

Comparing Bug-Fix Commits Between PyPI and Maven: A Study of Cross-Ecosystem Libraries

Higo Lab: B4 Kaya Kitada

Agenda

1

Background & Objective

2

Results

3

Discussion

4

Future Work

Agenda

1

Background & Objective

2

Results

3

Discussion

4

Future Work

Terminology

Library

- Reusable piece of software providing specific functionality.

Software Ecosystem

- **A collection of tools/libraries** built for a specific programming language.
 - Java is managed via **Maven**.
 - Python is managed via **PyPI**.

Cross Ecosystem Library

- Libraries with identical functionality **across multiple programming languages**.
- Enables users to access the same features within their preferred environment.

Related Work

1

Cross Ecosystem Dependency

2

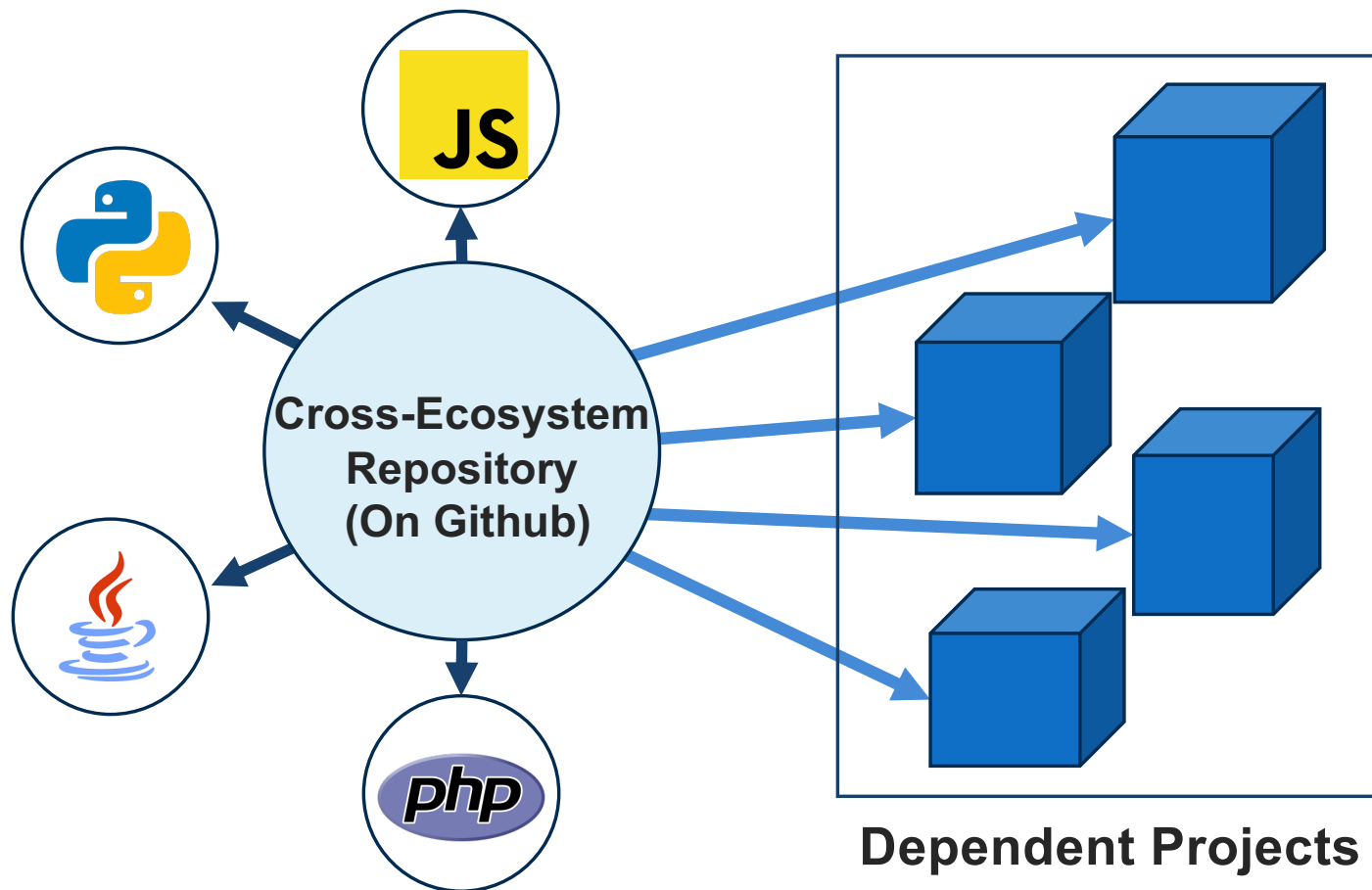
Maintenance Effort Cross-ecosystem

3

Practitioner Selection Criteria

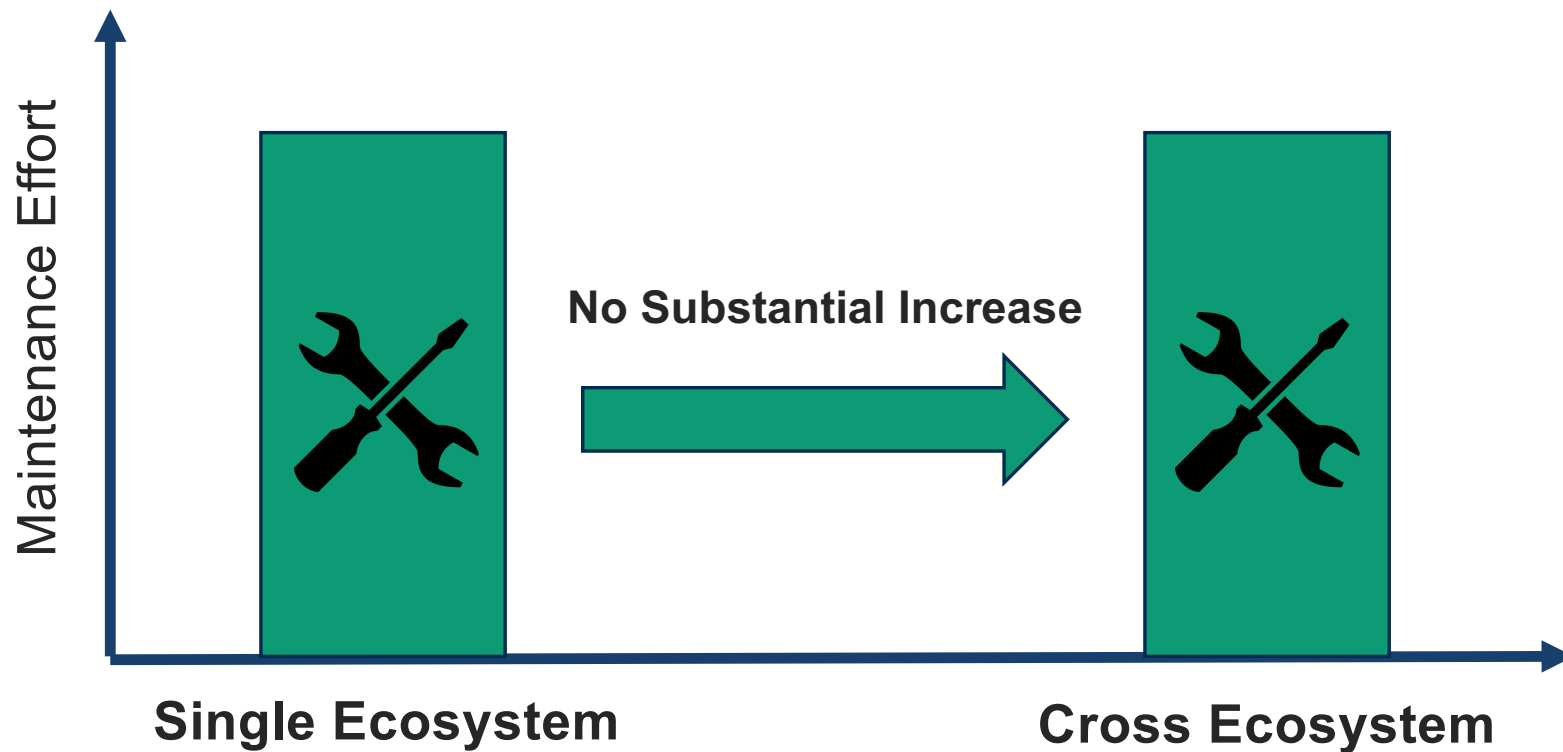
Cross Ecosystem Dependency

Thousands of cross-ecosystem libraries exist, Creating deep inter-dependencies[1].



