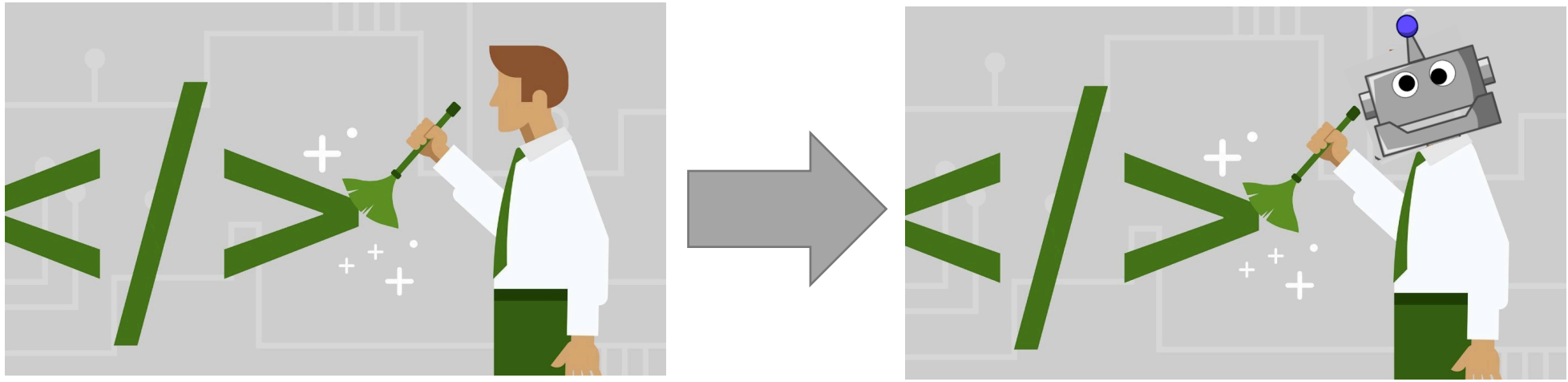
Towards Automated Refactoring of Code Clones in Object-Oriented Programming Languages

Authors: Simon Baars and Ana Oprescu

Location: Bolzano, Italy

Date: 8 July 2019



> *Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them.*
>
> —*Martin Fowler* & *Kent Beck* - *Refactoring*

Violations Overview | CPD View

Message	Class
Found suspect cut & paste (2 matches,5 lines)	
lines 21-25 in file DemoPMDThread.java	src.com.besthowtodo.tutorials.dynaweb.DemoPMD
lines 25-29 in file DemoPMDThread.java	src.com.besthowtodo.tutorials.dynaweb.DemoPMD
Found suspect cut & paste (2 matches,5 lines)	
lines 34-38 in file DemoPMDThread.java	src.com.besthowtodo.tutorials.dynaweb.DemoPMD
lines 10-14 in file DemoOMD.java	src.com.besthowtodo.tutorials.dynaweb.DemoOMI
Found suspect cut & paste (2 matches,1 line)	
Found suspect cut & paste (4 matches,2 lines)	
Found suspect cut & paste (2 matches,4 lines)	

RQ1.

How can we define clone types such that they ***can*** *be automatically refactored?*

RQ2.

What are the discriminating factors to decide when a clone ***should*** *be refactored?*

Clone types

How can we automatically refactor different types of clones?

> **Type 1.** Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

Shortcomings type 1

Textually equal code fragments

```java
1  package com.sb.fruitgame;
2
3  import com.sb.fruitgame.Apple;
4
5  public class GameScene extends Scene
6  {
7      public void collect(Apple apple) {
8          apple.setStatus(this);
9      }
10
11     public void addTen(int x) {
12         x = x + 10;
13         notifyChanged(x);
14         return x;
15     }
16 }
```

```java
1  package com.sb.fruitgame;
2
3  import com.sb.fruitgame.rotten.Apple;
4
5  public class LoseScene extends Scene
6  {
7      public void collect(Apple apple) {
8          apple.setStatus(this);
9      }
10
11     public void concatTen(String x) {
12         x = x + 10;
13         notifyChanged(x);
14         return x;
15     }
16 }
```

Type 1R clones

Functionally + textually equal code fragments

> Compare fully qualified identifier (FQI) of:

- Type references
  - Apple → com.sb.Apple
- Method signatures
  - eat(apple) → com.sb.Waste com.sb.EatController.eat(com.sb.Apple)

> Compare FQI of the types of variables

- myApple → (com.sb.Apple)myApple

Clone types

How can we automatically refactor different types of clones?

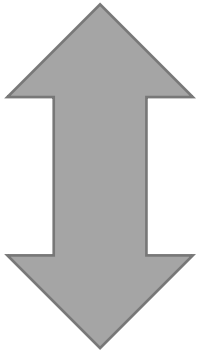
> **Type 1.** Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

> **Type 2.** Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout and comments.

