

# Tracking Code Clones in Evolving Software

Ekwa Duala-Ekoko

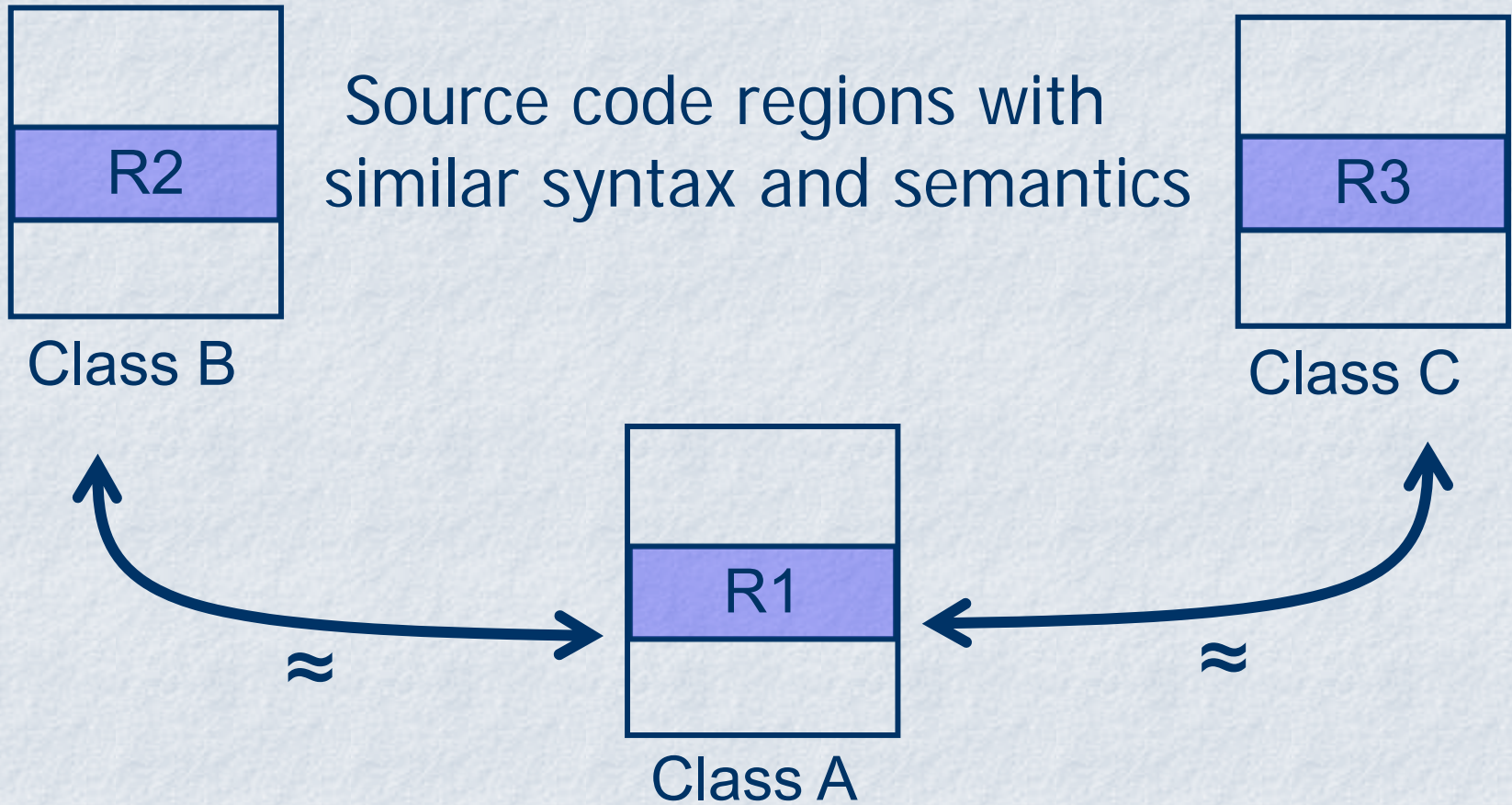
Martin P. Robillard



**McGill**

School of Computer Science

# What are code clones?



# Example Clone

```
for(int i = 0; i < methods.length; i++)
{
Method currMethod = methods[i];
if(name.equals(currMethod.getName()))
{
    Class[] param =
        currMethod.getParameterTypes();
    if( param.length != args.length )
        continue;
    try
    {
        for( int j = 0; j < param.length; j++)
            tempArgs[j] =
                Namespace.getAssignableForm( args[j],
                    param [j]);

        return currMethod;
    }
    catch( EvalError e )
    {...}
}}
```

```
