Representation, Analysis, and Refactoring Techniques to Support Code Clone Maintenance

Dissertation Research Defense

Robert Tairas

tairasr@cis.uab.edu

http://www.cis.uab.edu/tairasr

June 15, 2010

Committee:

Dr. Barrett Bryant (Chair)

Dr. Jeff Gray

Dr. Nicholas Kraft

Dr. Marjan Mernik

Dr. Brian Toone

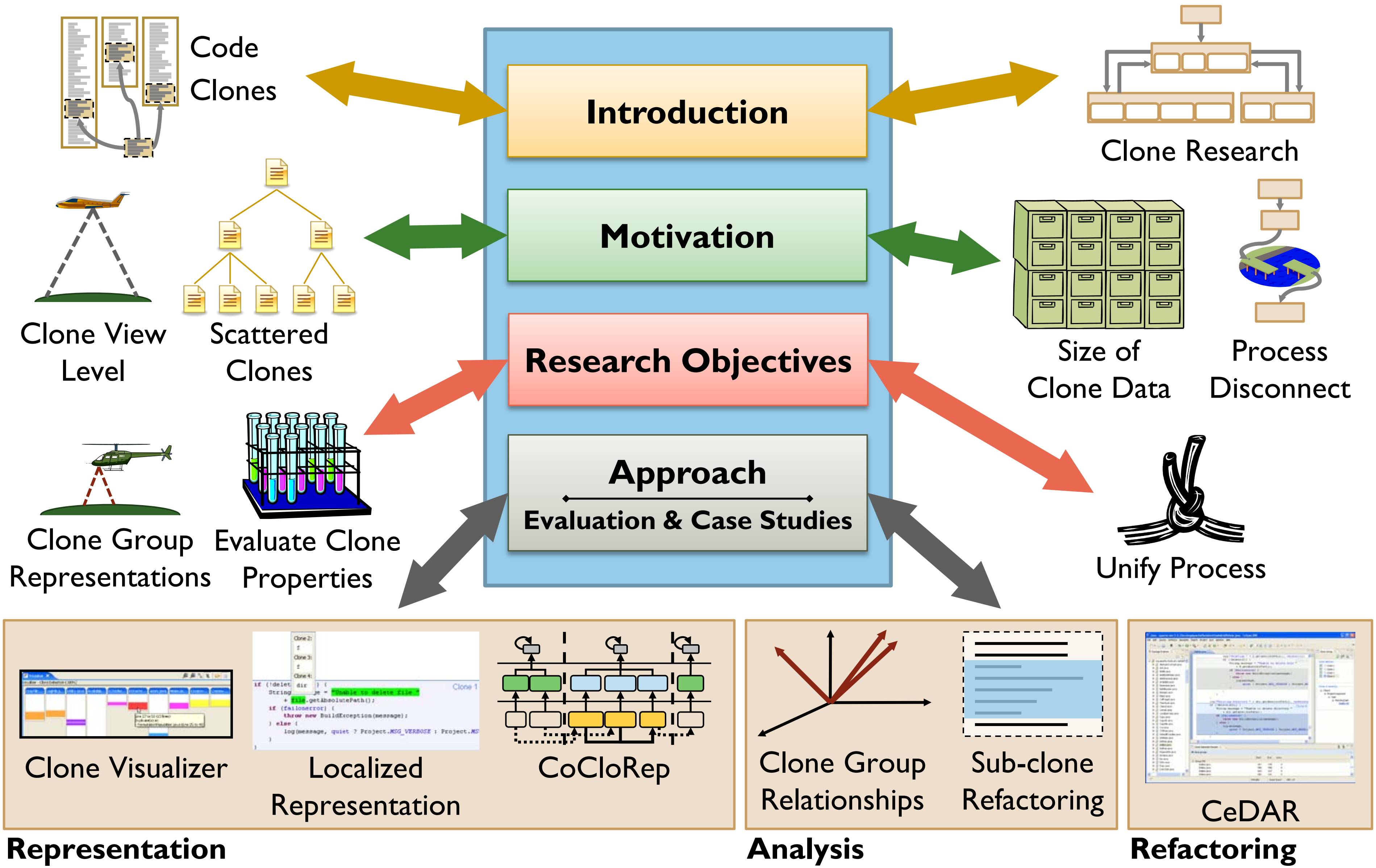
Dr. Chengcui Zhang

University of Alabama at Birmingham

Software Composition and Modeling Lab

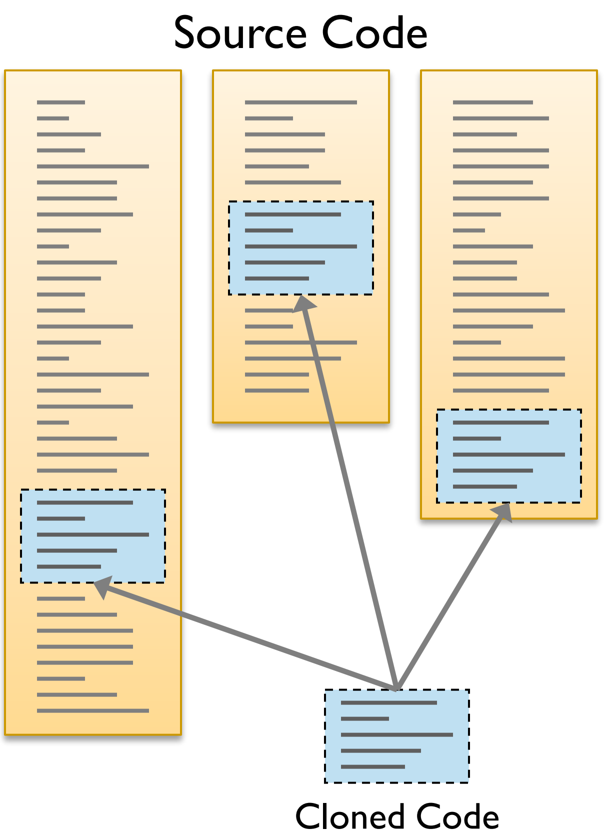
This research is supported by NSF grant CPA-0702764

Overview of Presentation



Cloning in Software

- Code Clones:
  - A section of code that is duplicated in multiple locations in a program
- Different granularity levels:
  - Statements, Block, Method, Class, Program
- Clone Group:
  - Clones of the same duplication



Types of Clones

Original code

```
int main() {
   int x = 1;
   int y = x + 5;
   return y;
}
```

```
int main() {
   int x = 1;
   int y = x + 5;
   return y;
}
```

Exact match
(Type I)

```
int func2() {
   int p = 1;
   int q = p + 5;
   return q;
}
```

Exact match with differing (parameterized) identifier names
(Type II)

```
int main() {
   int x = 1;
   int y = x + 5;
   x++;
   return y;
}
```

Near exact match
(Type III)

Bellon *et al.*, 2007

Reason for the Existence of Clones

- A section of code is copied and pasted into another part of the same program
  - Code performs some functionality correctly and copy-and-paste is relatively easy

- Simion[†] (Similar code fragments)
  - Behaviorally similar
  - Origins not from a common code fragment

[†]Juergens *et al.*, 2010

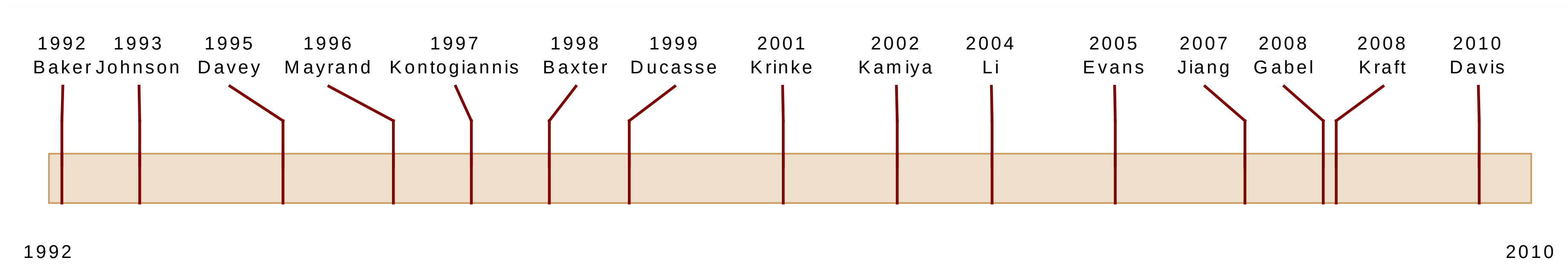
Clones in Software Maintenance

- Clone maintenance:
  - Fix an error, enhance the functionality, or to improve the structure and/or performance
  - Software maintenance consumes up to 90% of software development effort†
- Clone comprehension:
  - Knowledge of their existence, where the duplicates are located, and what kind of code is being duplicated
  - Program comprehension consumes at least 50% of maintenance cost‡

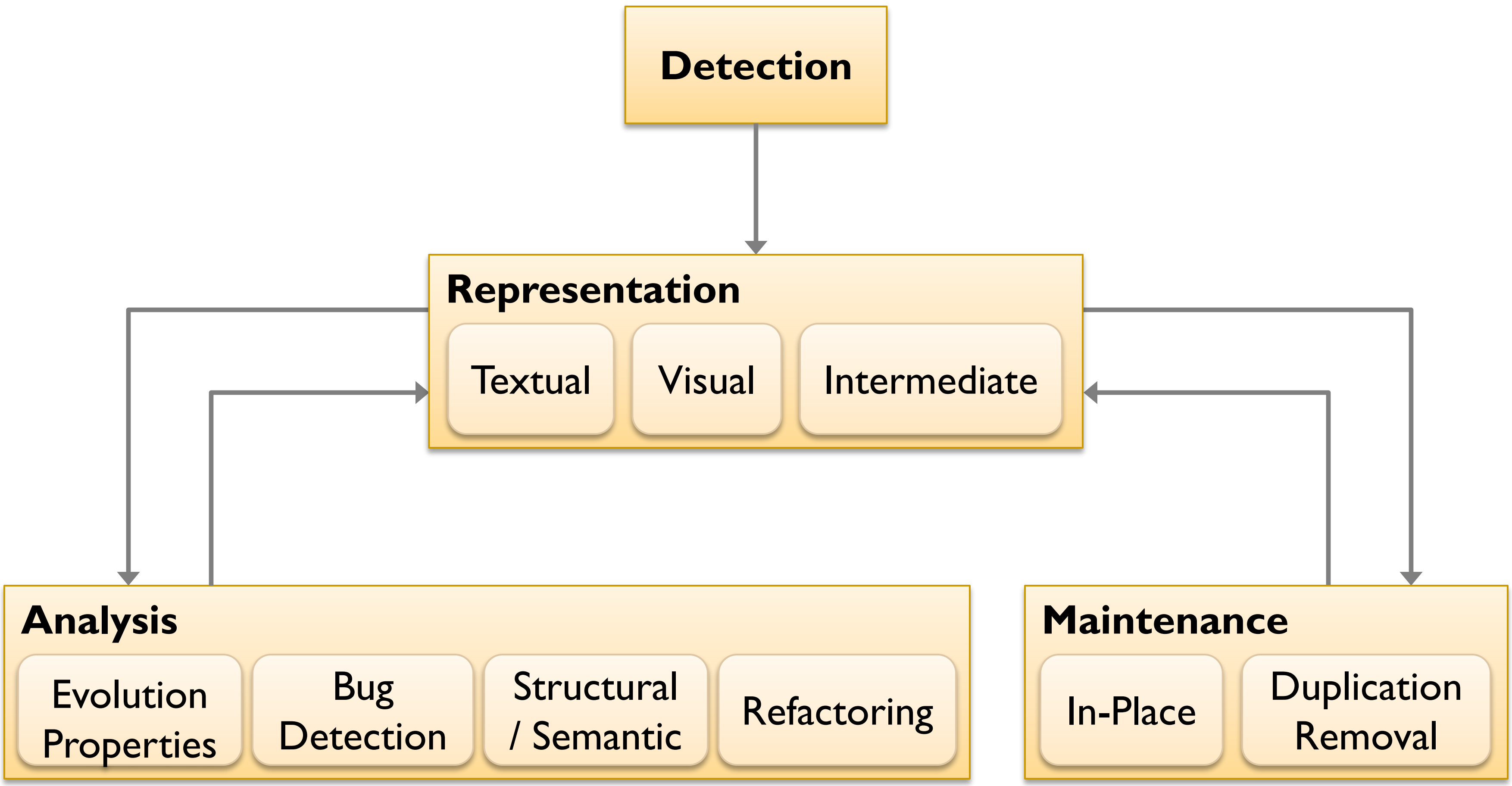
†Erlikh, 2000; ‡Standish, 1984

Clone Detection Techniques and Timeline

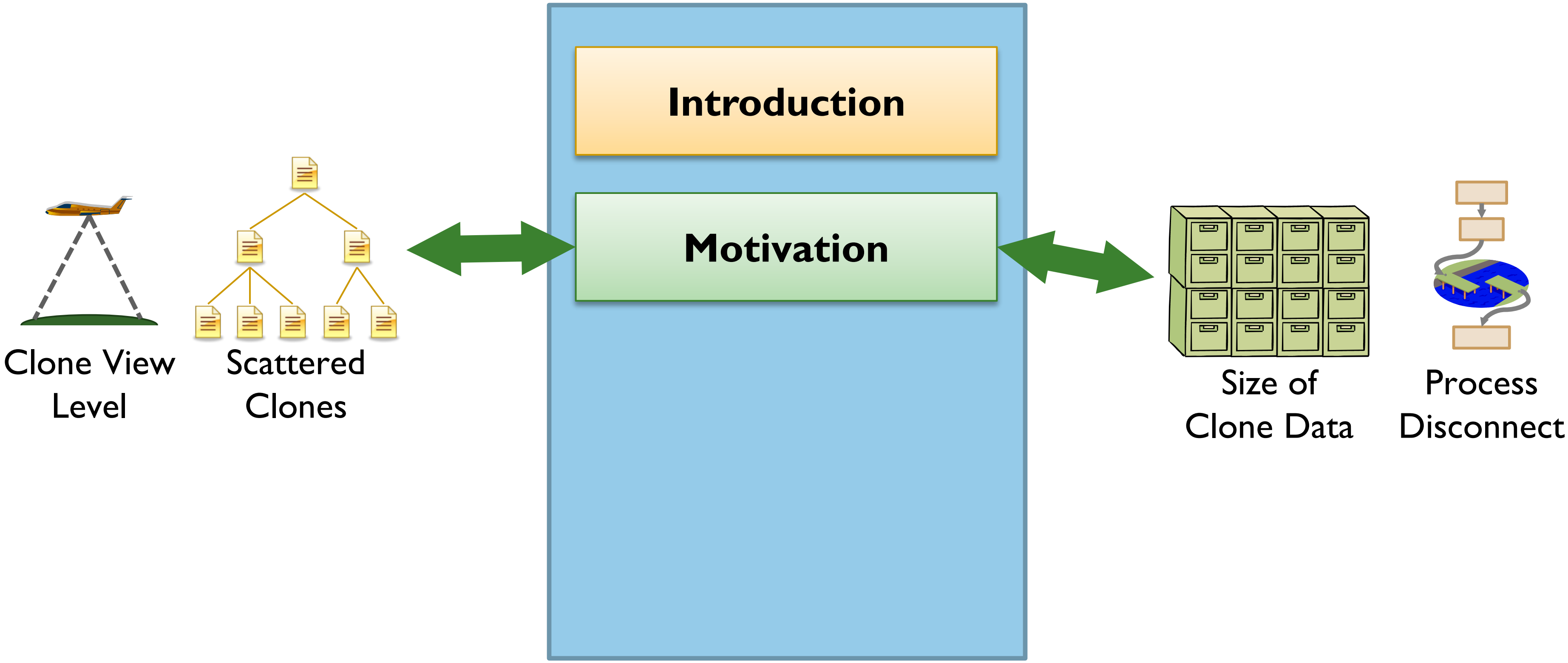
- *String*: Baker ‘92, Johnson ‘93, Davey ‘95, Ducasse ‘99
- *Token*: Kamiya ‘02, Li ’04
- *Tree*: Baxter ‘98, Evans ‘05, Jiang ’07, Kraft ‘08
- *Program Dependence Graph*: Krinke ’01, Gabel ‘08
- *Assembler*: Davis, ‘10
- *Metrics*: Mayrand ‘96, Kontogiannis ‘97



Clone Research



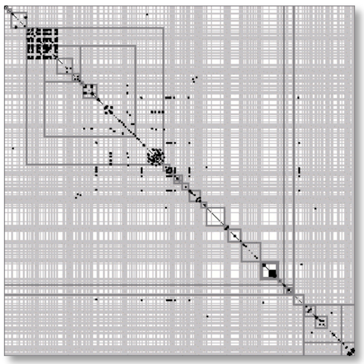
Overview of Presentation



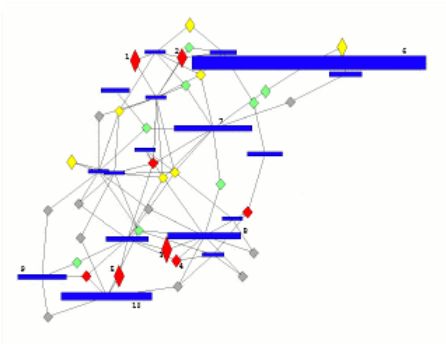
Representation Challenge: Evaluating Clone Groups

- Current representations and visualizations generally provide a system-level view

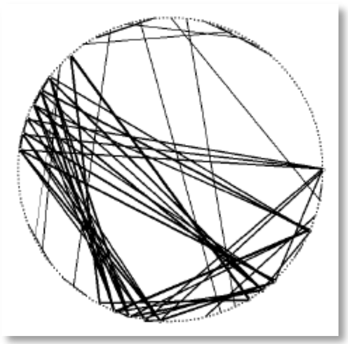
Scatter Plot†



Clone Cohesion & Coupling‡



Duplication Web¤



- Clones can be scattered in multiple source files

Clone pair distribution in Apache§

Same file	Same directory	2nd Cousin	3rd Cousin and more	Total
912	135	840	641	2528

†CCFInder, 2010; ‡Jiang and Hassan, 2007; ¤Rieger *et al.*, 2004; §Kapser and Godfrey, 2005

Example Detection Results (Textual)

Source File Starting Line Ending Line

Simian Output

```
76397-C:\...\CMPFieldMetaData.java:134-145,
   76296-C:\...\CMPFieldMetaData.java:117-129
433729-C:\...\UsersRolesLoginModuleTest.java:64-68,
   420696-C:\...\LoginModulesTest.java:312-316
164262-C:\...\ServerDataCollector.java:230-265,
   231230-C:\...\Scheduler.java:552-587
248103-C:\...\EJBVerifier11.java:448-480,
   249898-C:\...\EJBVerifier11.java:1073-1109,
   250532-C:\...\EJBVerifier11.java:1297-1337
...
```

Clone Group

Starting Line Ending Line Source File

SimScan Output

```
Found 6 duplicate lines in the following files:
 Between lines 201 and 207 in /.../WritableRaster.java
 Between lines 1305 and 1311 in /.../Raster.java
Found 6 duplicate lines in the following files:
 Between lines 920 and 926 in /.../JFIFMarkerSegment.java
 Between lines 908 and 914 in /.../JFIFMarkerSegment.java
...
```