Shortcomings type 2

Type 2 clones allow any(!) variation in identifiers, literals and types(!)

```
public boolean containsOnlyRedCircles(List<Circle> circles){
    return circles.stream().allMatch(Shape::isRed);
}
```



```
public Apple getEdibleApple(FruitBasket<Apple> basket){
    return basket.getFruit().getApple(Fruit::notEaten);
}
```

Type 2R clones

All about design tradeoffs

- No design tradeoff
  - Class/enum/interface names
  - Method names
  - Variable names (sometimes)

- Design tradeoffs
  - Literals
  - Variable references
  - Called methods

Type 2R clones

Tradeoff: different literal creates extra parameter

```java
// Original
public void printNice(String s){
    System.out.println("====================");
    System.out.println(s);
    System.out.println("====================");
}

public void printNice2(String s){
    System.out.println("--------------------");
    System.out.println(s);
    System.out.println("--------------------");
}
```

```java
// Refactored
public void printNice(String s){
    printNice(s, "====================");
}

public void printNice2(String s){
    printNice(s, "--------------------");
}

public void printNice(String s, String decoration){
    System.out.println(decoration);
    System.out.println(s);
    System.out.println(decoration);
}
```

Clone types

How can we automatically refactor different types of clones?

> **Type 1.** Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

> **Type 2.** Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout and comments.

> **Type 3.** Copied fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout and comments.

Type 3R clones

Gapped Statements

```
2   public String wordOfNumbers(int max){
3       Random r = new Random();
4       StringBuilder word = new StringBuilder();
5       for(int i = 0; i<=size; i++){
6           word.append(i);
7       }
8       word.append("that's it");
9       return word.toString();
10  }
```

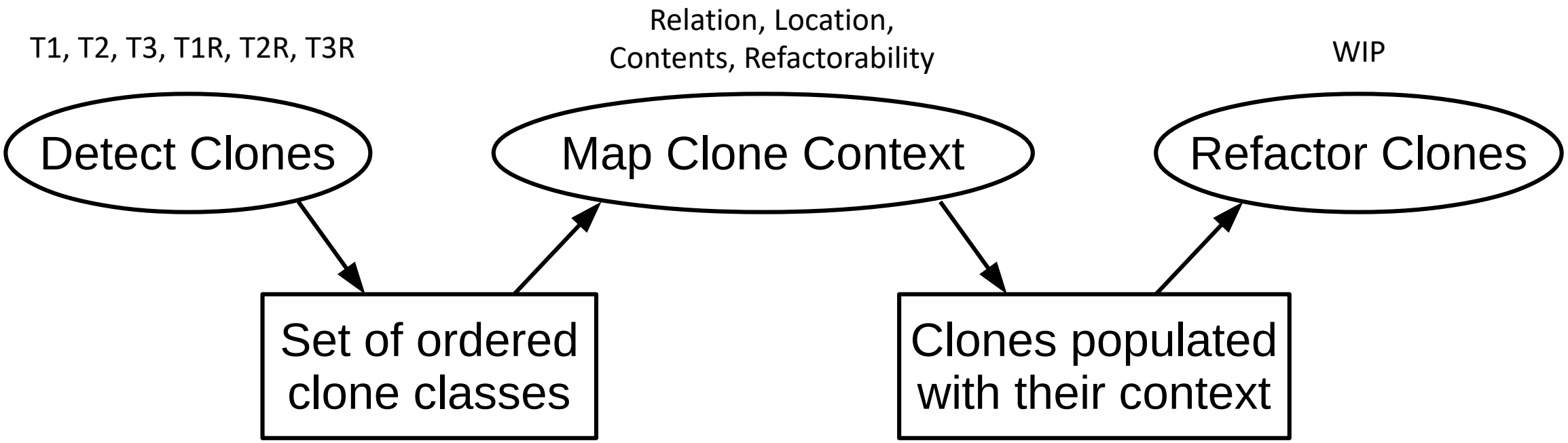
```
12  public void printWordOfNumbers(int max){
13      Random r = new Random();
14      StringBuilder word = new StringBuilder();
15      for(int i = 0; i<=size; i++){
16          word.append(i);
17          System.out.println("Appended "+i);
18      }
19      word.append("that's it");
20      System.out.println(word);
21  }
```