for(int i = 0; i < constructors.length; i++)
{
Constructor currCon = constructors[i];

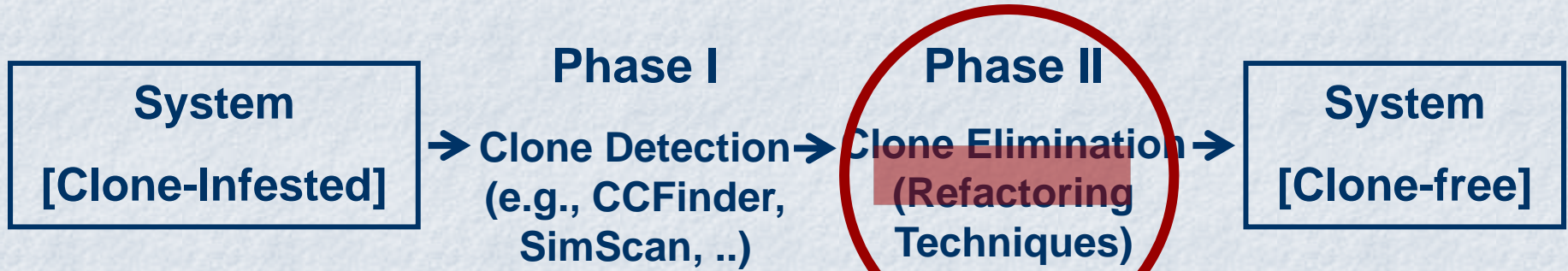
    Class[] param =
        currCon.getParameterTypes();
    if( param.length != args.length )
        continue;
    try
    {
        for( int j = 0; j < param.length; j++)
            tempArgs[j] =
                Namespace.getAssignableForm( args[j],
                    param [j]);

        return currCon;
    }
    catch( EvalError e )
    {... }
}
```

# Why Are Clones Problematic?

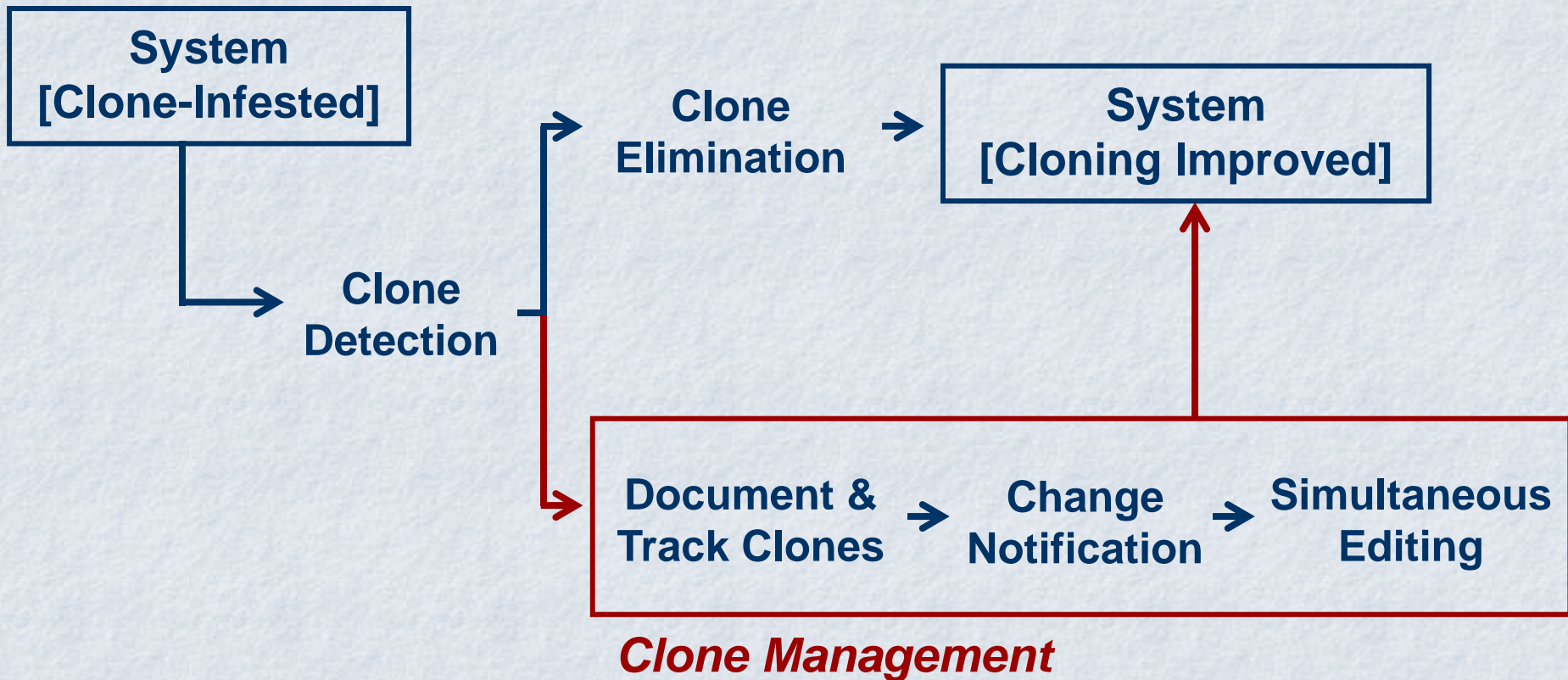
- Change coupling [Geiger et. al., 2006]
- Potential of bug introduction
- Non-negligible presence of clones in large-scale systems [up to 20%, Baker, 1995]