Maintenance Effort Cross-ecosystem

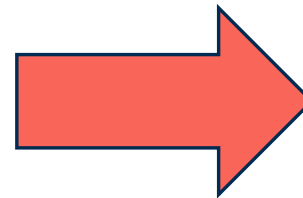
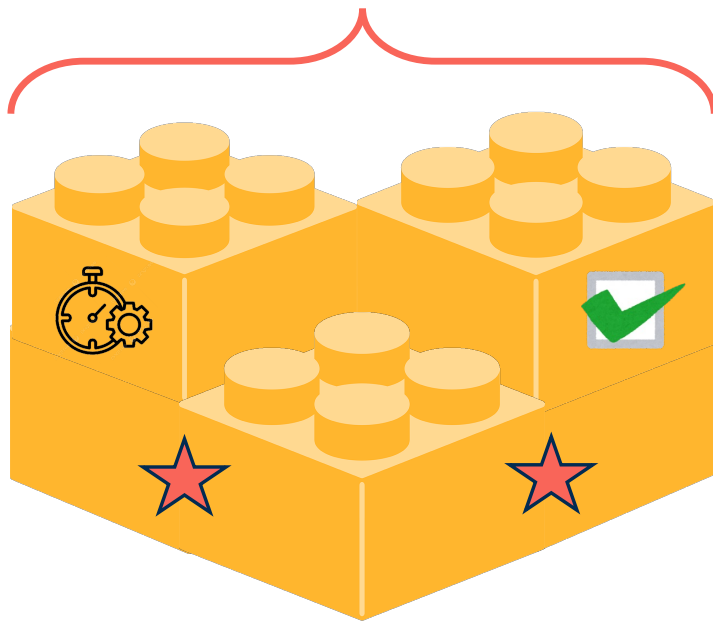
Effort typically shows no substantial increase[2].



Practitioner Selection Criteria

Consider both popularity and active maintenance for long-term safety.[3]

