MaRV

A Manually Validated Refactoring Dataset (FORGE 2025)

Henrique Nunes
Federal University of Minas Gerais

Eduardo Figueiredo
Federal University of Minas Gerais

Tushar Sharma
Dalhousie University

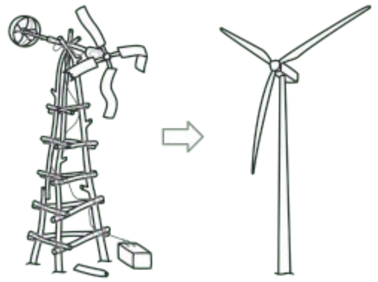
Whats is refactoring?

Refactoring reduces complexity and improves maintainability.

It removes code smells and addresses technical debt.

Research areas in refactoring:

- Cataloging techniques
- Identifying refactoring candidates
- Automated refactoring tools
- Measuring impact on maintainability



Challenges in Refactoring

Manual refactoring is costly and slow.

Automated tools are semi-automated and often inaccurate.

Large Language Models (LLMs) offer a new approach but have limitations:

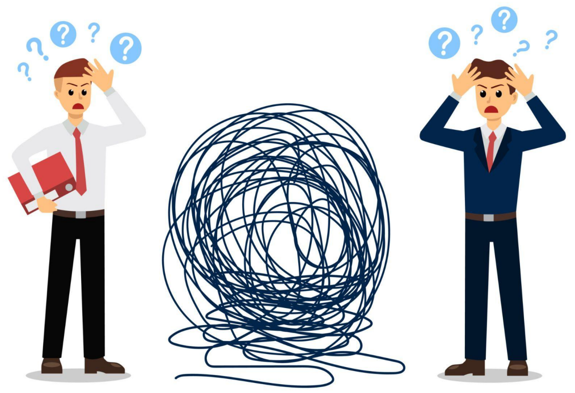
- Introduce new errors
- Cause unintended behaviors
- Require high-quality datasets

Lack of Refactoring Dataset

Refactoring datasets rely on tools like RefactoringMiner.

Issues with current datasets:

- Ambiguities
- False positives
- Lack of evaluation of tool outputs



MaRV: **Ma**nually **V**alidated **R**efactoring Dataset

Contains code snippets before and after refactoring.

Includes metadata:

- Manually annotated refactoring techniques
- Affected code elements
- Commit details

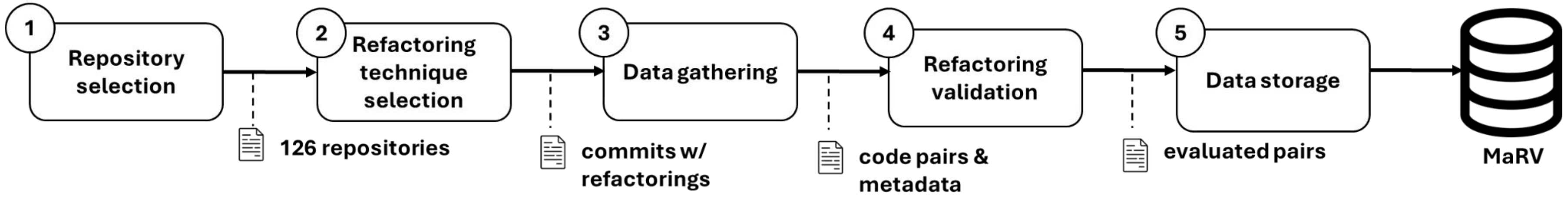
Why MaRV is Different?

Unlike previous datasets, MaRV:

- Captures manually validated refactorings
- Reduces noise and false positives.
- Provides detailed metadata for replicability.

Study Design

Steps to create MaRV dataset:



Repository Selection

Tool used: SEart GitHub search[1]

Criteria for selecting Java repositories:

- Active, popular: At least 10 contributors, **10,000 commits**, 1,000 stars
- At least one update in the past 12 months

Identified 172 repositories

[1] D. Ozren, E. Aghajani, and . Bavota. *"Sampling projects in github for MSR studies"*, MSR, 2021.

