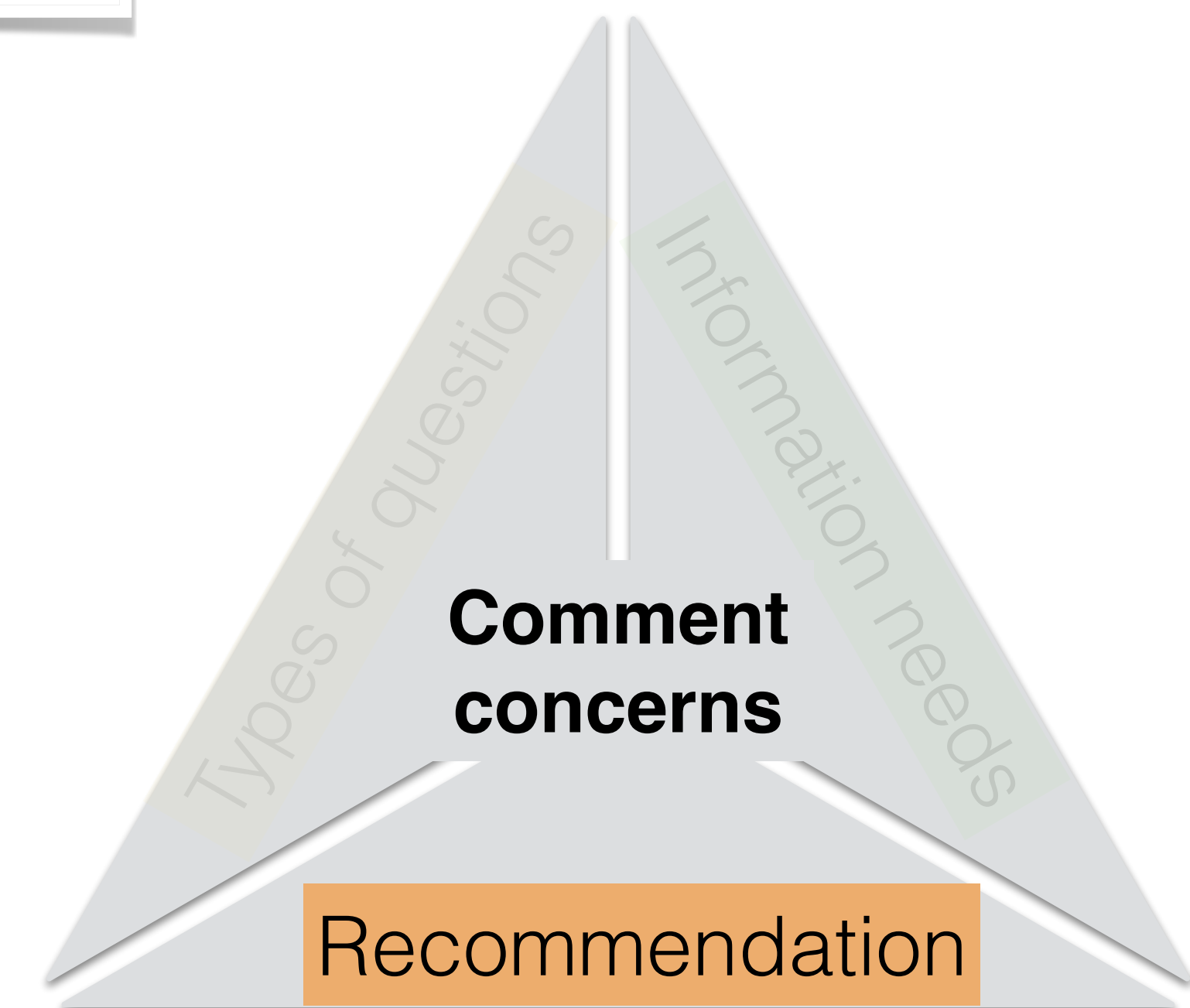
Software Documentation Issues Unveiled

Emad Aghajani*, Csaba Nagy*, Olga Lucero Vega-Márquez†
 Mario Linares-Vásquez†, Laura Moreno‡, Gabriele Bavota*, Michele Lanza*
 *Software Institute, Università della Svizzera italiana (USI), Switzerland
 †Systems and Computing Engineering Department, Universidad de los Andes, Colombia
 ‡Department of Computer Science, Colorado State University, USA



5 Taxonomy

[.net] long inline comments should start with a capital letter and end with a period.



stackoverflow Products Search...

Home PUBLIC Questions Tags Users COLLECTIVES Explore Collectives

Should Javadoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times

4

What is the recommended place to put Javadoc for a method with an annotation? Before or after the annotation?

```
@Test
/**
 * My doc
 */
public void testMyTest(){
}
```

OR

```
/**
 * My doc
 */
@Test
public void testMyTest(){
}
```


java coding-style annotations javadoc

I don't think it matters but second format is better. annotations are part of the code and play crucial role per their usage pattern. Better to keep all code related entries together.