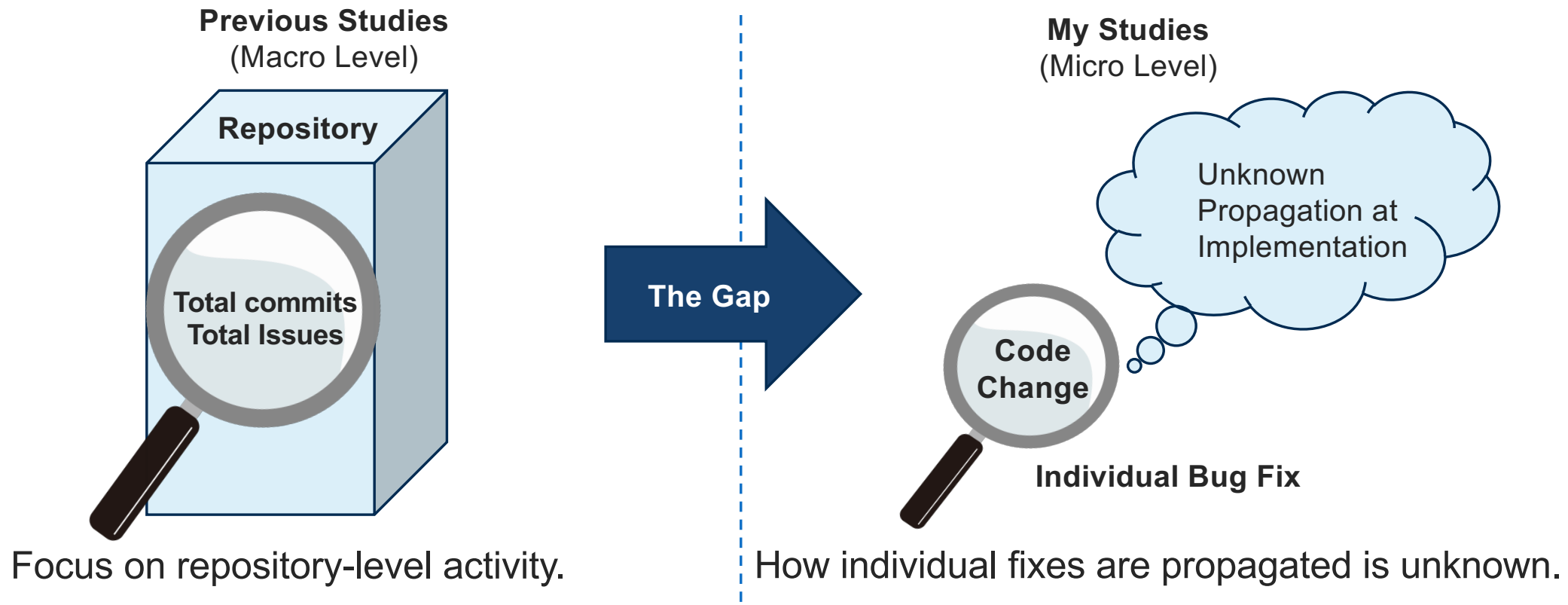
Not Only Popularity☆, But Also...