# Existing Solutions



- Some code clones are difficult/impossible to refactor [49%-64%, 2 open source projects, Kim et. al]
- Refactoring is not always cost-effective/beneficial [48%-72% disappeared within 8 check-ins, Kim et. al]

# Proposed Approach



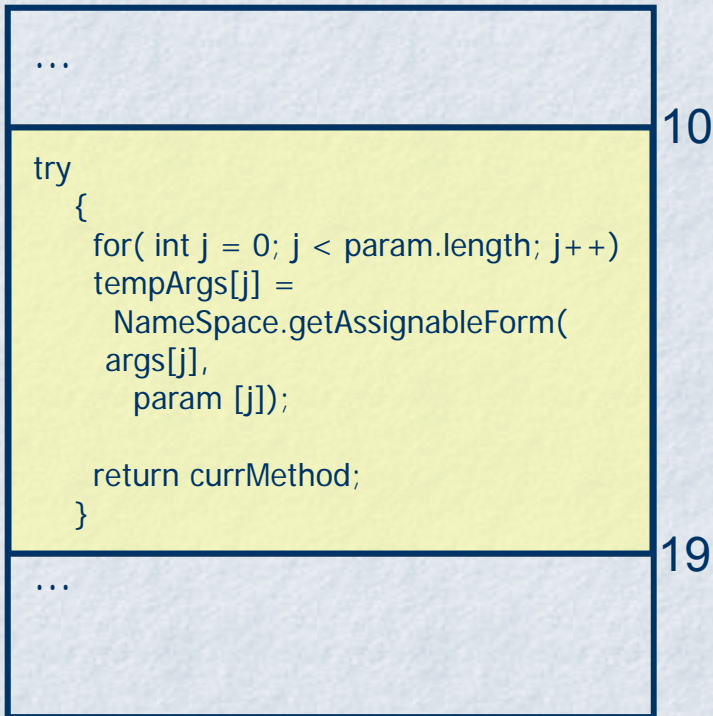
# Outline

1. Clone documentation model
2. Quantitative evaluation
3. Case study

# Describing Clone Regions (CRs)

## File/lines based descriptions

bsh.Reflect.java [10,19]



Line addition/removal will  
invalidate descriptions

## Looking for a better way...

### Question:

What characteristics can uniquely  
and robustly identify a clone  
region (CR)?

### Methodology:

Manully inspect ~600 CRs (in 4  
different Java systems)


### Answer (observations):

1. CR are constrained within the  
boundaries of code blocks
2. Some structural elements are  
unique at a given nesting  
level



**Clone Region Descriptors (CRDs)**


# Clone Region Descriptors: Partial Example

```
public class DeleteAction
{
  ...
  public void run(int x)
  {
    ...
    for(int i=x;i<map.size(); i++)
    {
      
    }
    ...
  }
}
```

## Description of Region A (simplified)

*<file>* = DeleteAction.java  
*<class>* = DeleteAction  
*<method>* = run(int)  
*<block type>* = for  
*<anchor>* = "i<map.size()"



# Clone Region Descriptors: Partial Example

```
public class DeleteAction
{
...
  public void run(int x){
...
    try{
      
    }catch(IOException e){...}
    }catch(Exception e){...}
...
  }
...
}
```

## Description of Region B (simplified)

```
DeleteAction.java
DeleteAction
run(int)
try
IOException, Exception
```

# Clone Region Descriptors: Conflicts

```
public class DeleteAction
{
    ...
    public void run(int x)
    {
        ...
        for(int i=x;i<map.size(); i++)
        {
            
        }
        ...
        for(int i=x;i<map.size(); i++)
        {
            
        }
    }
}
```

- Basic CRDs are not always unique
- Non-trivial differences generally exist in the logic implemented by each block
- CRD capture these differences in a **corroboration metric** (sum of **fan-out** and **cyclomatic complexity**)