Clone Group

Analysis Challenge: Large Amounts of Data

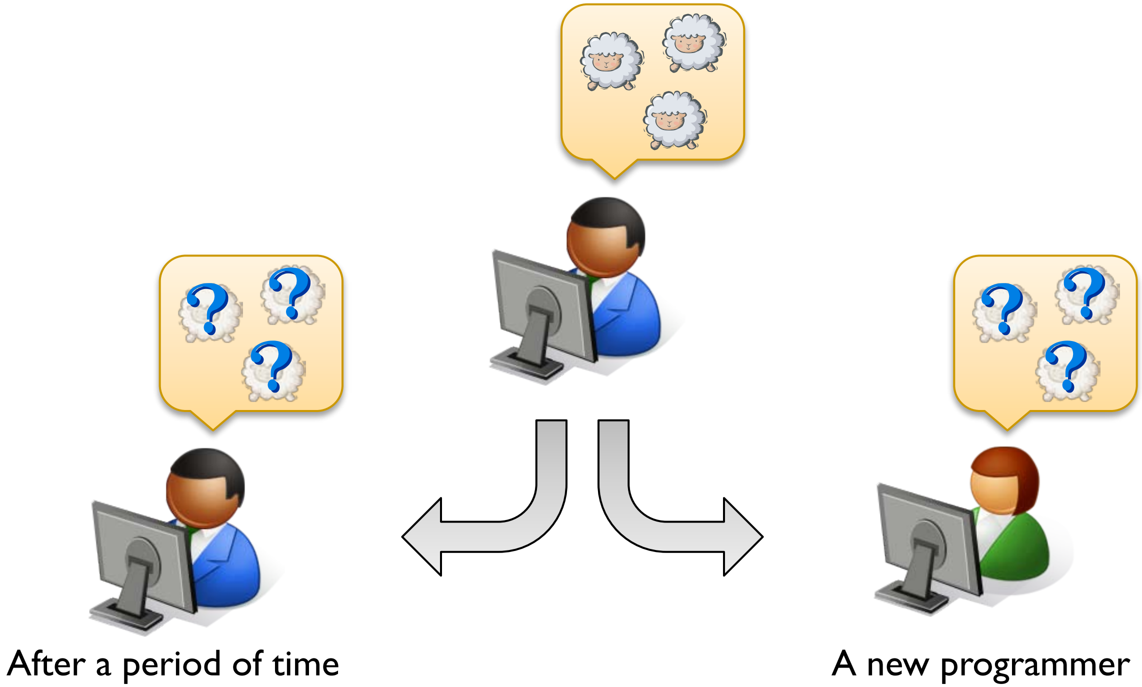
- Clone coverage in software of various sizes and languages reported by various clone detection tools
- Detection results can yield large amounts of data

Clone coverage percentages in different programs

Program	LoC	% of Clones
Linux Kernel	4,365K†	15%
JDK 1.4.2	2,418K‡	8%
JDK 1.3.0	570K¤	9%
Process-Control System	400K§	12%
JHotDraw 7.0.7	71K¥	19%
JavaGenes 0.7.68	45K¥	10%

†Li *et al.*, 2004; ‡Jiang *et al.*, 2007; ¤Kamiya *et al.*, 2002; §Baxter *et al.*, 1998, ¥CloneDR, 2010

Maintaining Clones



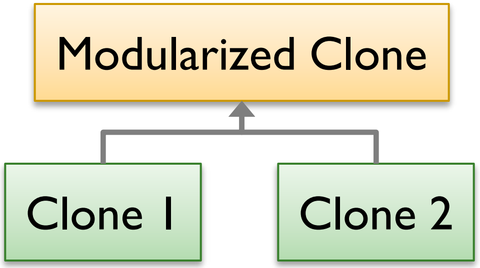
Activity	Class Containing Clones	Correction Date
New statement insertion	ClassDiagramModel	March 2002
	DeploymentDiagramModel	August 2002
Bug fix	SelectionComponentInstance	October 2002
	SelectionComponent	February 2003

Updates of clones in ArgoUML†

†Aversano *et al.*, 2007

Removing Clones through Refactoring

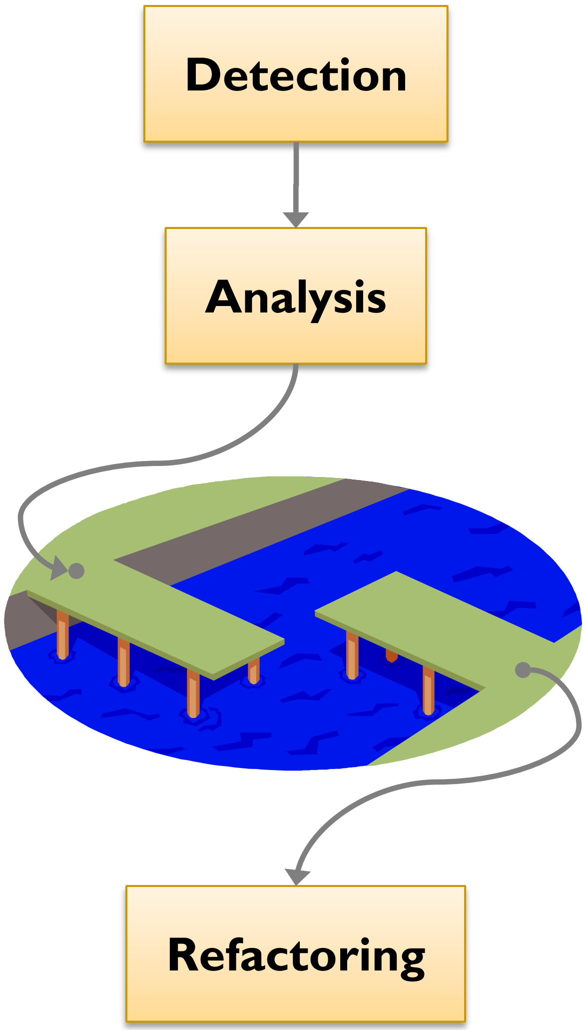
- Modularizing the code represented by clones through appropriate abstractions may improve code quality
  - Less duplicated code to maintain
  - Ease of future maintenance efforts



- *Refactoring* is one means of improving the quality of code
  - The goal of refactoring is to preserve the external behavior of code while improving its internal structure†

†Fowler, 1999

Refactoring Challenge: Process Disconnect



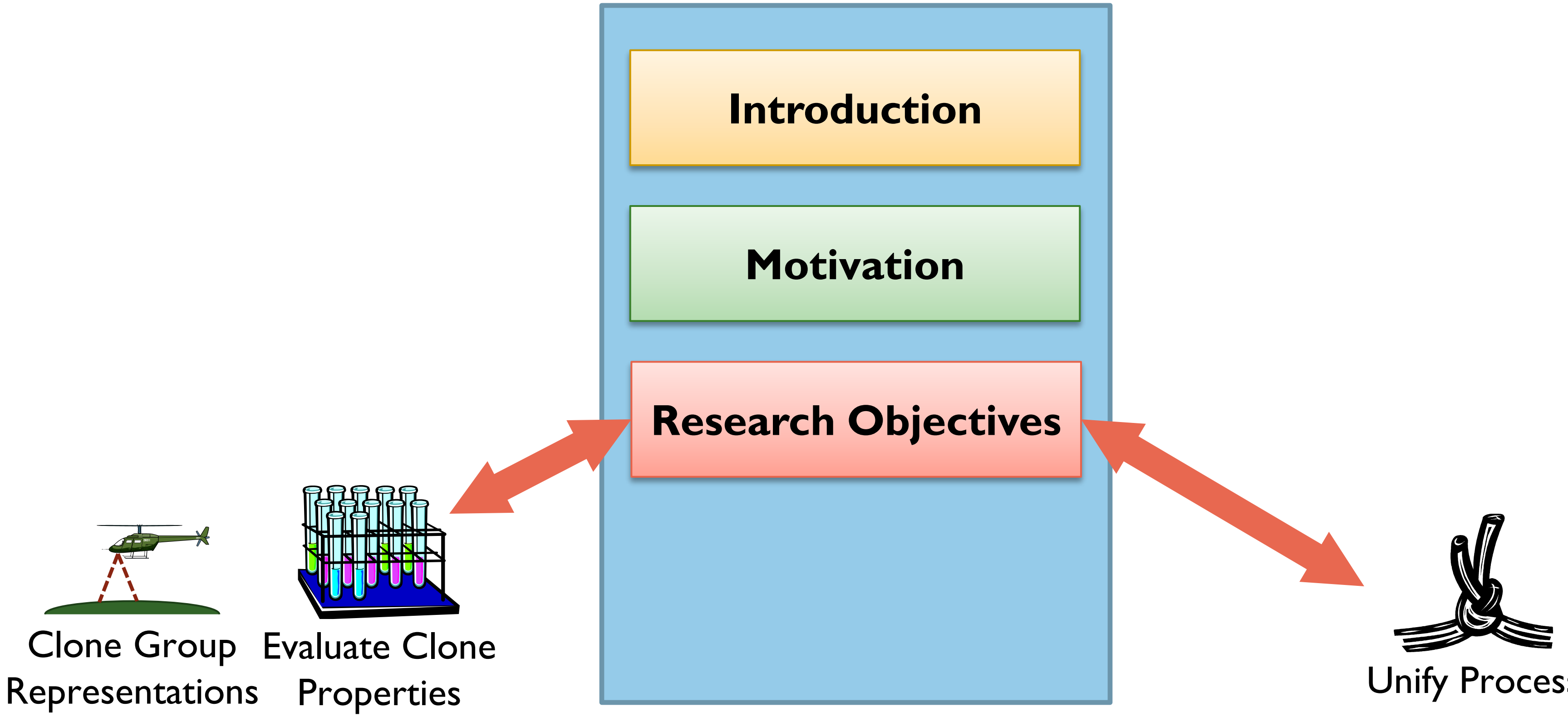
- Techniques such as ARIES[†] and SUPREMO[‡] can assist in determining clones that can potentially be refactored
- However, the task of refactoring clones is delegated to the programmer
- The programmer must either manually refactor the clones or forward the information about the clones to a refactoring engine

[†]Higo *et al.*, 2004; [‡]Koni-N'Sapu, 2001

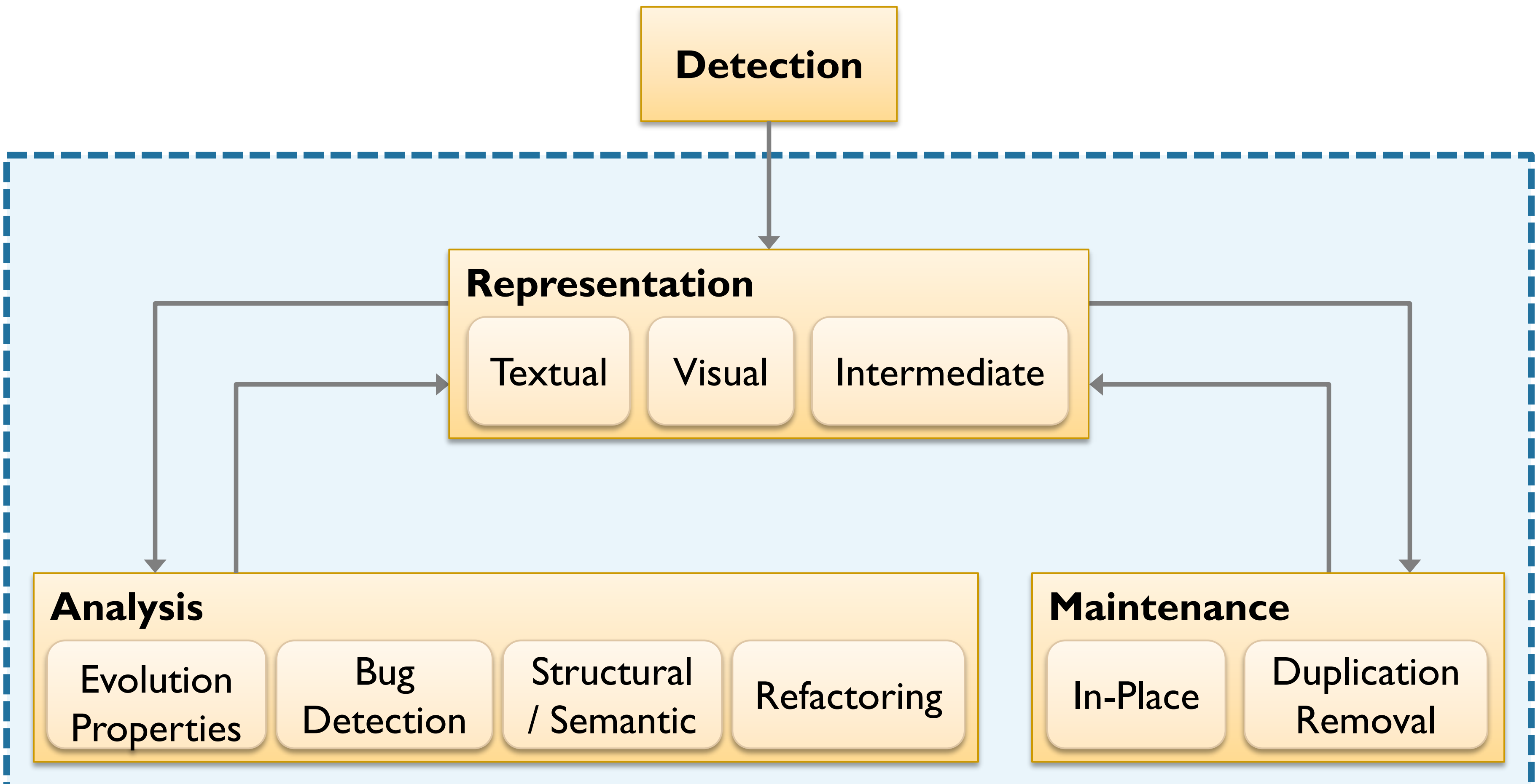
Summary of Challenges

- Representation
  - System-level Views / Scattered Clones
- Analysis
  - Large Amounts of Data
- Refactoring
  - Process Disconnect

Overview of Presentation



Research Scope



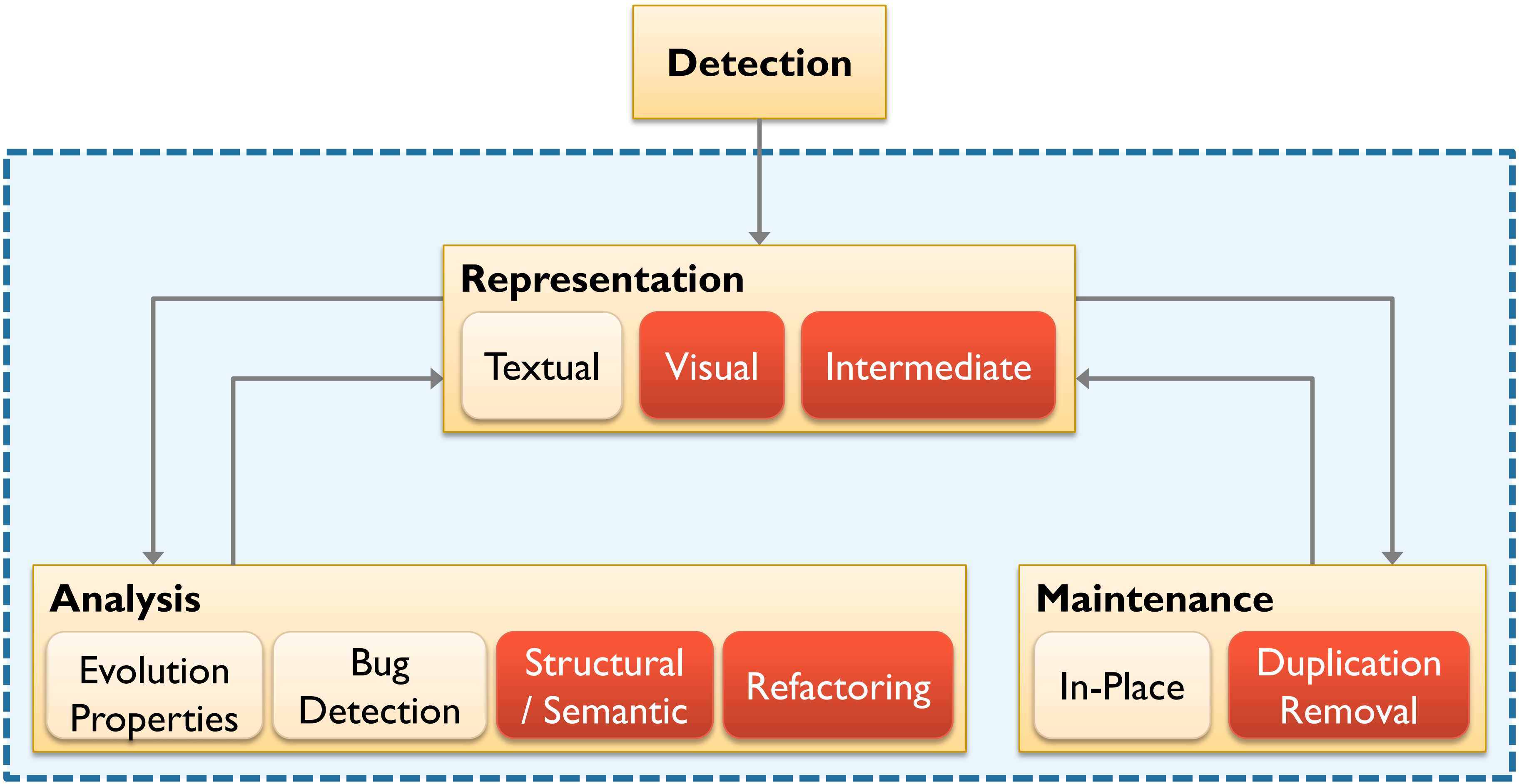
Research Scope

Detection

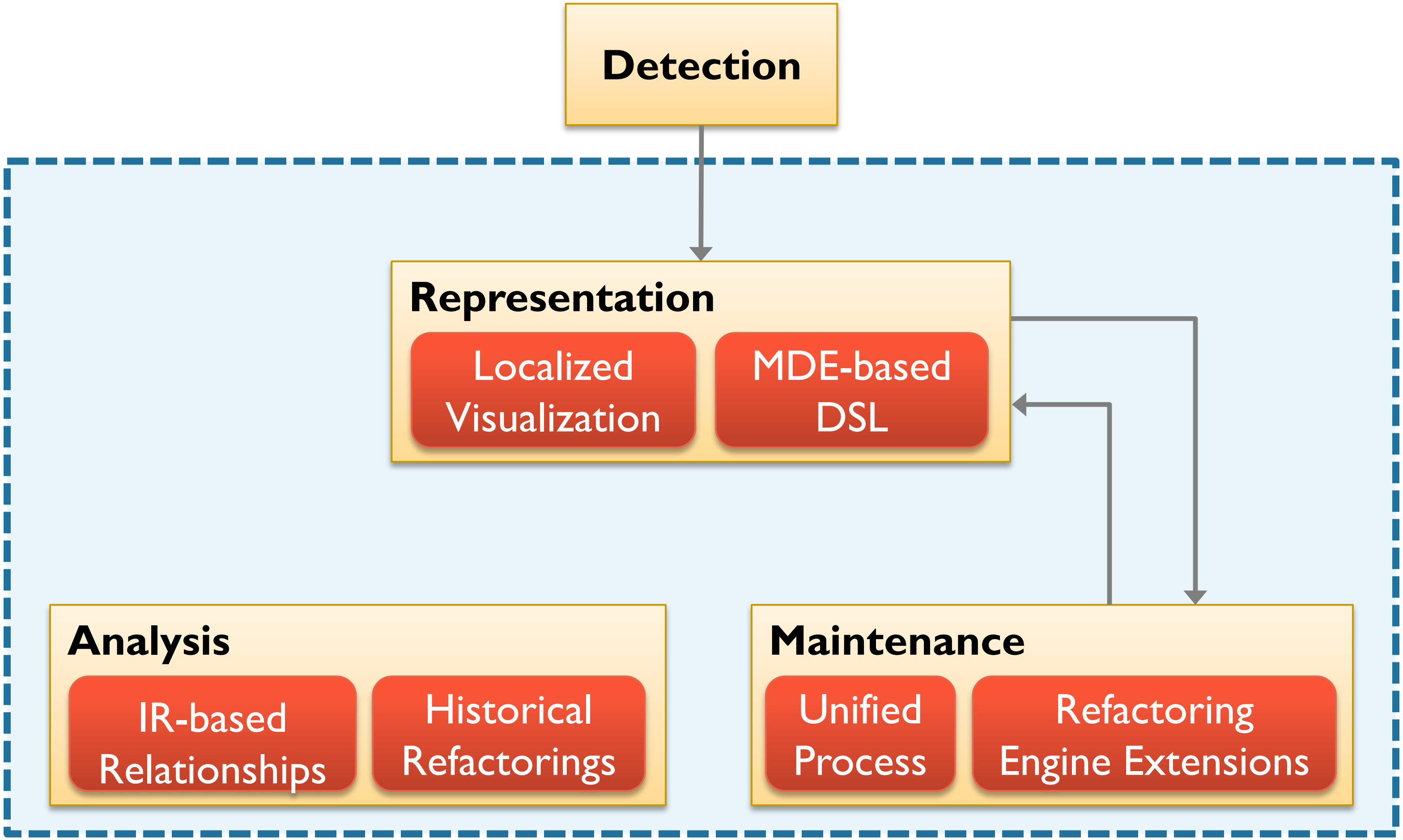
We focus on supporting two aspects related to the maintenance of code clones:

1) clone comprehension through its representation and analysis
2) clone maintenance with a focus on the removal of the duplication associated with the clones

Research Objectives

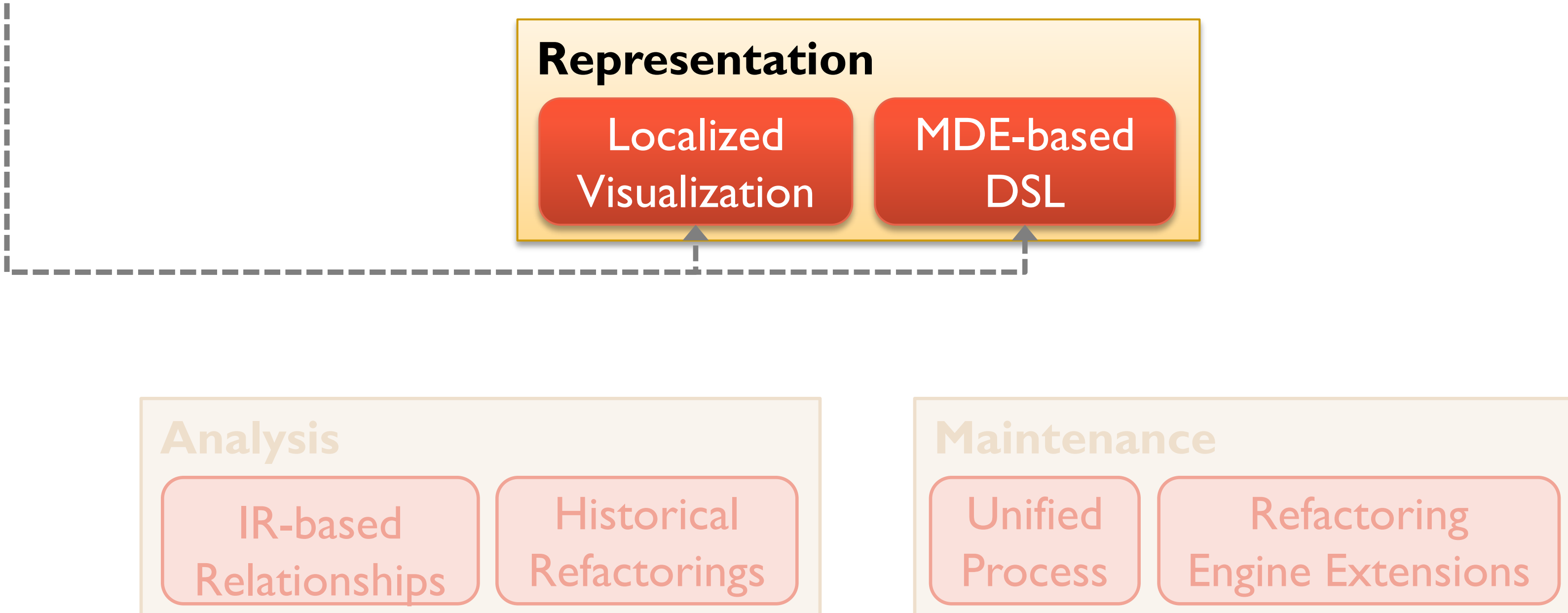


Research Objectives



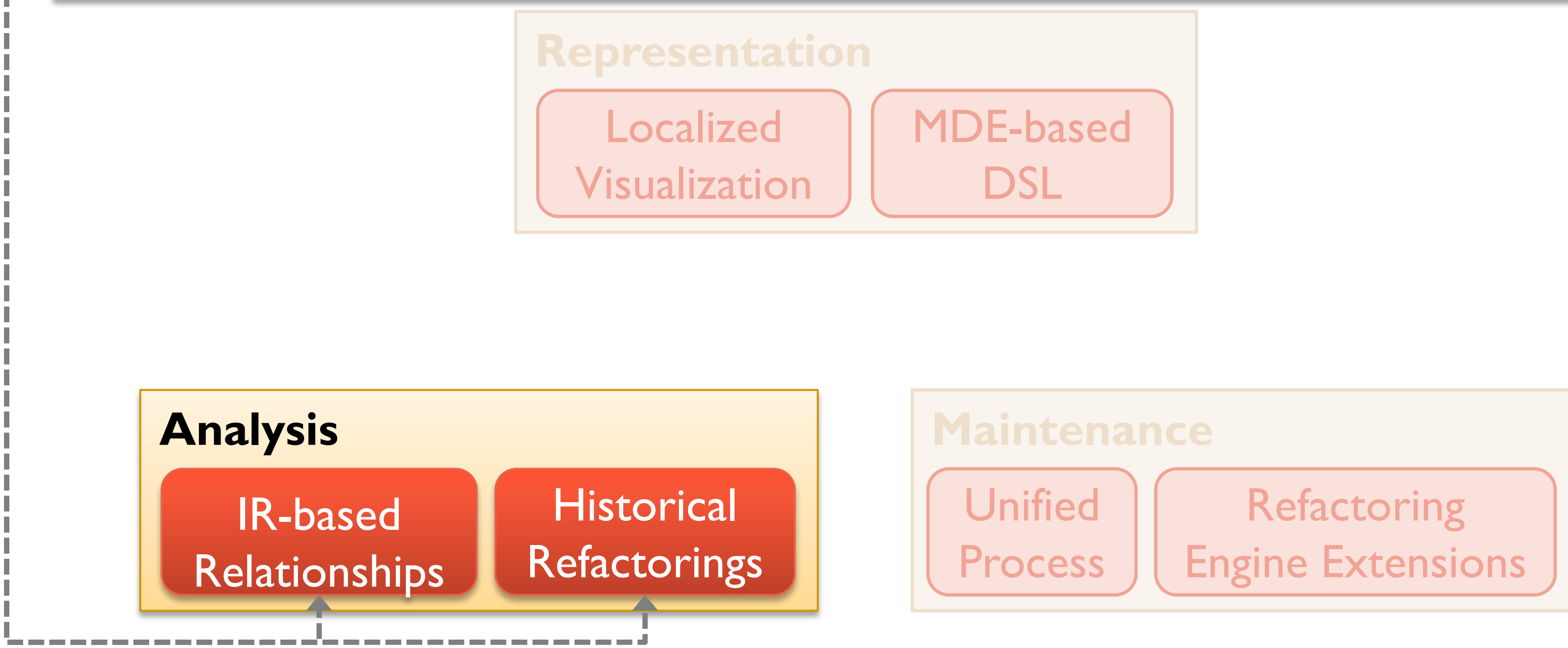
Research Objectives: Representation

- Contribute novel visualizations of clone groups
- Investigate the utilization of Model-Driven Engineering (MDE) techniques to represent and analyze clone groups



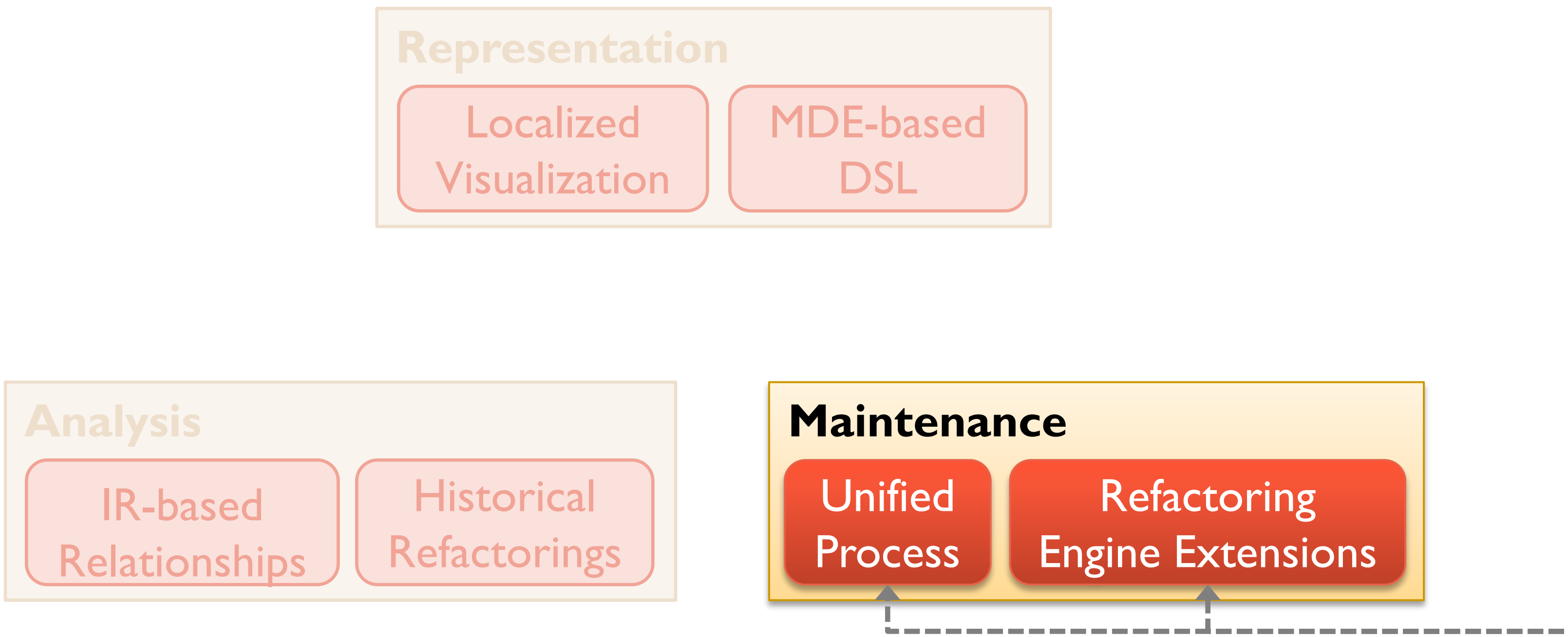
Research Objectives: Analysis

- Discover relationships of clone groups using an Information Retrieval (IR) technique
- Observe relationships of clones and actual historic refactorings

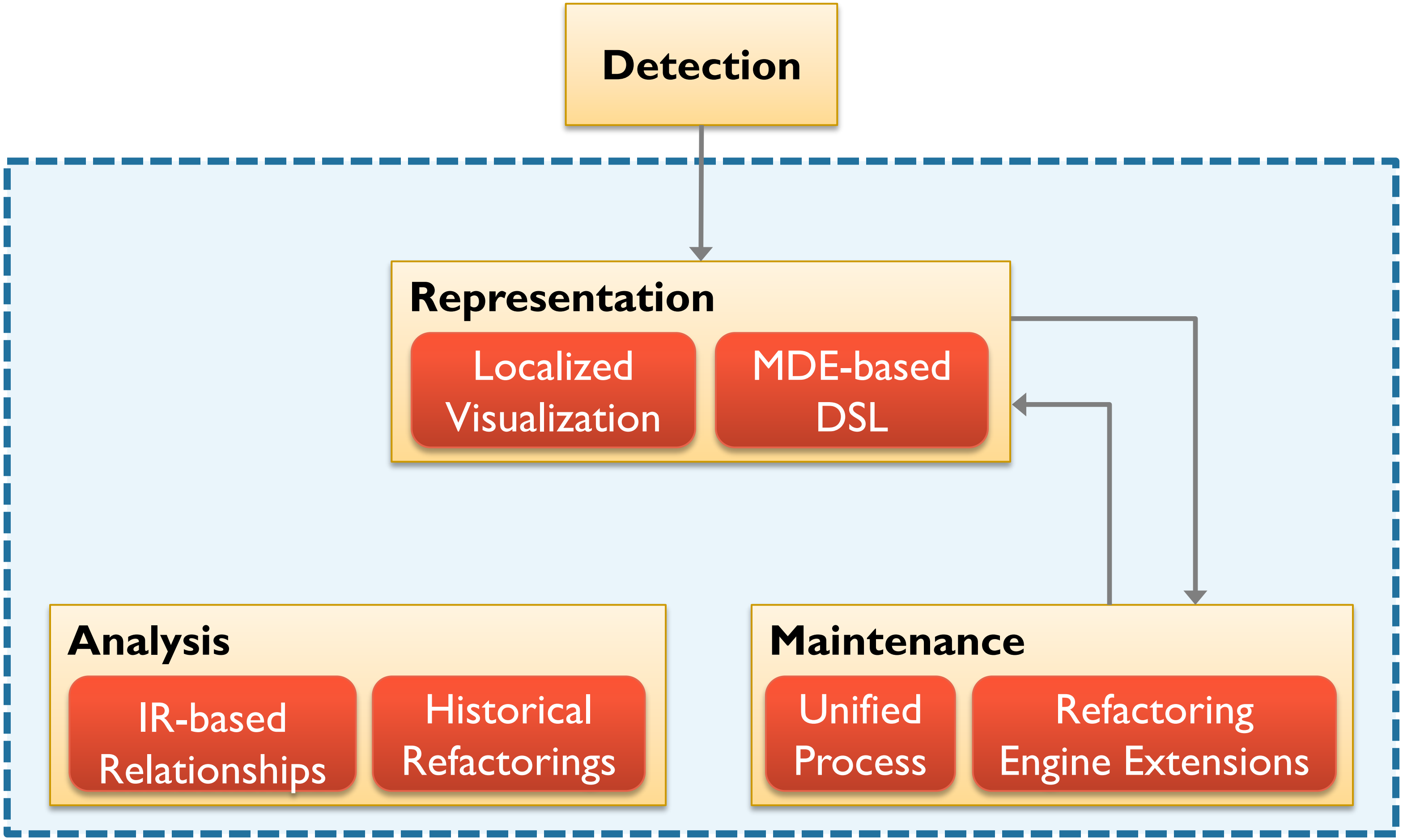


Research Objectives: Refactoring

- Extend the capabilities of an IDE to unify the phases of clone detection, analysis, and refactoring