Mitigate Long-term Risks

Research Gap

Granularity & The Missing Link



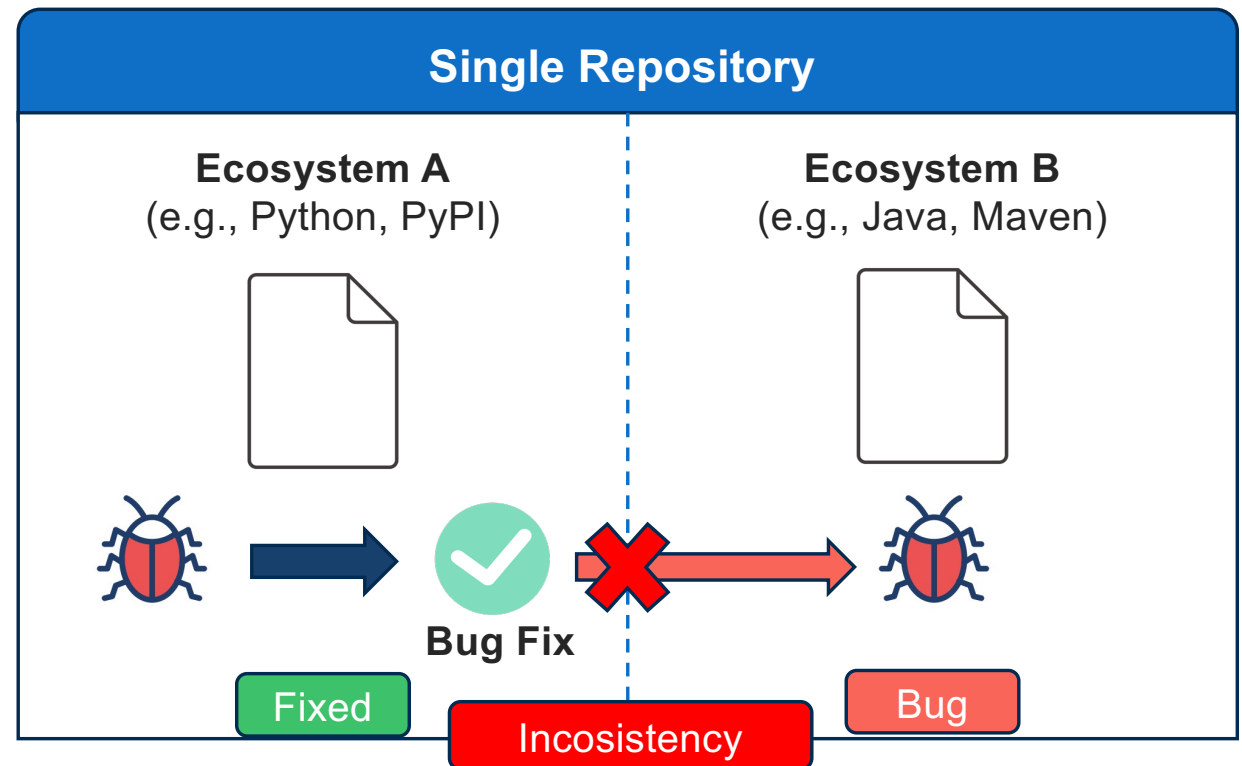
Challenge

Inconsistency In Multi Ecosystem Repositories

Asynchronous Tracking: Separate cycles obscure bug-fix synchronization.

Hidden Inconsistency: Unclear if fixes are propagated across ecosystems.

Reliability Gap: One fix might not mean "fixed in all environments."



Research Objective & Questions

- Objective