Type 3R clones

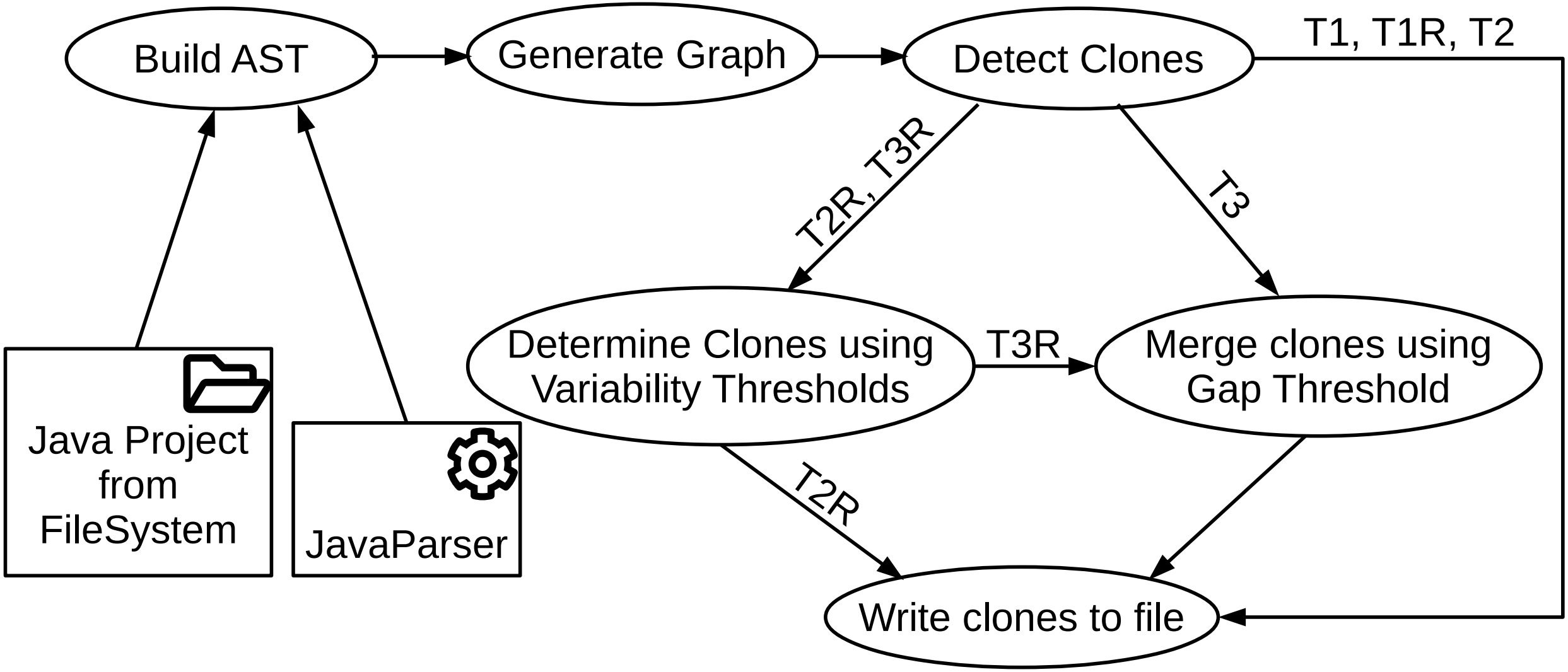
Added/removed/changed statement → solve with conditional

```
public List<String> wordOfNumbers(int size){
    return wordOfNumbers(n, size, false);
}

public void printWordOfNumbers(int size){
    System.out.println(wordOfNumbers(n, size, true));
}

public StringBuilder wordOfNumbers(int size,
                                   boolean doPrint){
    Random r = new Random();
    StringBuilder word = new StringBuilder();
    for(int i = 0; i<size; i++){
        word.append(i);
        if(doPrint) System.out.println("Appended "+i);
    }
    word.append("that's it");
    return word;
}
```