**Current corroboration metric:**  
**fan-out + cyclomatic complexity**

# Current CRD Model

<CRD> ::= <file> <class> <CM> [<method>]  
<method> ::= <signature> <CM> <block> \*  
<block> ::= <btype> <anchor> <CM>  
<btype> ::= 'for' | 'while' | 'do' | 'if' | 'switch' | 'try' | 'catch'


## Block types and anchors

---

<b>for</b> <b>(all loops)</b>	<b>if</b>	<b>switch</b>	<b>try</b>	<b>catch</b>	<b>method</b>
Termination condition	Branching predicate	Switch expression	List of exceptions thrown	Type of exception caught	Method signature

---

# Clone Region Descriptors: Complete Example

```
public class DeleteAction
{
  ...
  public void run(int x)
  {
    ...
    for(int i=x;i<map.size(); i++)
    {
      
    }
    ...
  }
}
```

## Description of Region A (with corroboration metrics)

*<file>* = DeleteAction.java  
*<class>*, *<CM>* = DeleteAction, **21**  
*<method>*, *<CM>* = run(int), **11**  
*<btype>*, *<CM>* = for, **6**  
*<anchor>* = "i<map.size()"

CRD constructed from the Abstract Syntax Tree

# Clone Region Lookup Algorithm

**CRD**<fileName, className, methodSignature, blockType, anchor>



**Class