 - Investigating bug-fix propagation across dual ecosystems.
- RQ1
 - To what extent do bug-fix commits from dual-ecosystem libraries propagate to both code bases (Python and Java)?
- RQ2
 - Are there different to the kinds of bug fixes that propagate to the dual ecosystems?

Agenda



Background & Objective



Results



Discussion



Future Work

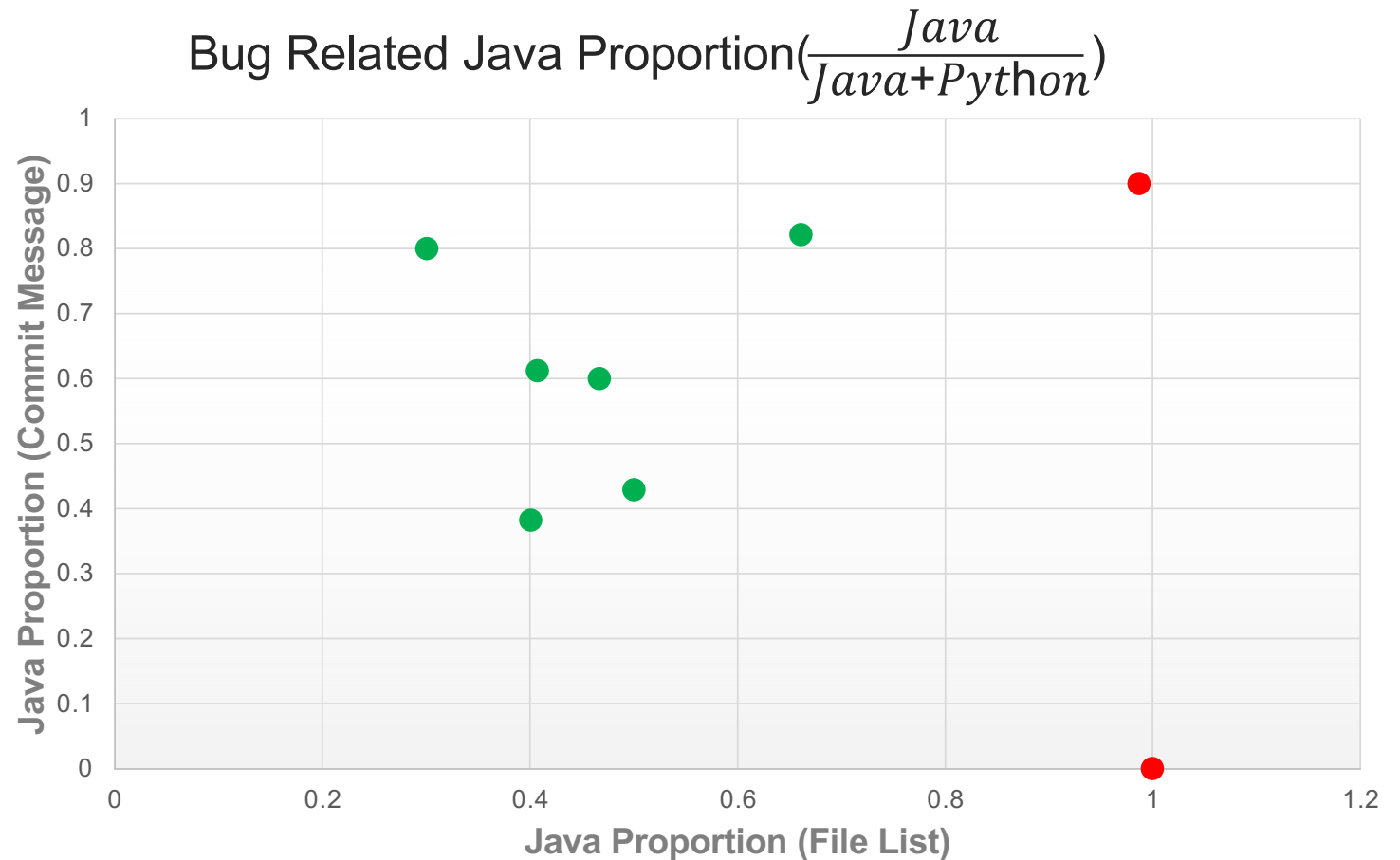
RQ1 Result: Target Repositories for Detailed Analysis