CloneRefactor



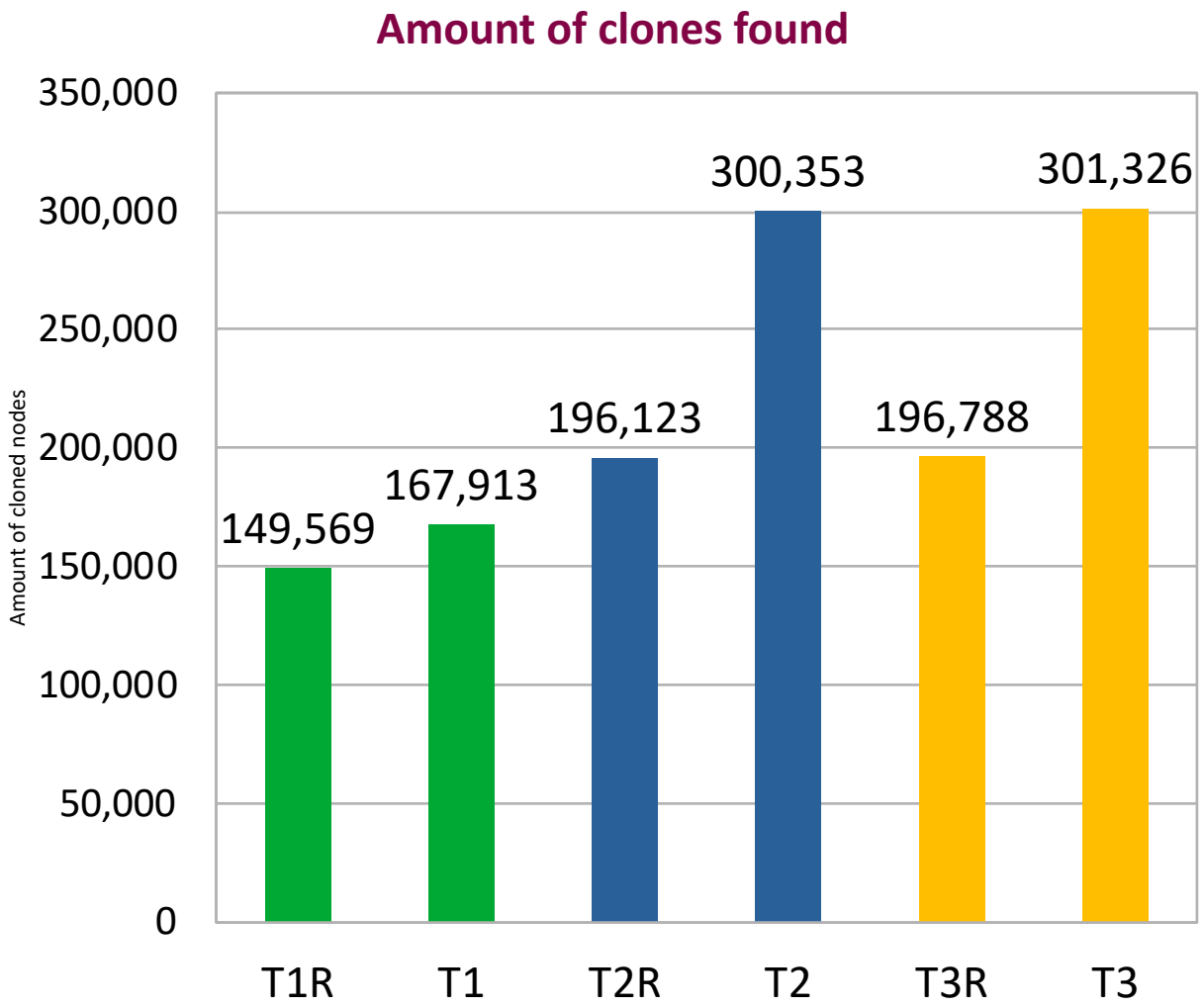
CloneRefactor

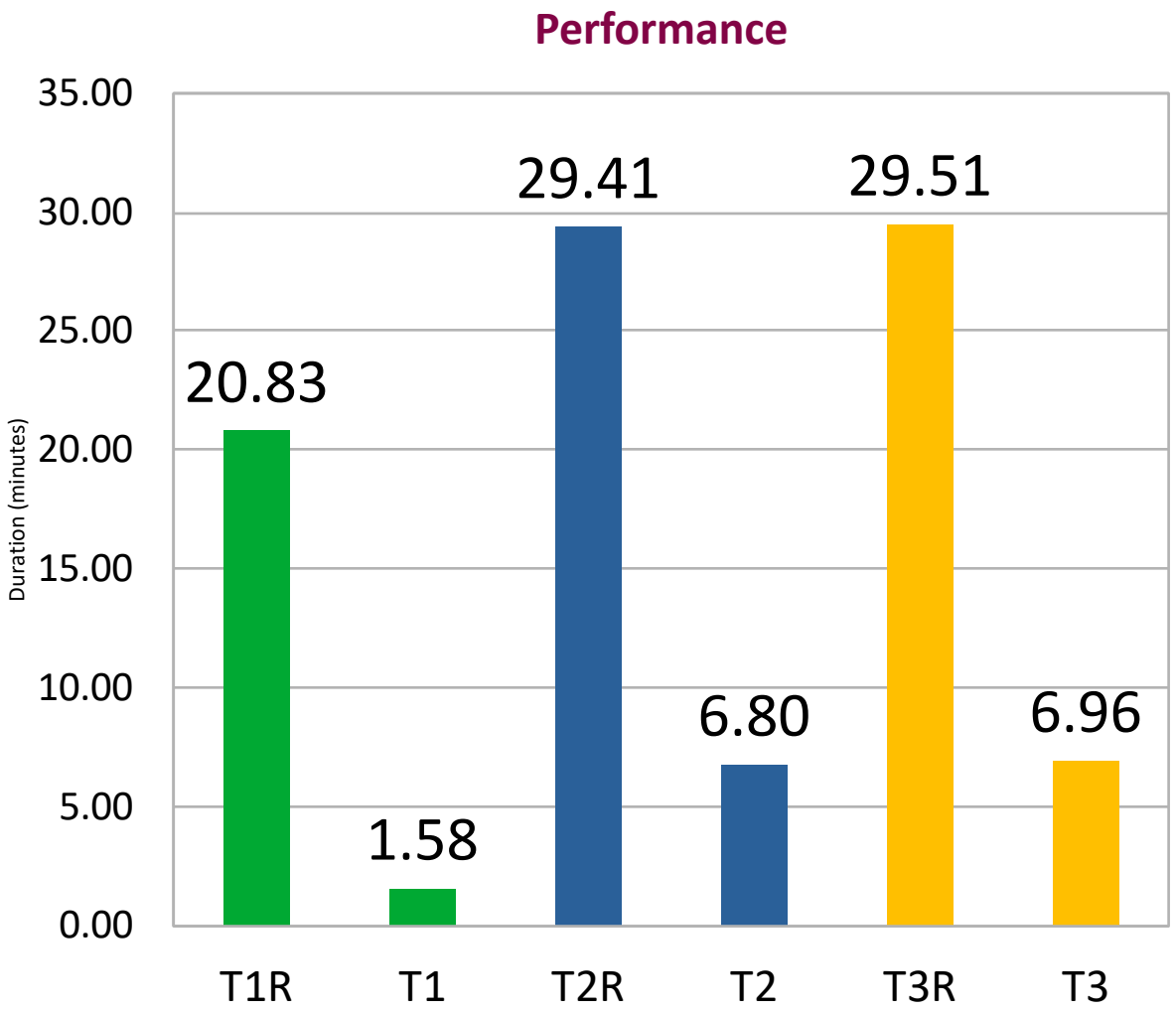


Experiments

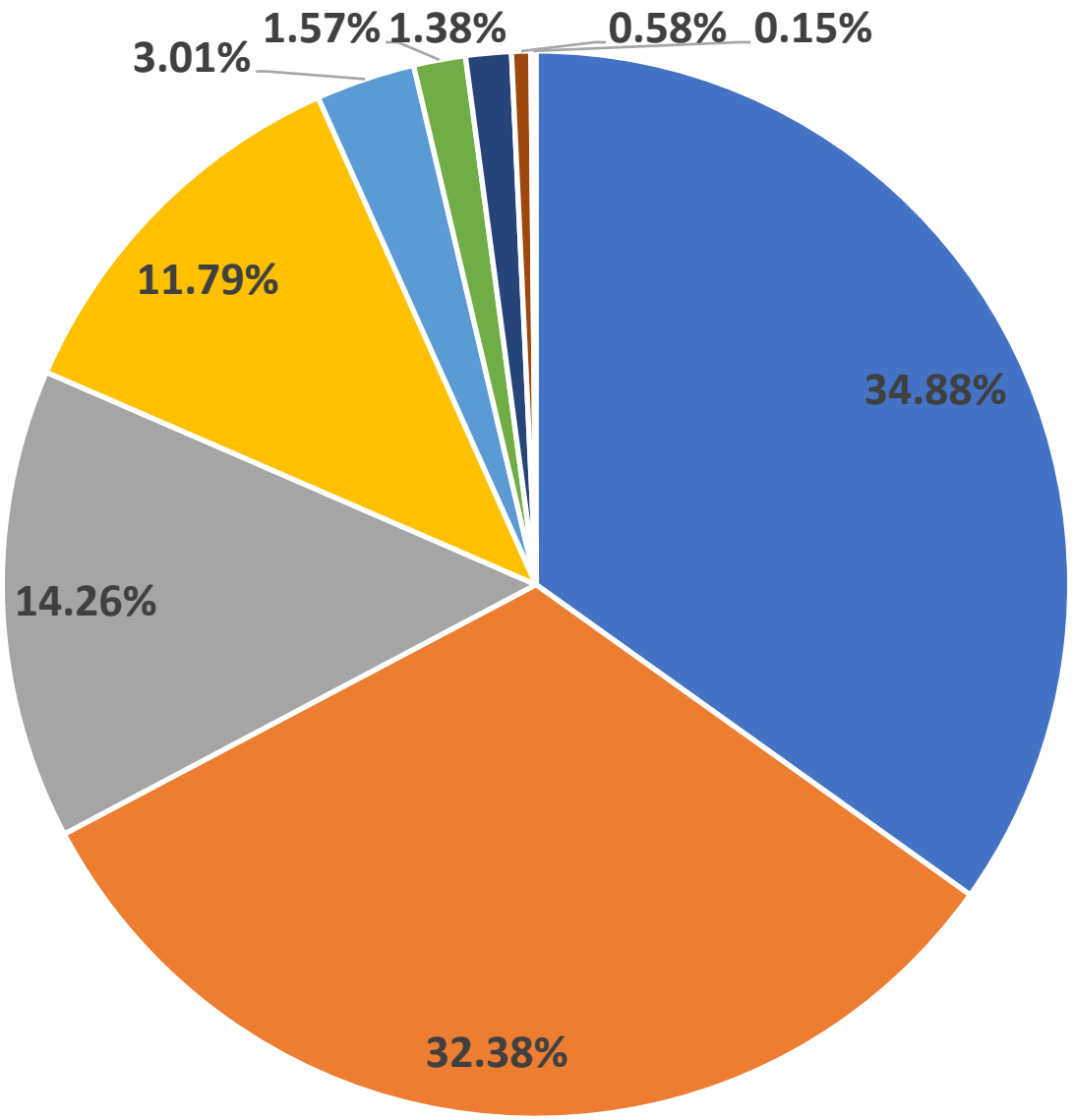
- Corpus with 1.343 Java projects
  - GitHub projects with relatively high number of forks
  - Relatively small, averaging 980 statements
- Clone Types
- Thresholds (WIP)
- Context
- Refactoring (WIP)