7

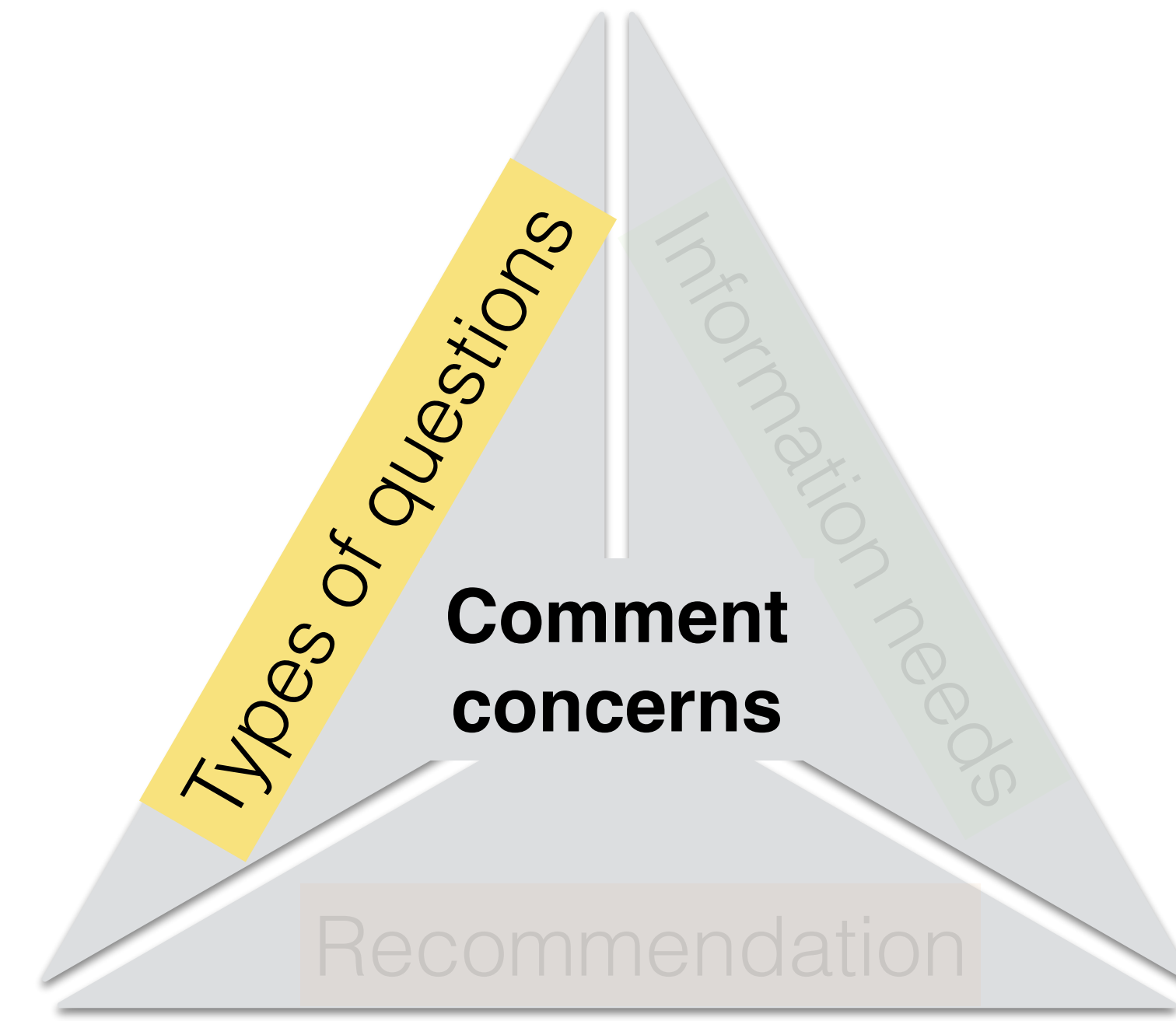
Share Edit Follow

answered Nov 14 '12 at 18:12

 **Yogendra Singh**
32.8k ● 6 ● 60 ● 71

✓

Best Practice



stackoverflow Products Search...

Home PUBLIC Questions Tags Users COLLECTIVES Explore Collectives

Should JavaDoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times

4 What is the recommended place to put JavaDoc for a method with an annotation? Before or after the annotation?

```
@Test
/**
 * My doc
 */
public void testMyTest(){
}
```

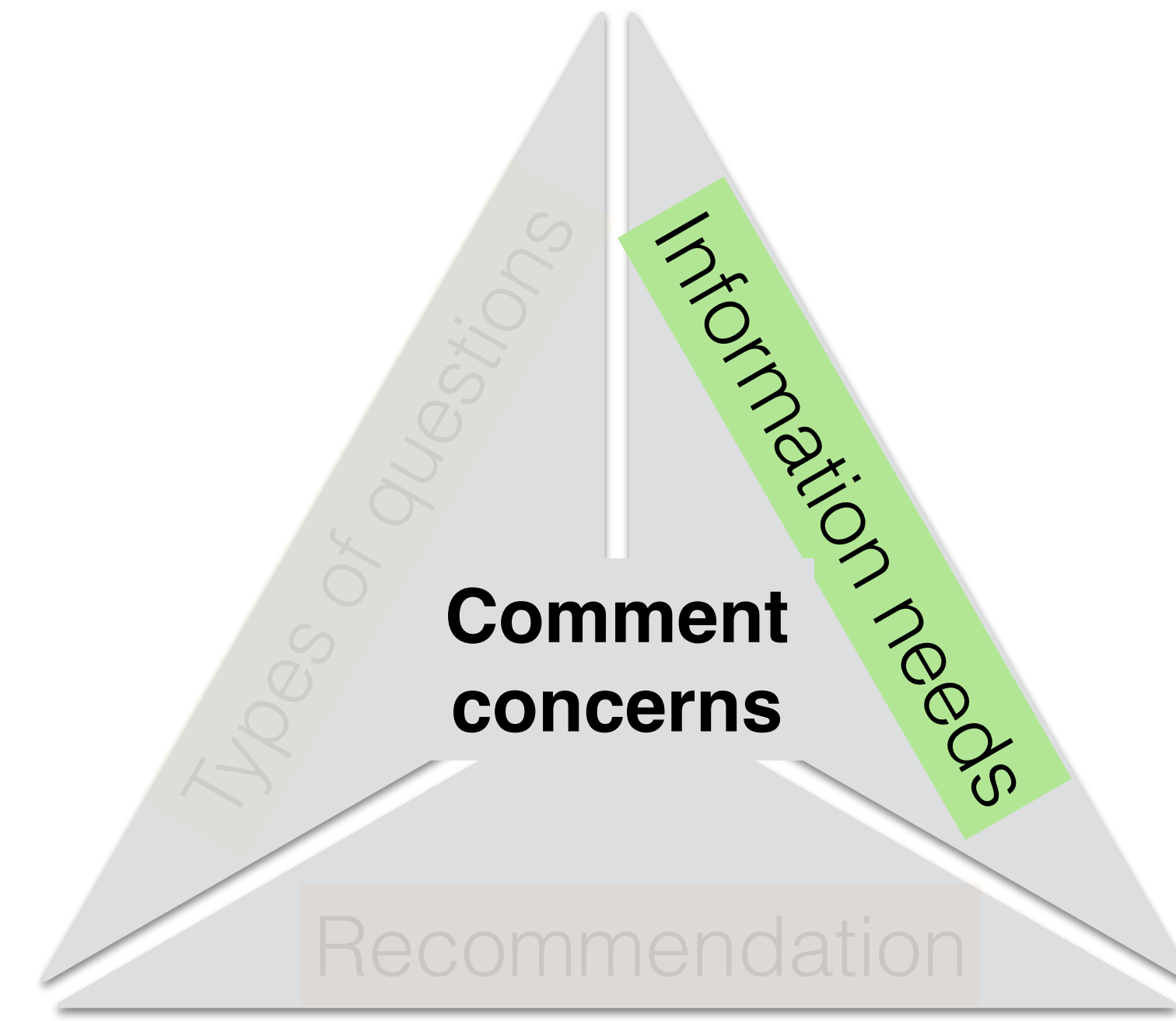
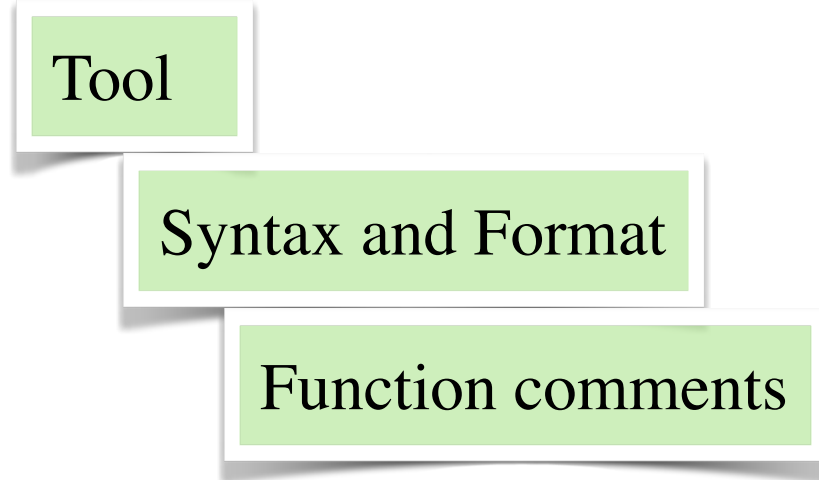
OR

```
/**
 * My doc
 */
@Test
public void testMyTest(){
}
```

java coding-style annotations javadoc

7 I don't think it matters but second format is better. annotations are part of the code and play crucial role per their usage pattern. Better to keep all code related entries together.

answered Nov 14 '12 at 18:12 **Yogendra Singh** 32.8k 6 60 71



stackoverflow Products Search...

