An Empirical Evaluation of Method Signature Similarity in Java Codebases

Mohammad Taha Khan, Mohamed Elhussiny, Billy Tobin and Muhammad Ali Gulzar

Method Signatures in Java

A method signatures uniquely identify methods within a class

It includes the method name, its parameters, and return type

Method Signatures in Java

A method signatures uniquely identify methods within a class

It includes the method name, its parameters, and return type

```
package java.util;
public class Arrays {

 @param a the array to be sorted */
 public static void parallelSort(short[] a) {
   int n = a.length;
   if (n <= MIN_ARRAY_SORT_GRAN) {
       . . .
   }
. . .
}
```

Implementation of the **`parallelSort`** method from the **`java.util.Arrays`** class in the Java Standard Library.

Method Signatures in Java

A method signatures uniquely identify methods within a class

It includes the method name, its parameters, and return type

Class Name	Arrays
Method Name	parallelSort
Return Type	void
Input Parameters	[short[] a]

Method Signature

```
package java.util;
public class Arrays {

    @param a the array to be sorted */
    public static void parallelSort(short[] a) {
        int n = a.length;
        if (n <= MIN_ARRAY_SORT_GRAN) {
            . . .
        }
    . . .
    }
```

Implementation of the **`parallelSort`** method from the **`java.util.Arrays`** class in the Java Standard Library.

Method Signature Similarity

Overloaded Methods

```
class Array {
// . . .
 public static void parallelSort (short [] a) {
  int n = a.length , p , g ;
  if ( n <= MIN_ARRAY_SORT_GRAN || // . . .
}

 public static void parallelSort (int [] a) {
  int n = a.length , p , g ;
  if ( n <= MIN_ARRAY_SORT_GRAN || // . . .
}
```

Method Signature Similarity

Overloaded Methods

```
class Array {
// . . .
 public static void parallelSort (short [] a) {
  int n = a.length , p , g ;
  if ( n <= MIN_ARRAY_SORT_GRAN || // . . .
}

 public static void parallelSort (int [] a) {
  int n = a.length , p , g ;
  if ( n <= MIN_ARRAY_SORT_GRAN || // . . .
}
```

Textually Similar Methods

```
public class ClearOperation {
 @Override
 public int getSyncBackupCount () {
  return
mapServiceContext.getMapContainer(name).getBackupCount();
 }


 @Override
 public int getAsyncBackupCount () {
  return
mapServiceContext.getMapContainer(name).getAsyncBackupCount()
;
}
```

Method Signature Similarity

Overloaded Methods

Textually Similar Methods

Methods with similar names or ones overloaded frequently can lead to **confusion** and increased **cognitive load** for developers.

This leads to potential **misuse** of these methods, leading to **errors** that are hard to **identify** and **debug**.

Method Signature Similarity

```
public class ClearOperation {
 @Override
 public int getSyncBackupCount () {
  return
mapServiceContext.getMapContainer(name).getBackupCount();
 }


 @Override
 public int getAsyncBackupCount () {
  retur
mapServiceContext.getMapContainer(name).getAsyncBackupCount()
;
}
```

Research Goals

Assess how **widespread** method signature similarity in real-world Java codebases

Explore the effects of both overloaded and textually similar methods on code **quality**, **maintainability** and developer **productivity**

Research Questions

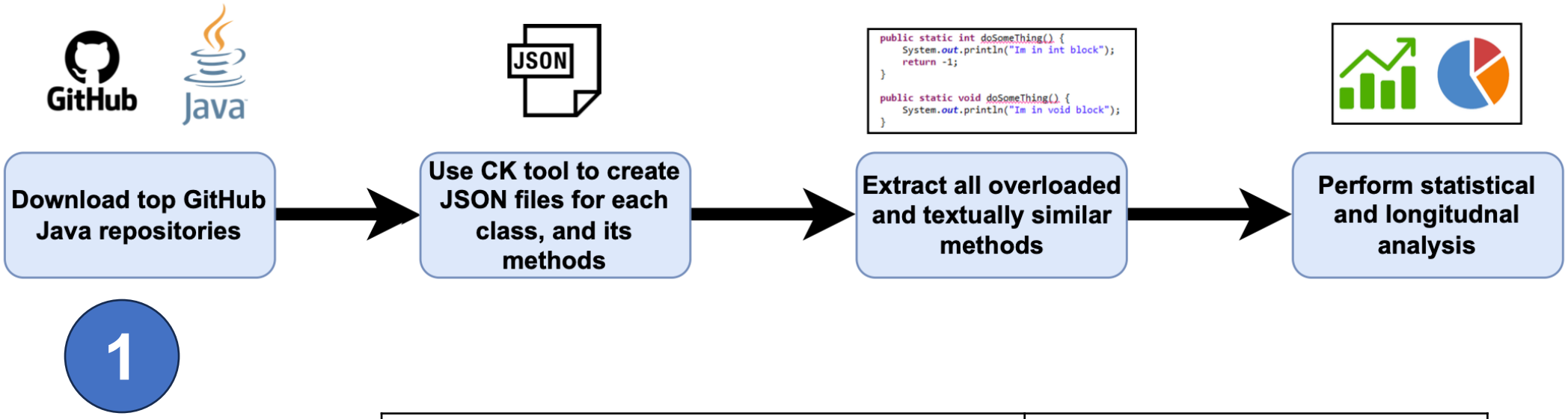
How **prevalent** are methods with similar signatures in large scale codebases?

How **frequently** developers use methods with similar signatures?

Do methods with similar signatures have **strong correlation** with certain **codebase** characteristics?