Refactoring Technique Selection

102 different refactoring techniques identified by RefactoringMiner

Focused on techniques in the top 75% of occurrence frequency (Q3): It results in 7 refactoring techniques.

We selected 4 refactoring techniques: (1) Extract Method, (2) Rename Method, (3) Remove Parameter and (4) Rename Variable

These techniques are frequently used and applicable in refactoring research

Data Gathering Process

Automatic processing refactoring data:

1. Filter the 4 refactoring techniques
2. Extracts code diffs, before and after the refactoring
3. Stores metadata (commit SHA, repository name, etc.)

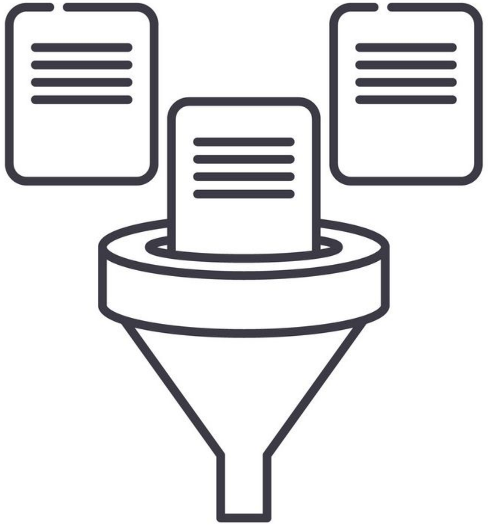
Each refactoring generates 3 files: (1) Original snippet, (2) Refactored snippet and (3) Metadata

Data Gathering Process: Filter

Filters applied to ensure suitable snippets:

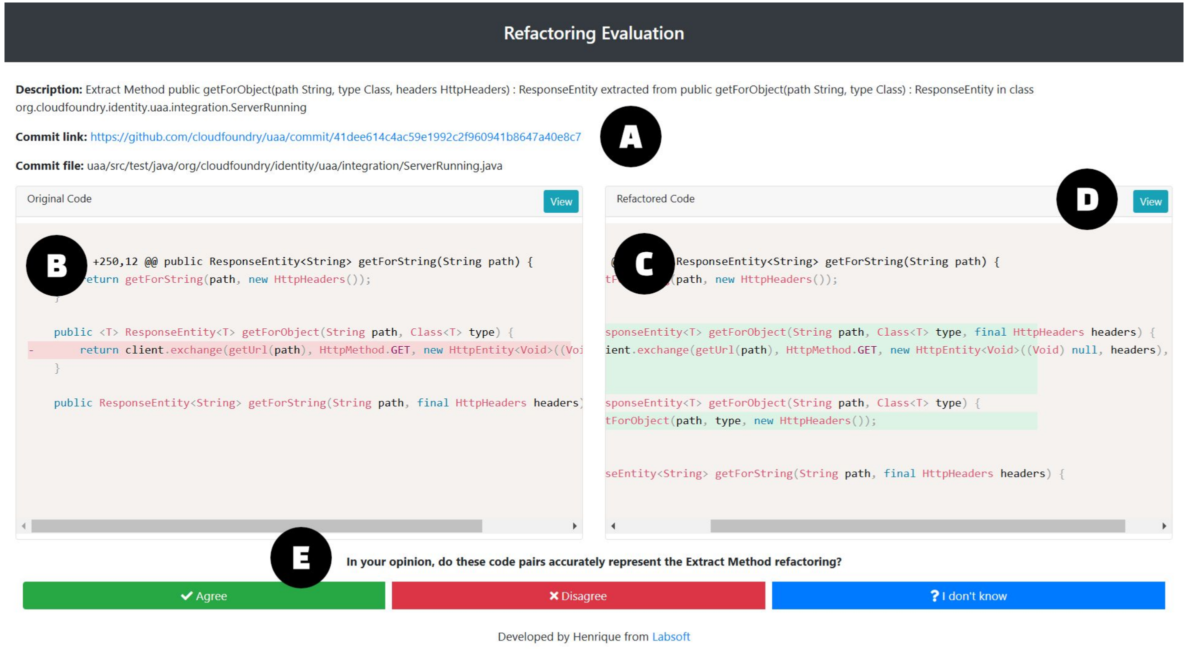
1. Snippets with at least 100 lines
2. Non-empty files
3. At least one method declaration