Home PUBLIC Questions Tags Users COLLECTIVES Explore Collectives

Should Javadoc go before or after a method-level annotation?

Asked 9 years, 2 months ago Active 9 years, 2 months ago Viewed 2k times

4 ▲ What is the recommended place to put Javadoc for a method with an annotation? Before or after the annotation?

```
@Test
/**
 * My doc
 */
public void testMyTest(){
}
```

OR

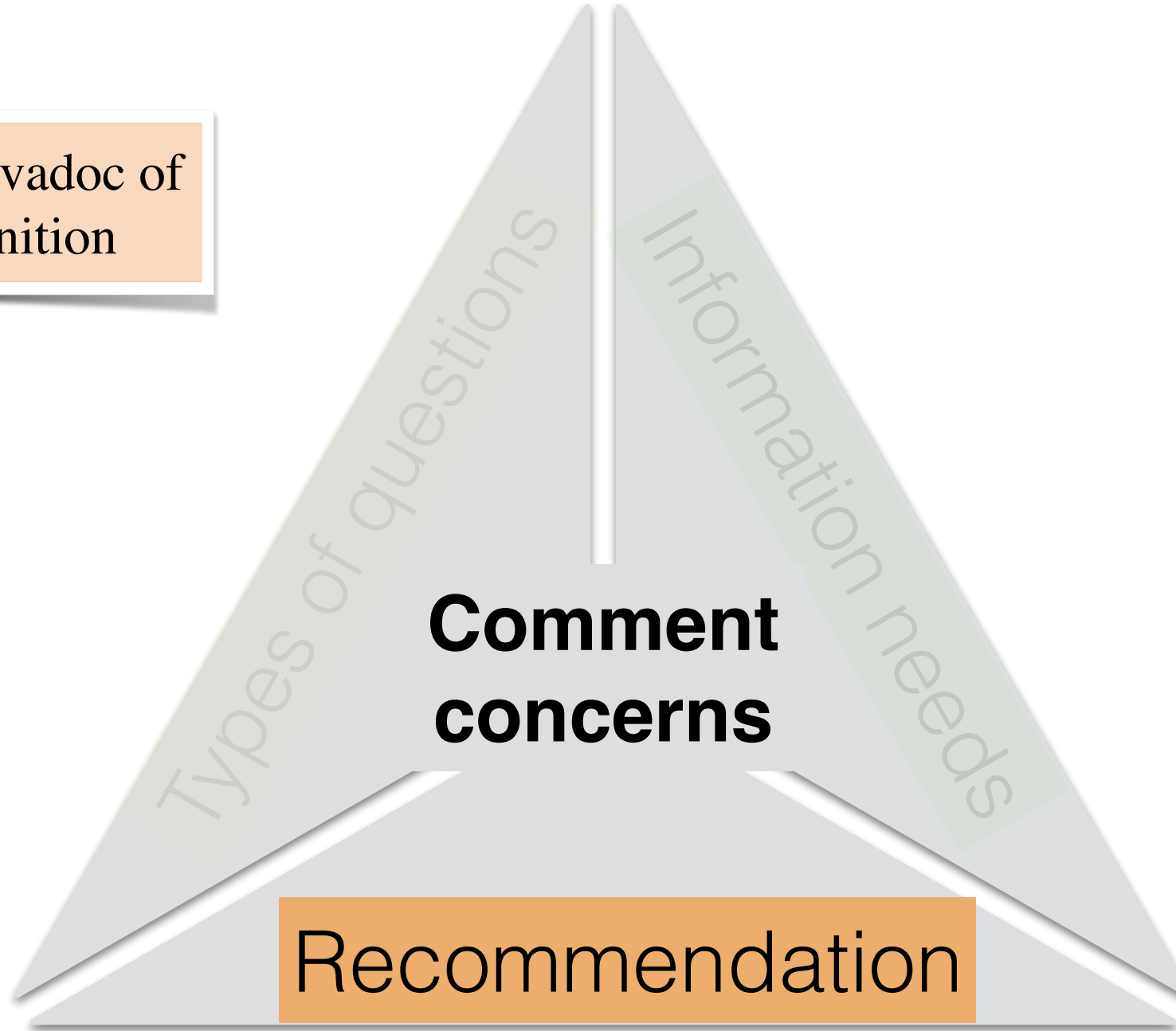
```
/**
 * My doc
 */
@Test
public void testMyTest(){
}
```

java coding-style annotations javadoc

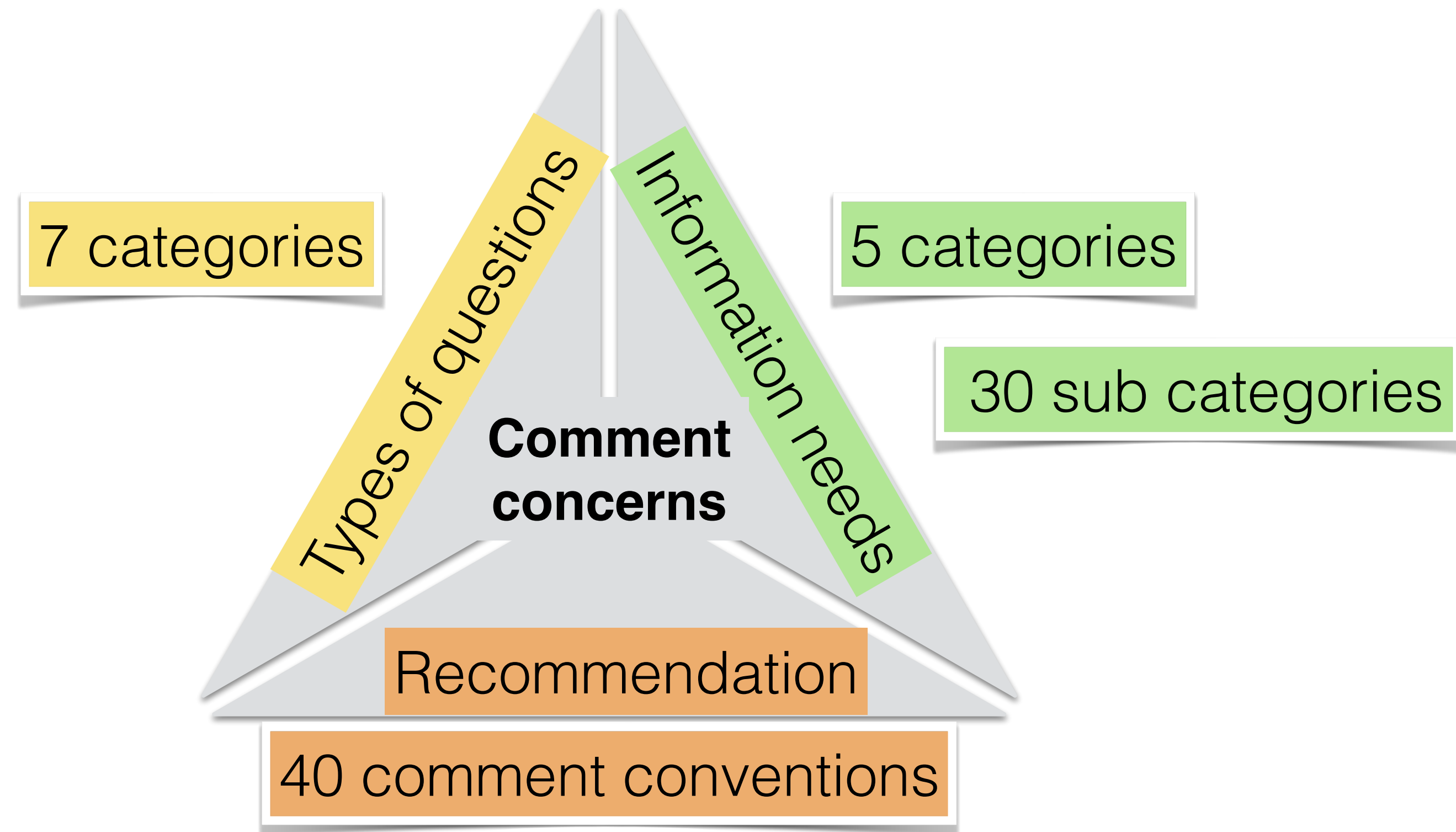
7 ▲ I don't think it matters but second format is better. annotations are part of the code and play crucial role per their usage pattern. Better to keep all code related entries together.