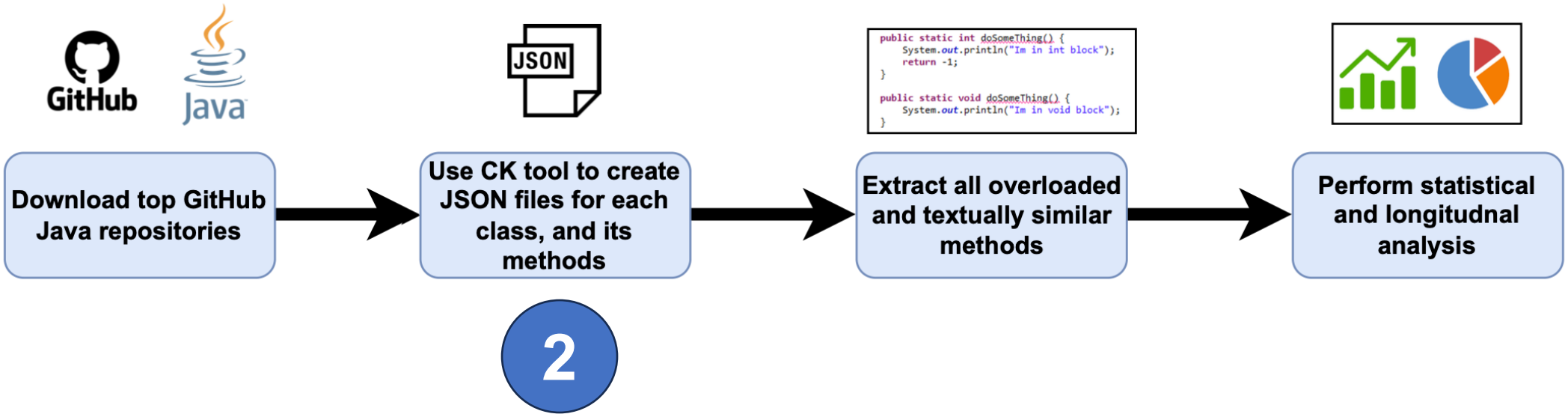
How do methods with similar signatures **evolve** as software matures?

Methodology



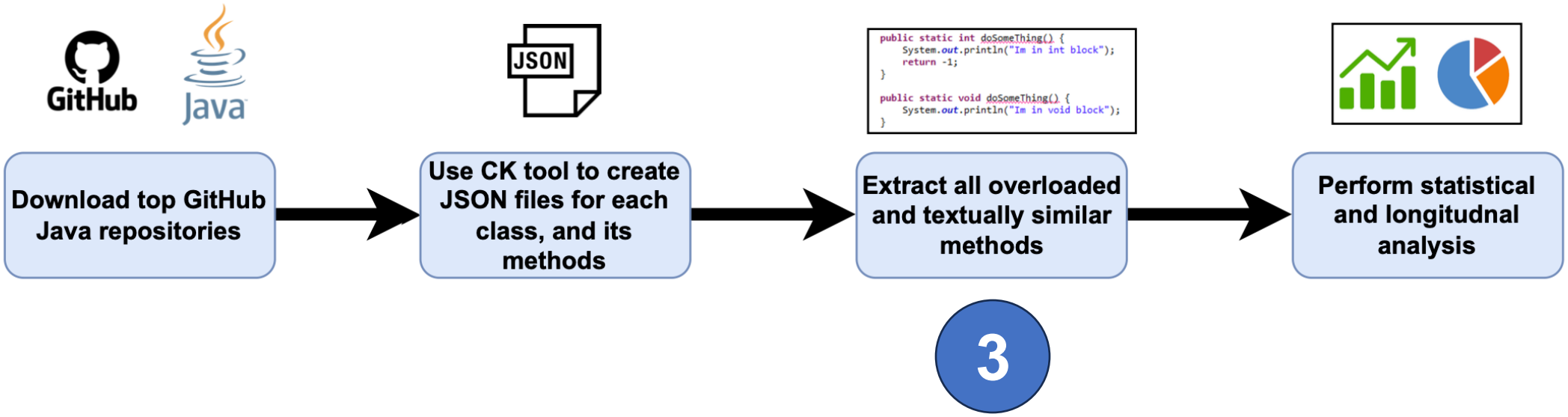
Total Repositories Collected	167 Repositories
Github Stars Range	>3500 Stars
Total Lines of Code Analyzed	~6,400,000 LOC
Total Methods	~1,900,000 Methods
Average Age of Repositories	~ 9 years

Methodology



CK Java analysis tool is an **open-source** tool that specializes in **measuring** software **metrics** in Java codebases