Research Objectives



Overview of Presentation

Introduction

Motivation

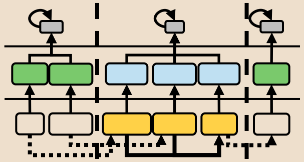
Research Objectives

Approach

Evaluation & Case Studies

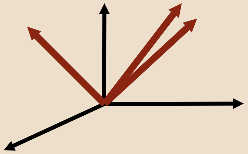
Clone Visualizer

Localized Representation



CoCloRep

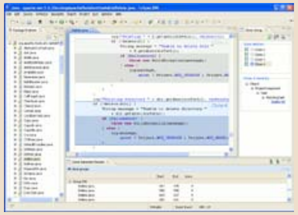
Representation



Clone Group Relationships

Sub-clone Refactoring

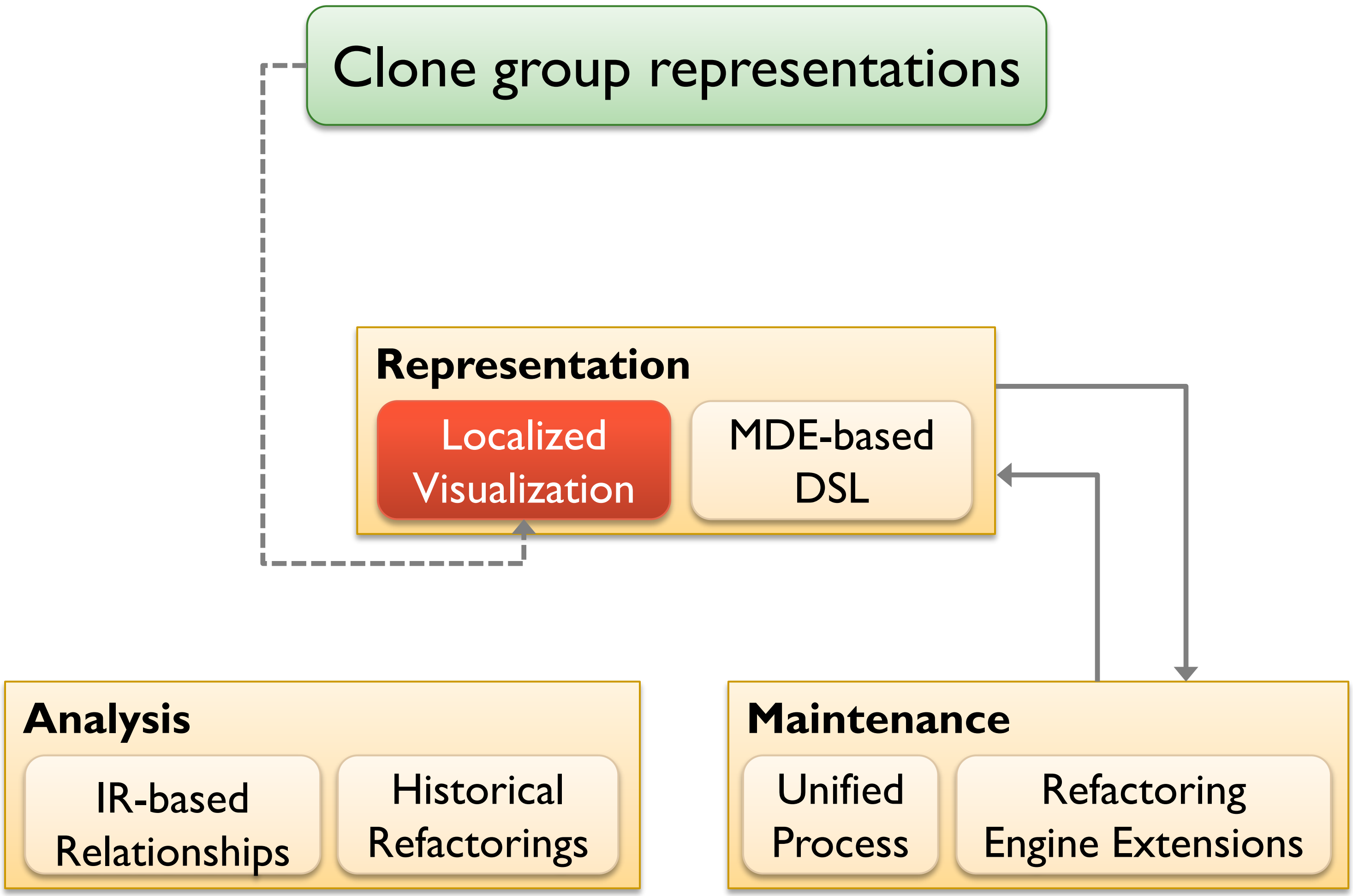
Analysis



CeDAR

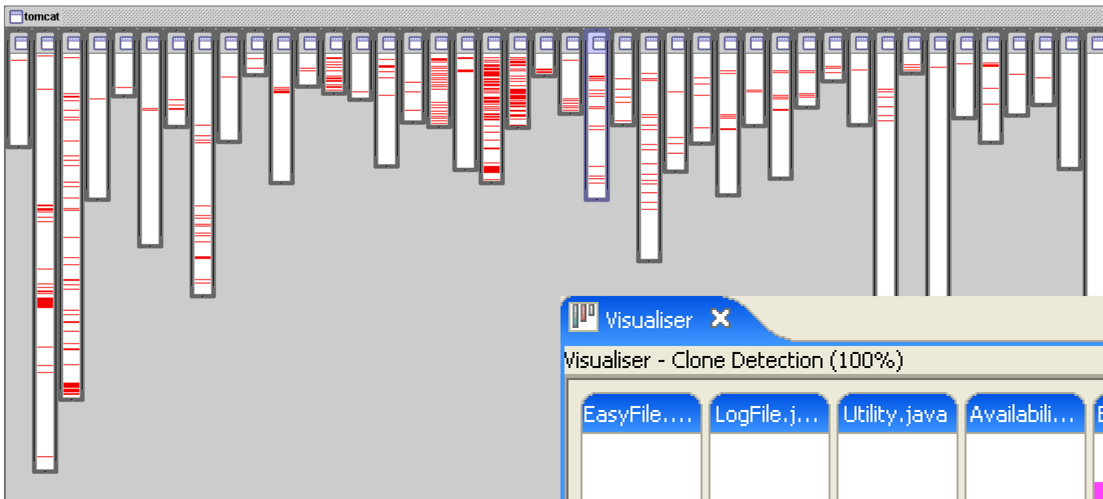
Refactoring

Clone Group Representations



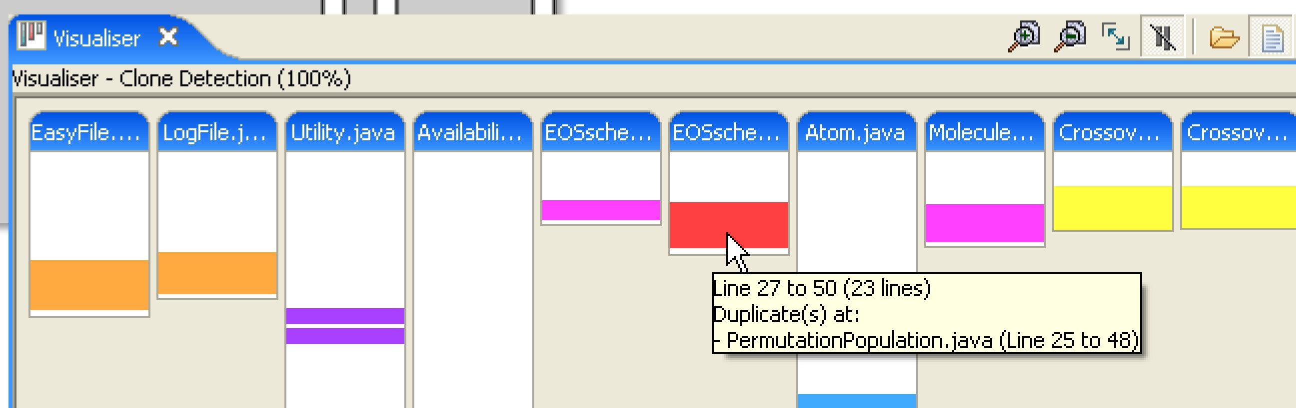
CloViz: Visualization of Clone Detection Results

- Provide an alternative method of viewing clone detection results from the widely used scatter plot
- Extended from the AspectJ Development Tools Visualiser plug-in



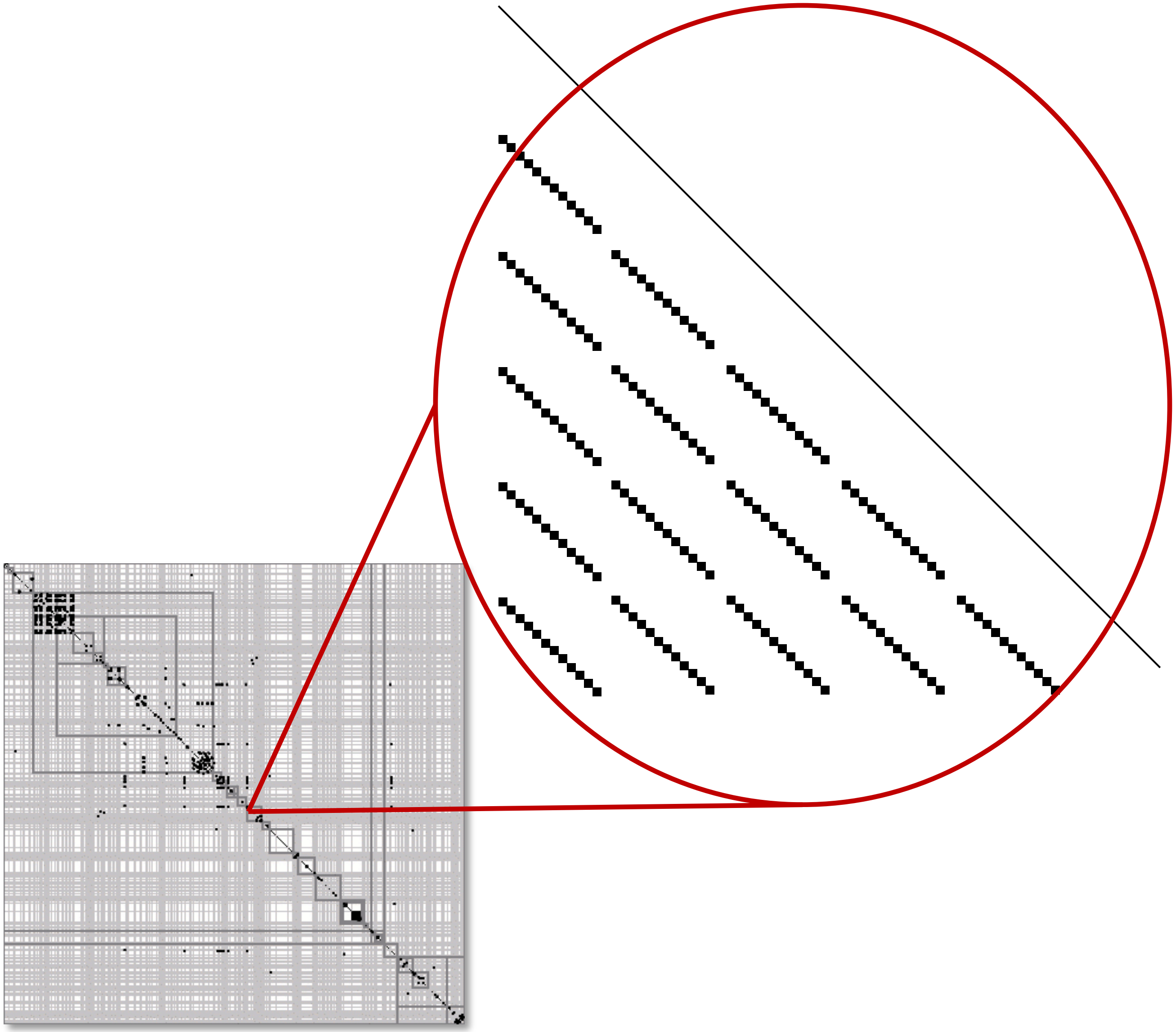
Logging concern in Tomcat†

Visualization view in CloViz

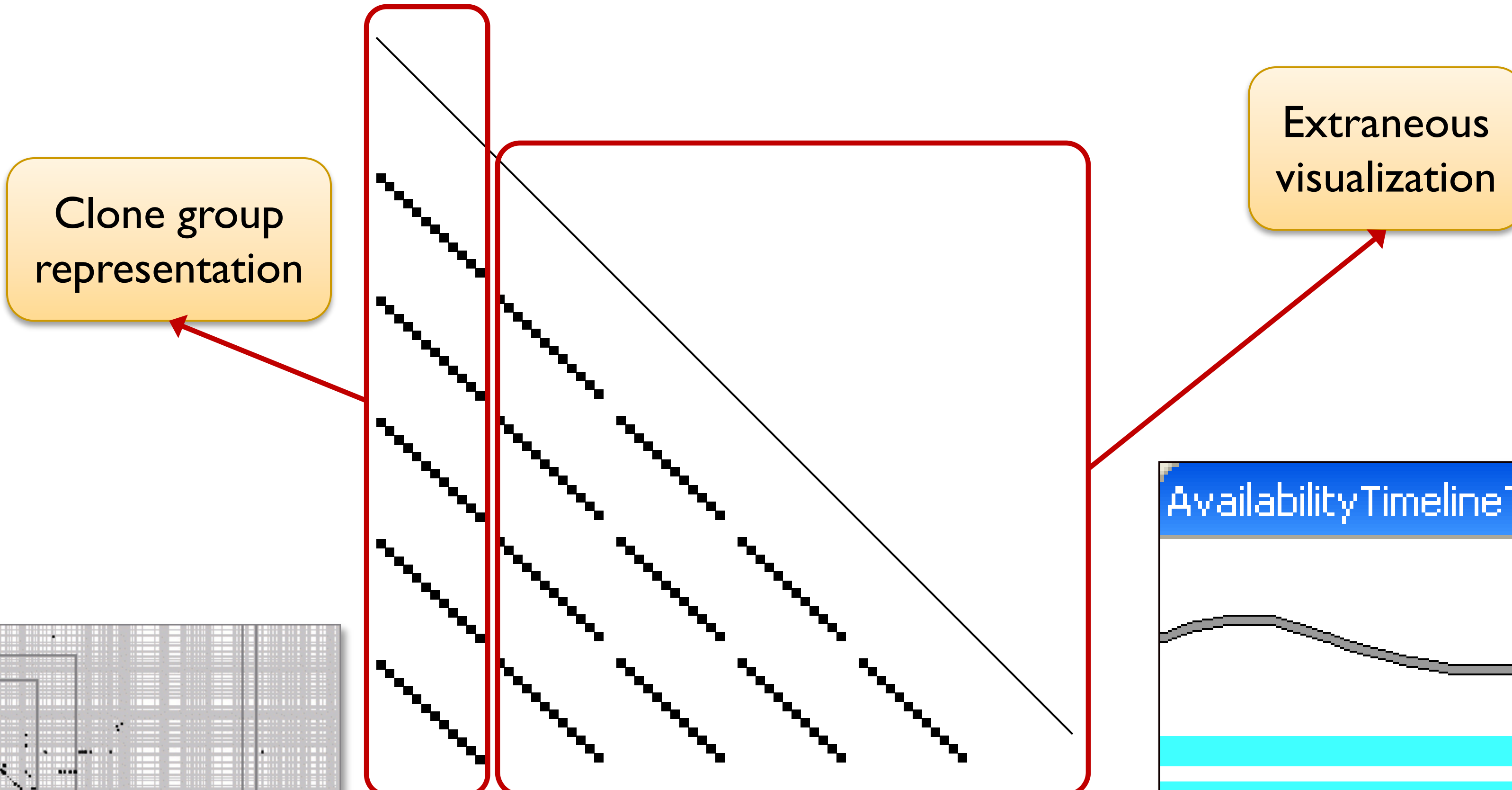


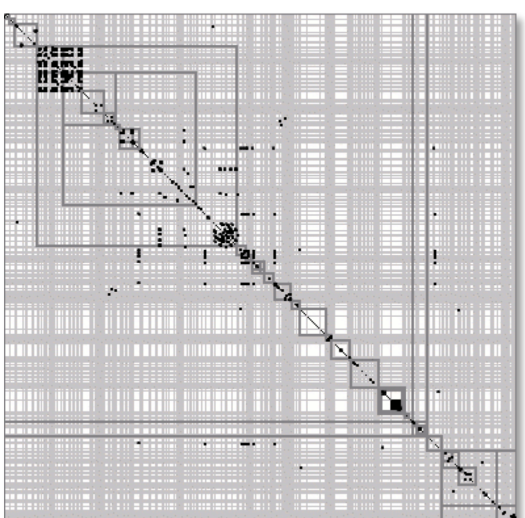
†Hilsdale and Kersten, 2004

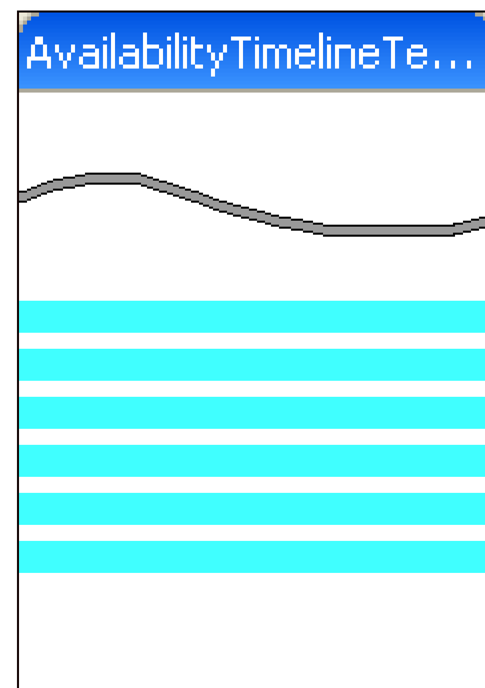
Comparison with Scatter Plot



Comparison with Scatter Plot



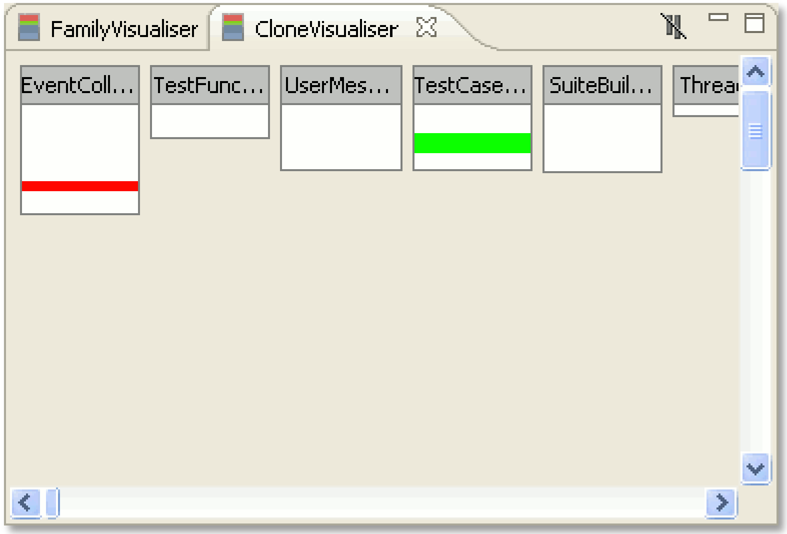




Visualizer Utilization

- Visualization technique included in clone detection plug-in developed at Technische Universität München
  - Part of ConQAT (Continuous Quality Assessment Toolkit)

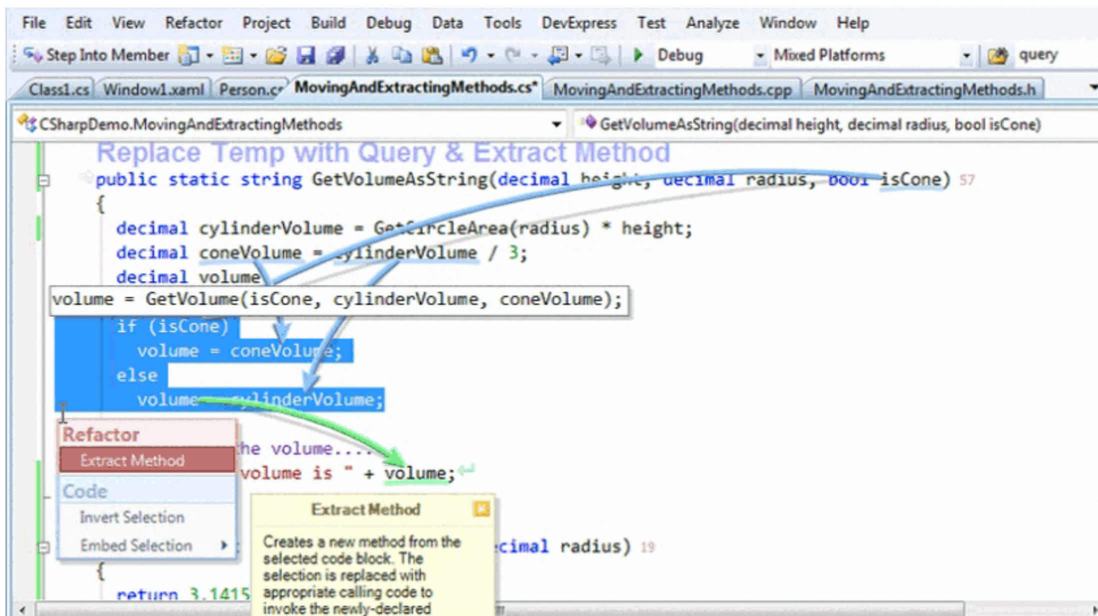
Screen shot of visualizer view in ConQAT†