SQL file used to populate the database (MySQL)



Refactoring Validation Tool

Developed Web tool for manual validation



Description: Extract Method public getForObject(path String, type Class, headers HttpHeaders) : ResponseEntity extracted from public getForObject(path String, type Class) : ResponseEntity in class org.cloudfoundry.identity.uaa.integration.ServerRunning

Commit link: https://github.com/cloudfoundry/uaa/commit/41dee614c4ac59e1992c2f960941b8647a40e8c7

Commit file: uaa/src/test/java/org/cloudfoundry/identity/uaa/integration/ServerRunning.java

```
+250,12 @@ public ResponseEntity<String> getForString(String path) {
eturn getForString(path, new HttpHeaders());


    public <T> ResponseEntity<T> getForObject(String path, Class<T> type) {
-       return client.exchange(getUrl(path), HttpMethod.GET, new HttpEntity<Void>((Voi
    }

    public ResponseEntity<String> getForString(String path, final HttpHeaders headers
```

```
 ResponseEntity<String> getForString(String path) {
tF  (path, new HttpHeaders());


sponseEntity<T> getForObject(String path, Class<T> type, final HttpHeaders headers) {
ient.exchange(getUrl(path), HttpMethod.GET, new HttpEntity<Void>((Void) null, headers),


sponseEntity<T> getForObject(String path, Class<T> type) {
tForObject(path, type, new HttpHeaders());


seEntity<String> getForString(String path, final HttpHeaders headers) {
```

In your opinion, do these code pairs accurately represent the Extract Method refactoring?

Developed by Henrique from Labsoft

Manual Validation and Data Storage

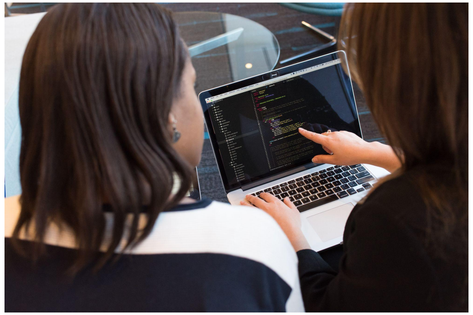
Code snippets presented side by side

Reviewers mark if the refactoring is correct

Feedback stored (in database) and converted for JSON format

40 participants evaluated 693 refactorings

Each refactoring was evaluated by 2 participants



Results

Quantity of refactorings evaluated by a pair of participants:

	Votes	Count	Total
Consensus	[disagree, disagree]	84	405
	[agree, agree]	321	
Conflict	[disagree, agree]	217	285
	[I don’t know, disagree]	27	
	[I don’t know, agree]	41	
Other	[I don’t know, I don’t know]	3	3
	Total		**693**

Potential Applications: Refactoring Benchmark

Manually validated refactorings provide a ground truth

Improved accuracy and reliability of refactoring identification tools

Potential Applications: Refactored Code Generation

Provides examples of code before and after refactoring

Helps train models to generate refactored versions of existing code

Eg. Few-shot learning (examples for contextualization)

Threats to Validity

Dataset size:

- Number of refactorings instances
- Quantity of refactoring techniques

Compilation and tests for the instances.

Human evaluation conflicts

Contributions

A manually validated dataset of 693 refactored code pairs.

A web-based tool for easy validation of refactorings.

Raw RefactoringMiner output and extraction scripts for 126 repositories.

Current Work

Addressing the conflicts in human evaluations

Creating a benchmark (compilation and tests)

Performing a new round of human evaluation

Thank you!