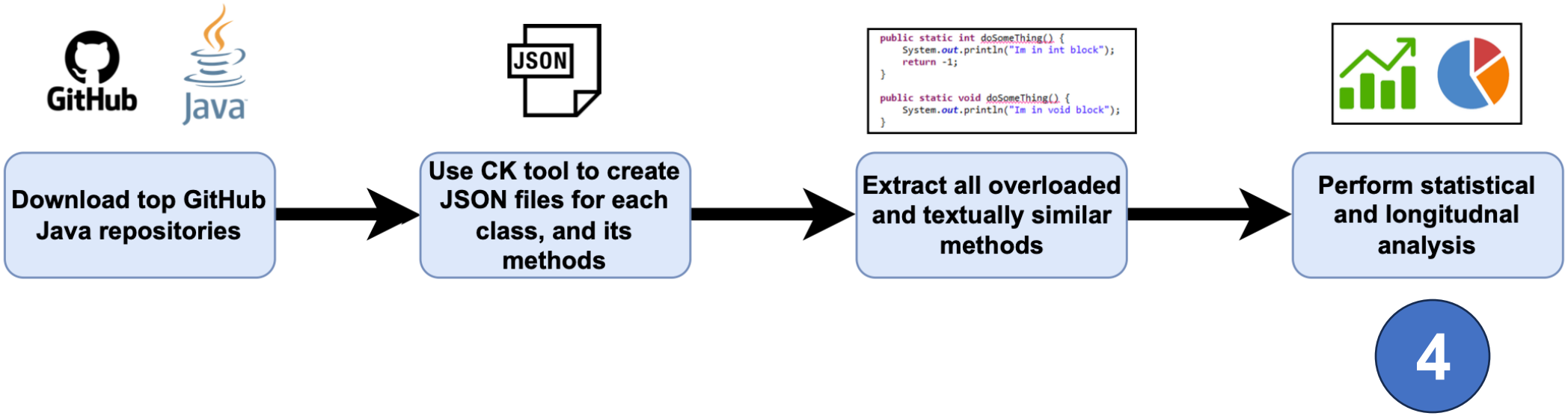
Methodology



Extracting **overloaded** methods was **straightforward**

To identify **textually similar methods**, used the **edit distance** between method names

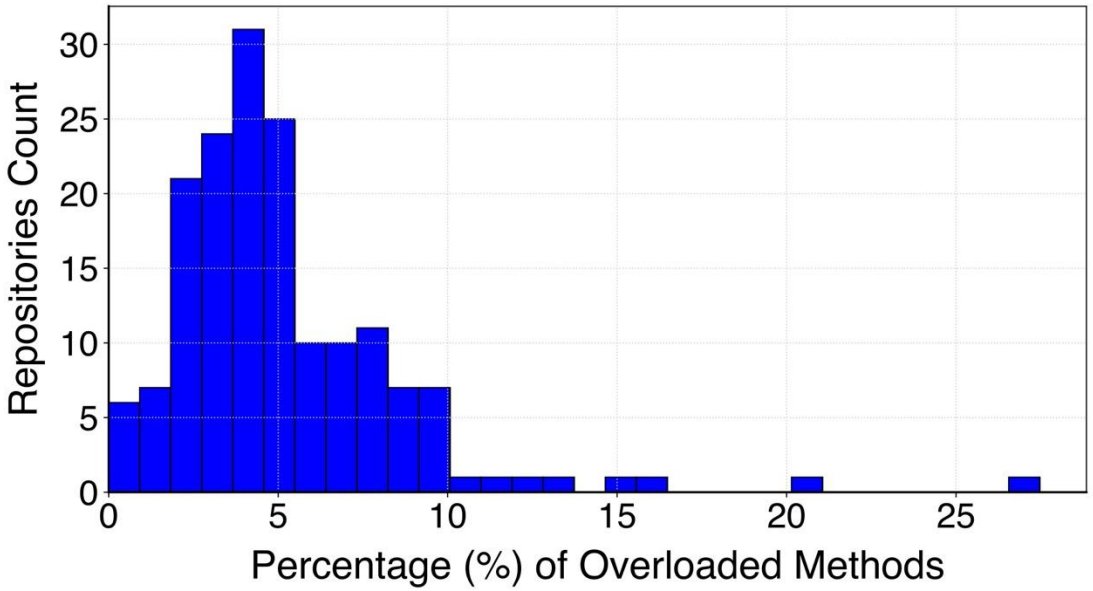
Methodology



For **longitudinal analysis** we filtered out **40** repositories at random

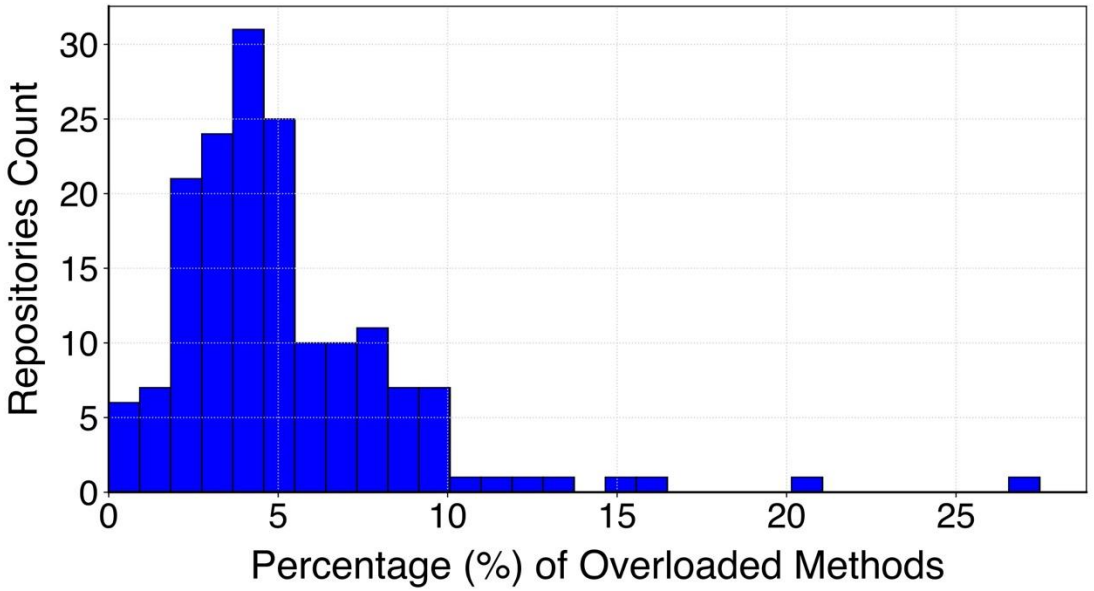
Collected **25 snapshots** evenly distributed across the repository **lifetime**