†ConQAT, 2010

Representation within Source Editor

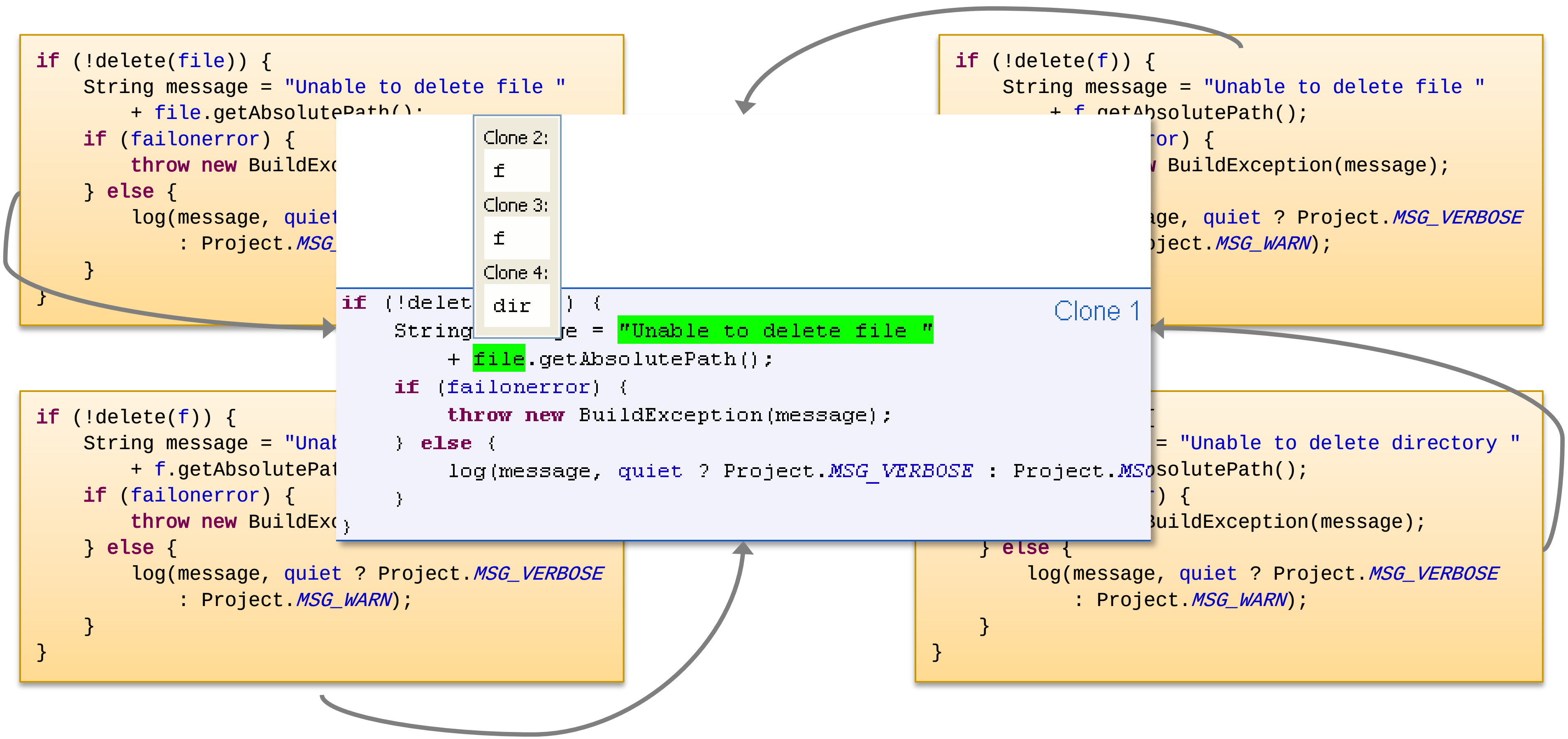
- Refactoring activity requires multiple modal dialog boxes
  - Separation between program editing and refactoring tasks
- A solution: visualize refactoring changes directly in the source editor



Screen shot of Refactor! Pro†

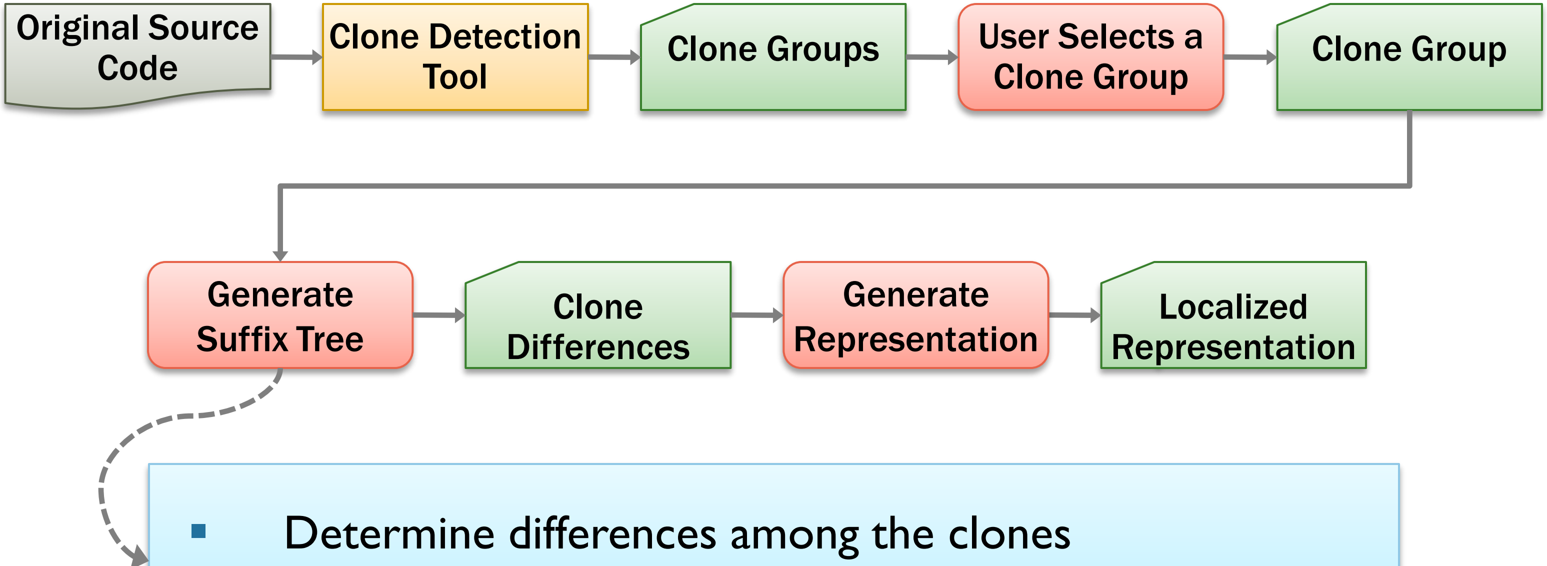
Localized Clone Representation

- Represent a clone group in a localized manner
  - Parameterized differences visualized in representation



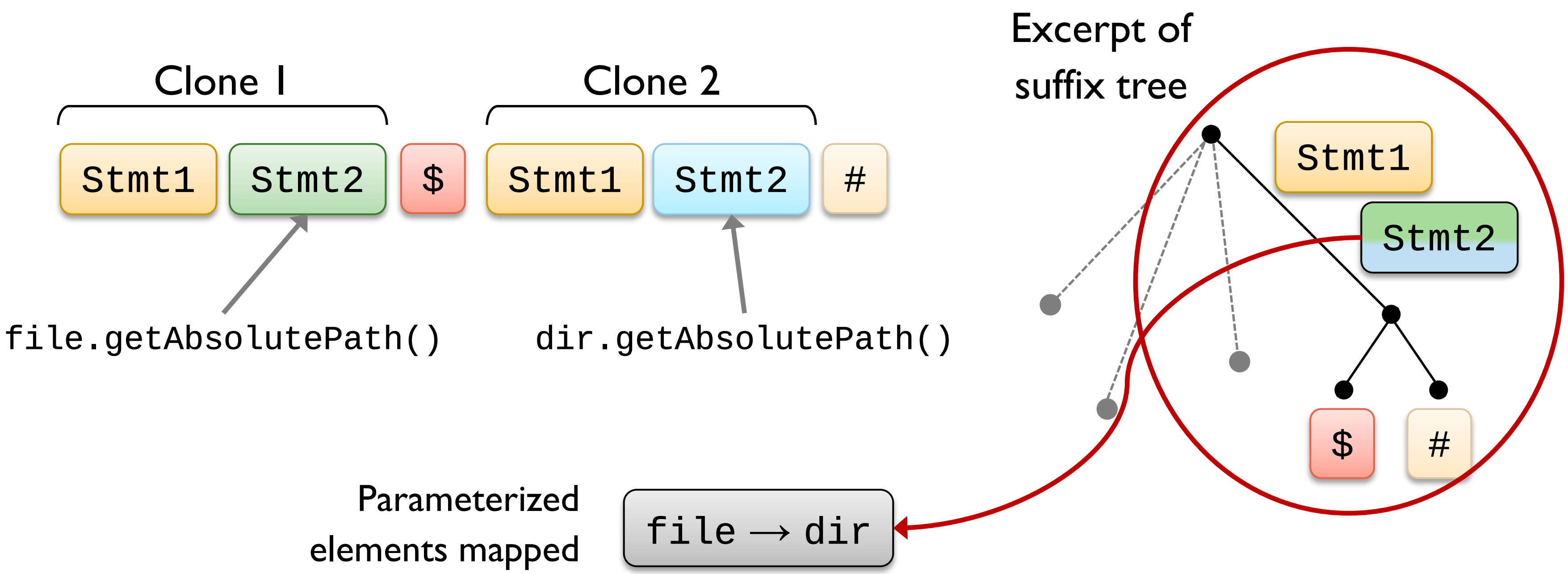
Displaying Clones in a Localized Manner

- Localized representation is displayed after a user selects a clone group



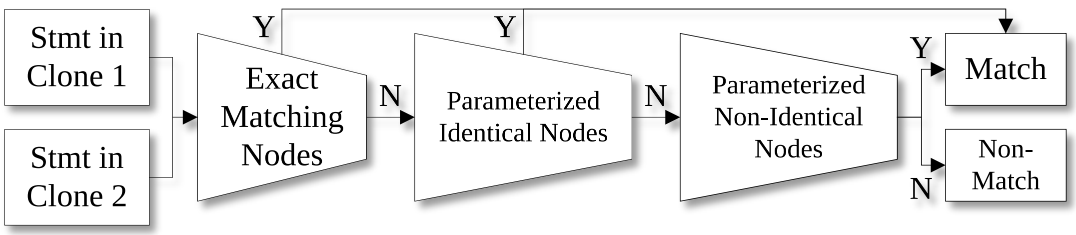
- Determine differences among the clones
- Differences based on first-level statement comparisons

Detecting Parameterized Elements



- A *suffix tree* is generated on the AST nodes representing the statements of a group of clones
- Elements in nodes containing *allowed* differences are mapped together

Statement Similarity Levels



- Comparing two statements of two clones
  - Level 1: Corresponding nodes are identical and match each other exactly
  - Level 2: Corresponding nodes are identical, but can contain allowed parameterized differences
    - `MethodInvocation`, `NumberLiteral`, `QualifiedName`, `SimpleName`, and `StringLiteral`
  - Level 3: Corresponding nodes are *not* identical, but both are correspond to types from the Level 2 comparison

Example Representations

- Exact statements, statements with parameterized differences, and non-matching statements

Level 1

Level 2

Non-match

Level 3

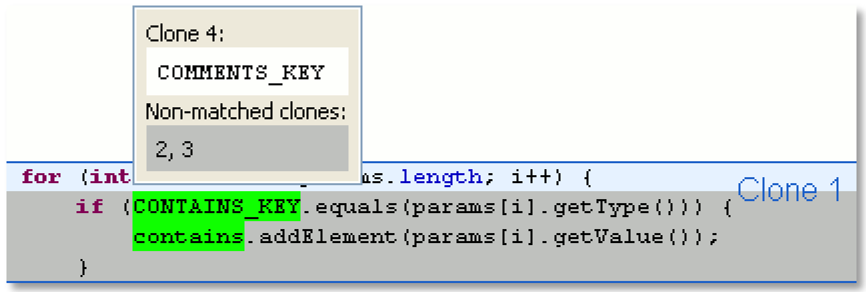
Clone 1

```
String classname = (String) e.nextElement();
String location
    = classname.replace('.', File.separatorChar) + ".class";
File classFile = new File(config.srcDir, location);
if (classFile.exists()) {
    checkEntries.put(location, classFile);
    log("dependent class: " + classname + " - " + classFile,
        Project.MSG_VERBOSE);
}
```

```
String classname = (String) e.nextElement();
String filename = classname.replace('.', File.separatorChar);
filename = filename + ".class";
File depFile = new File(basedir, filename);
if (depFile.exists() && parentSet.containsKey(filename)) {
    // This is included
    included.addElement(filename);
}
```

Example Representations

- Sub-groups of clones
  - Tighter similarities: Clones 1 and 4 vs. Clones 2 and 3



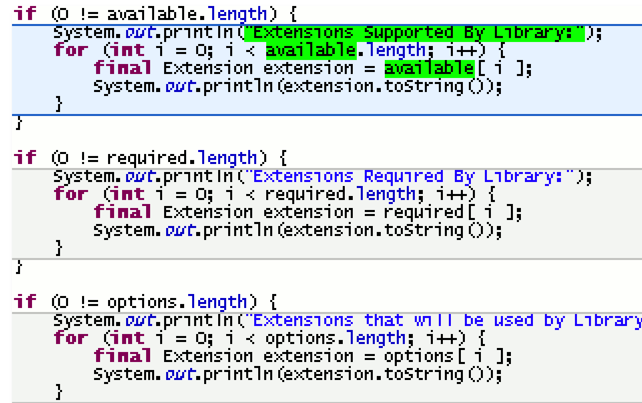

```
for (int i = 0; i < params.length; i++) {
    if (CONTAINS_KEY.equals(params[i].getType())) {
        contains.addElement(params[i].getValue());
    }
}
```
Clone 1

```
for (int i = 0; i < params.length; i++) {
    if (PREFIX_KEY.equals(params[i].getName())) {
        prefix = params[i].getValue();
        break;
    }
}
```
Clone 2

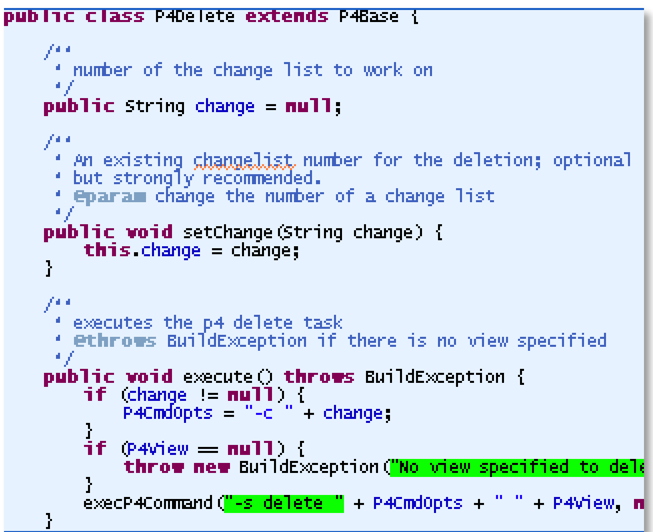
```
for (int i = 0; i < params.length; i++) {
    if (LINE_BREAKS_KEY.equals(params[i].getName())) {
        userDefinedLineBreaks = params[i].getValue();
        break;
    }
}
```
Clone 3

```
for (int i = 0; i < params.length; i++) {
    if (COMMENTS_KEY.equals(params[i].getType())) {
        comments.addElement(params[i].getValue());
    }
}
```
Clone 4

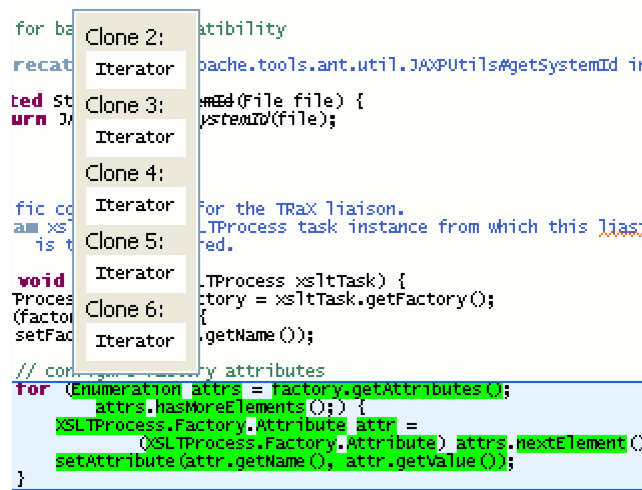
Clone Properties Based on Visualizations



Quick summary of neighboring clones



Clones with small differences



Identifying clone with more difference

Evaluation: Fully Representing Clones

- Considers the number of clone groups (i.e., #CG) that can be appropriately represented
  - Evaluated on multiple open source Java projects

Project	#CG	Exact (%)	Param (%)	StmtDiff (%)	Mixed (%)
Apache Ant 1.6.5	429	61 (14%)	152 (35%)	131 (31%)	85 (20%)
ArgoUML 0.26	650	61 (9%)	214 (33%)	124 (19%)	251 (39%)
Jakarta-JMeter 2.3.2	377	77 (20%)	158 (42%)	71 (19%)	71 (19%)
JBoss AOP 2.1.5	159	51 (32%)	81 (51%)	14 (9%)	13 (8%)
JFreeChart 1.0.10	847	151 (18%)	415 (49%)	168 (20%)	113 (13%)
JRuby 1.4.0	318	113 (36%)	70 (22%)	63 (20%)	72 (23%)
EMF 2.4.1	285	54 (19%)	136 (48%)	52 (18%)	42 (15%)
JEdit 4.2	345	91 (26%)	120 (35%)	88 (26%)	46 (13%)
Squirrel-SQL 3.0.3	428	78 (18%)	164 (38%)	70 (16%)	116 (27%)

Evaluation: Fully Representing Clones

- "Exact" → Clones that match each other exactly

Project	#CG	Exact (%)	Param (%)	StmtDiff (%)	Mixed (%)
Apache Ant 1.6.5	429	61 (14%)	152 (35%)	131 (31%)	85 (20%)
ArgoUML 0.26	650	61 (9%)	214 (33%)	124 (19%)	251 (39%)
Jakarta-JMeter 2.3.2	377	77 (20%)	158 (42%)	71 (19%)	71 (19%)
JBoss AOP 2.1.5	159	51 (32%)	81 (51%)	14 (9%)	13 (8%)
JFreeChart 1.0.10	847	151 (18%)	415 (49%)	168 (20%)	113 (13%)
JRuby 1.4.0	318	113 (36%)	70 (22%)	63 (20%)	72 (23%)
EMF 2.4.1	285	54 (19%)	136 (48%)	52 (18%)	42 (15%)
JEdit 4.2	345	91 (26%)	120 (35%)	88 (26%)	46 (13%)
Squirrel-SQL 3.0.3	428	78 (18%)	164 (38%)	70 (16%)	116 (27%)