Share Edit Follow answered Nov 14 '12 at 18:12 Yogendra Singh 32.8k ● 6 ● 60 ● 71

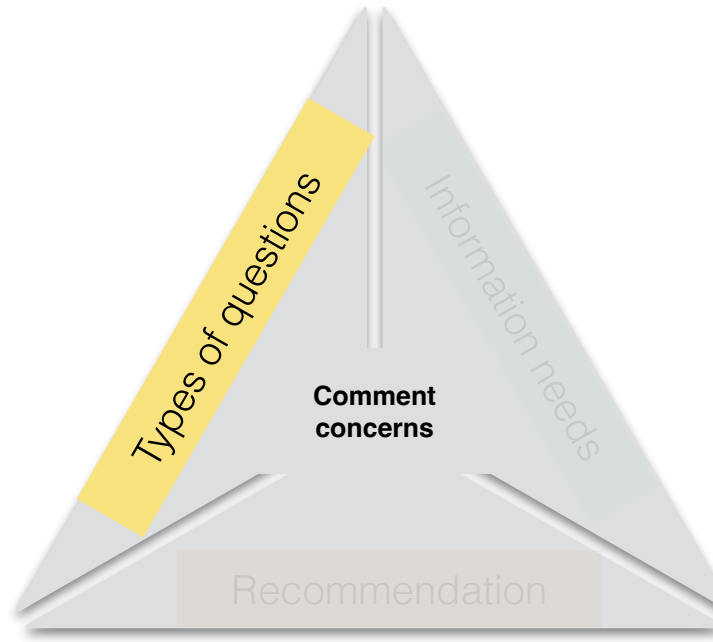
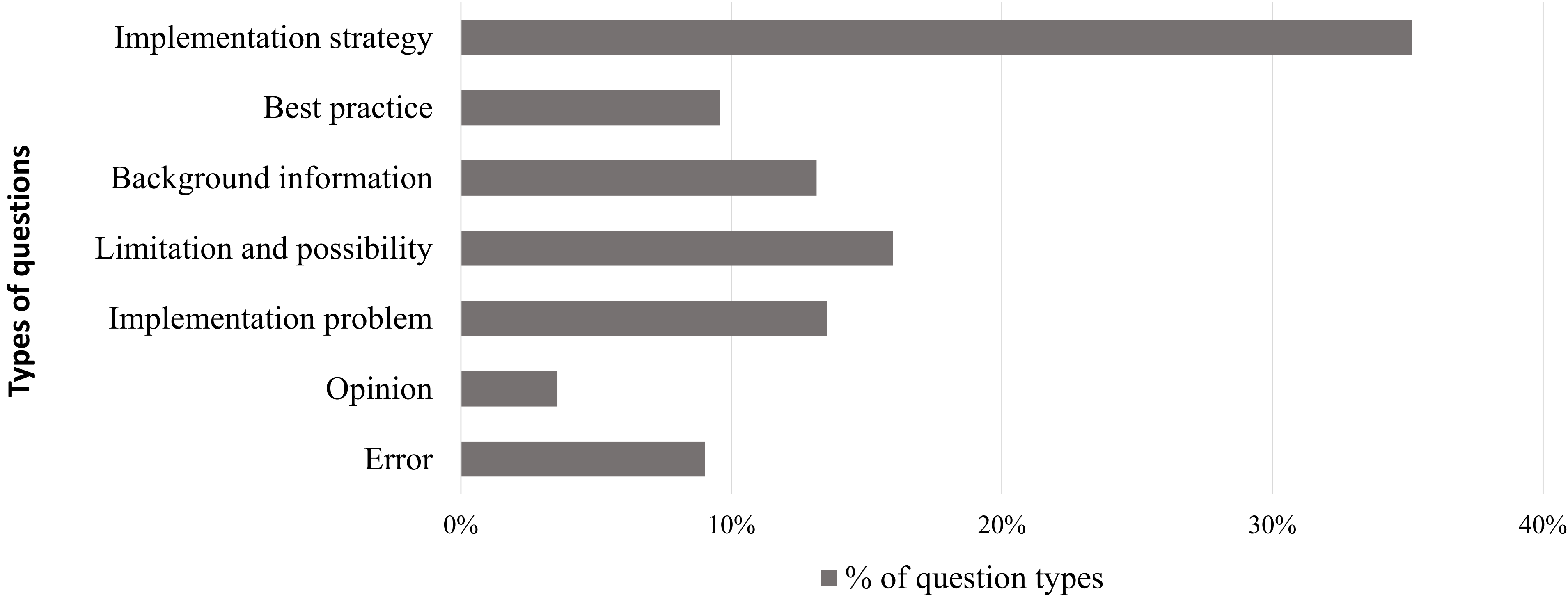
[Javadoc] Write annotations after the Javadoc of the method and before the method definition