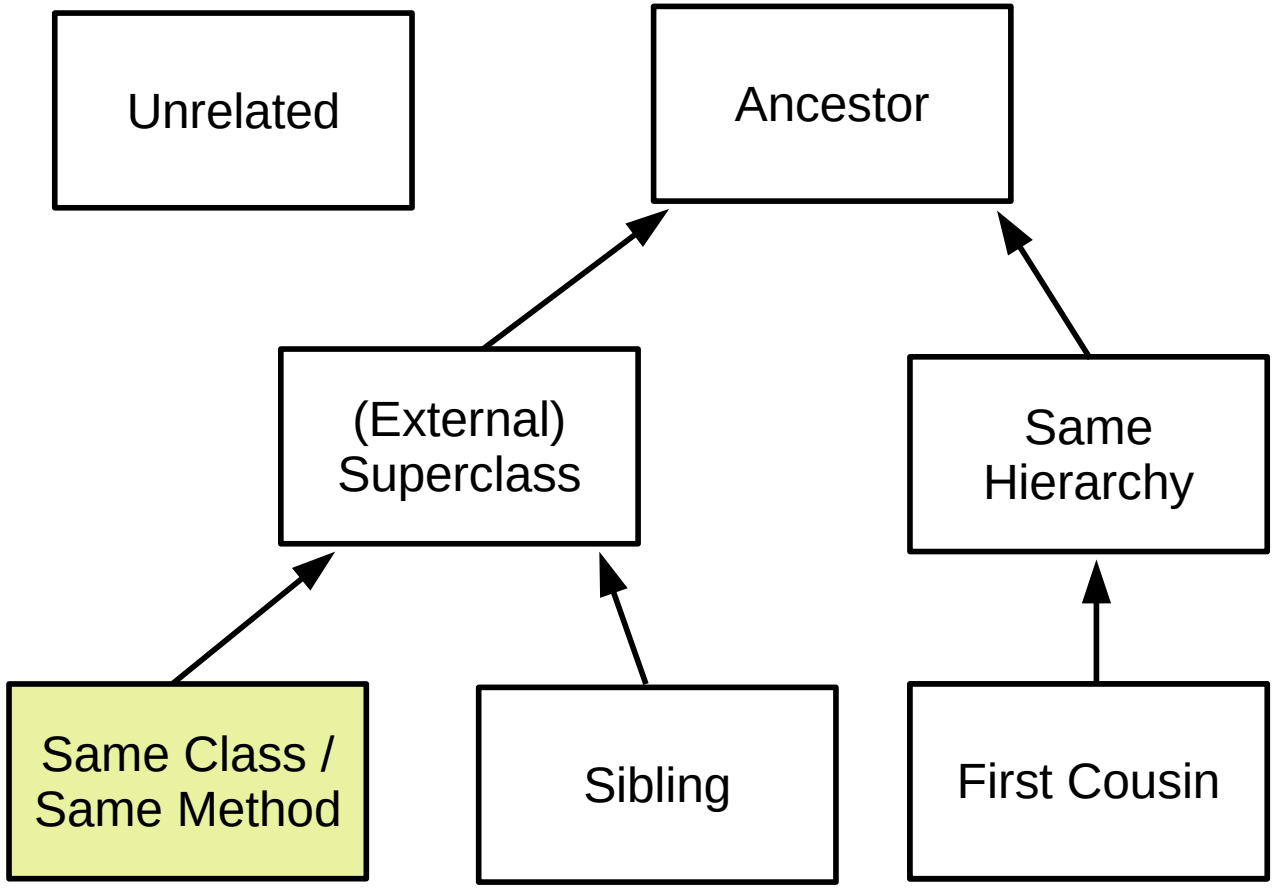
Clone Types


Amount of clones found
Amount of cloned nodes
350,000
300,000
250,000
200,000
150,000
100,000
50,000
0
149,569
167,913
196,123
300,353
196,788
301,326
T1R
T1
T2R
T2
T3R
T3


Performance
Duration (minutes)
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
20.83
1.58
29.41
6.80
29.51
6.96
T1R
T1
T2R
T2
T3R
T3