Evaluation: Fully Representing Clones

- "Param" → Clone groups with parameterized differences
  - Majority of the cases except ArgoUML and JRuby
  - Four cases almost half of the instances

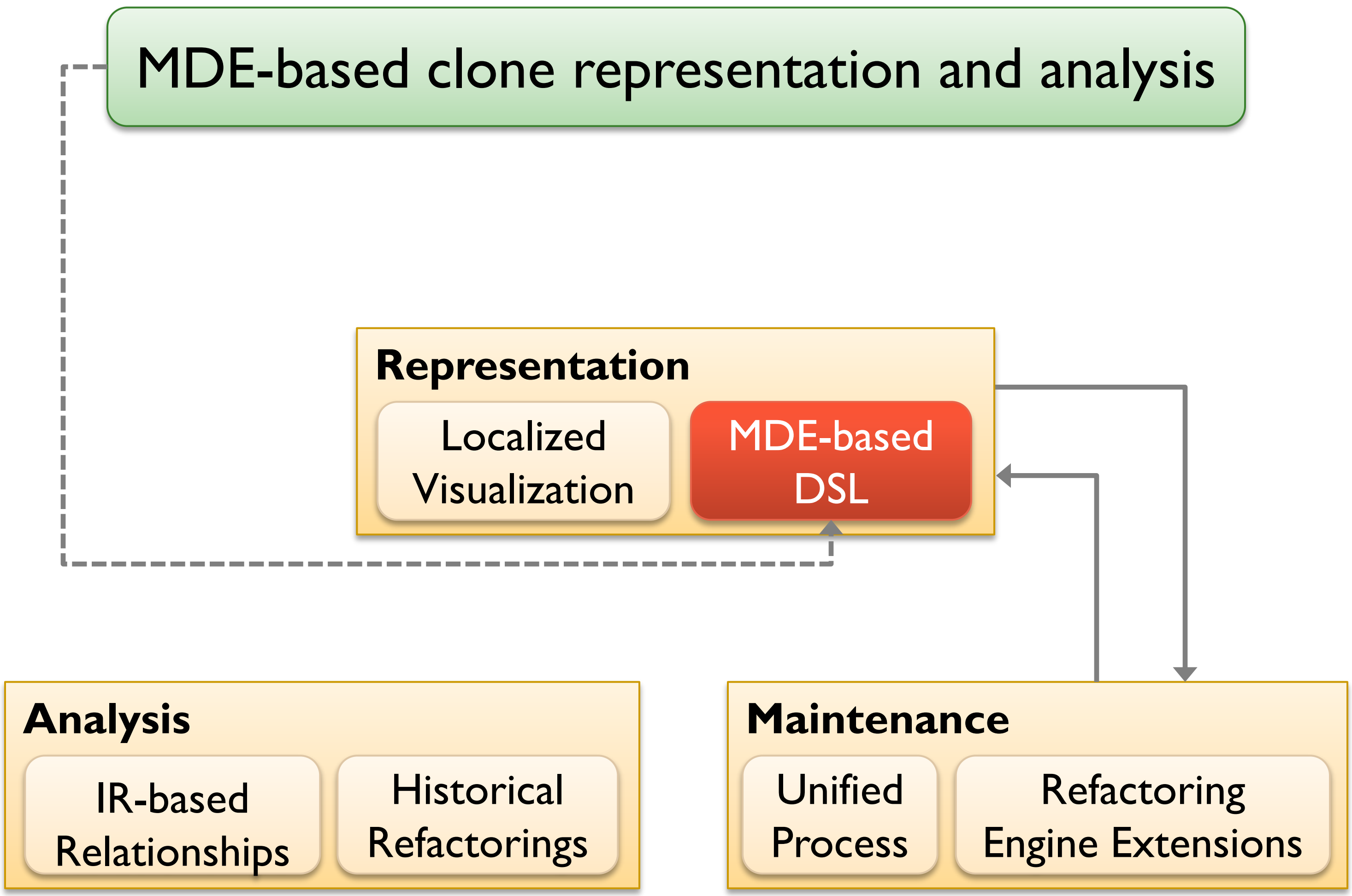
Project	#CG	Exact (%)	Param (%)	StmtDiff (%)	Mixed (%)
Apache Ant 1.6.5	429	61 (14%)	152 (35%)	131 (31%)	85 (20%)
ArgoUML 0.26	650	61 (9%)	214 (33%)	124 (19%)	251 (39%)
Jakarta-JMeter 2.3.2	377	77 (20%)	158 (42%)	71 (19%)	71 (19%)
JBoss AOP 2.1.5	159	51 (32%)	81 (51%)	14 (9%)	13 (8%)
JFreeChart 1.0.10	847	151 (18%)	415 (49%)	168 (20%)	113 (13%)
JRuby 1.4.0	318	113 (36%)	70 (22%)	63 (20%)	72 (23%)
EMF 2.4.1	285	54 (19%)	136 (48%)	52 (18%)	42 (15%)
JEdit 4.2	345	91 (26%)	120 (35%)	88 (26%)	46 (13%)
Squirrel-SQL 3.0.3	428	78 (18%)	164 (38%)	70 (16%)	116 (27%)

Evaluation: Fully Representing Clones

- "StmtDiff" → Clone groups with statement differences
- "Mixed" → Clone groups containing both "Param" and "StmtDiff"

Project	#CG	Exact (%)	Param (%)	StmtDiff (%)	Mixed (%)
Apache Ant 1.6.5	429	61 (14%)	152 (35%)	131 (31%)	85 (20%)
ArgoUML 0.26	650	61 (9%)	214 (33%)	124 (19%)	251 (39%)
Jakarta-JMeter 2.3.2	377	77 (20%)	158 (42%)	71 (19%)	71 (19%)
JBoss AOP 2.1.5	159	51 (32%)	81 (51%)	14 (9%)	13 (8%)
JFreeChart 1.0.10	847	151 (18%)	415 (49%)	168 (20%)	113 (13%)
JRuby 1.4.0	318	113 (36%)	70 (22%)	63 (20%)	72 (23%)
EMF 2.4.1	285	54 (19%)	136 (48%)	52 (18%)	42 (15%)
JEdit 4.2	345	91 (26%)	120 (35%)	88 (26%)	46 (13%)
Squirrel-SQL 3.0.3	428	78 (18%)	164 (38%)	70 (16%)	116 (27%)

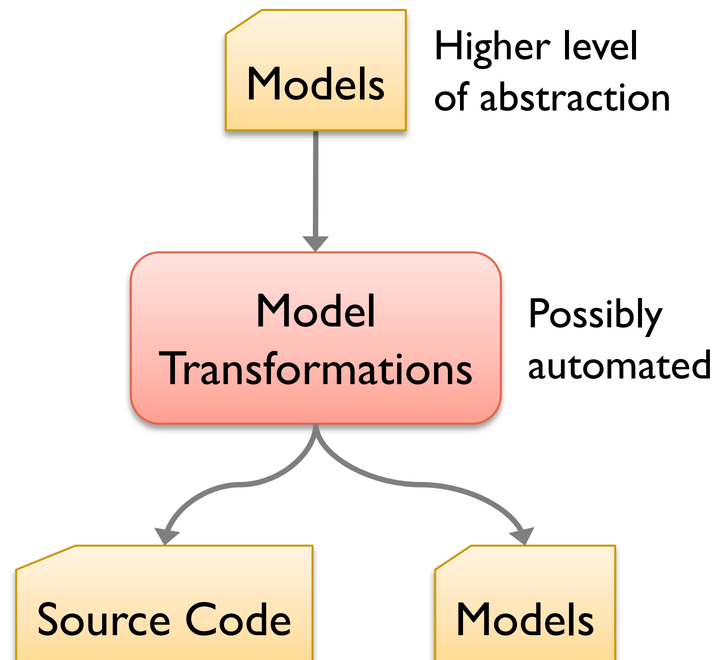
CoCloRep: Code Clone Representation



CoCloRep: Code Clone Representation

- An investigation into the development of a Domain-Specific Language (DSL) for representing code clones
- Utilizing Model-Driven Engineering (MDE) in the context of clone analysis

MDE is concerned with raising the abstraction level of software development by utilizing models to specify the application



First DSL: Clone Representation

```
-- clone 1
int g;
int f = g + 3;
i = i + 1;
c = f + m;
```

```
-- clone 2
int q;
int p = q + 3;
c = p + m;
```

Variabilities

Commonalities

```
-- clone instances
instance r = cg(f, g) {
  t {
    i = i + 1;
  }
};

instance s = cg(p, q);
```

```
-- clone group
clone cg($a, $b) {
  int $b;
  int $a = $b + 3;
  {{ t }}
  c = $a + m;
}
```

Second DSL: Commands

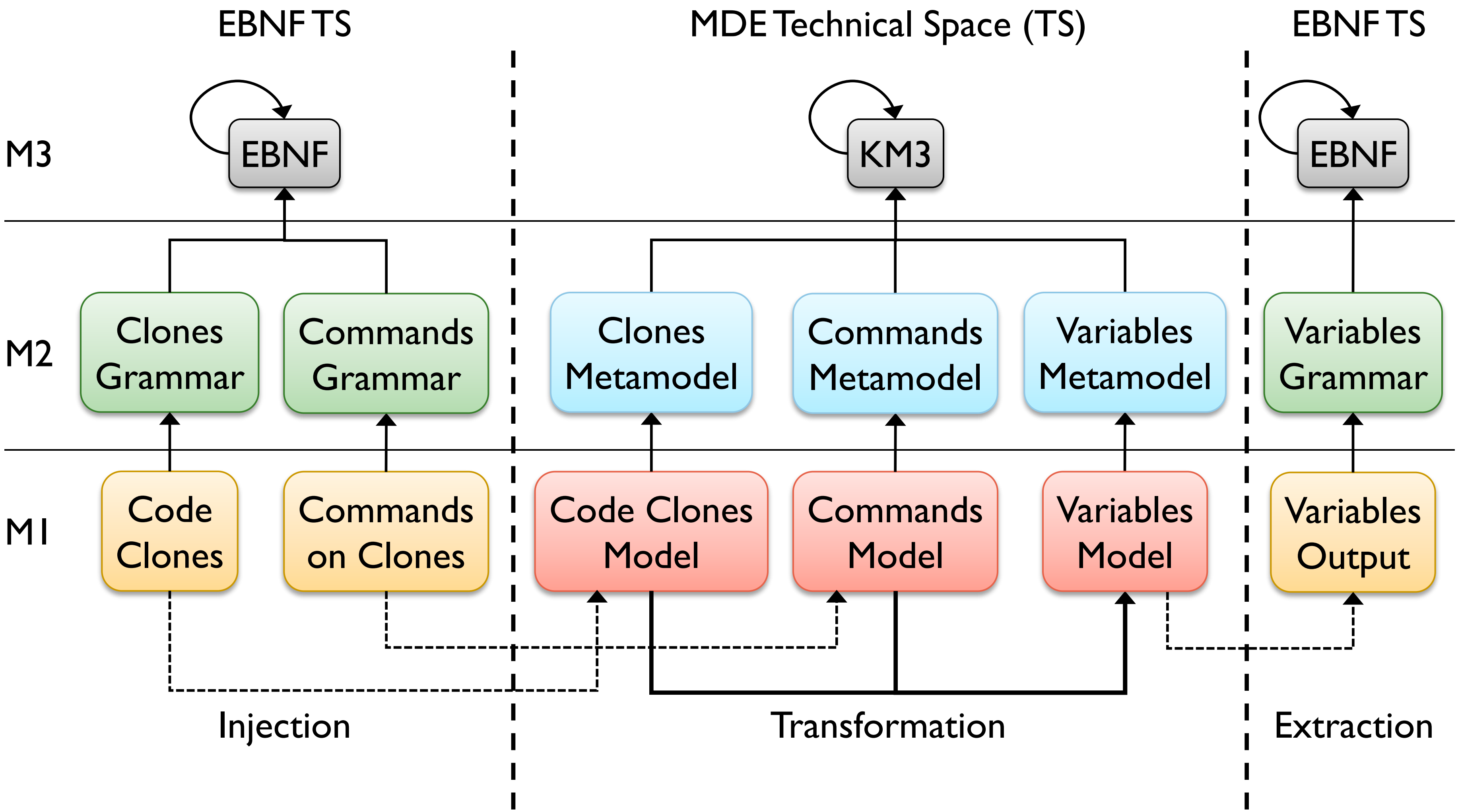
- Input

```
variables cg;
```

- Output

```
Variable information for clone group cg
Declared variables:
  b
  a
Outside assigned variables:
  c
  i (in instance r)
Outside non-assigned variables:
  m
```

Model Transformation Process



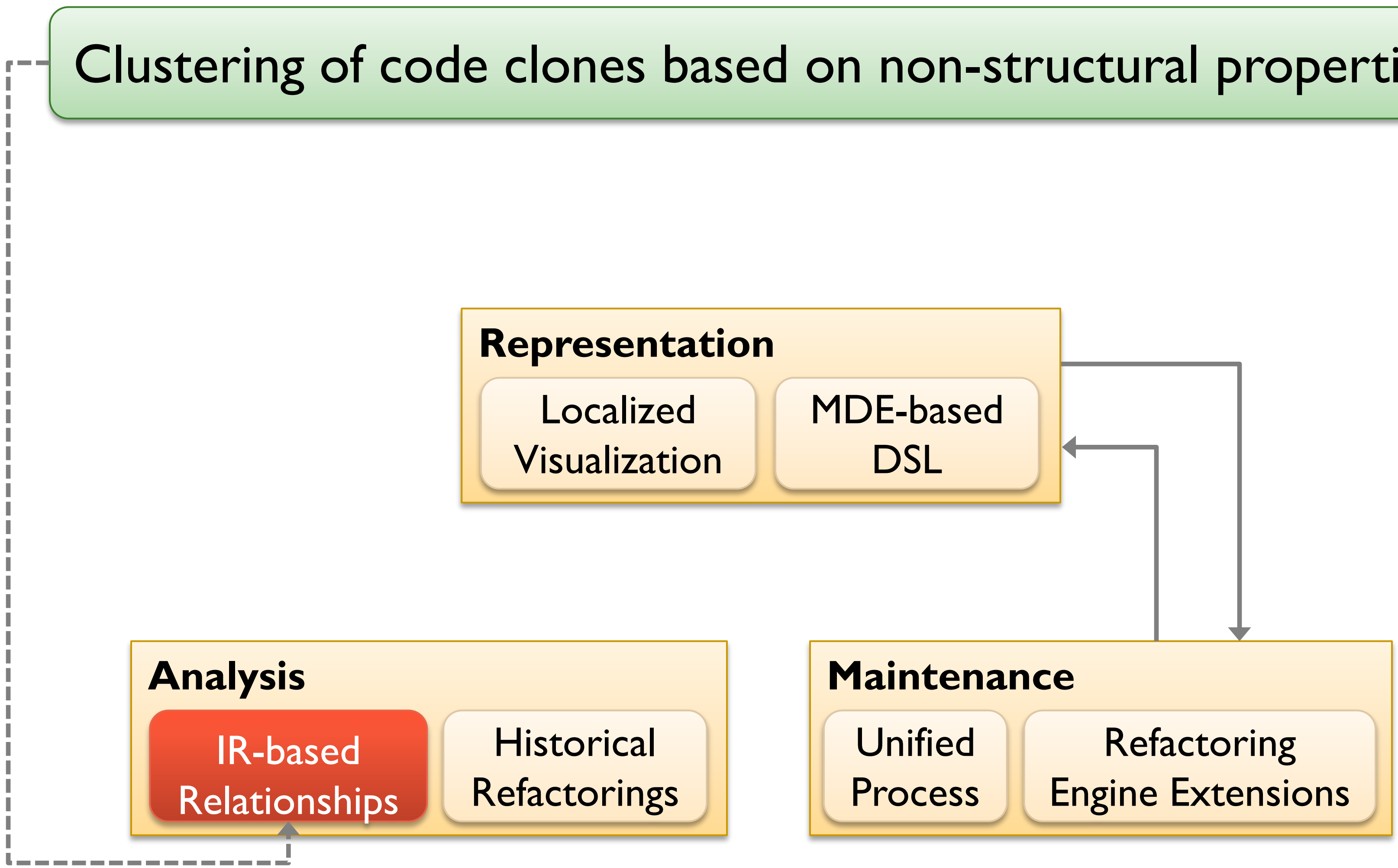
Representation and Analysis in CoCloRep

- Representation of clones (as models)
  - Commonalities stored in clone groups
  - Variabilities stored in clone instances
  - Modified / combined AST of all clone instances
- Analysis of clones (via model transformations)
  - Transformations with both declarative and imperative constructs
  - Requires more complex transformations
    - Not one-to-one

Summary

- Clone group representation
  - Representations provide a low-level view of clones and a centralized location to view clone properties
- Maintenance
  - Visual representations provide a quick summary of clone properties
    - i.e., location of clones, complexity of clone differences
  - Preliminary investigation of using MDE for clone refactoring

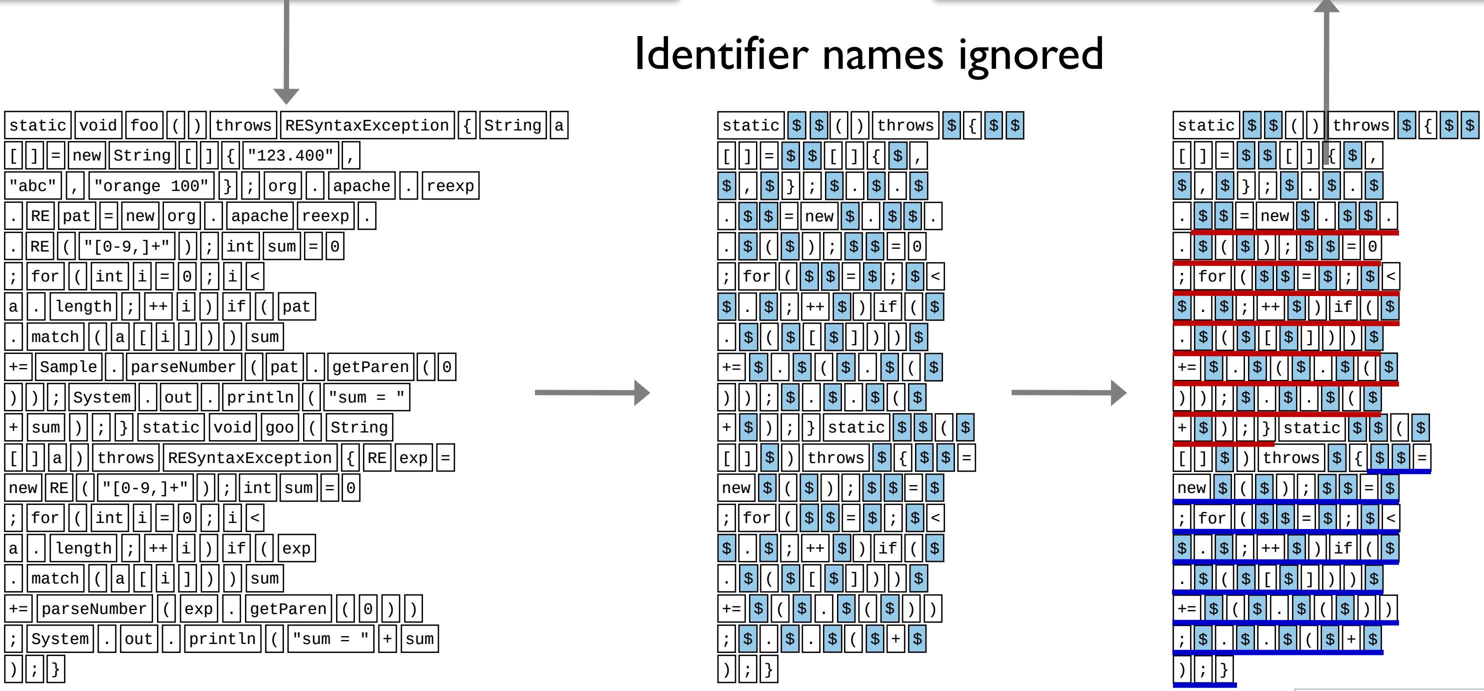
Clone analysis using Information Retrieval