Prevalence of Methods

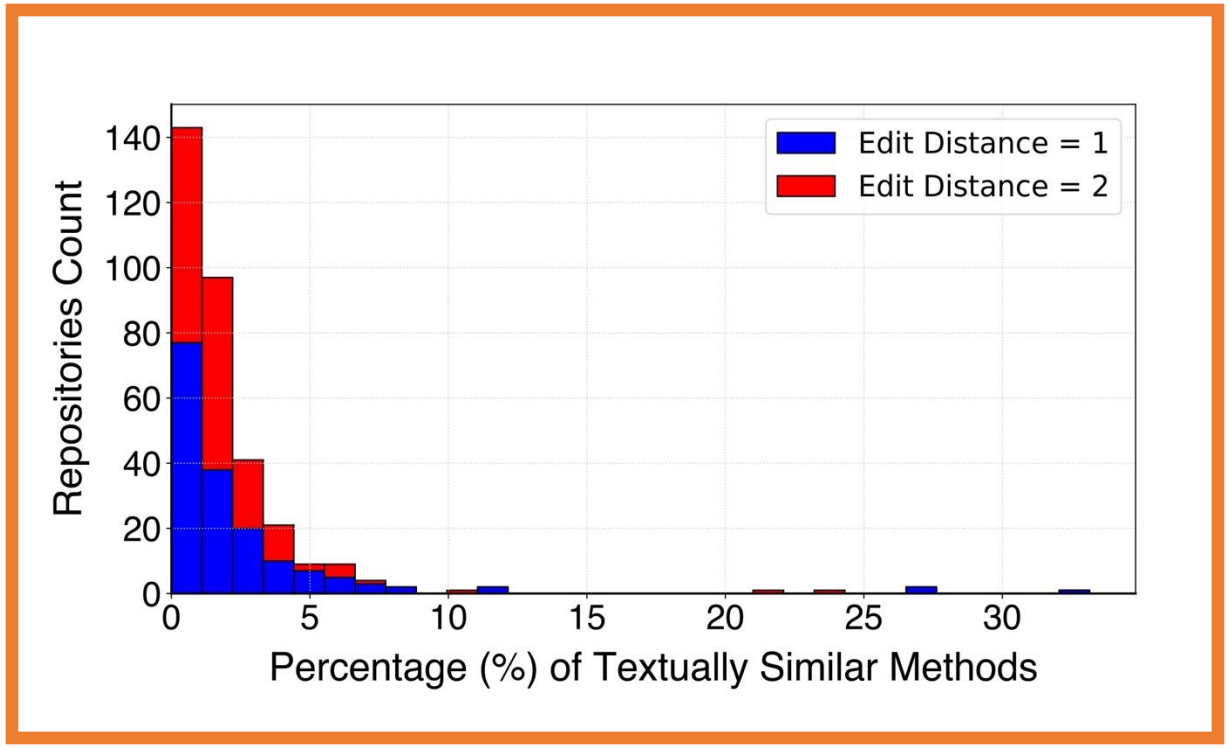


Overloaded Methods

Prevalence of Methods

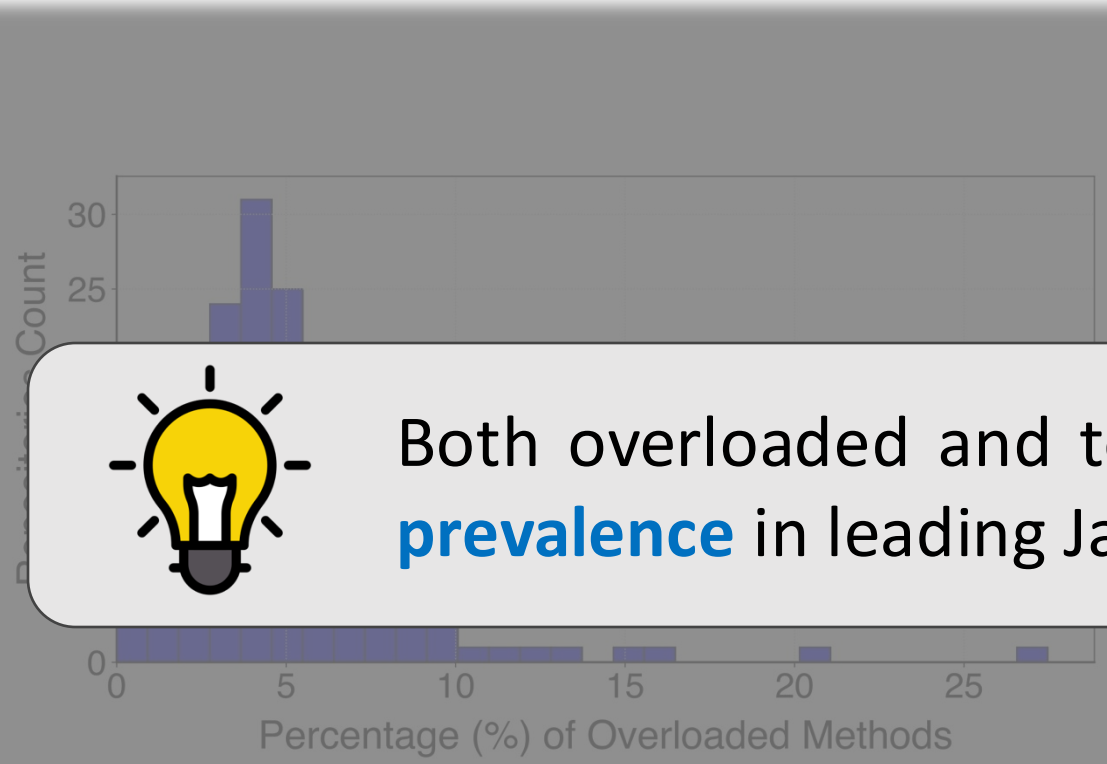


Overloaded Methods

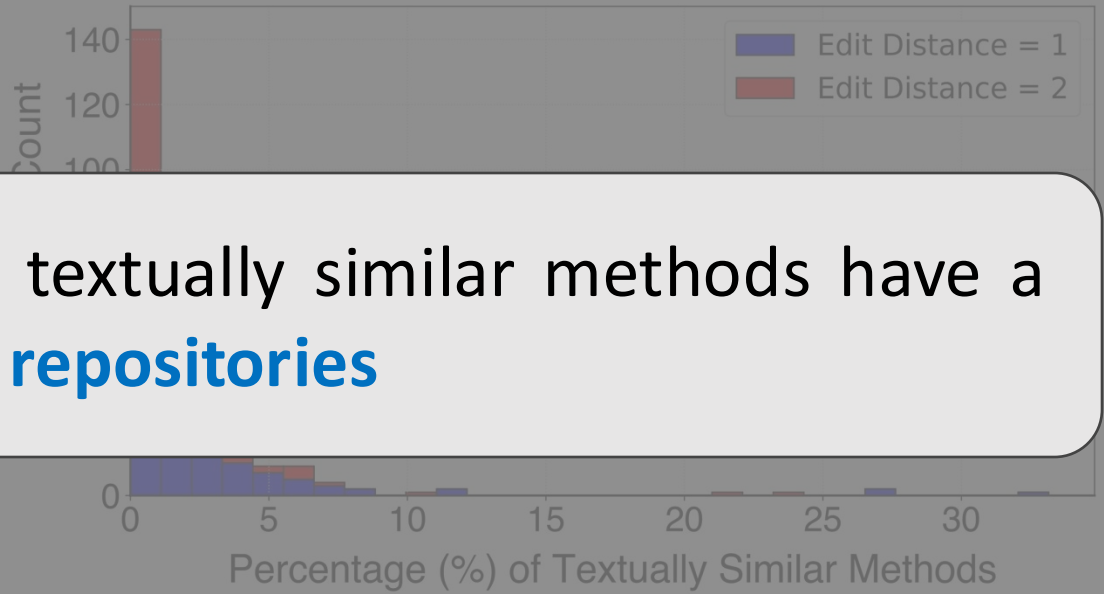


Textually Similar Methods

Prevalence of Methods