- **Balanced Maintenance Group**

- 6 projects
- 1047 total commits

- **Biased Group**

- 2 projects 429 total commits

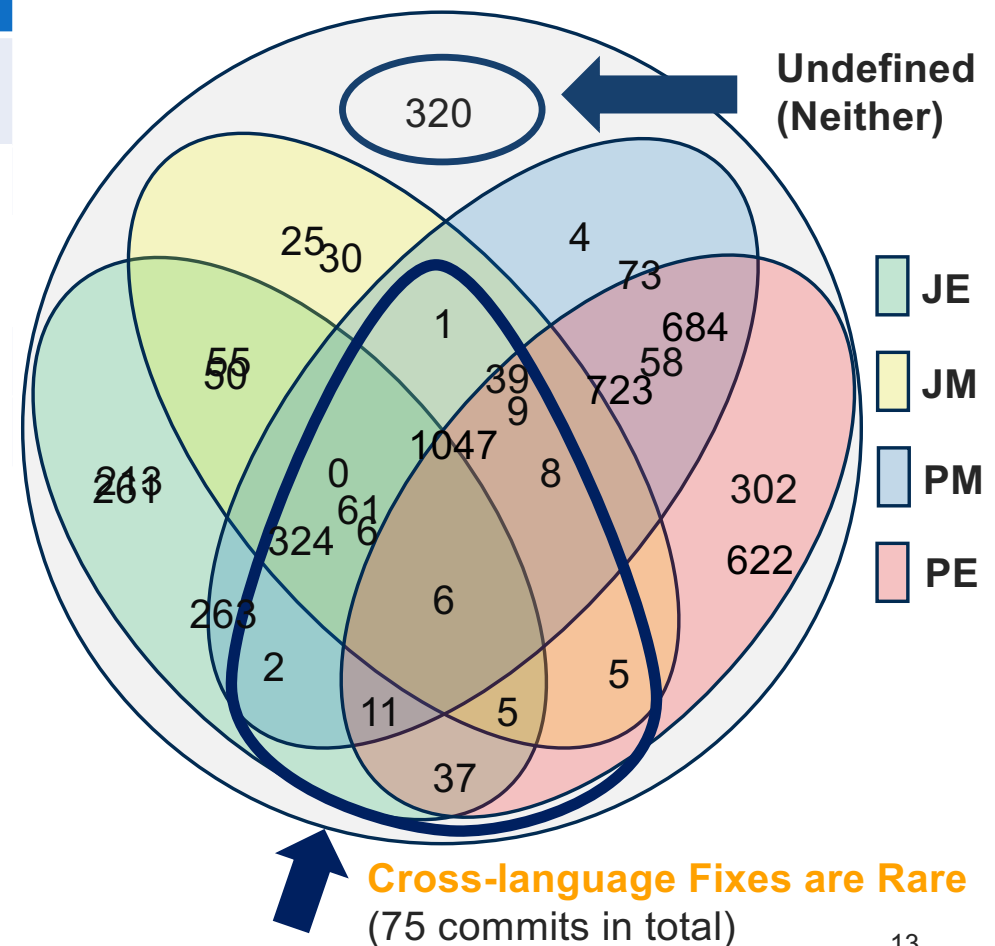


RQ1 Result - Detail

Category	Count	Percentage
Total Bug-Fix Commits	1047	100%



- **Physical fixes(Files)** are more frequent than **intent mentions(Messages)**
- **True cross-language** commits (touching both Java & Python) are a small fraction
- The large “**Neither**” group suggests limitation in keyword-based detection or commit unrelated to language specifics.



RQ1 Result – Simultaneous Maintenance Rates

Simultaneous Bug-Fix across Java & Python are Extremely Rare

- **Physical Synchronization (File level):** Modifying Java and Python files in a single commit.
- **Logical Synchronization (Message level):** Monitoring Java and Python in the commit message.

Repository	Commit Count	Physical Synchronization (%)	Logical Synchronization (%)
py4j	218	24.31%	5.96%
appengine-pipelines	52	5.77%	0.00%
appengine-mapreduce	147	0.68%	1.36%
HanLP	190	0.53%	0.53%
ome-model	392	0.51%	0.26%
appengine-gcs-client	48	0.00%	0.00%

- Most projects show extremely low synchronization, with rates often failing to reach even 1%.
- Only py4j stands out with significantly higher rates in both categories.

Agenda

1

Background & Objective

2

Results

3

Discussion

4

Future Work

Answer to RQ1 – Synchronized vs. Fragmented Maintenance

Answer: Simultaneous fixes to Java and Python files within a single repository are extremely rare (**synchronization rate $< 1\%$**).

Conclusion: Cross-ecosystem library maintenance is physically and logically "**fragmented by language**".

Discussion – Reasons for Maintenance Fragmentation

1

Dependency Structure & Fix Trends

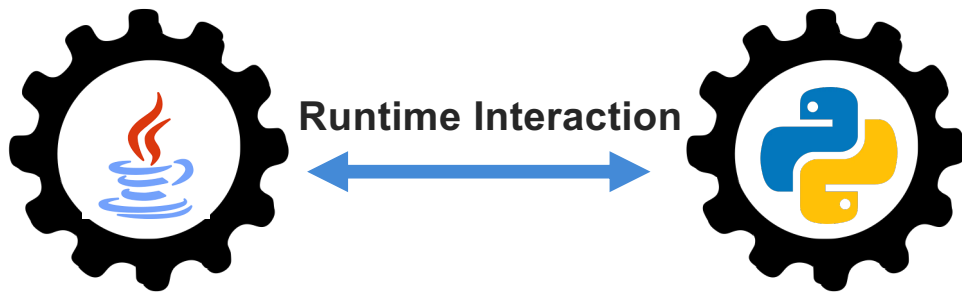
2

Observing Inconsistencies and Explicit Mentions

Dependency Structure & Fix Trends

Type1: Runtime Interdependency (e.g., py4j)

High Simultaneous Fix Rate



- Both languages interact at runtime.
- Simultaneous fixes are mandatory to prevent runtime errors

Type2: Independent Implementation (e.g., HanLP)