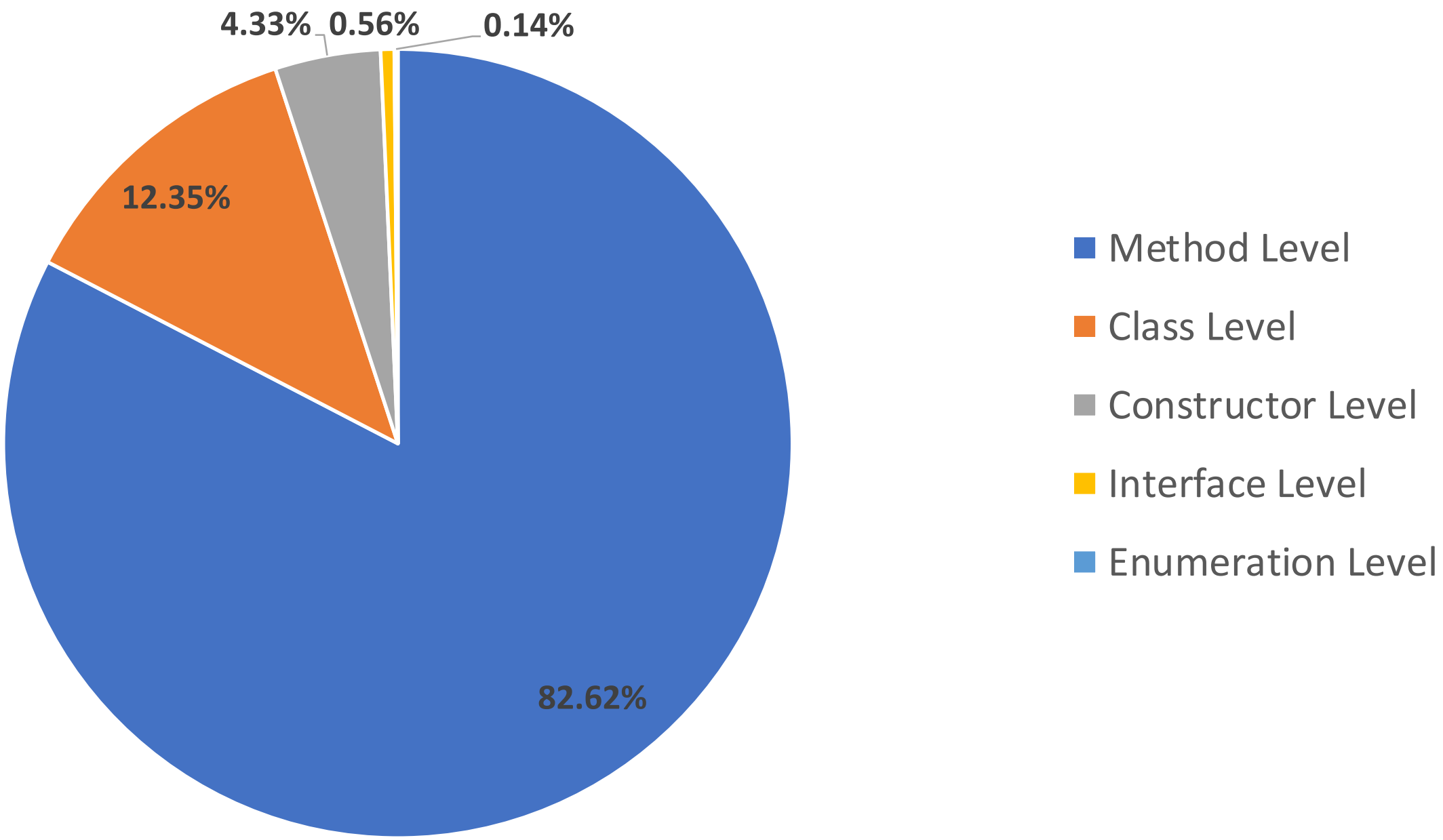
Relation between clone instances



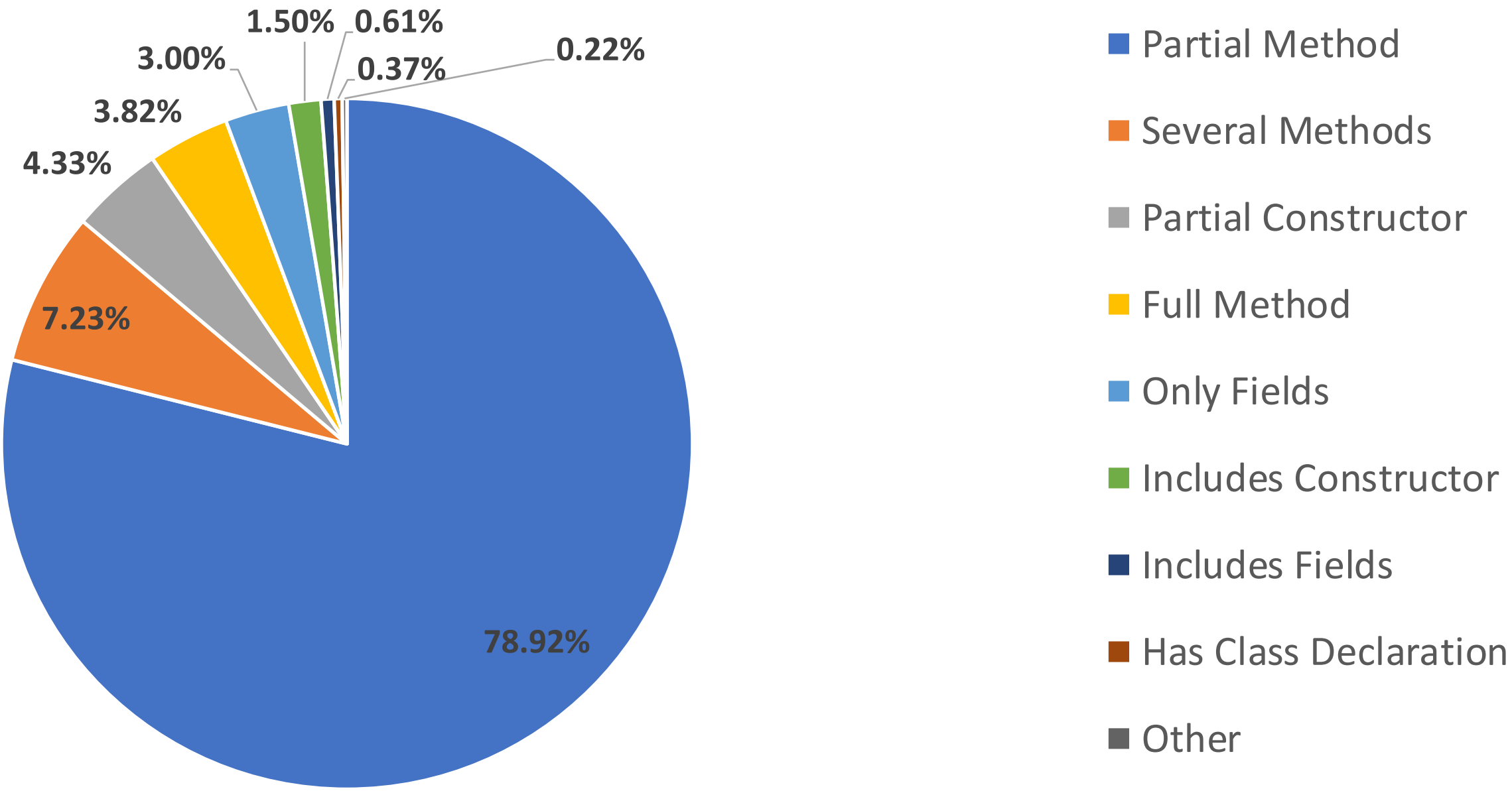


Unrelated | Same Class | Same Method | Sibling | External Superclass | Superclass | First Cousin | Common Hierarchy | Ancestor