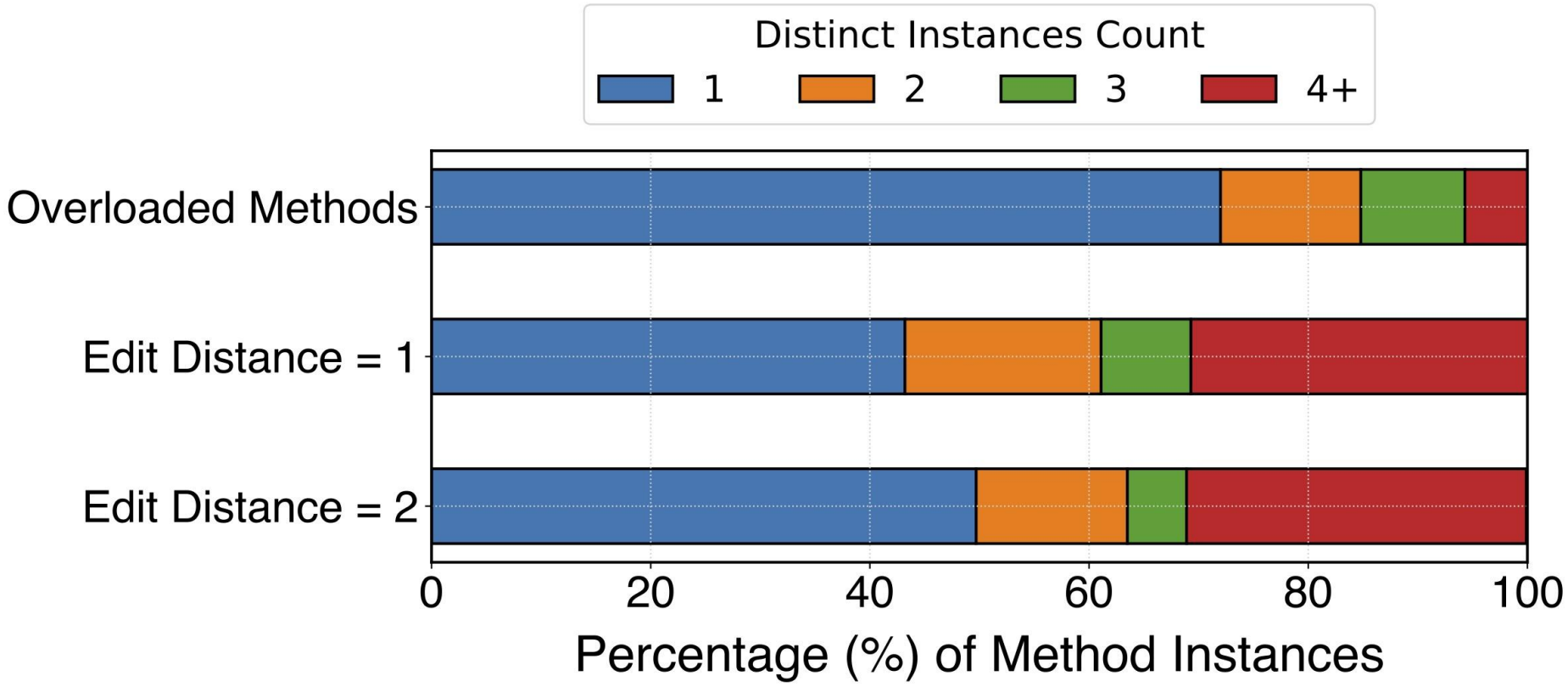
Overloaded Methods



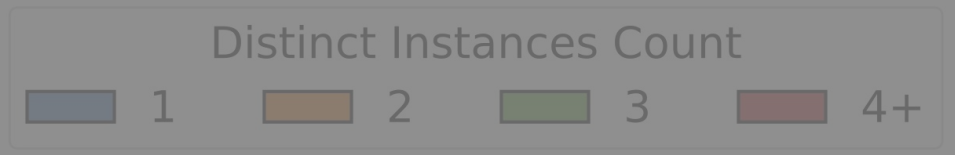
Textually Similar Methods

Both overloaded and text textually similar methods have a **prevalence** in leading Java **repositories**

Frequency of Methods