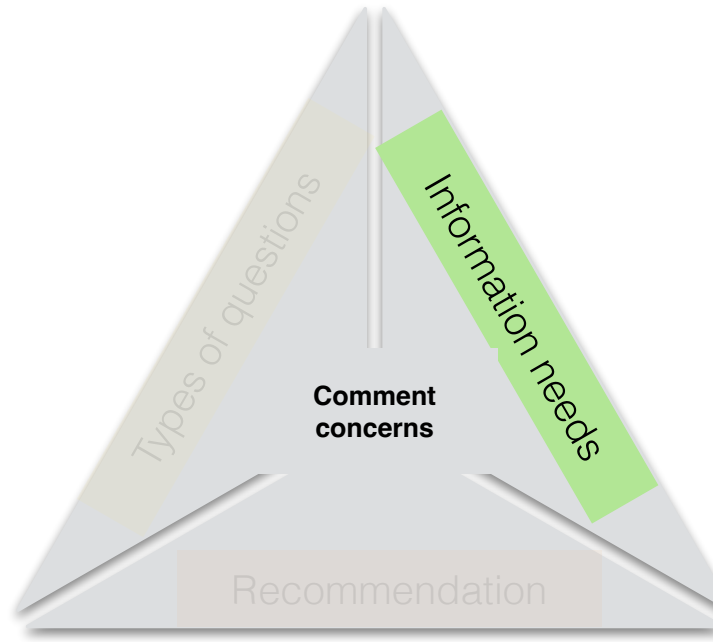
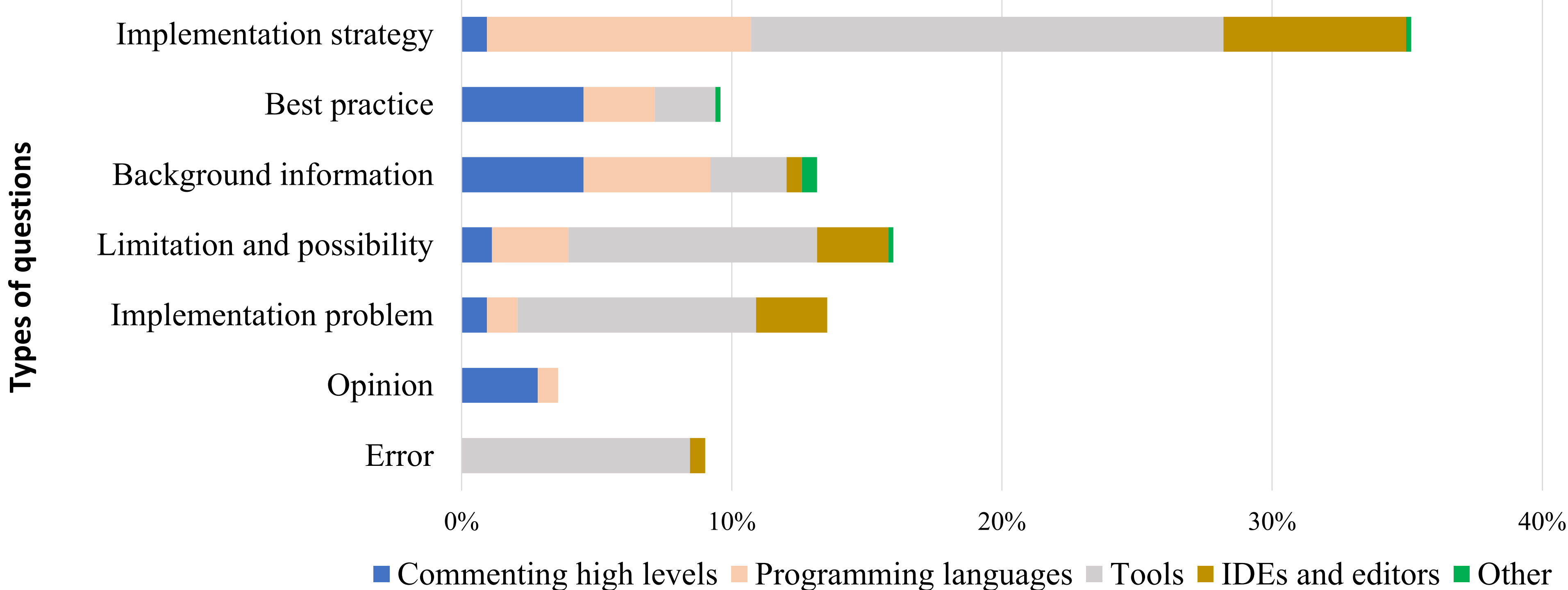
5 Taxonomy