Structure-based Clone Detection

```
static void foo() throws RESyntaxException {
  String a[] = new String[] {“123.400”,”abc”,”orange 100”};
  org.apache.regexp.RE pat = new org.apache.regexp.RE(“[0-9,]+”);
  int sum = 0;
  for(int i = 0; i < a.length; ++i)
    if(pat.match(a[i]))
      sum += Sample.parseNumber(pat.getParen(0));
  System.out.println(“sum =“ + sum);
}
static void goo(String[] a) throws RESyntaxException {
  RE exp = new RE(“[0-9,]+”);
  int sum = 0;
  for(int i = 0; i < a.length; ++i)
    if(exp.match(a[i]))
      sum += parseNumber(exp.getParen(0));
  System.out.println(“sum =“ + sum);
}
```

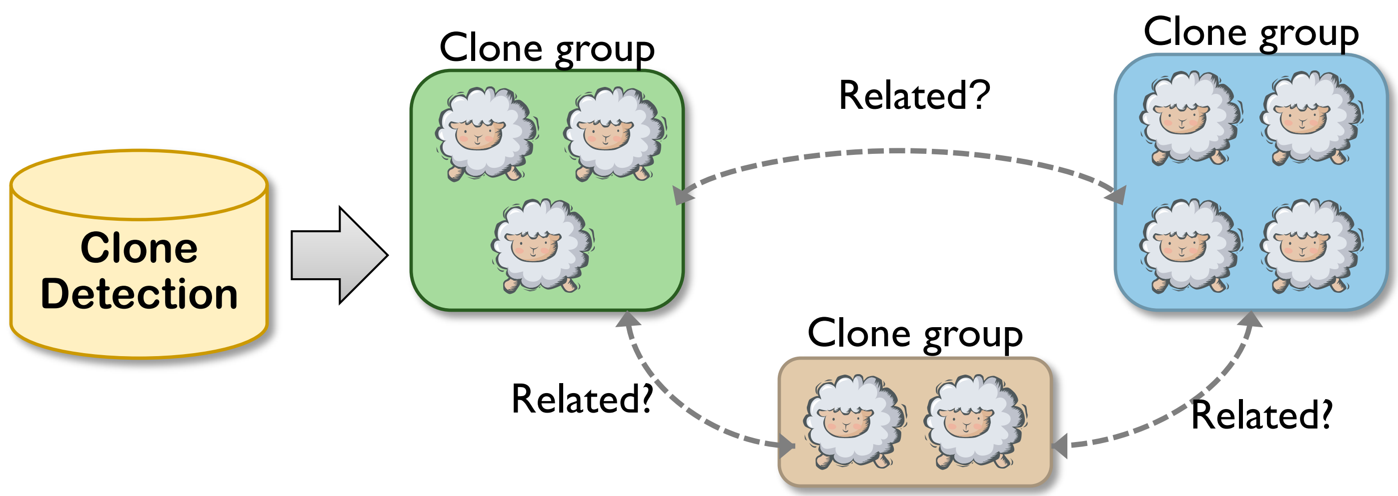
```
static void foo() throws RESyntaxException {
  String a[] = new String[] {“123.400”,”abc”,”orange 100”};
  org.apache.regexp.RE pat = new org.apache.regexp.RE(“[0-9,]+”);
  int sum = 0;
  for(int i = 0; i < a.length; ++i)
    if(pat.match(a[i]))
      sum += Sample.parseNumber(pat.getParen(0));
  System.out.println(“sum =“ + sum);
}
static void goo(String[] a) throws RESyntaxException {
  RE exp = new RE(“[0-9,]+”);
  int sum = 0;
  for(int i = 0; i < a.length; ++i)
    if(exp.match(a[i]))
      sum += parseNumber(exp.getParen(0));
  System.out.println(“sum =“ + sum);
}
```



Code Clone Analysis Tool: ICCA

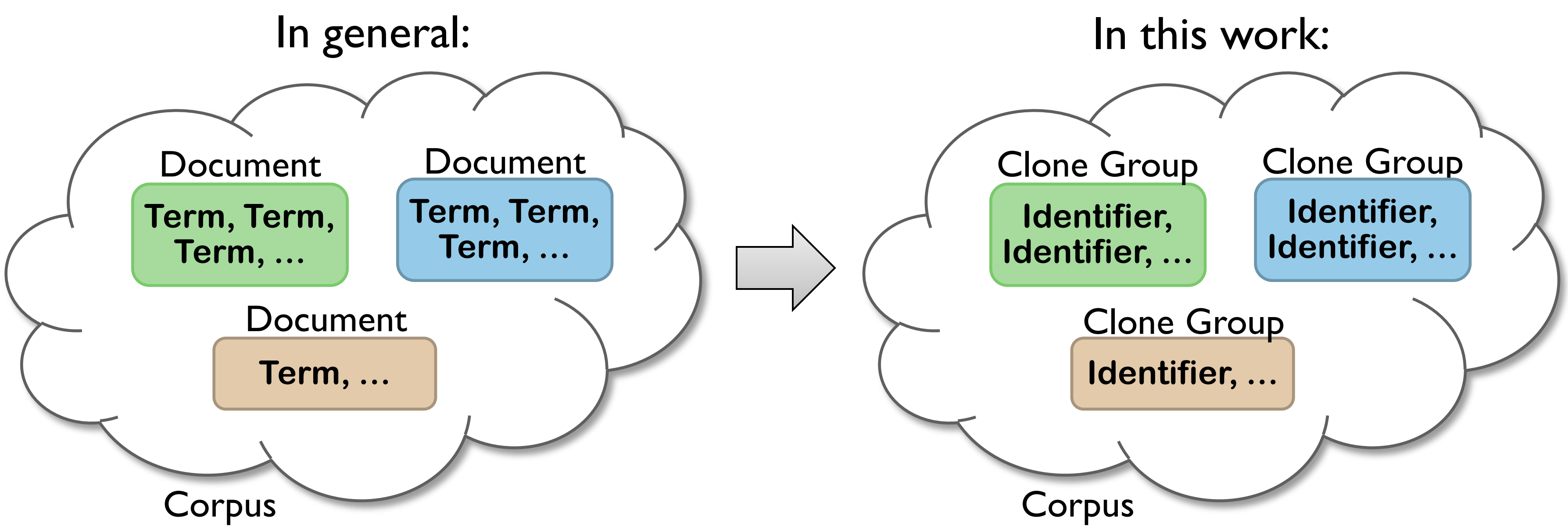
Clone analysis using Information Retrieval

- Investigate additional relationships among clone groups based on non-structured properties
- Latent Semantic Indexing (LSI) used to cluster clone groups based on the identifier names in the clones



Latent Semantic Indexing

- Latent Semantic Indexing (LSI) can be used to provide relationships among terms and documents in a corpus
- *Document to Document* relationships are determined based on terms in documents



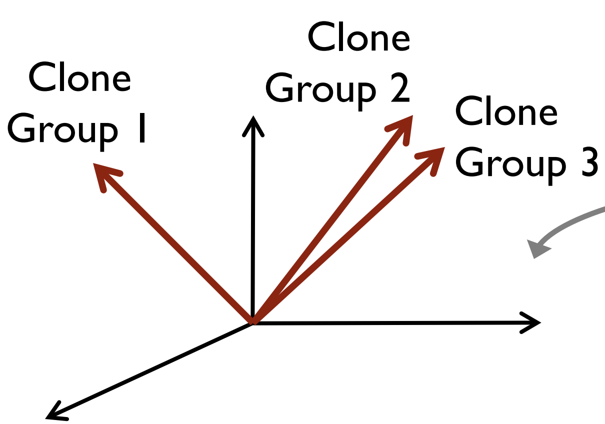
Approach: Clone Group Clustering

Clone Group 1

```
static void foo() throws RESyntaxException {
  String a[] = new String[] { "123,400", "abc“,
    "orange 100"};
  org.apache.regexp.RE pat = new
    org.apache.regexp.RE("[0-9,]+");
  int sum = 0;
  for (int i = 0; i < a.length; ++i)
    if (pat.match(a[i]))
      sum += Sample.parseNumber(pat.getParen(0));
  System.out.println("sum = " + sum );
}
```

Term-Document Matrix

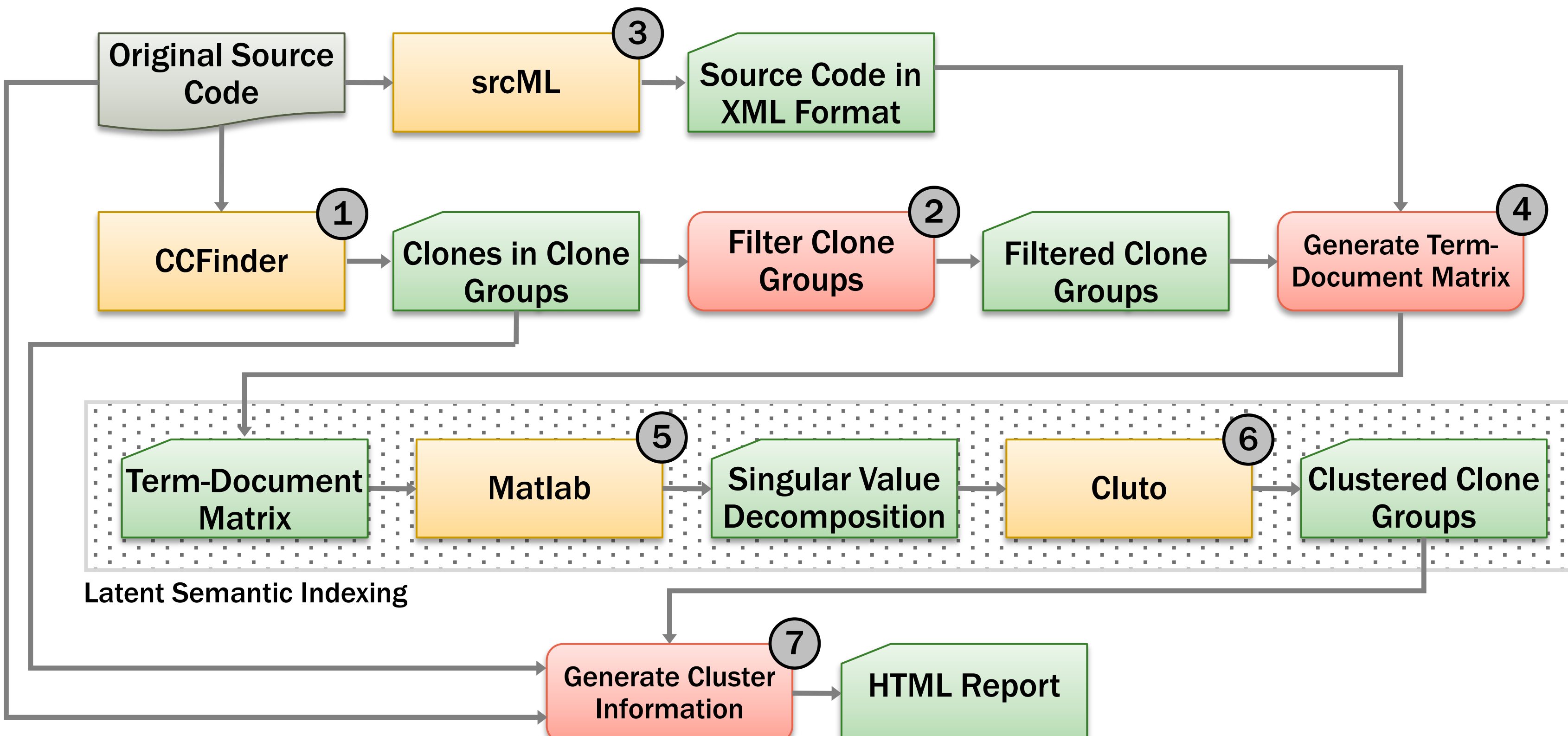
	CG1	CG2	…
a	3	X	…
apache	2	X	…
foo	1	X	…
getParen	1	X	…
i	4	X	…
…	…	…	



Singular Value Decomposition (SVD)

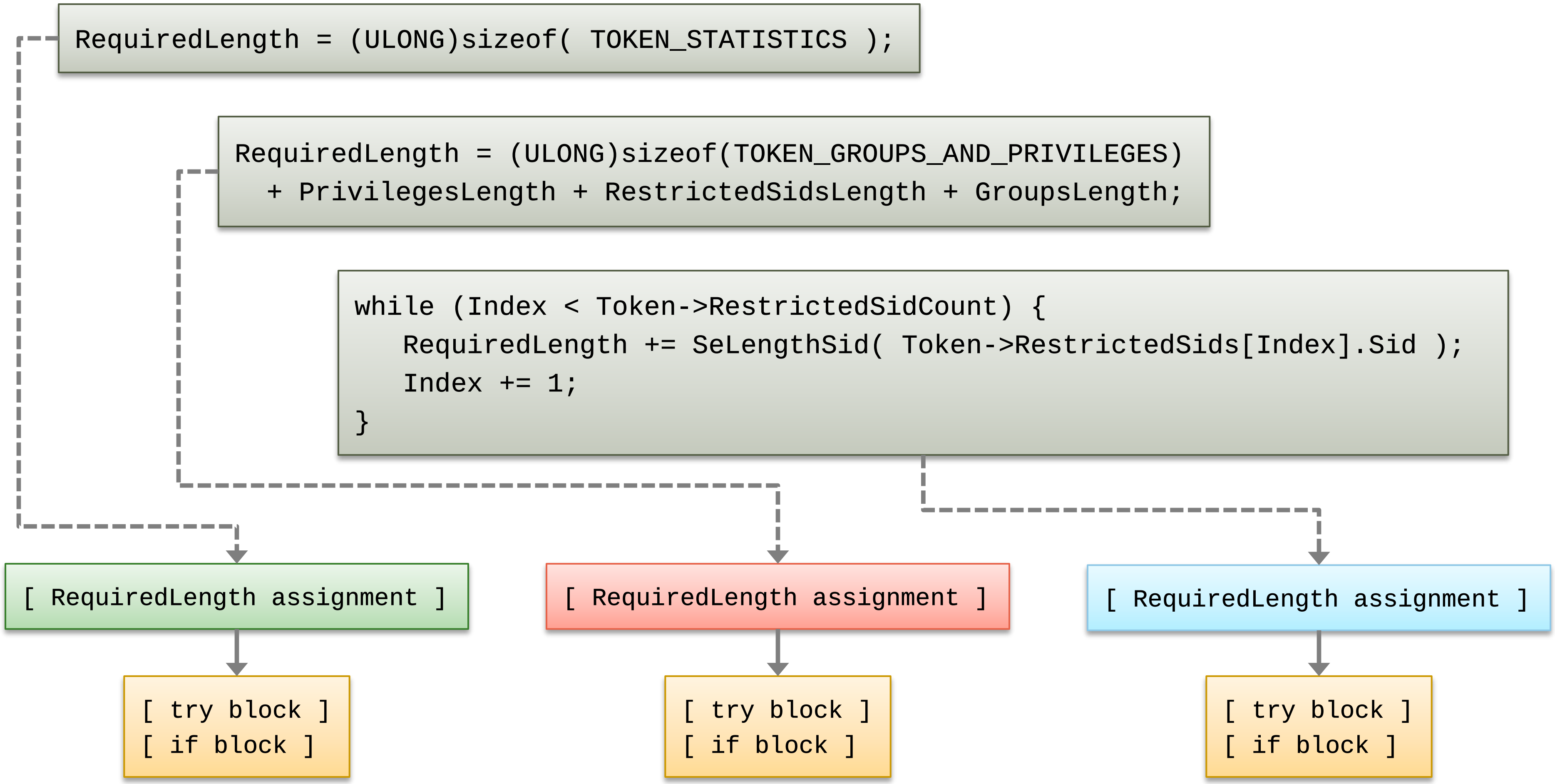
Information Retrieval-based Process

- Case Study: Microsoft Research Kernel
  - Available for academic teaching and research
  - Basic operating system implementations for the NT Kernel