Frequency of Methods

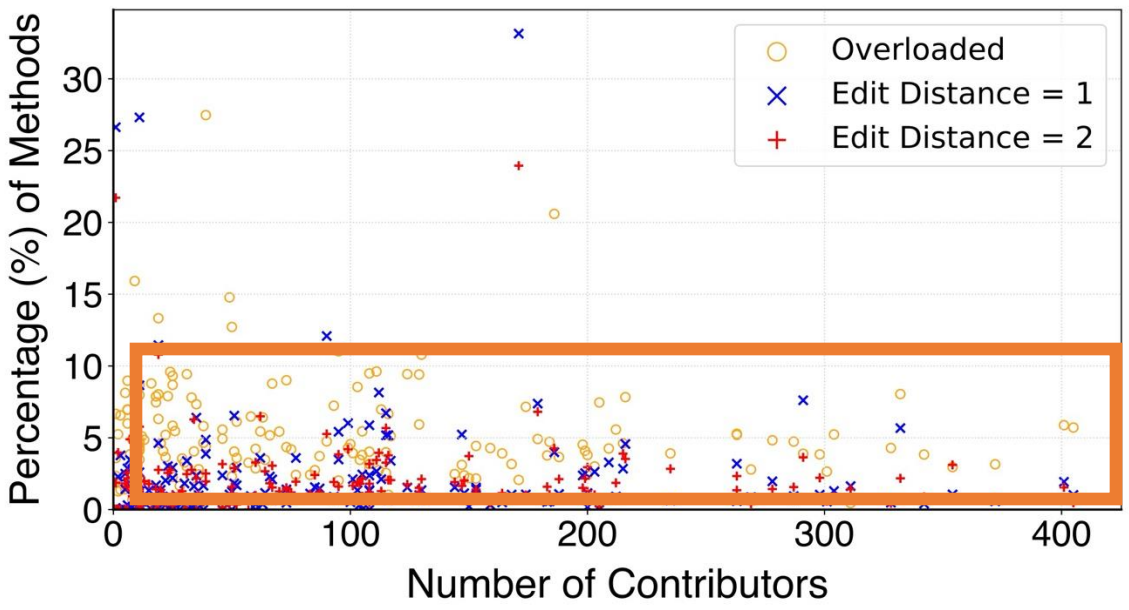


It is **uncommon** for a method to be **overloaded** more than **three times**

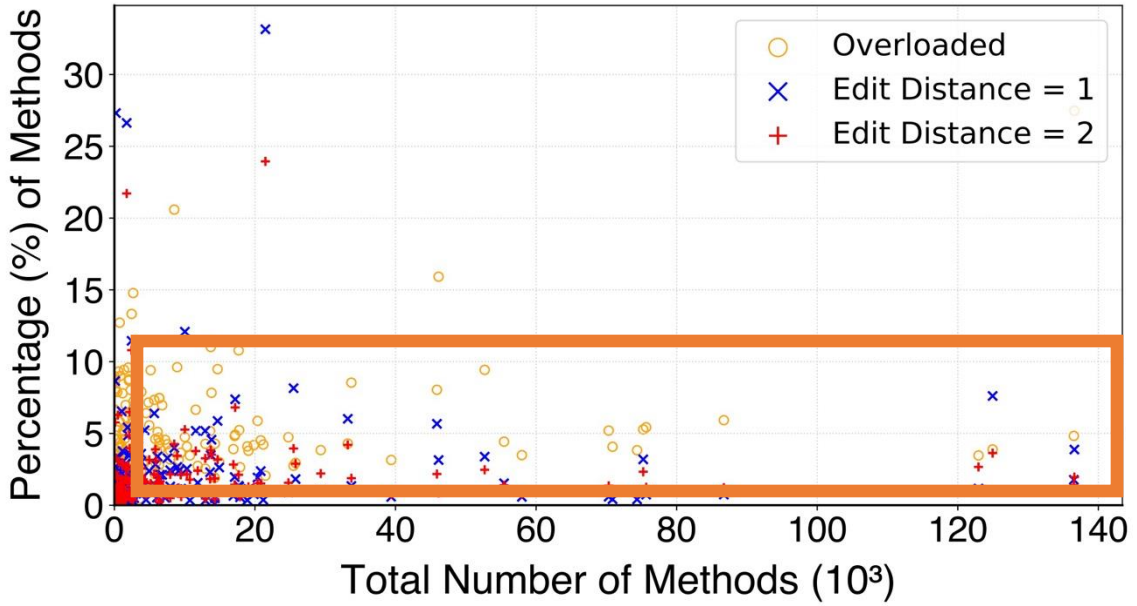
Method **variations** with small **edit distances** are more **widespread**