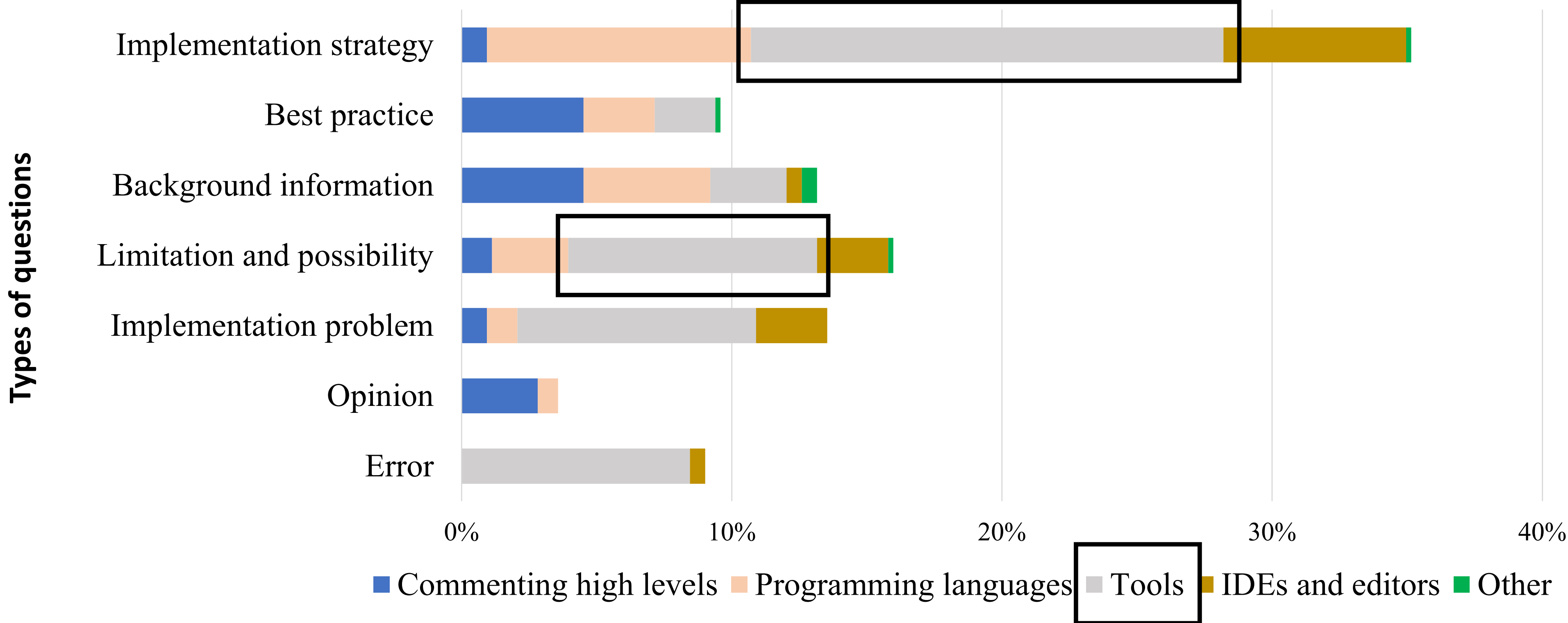
Developer concerns about comments



Developer concerns about comments



Developer concerns about comments



Developers gets confused on how to write comments using various tools

Future work

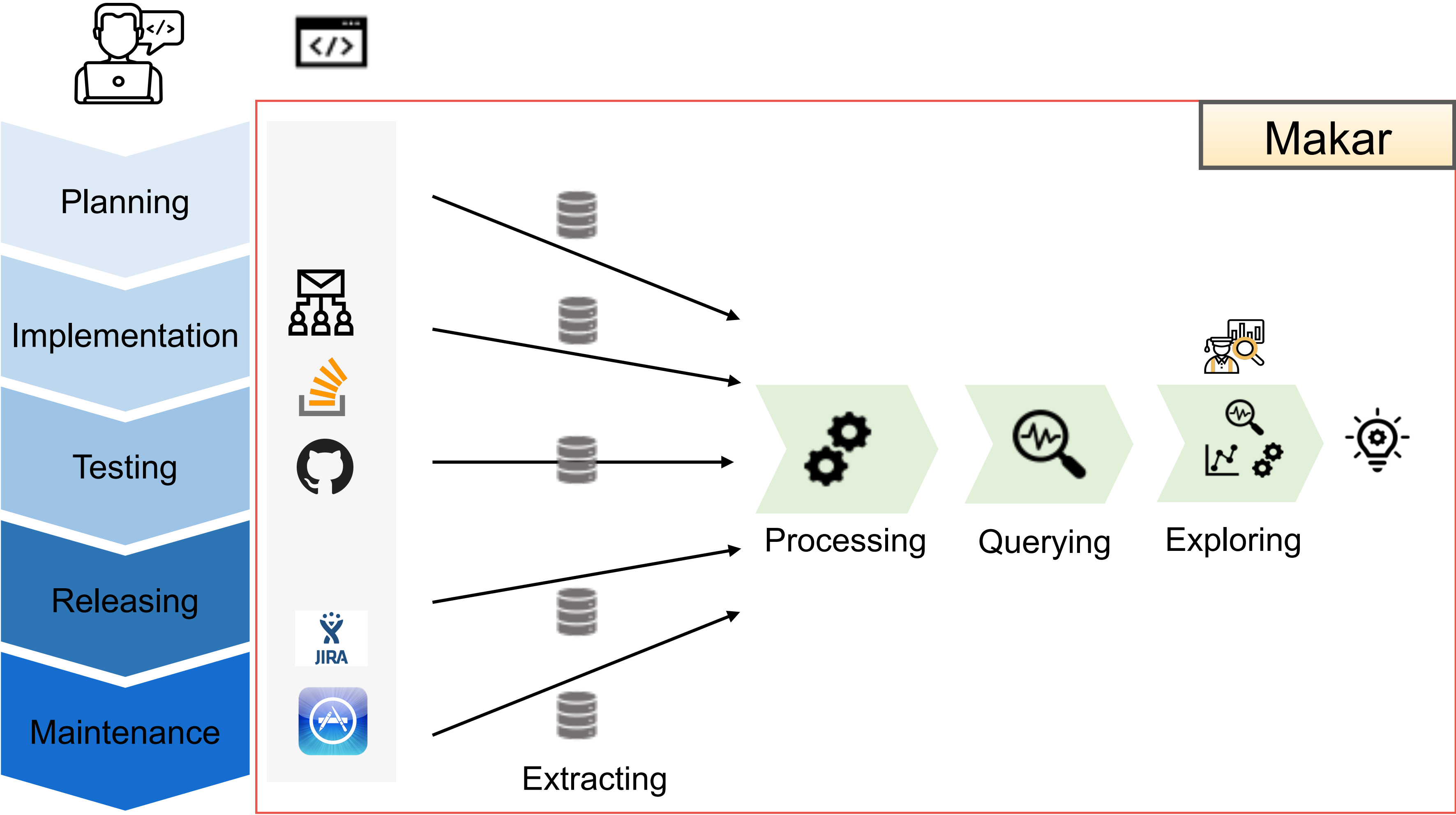
Improve documentation of tools and availability of coding style guidelines

Investigate more sources (e.g., GitHub, Jira, Mailing lists)

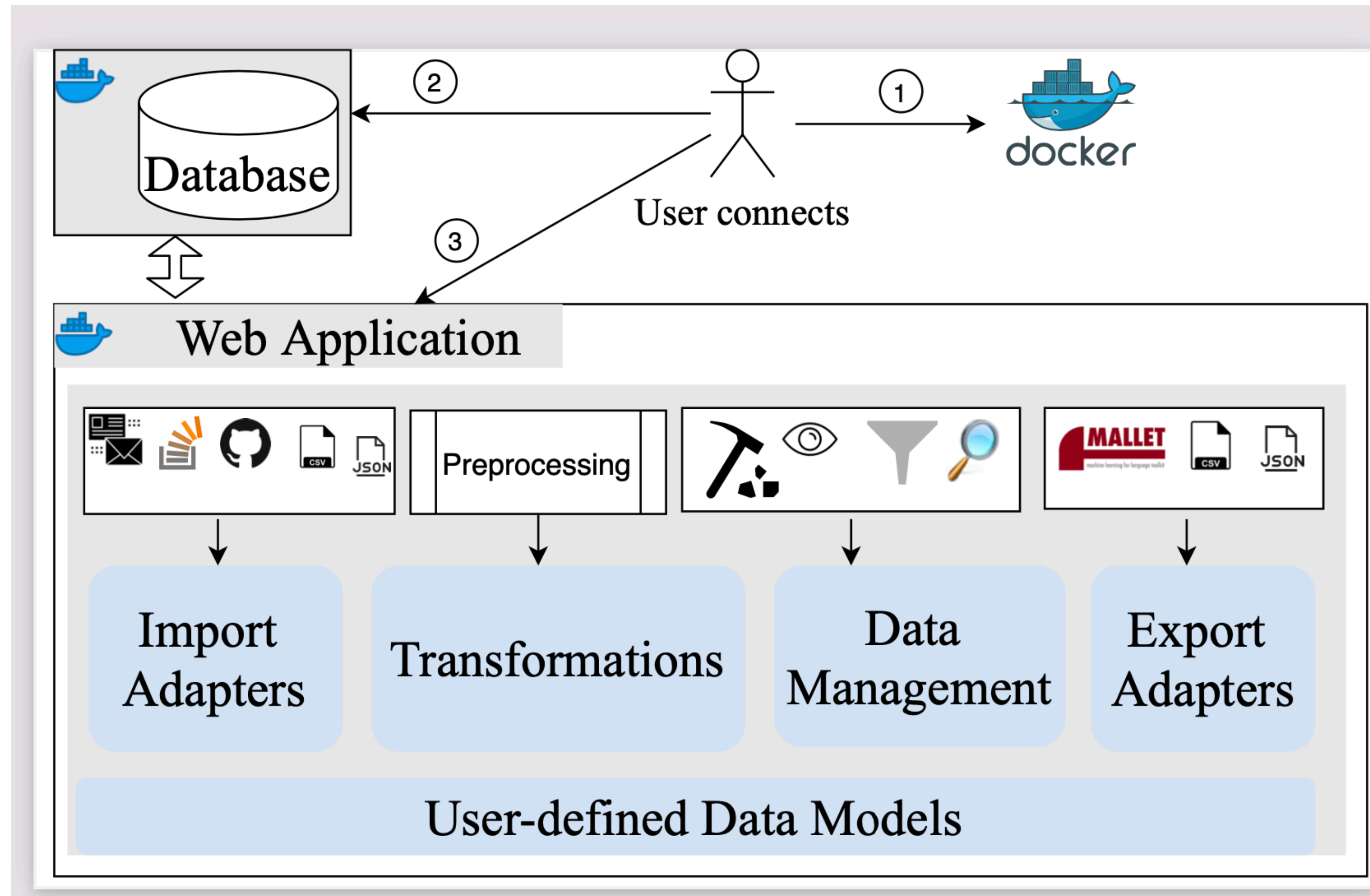
Survey developers to know which concerns are more important than others