Identification** <fileName, className>: returns ASTNode\_Class



**[Method Identification** <ASTNode\_Class, methodSignature>:  
returns ASTNode\_Method]

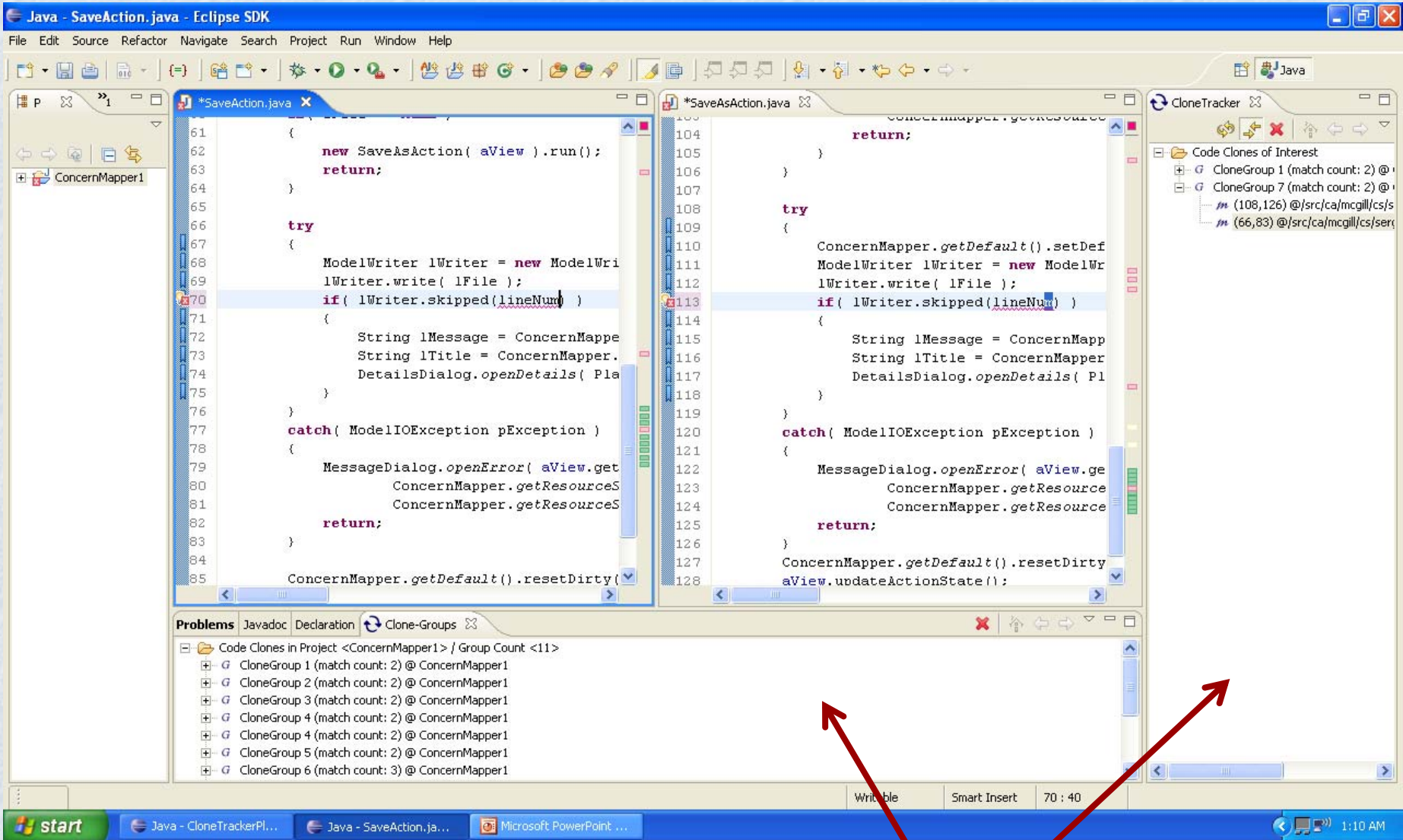


**[Block Identification** <ASTNode\_Method, blockType, anchor >]



**Clone Region**

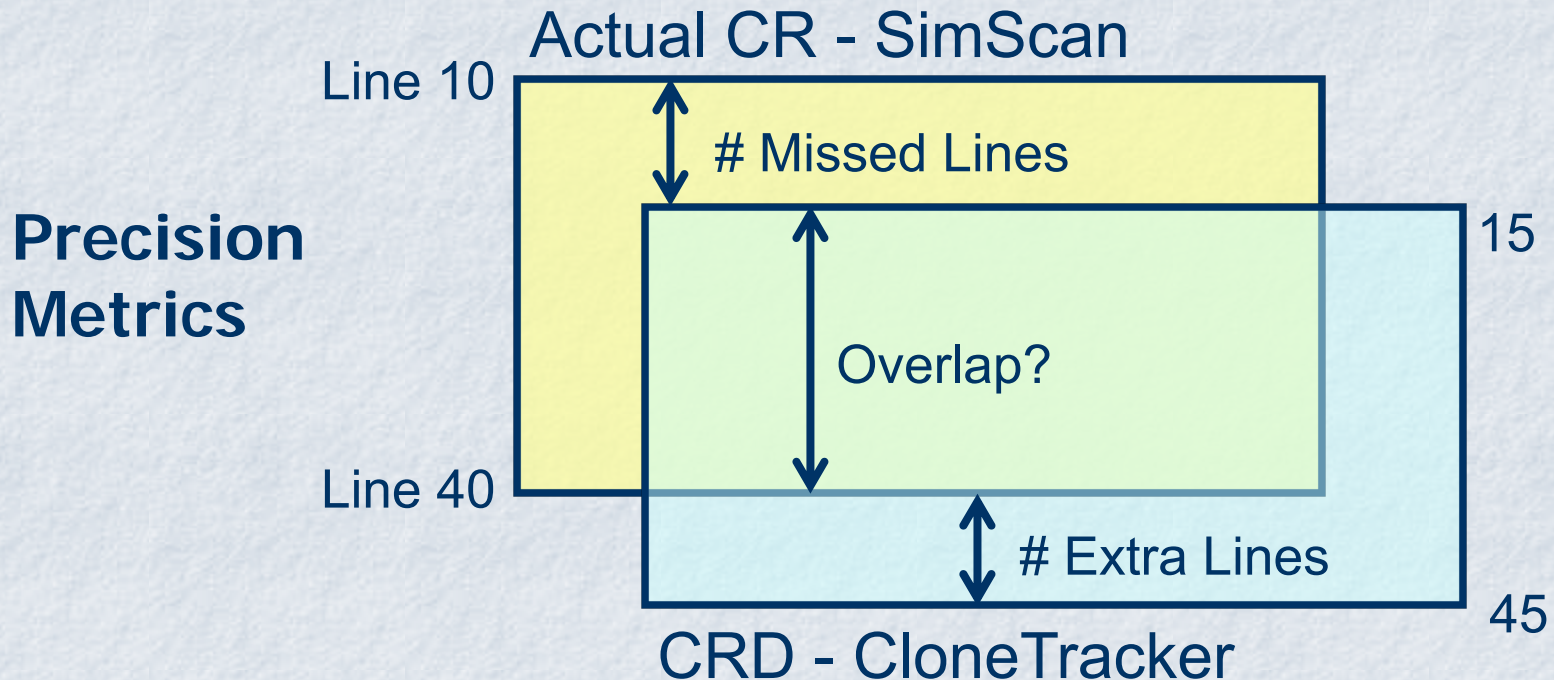
# The CloneTracker Eclipse Plug-in



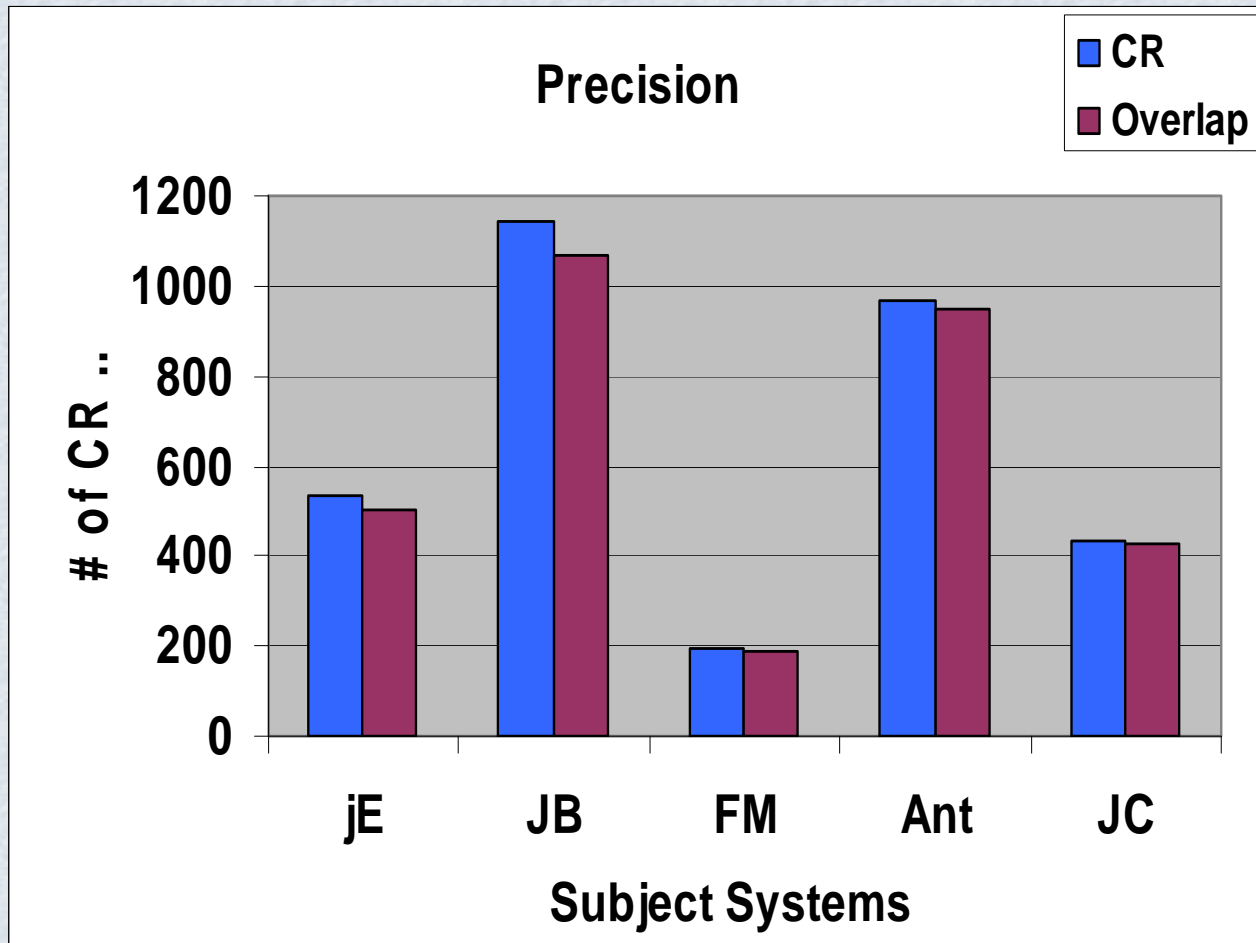
**CloneTracker Views**

# Quantitative Evaluation: Precision of CRDs

- Do CRDs accurately represent clone regions?
- Subject systems:  
**jEdit, JBossAOP, Ant, FreeMind, JCommander.**