Percentage (%) of Method Instances

Correlation with Repository Attributes

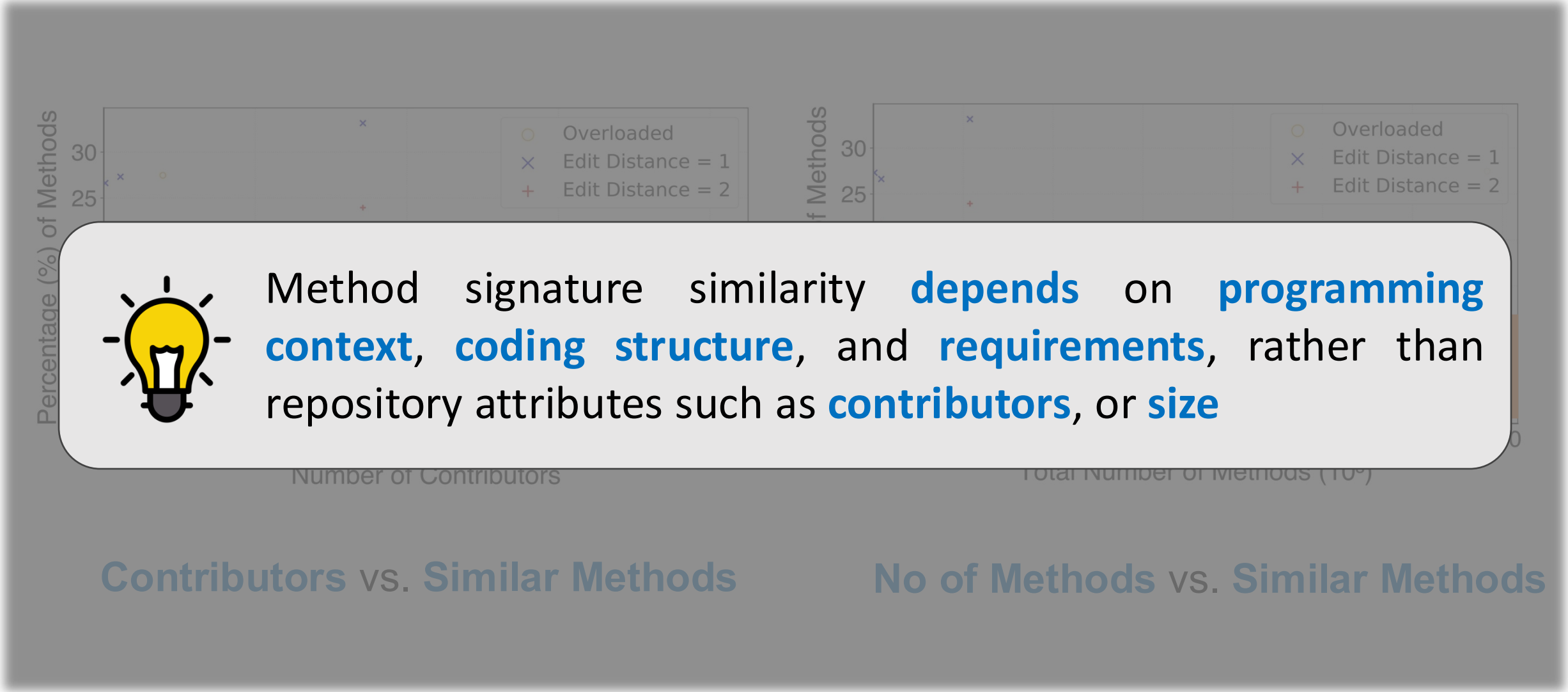


Contributors vs. **Similar Methods**



No of Methods vs. **Similar Methods**

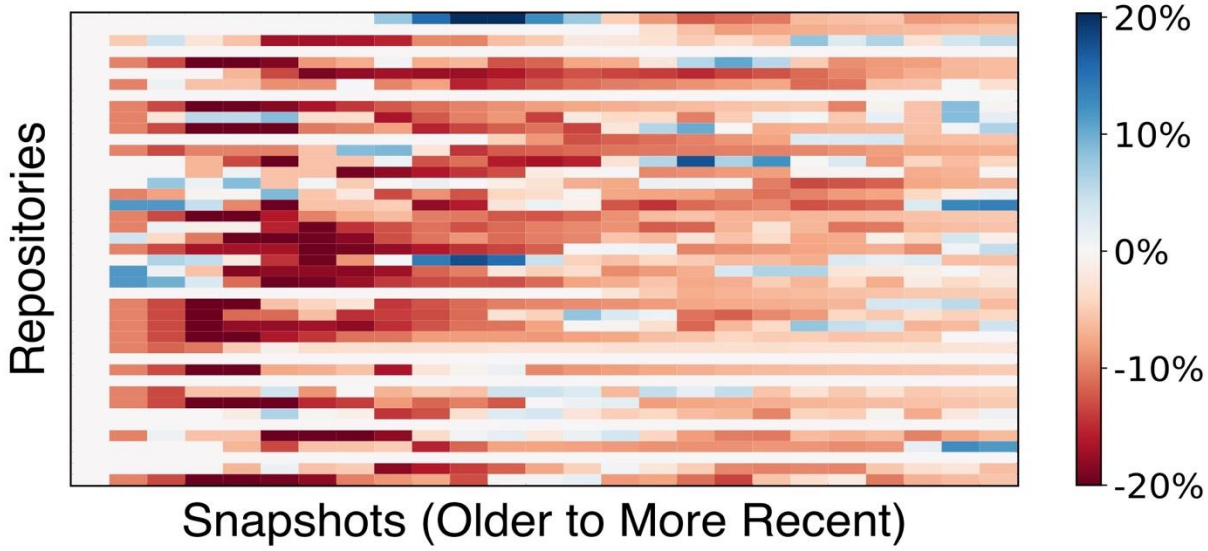
Correlation with Repository Attributes



Method signature similarity **depends** on **programming context**, **coding structure**, and **requirements**, rather than repository attributes such as **contributors**, or **size**

Evolution of Methods

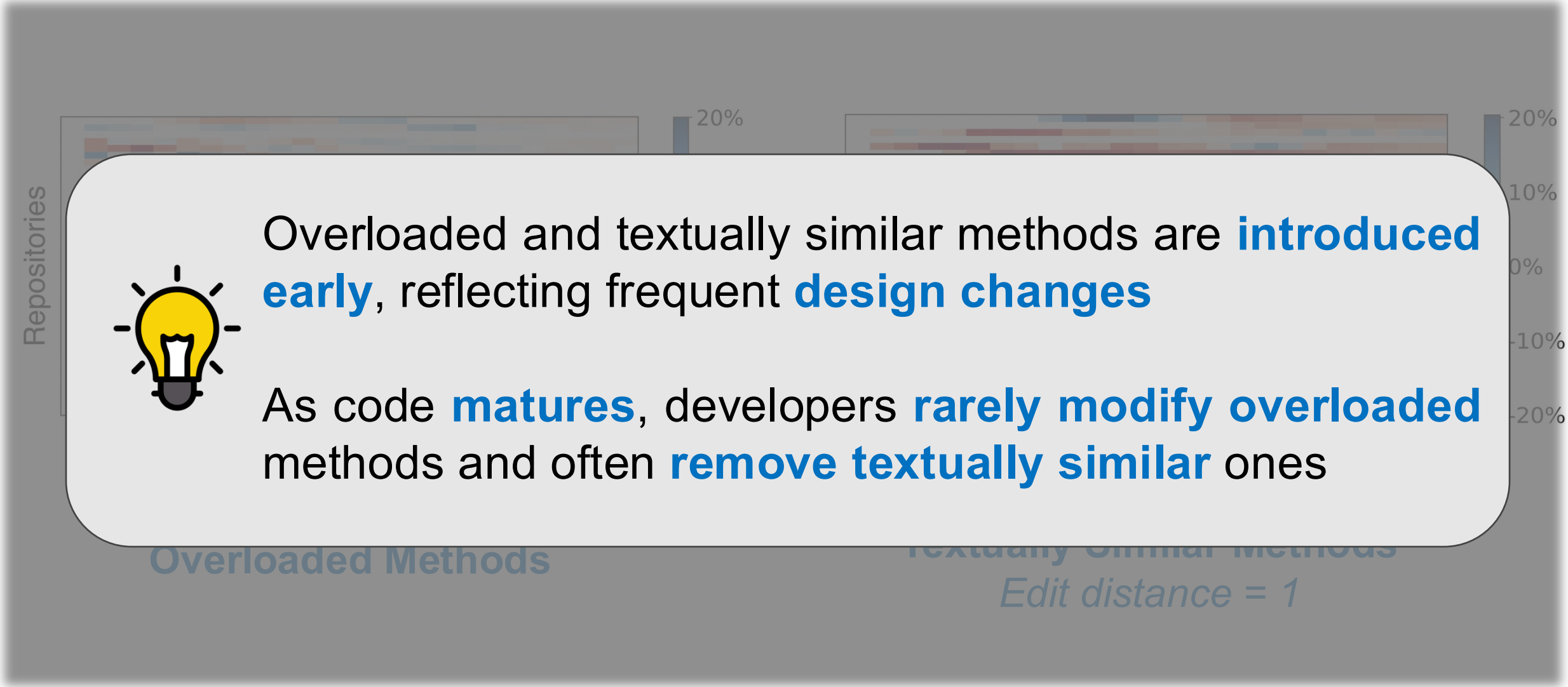
Overloaded Methods



Textually Similar Methods

Edit distance = 1

Evolution of Methods



Overloaded and textually similar methods are **introduced early**, reflecting frequent **design changes**

As code **matures**, developers **rarely modify overloaded** methods and often **remove textually similar** ones

Takeaways and Future Work

Methods with similar signatures are **context dependent** and arise from project requirements

Developer teams should establish naming **conventions** early to avoid confusion and cleanup later.

Use tools like **Maven** and **CheckStyle** to enforce naming conventions automatically.

Future Work: Focus on user studies and deeper investigation into commit notes, issues where we see a significant change

Conclusion

Our study reveals the **prevalence** and **evolution** of method signature similarity in **Java** and its **impact** on **development** practices.

Call to Action: Developers should manage method names carefully to enhance code maintainability, productivity, and prevent errors.

An Empirical Evaluation of Method Signature Similarity in Java Codebases

Mohammad Taha Khan, Mohamed Elhussiny, Billy Tobin and Muhammad Ali Gulzar