Low Simultaneous Fix Rate

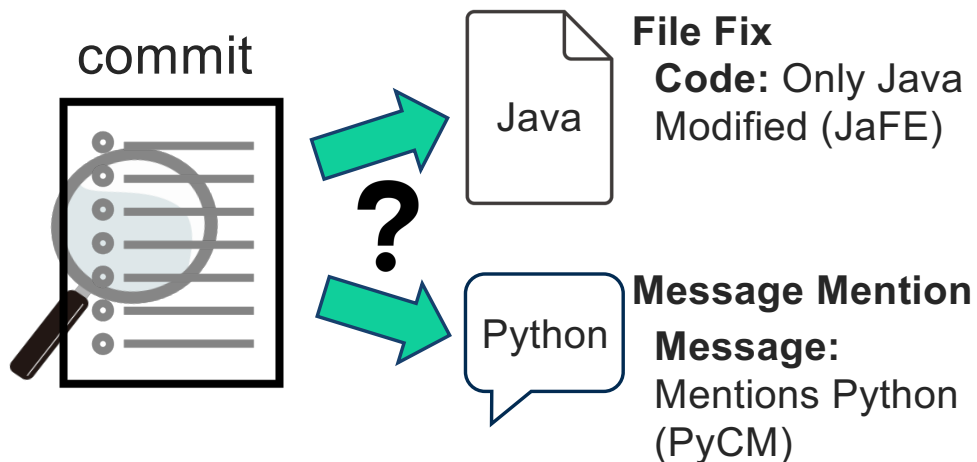


- Same functionality implemented independently.
- Fixes are typically applied separately; single-commit synchronization is rare

Key Insight: Maintenance fragmentation is strongly influenced by the project's **architectural structure**, necessitating different analysis approaches for each type.

Observing Inconsistencies and Explicit Mentions

The “Mismatch” (Files vs. Messages)



Inconsistency between modified files and mentioned language.

The “Explicit Mention” (Rarely Needed)

Usually Implicit (Omitted)

Java File Extension (JE)	324	Language obvious from context (directory). Name rarely written
Java Related Message (JM)	100	
Python Related Message (PM)	90	
Python File Extension (PE)	432	

Java & Python

Explicit (Rare):
Language named in message

Language specifically named in the message, even though often redundant

Key Insight: Mismatch are evidence of developers’ **awareness of cross-ecosystem consistency**, even when simultaneous code changes are not possible or necessary

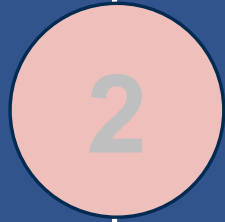
Lessons & Learns

- Sync Driven by Architectural Necessity
 - Simultaneous fixes are more prominent in structures where they are "mandatory" to avoid execution errors.
- Re-evaluating "Low Sync Rates"
 - Low sync rates in a single commit alone are insufficient to judge a project as poorly maintained.
- Toward Long-term Reliability Assessment
 - While messages show "Intent," the final state of code synchronization over time remains unverified.

Agenda



Background & Objective



Results



Discussion



Future Work

Future Work

- Precise Detection via Source Code Diff
 - Capturing "silent propagation" missed by keyword analysis.
 - Asynchronous Propagation & Time-lag Analysis
- Tracking fix delays across commits using both Issue IDs and code similarity (diffs).
 - Identifying "hidden propagation" that lacks explicit mentions.
- Generalization of Maintenance Roles
 - Expanding dataset to other pairs (e.g., C++/Rust + Python).

Reference

[1]

L. N. Erit, Y. Fan, B. Mirou, K. Matsumoto, and R. G. Kula,
“Intertwining Communities: Exploring Libraries that Cross Software Ecosystems,”
arXiv:2303.09177 [cs.SE], Mar. 2023.

[2]

K. Kannee, R. G. Kula, S. Wattanakriengkrai, and K. Matsumoto,
“Should Maintainers Deploy to Cross-Ecosystems? An Analysis of Packages from PyPI and NPM,”
IEICE Transactions on Information and Systems, Advance Publication, 2025.

[3]

E. Larios Vargas, M. F. Aniche, C. Treude, M. Bruntink, and G. Gousios,
Selecting Third-Party Libraries: The Practitioners’ Perspective,
Proc. ESEC/FSE 2020, pp. xxx–xxx.
doi:10.1145/3368089.3409711

[4]

GitHub, “GitHub REST API Documentation,” <https://docs.github.com/en/rest>, accessed 2025.

[5]

D. Spadini, M. Aniche, and A. Bacchelli,
“PyDriller: Python Framework for Mining Software Repositories,”
SANER 2019.