# Quantitative Evaluation: Precision of CRDs

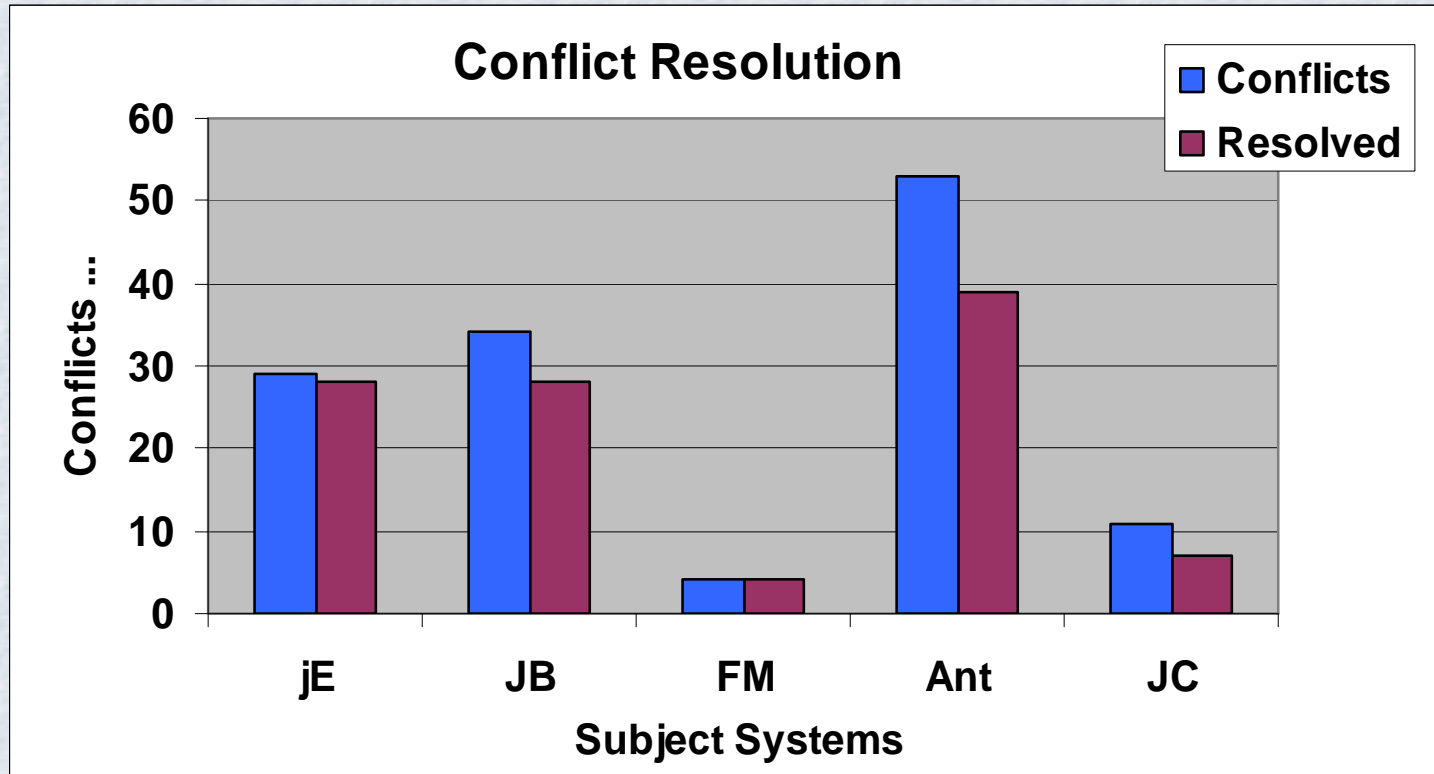


Avg. CR	24.6
Avg. ML	1.8
Avg. EL	3.5

Total # of CR: **3,275**

Total Overlap: **96%**

# Quantitative Evaluation: Conflict Resolution



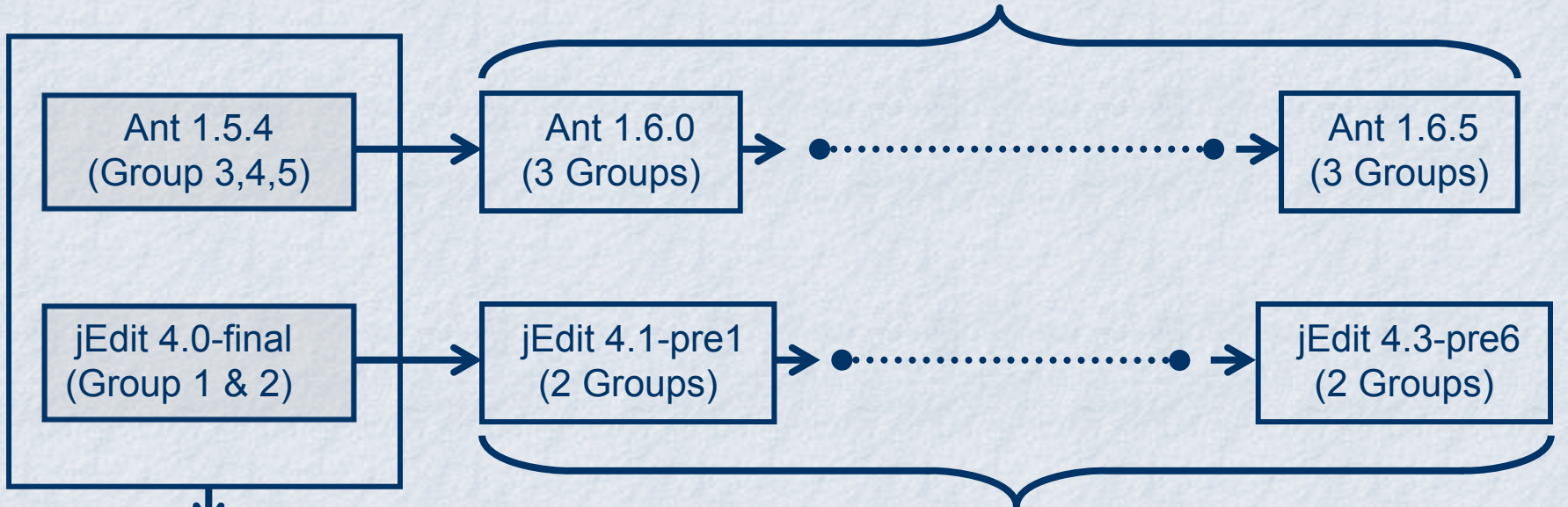
Total Conflicts: **131**

Total Resolved: **81%**

# Case Study: Tracking Clones in Evolving Software

Can CRDs track clones across subsequent versions?

Can we track clones across 6 versions using CRD?



Can we track clones across 27 versions using CRD?

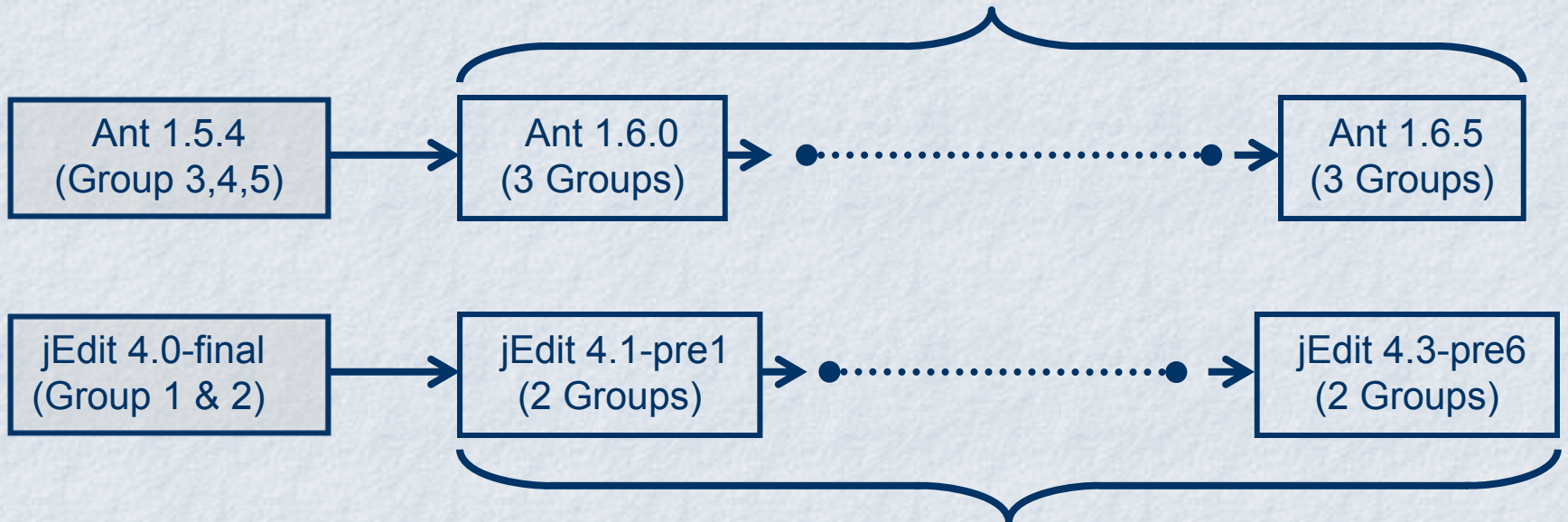
**CloneTracker**



**CRD of Group 1 to 5**

# Case Study: Tracking Clones in Evolving Software

All 3 groups tracked across all 6 versions of Ant



**Group 1** tracked across next 16 versions of jEdit  
(then disappears)

**Group 2** tracked across next 26 versions of jEdit

# Future Work

- Make CRDs robust to changes in nesting levels
- Design a robust (non-textual) representation for anchors
- Experiment with different corroboration metrics

# Conclusion

- Refactoring of code clones is not always feasible or cost-effective
- We proposed ***Clone Region Descriptors*** : **(Increased Abstraction, Lightweight, Robust, and Reusable)** a technique that enable developers to:
  - Document and keep track of clone groups across different versions of a system
  - Be notified upon changes that affect a documented clone group
  - Support simultaneous modification to clone regions

# Questions