Verifying the tool support for these concerns

Makar: A tool for Multi-source Studies



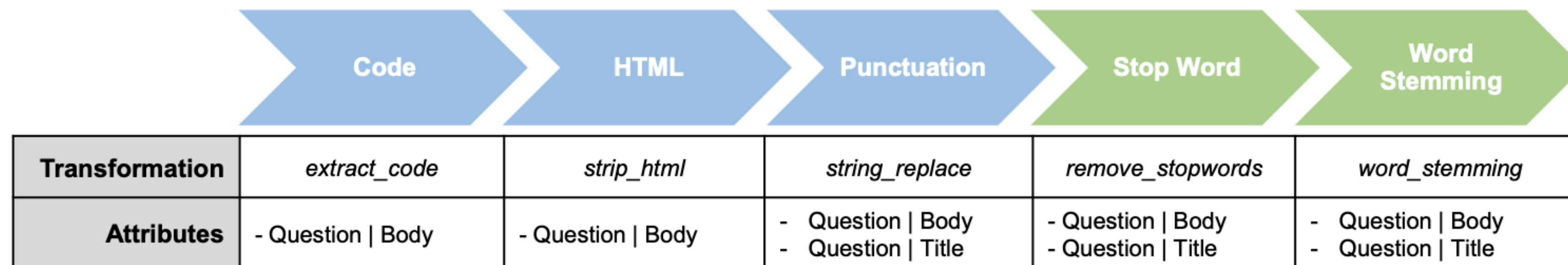
Makar Architecture



Case Study

Import adapters: Stack overflow, CSV, Apache mailing list

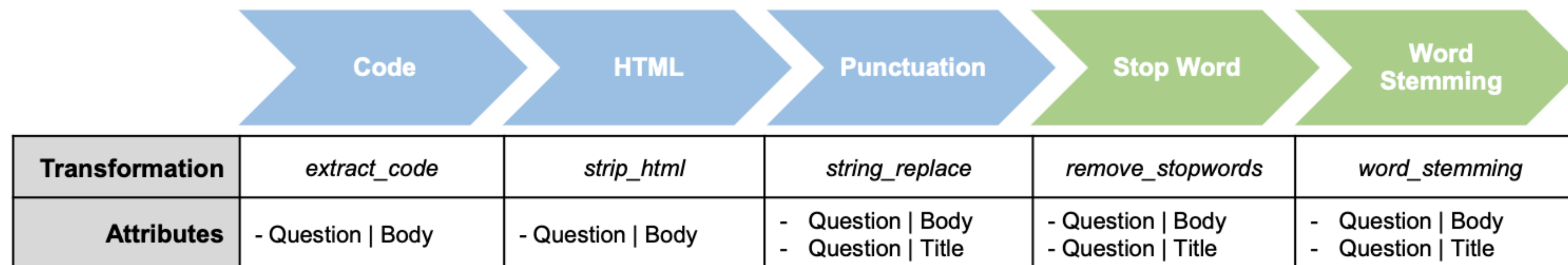
Preprocess the data:



Case Study

Import adapters: Stack overflow, CSV, Apache mailing list

Preprocess the data:



Export adapters: CSV adapters

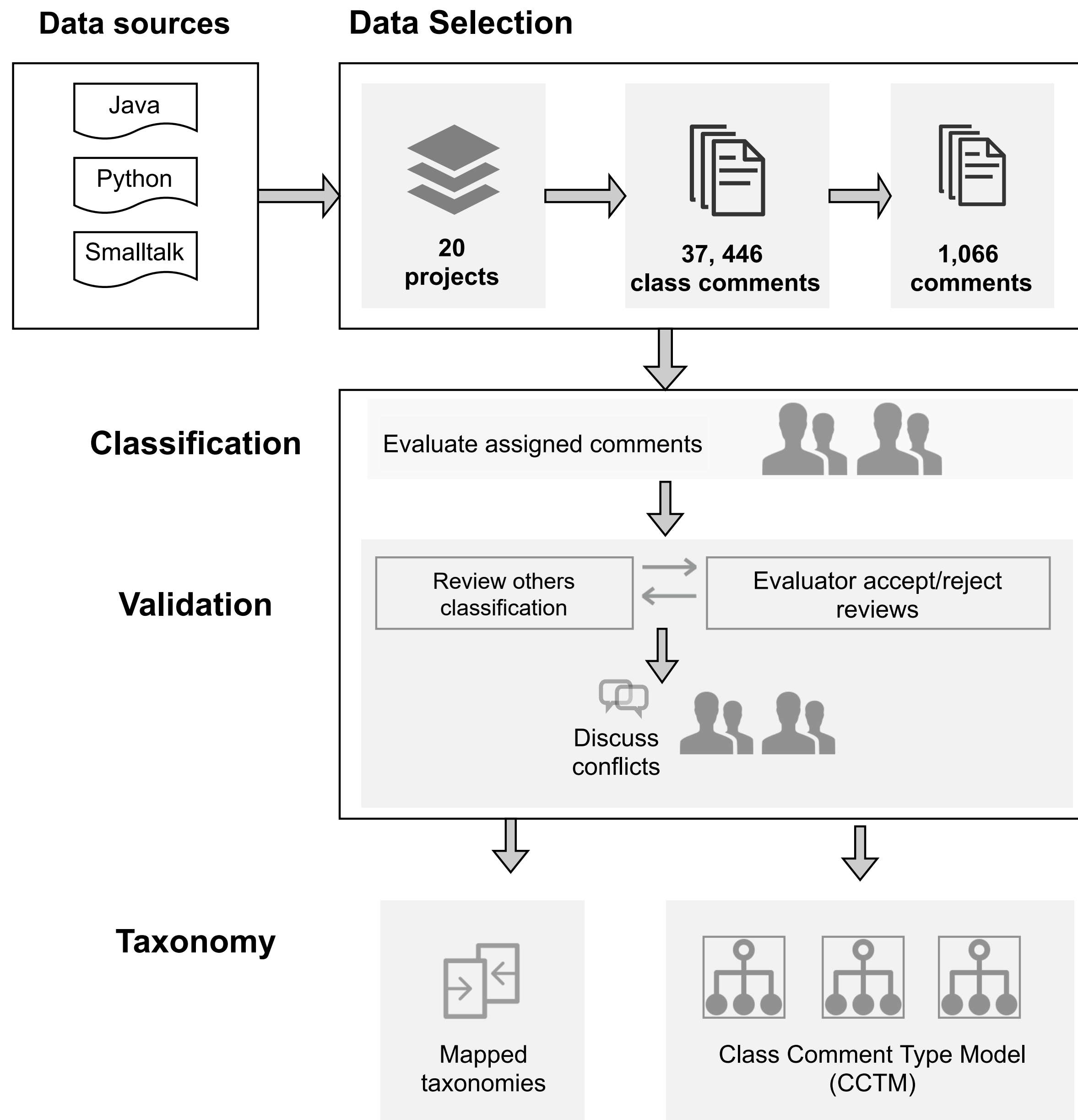
Features

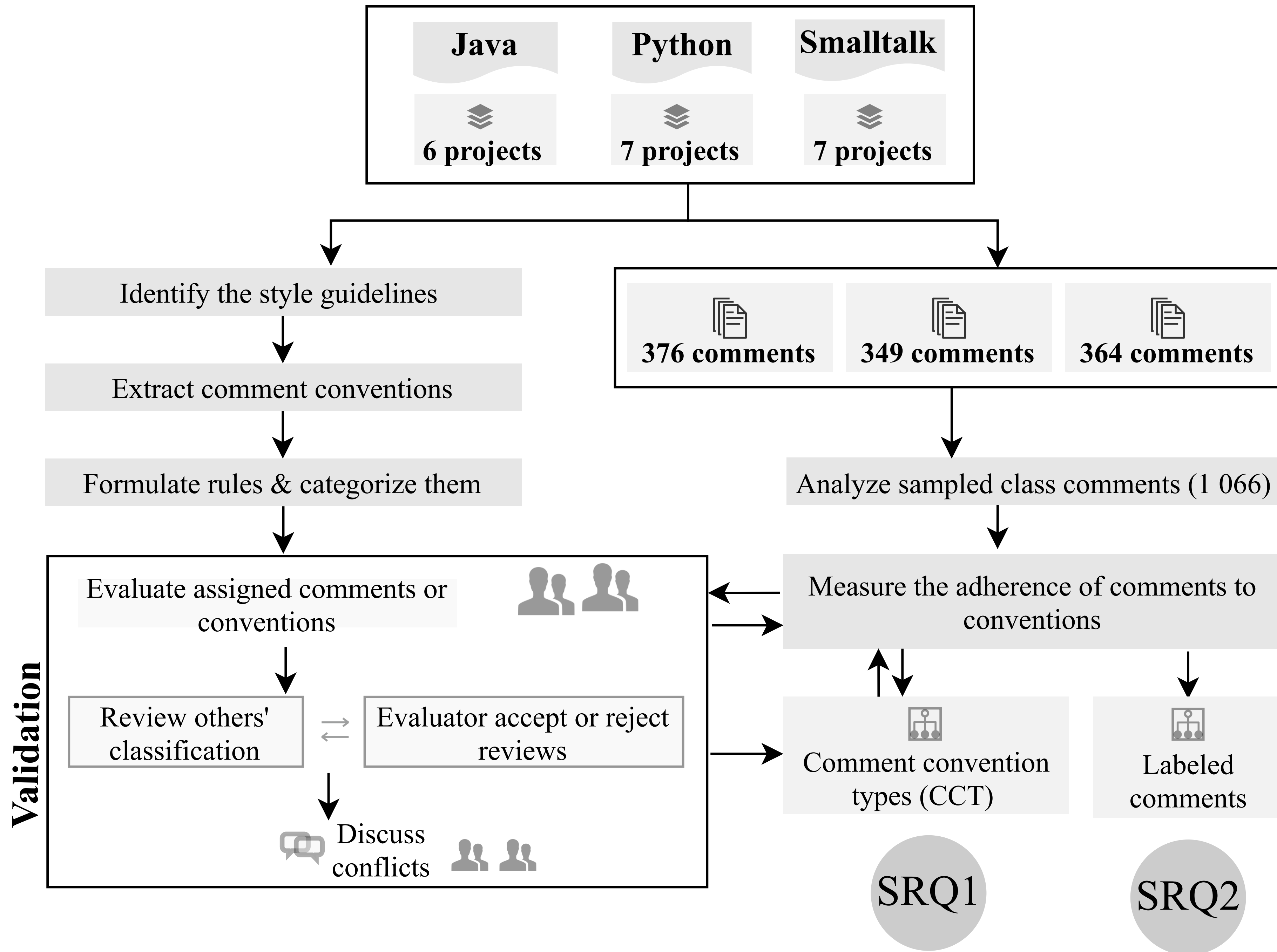
Extract data from different sources
e.g., Stack Overflow, Github, Mailing Lists

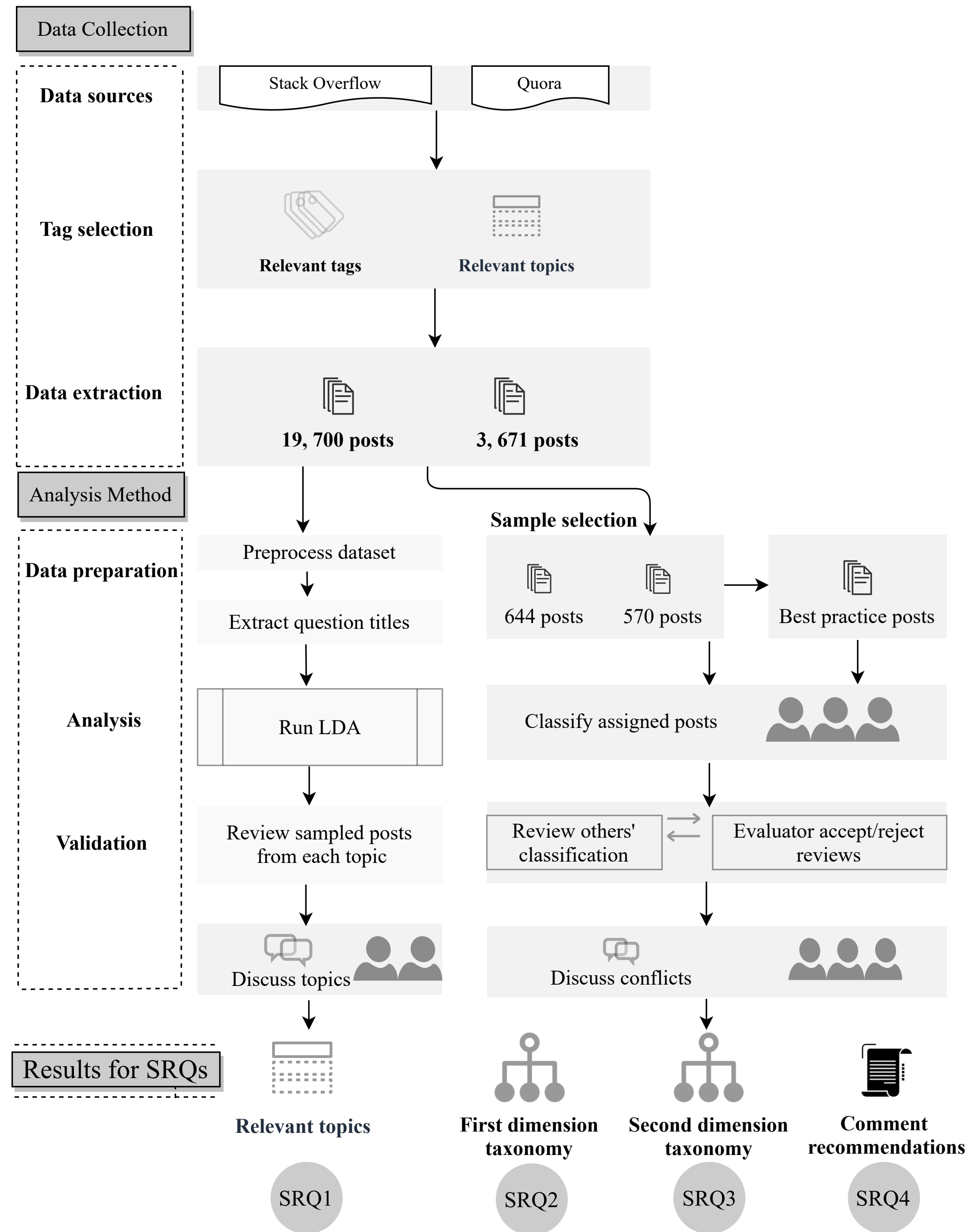
Support mapping and processing the data

Explore and perform ad-hoc searches

Extending the dataset easily







For more details, refer to the thesis