Location of clones

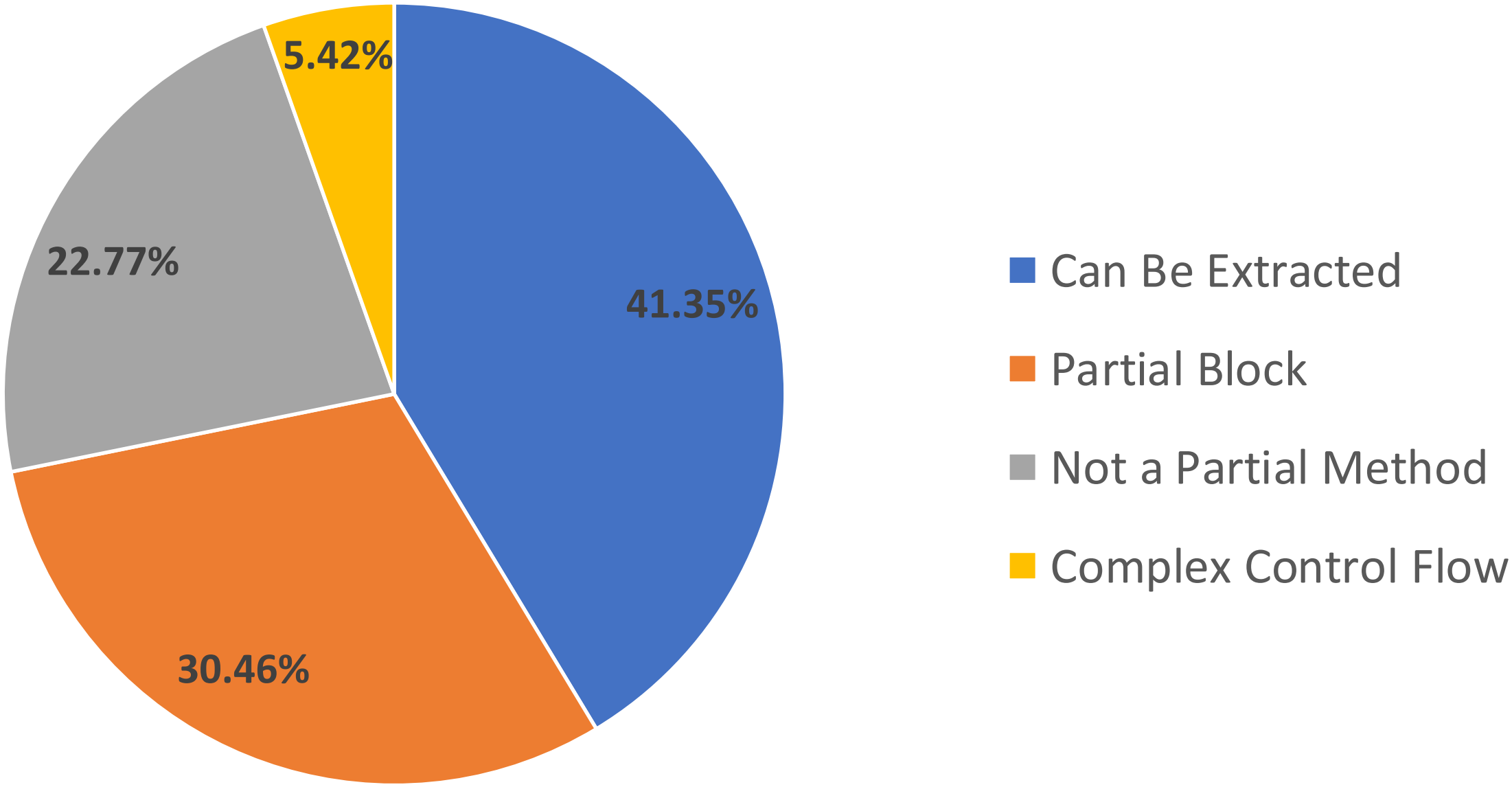


Contents of clones



Refactorability

To what extend can we extract methods?



Next steps

Analyze impact of “Extract Method” refactorings

RQ3.

*To what extent can we **automate** the process of refactoring clones?*

- Analyze Refactored Source Code
  - Maintainability index
    - Volume
    - Duplication
    - Complexity
  - Survey
- Dynamic Thresholds based on tradeoff
  - Relation
  - Number of parameters

Summary

Refactoring-Oriented clone types

> **T1R.** Textually and functionally identical code fragments.

> **T2R.** Allow change in controlled features of a codebase. Refactoring opportunities for these controlled features are known.

> **T3R.** Allows a gap of different statements between clones, constrained by a threshold.

> Refactoring oriented clone types take about **5** times as long to detect and yield fewer clones.

> Almost half of all clones are in the **same class/method**, about 35% is **unrelated**.

> About **80%** of clones is found in **methods**.

> Applying refactorings will be a goldmine for threshold related information.