Cluster Observations: Example

- Clones were grouped based on the method of assigning `RequiredLength`



Cluster Observations: Example

- Clones were grouped based on statement sequences

Clone Group

[Array Initialization]
[Code Sequence 1]
[Code Sequence 2]
Clone

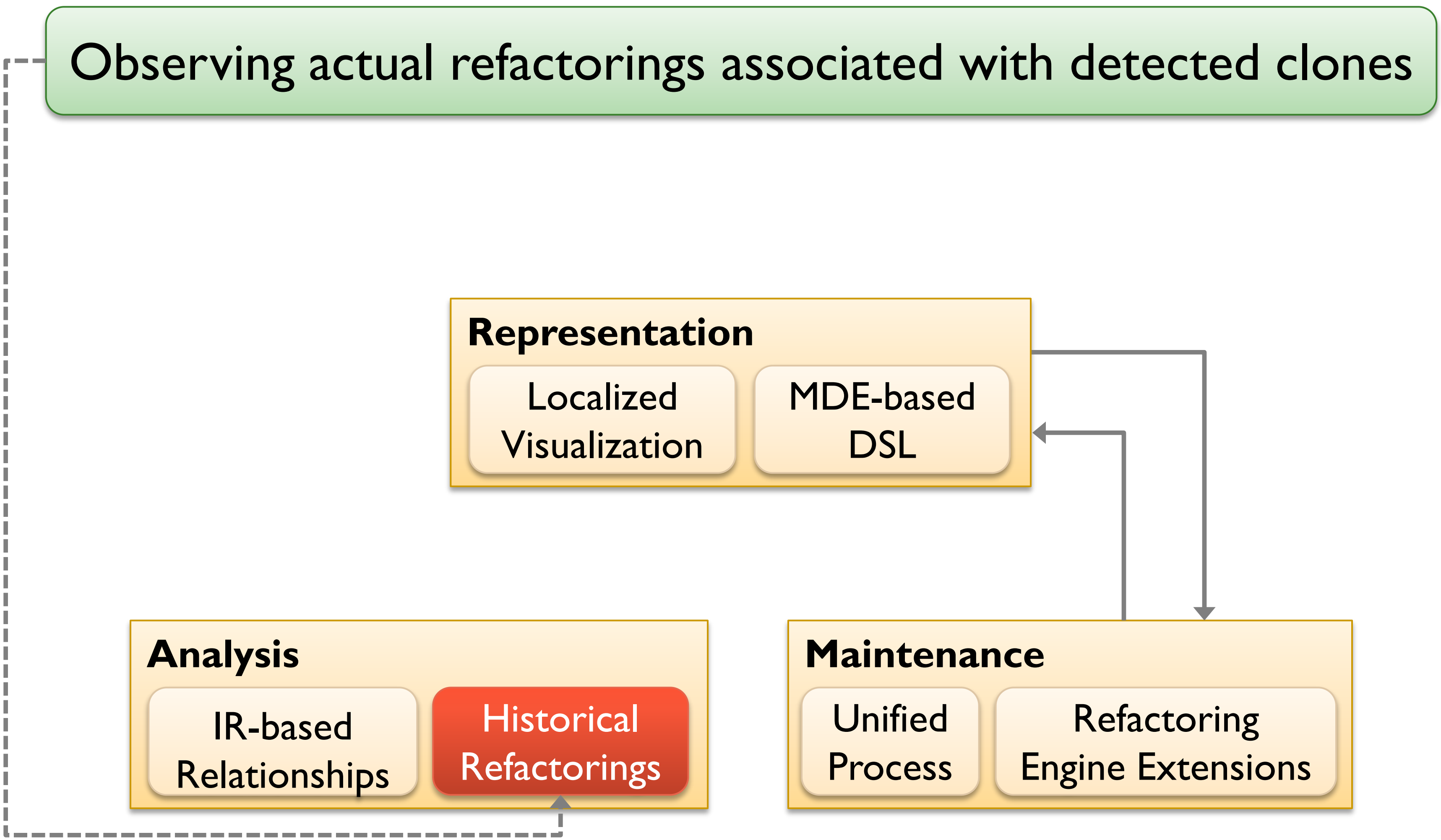
Clone Group

[Code Sequence 1]
[Array Initialization]
[Code Sequence 2]
Clone

- Clones grouped based on existence of a statement and arguments

```
irp = IoAllocateIrp( deviceObject->StackSize, (...) );
if (!irp) {
  if (...) {
    ExFreePool( event );
  }
  IopAllocateIrpCleanup( fileObject, (...) );
  return STATUS_INSUFFICIENT_RESOURCES;
}
irp->Tail.Overlay.OriginalFileObject = fileObject;
irp->Tail.Overlay.Thread = CurrentThread;
```

Sub-Clone Refactoring



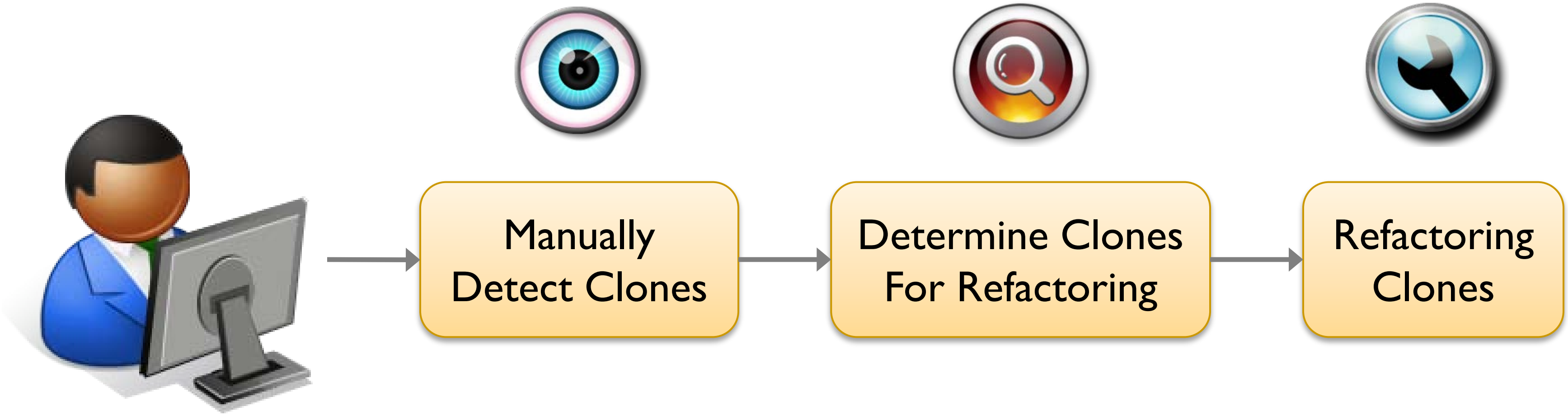
Refactoring Clones

```
public class A {

  public void method() {
    {cloned statements}
    {cloned statements}
    {cloned statements}
    ...
    {cloned statements}
    {cloned statements}
    {cloned statements}
  }

}
```

Extract-Method Refactoring

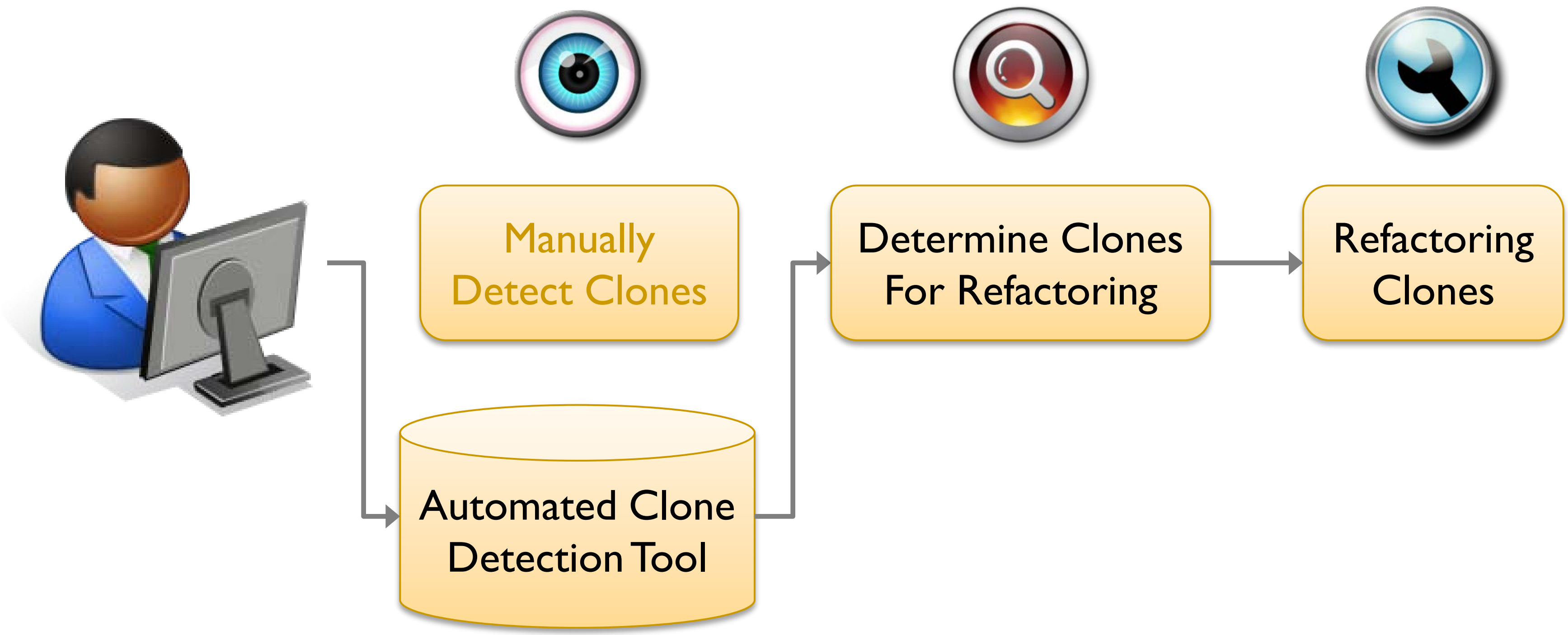
```
public class A {

  public void method() {
    newMethod();
    ...
    newMethod();
  }

  public void newMethod() {
    {cloned statements}
    {cloned statements}
    {cloned statements}
  }

}
```

Clone Refactoring Process



- Changes between two versions
  - First version contains original code
  - Second version contains refactored code

Clone Refactoring Process

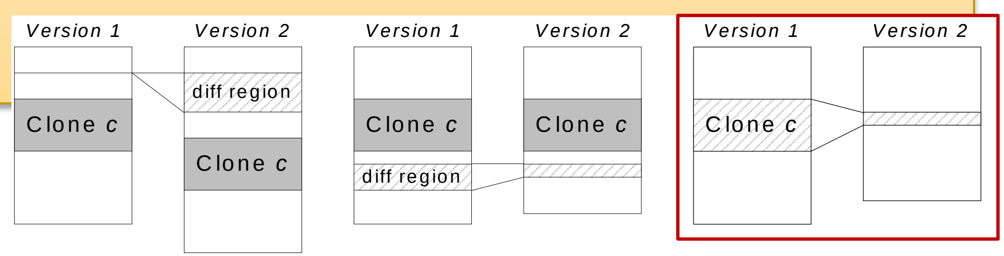


- What are the refactoring characteristics of clones detected by a clone detection tool, if such a tool was used in the clone maintenance process?

Approach: Observing Refactorings

- Observing actual clone-related refactorings in multiple release versions of JBoss


```
 @@ -2471,13 +2469,7 @@
             scan_position.current_slot = Page.INVALID_SLOT_NUMBER;

             // release the scan lock now that we have saved away the row.
-
-            if (scan_position.current_scan_pageno != 0)
-            {
-               this.getLockingPolicy().unlockScan(
-                  scan_position.current_scan_pageno);
-               scan_position.current_scan_pageno = 0;
-            }
+            unlockCurrentScan(scan_position);
          }
       }
    }
```



Refactoring in Clone Ranges

```
1 2   4 5   protected String getValue(String name, String value) {
1 2   4 5     if (value.startsWith("${") && value.endsWith("}")) {
1 2 3 4 5 -     try {
1 2 3 4 5 -       String propertyName = value.substring(2, value.length()-1);
1 2 3 4 5 -       ObjectName propertyServiceON = new ObjectName("...");
1 2 3 4 5 -       KernelAbstraction kernelAbstraction = KernelAbstractionFactory.getInstance();
1 2 3 4 5 -       String propertyValue = (String)kernelAbstraction.invoke(...);
1 2 3   5 -       log.debug("Replaced ejb-jar.xml element " + name + " with value " + propertyValue);
1 2 3   5 -       return propertyValue;
1 2 3   5 -     } catch (Exception e) {
1 2 3   5 -       log.warn("Unable to look up property service for ejb-jar.xml element " + ...);
1 2 3   5 -     }
          +     String replacement = StringPropertyReplacer.replaceProperties(value);
          +     if (replacement != null)
          +       value = replacement;
1 2     5     }
1 2     5     return value;
1 2     5   }
```

```
 if (edge instanceof MTransition) {
    MTransition tr = (MTransition) edge;
-   FigTrans trFig = new FigTrans(tr);
-   // set source and dest
-   // set any arrowheads, labels, or colors
-   MStateVertex sourceSV = tr.getSource();
-   MStateVertex destSV = tr.getTarget();
-   FigNode sourceFN = (FigNode) lay...
-   FigNode destFN = (FigNode) lay...
-   trFig.setSourcePortFig(sourceFN);
-   trFig.setSourceFigNode(sourceFN);
-   trFig.setDestPortFig(destFN);
-   trFig.setDestFigNode(destFN);
+   FigTrans trFig = new FigTrans(tr, lay);
    return trFig;
  }
```

- Refactoring performed on only part of the reported clone range
  - Sub-clone refactoring

Evaluation: Tool Coverage

- 21 *Extract Method*-type Refactoring in JBoss (v2.2.0–4.2.3)
  - Clones initially detected by Simian
  - Further evaluated with four other tools

Tool	Exact Coverage	Larger Coverage
1. CCFinder	4 (19%)	8 (38%)
2. CloneDR	6 (28%)	9 (42%)
3. Deckard	8 (38%)	3 (14%)
4. Simian	2 (9%)	0 (0%)
5. Simscan	6 (28%)	12 (57%)

- These tools mainly look for the maximal sized clone

Evaluation: Focus on Deckard

- Deckard selected due to tree-based tool performance
  - JBoss re-evaluated
  - Additional artifacts: ArgoUML (v0.10.1–0.26) and Apache Derby (v10.1.1.0–10.5.3.0)

Property		JBoss	ArgoUML	Derby
Refactoring Coverage	Exact coverage	19	17	12
	Sub-clone coverage	14	9	15
Coverage Levels	Same level	4	4	6
	1 level above	9	2	8
	> 1 level above	1	3	1
Clone Differences	Refactorable	7	4	8
	Not Refactorable	7	5	7

Evaluation: Focus on Deckard

- Reported clone range mainly the same level or one syntactic level above the actual refactored code
  - Possibly to keep some logic in the original location

Property		JBoss	ArgoUML	Derby
Refactoring Coverage	Exact coverage	19	17	12
	Sub-clone coverage	14	9	15
Coverage Levels	Same level	4	4	6
	1 level above	9	2	8
	> 1 level above	1	3	1
Clone Differences	Refactorable	7	4	8
	Not Refactorable	7	5	7

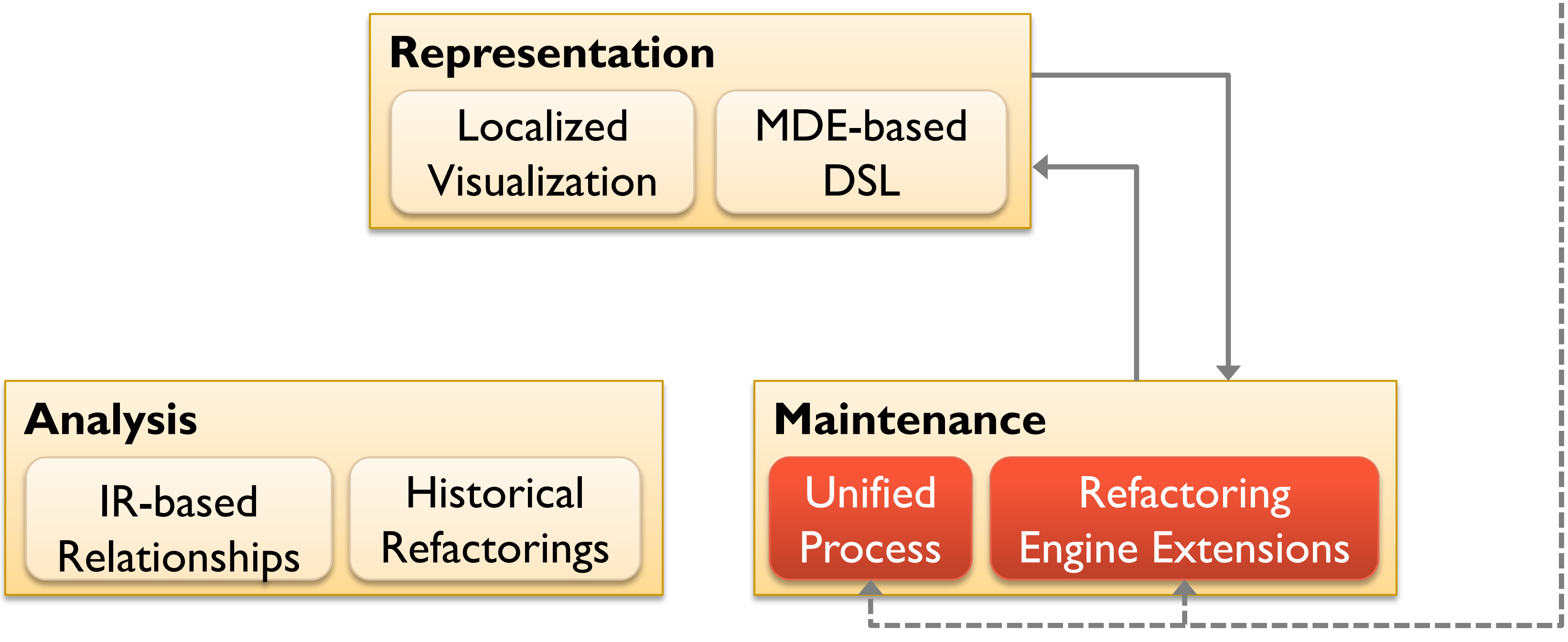
- Programmers only refactored a sub-clone even when the entire clone was refactorable

Summary

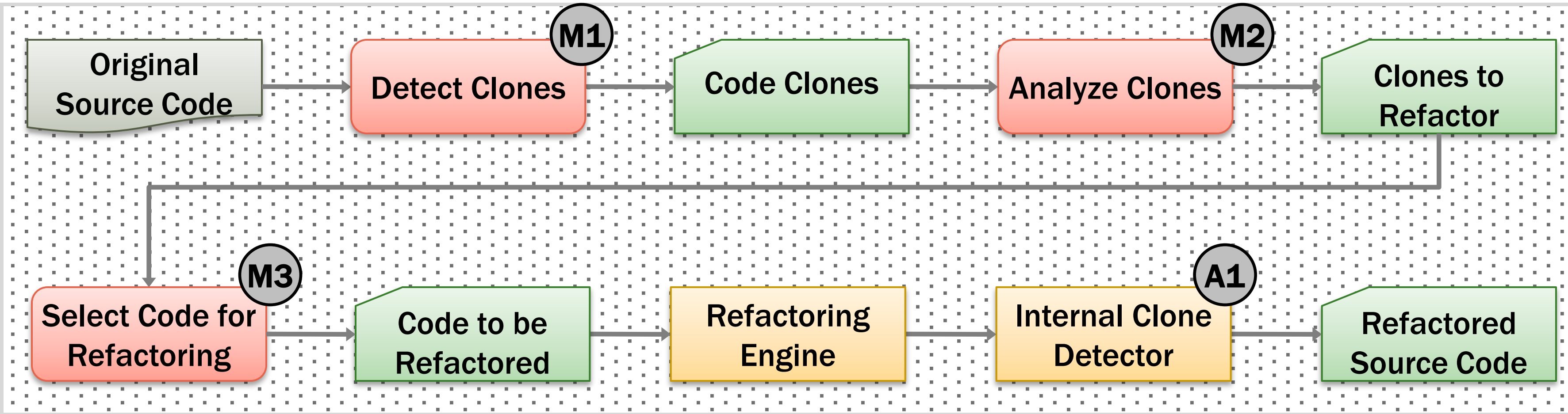
- Analysis of large amounts of clone data
  - “Super-clones”
    - Clone group clustering based on non-structural information
  - “Sub-clones”
    - Refactoring performed on partial range of clones
- Maintenance
  - Clone groups that are related could be considered for similar updating
  - Support for sub-clone refactoring should be part of maintenance process

CeDAR: Clone Detection, Analysis, and Refactoring

Unifying the process of clone detection, analysis, and refactoring



Current Refactoring Process



Eclipse IDE

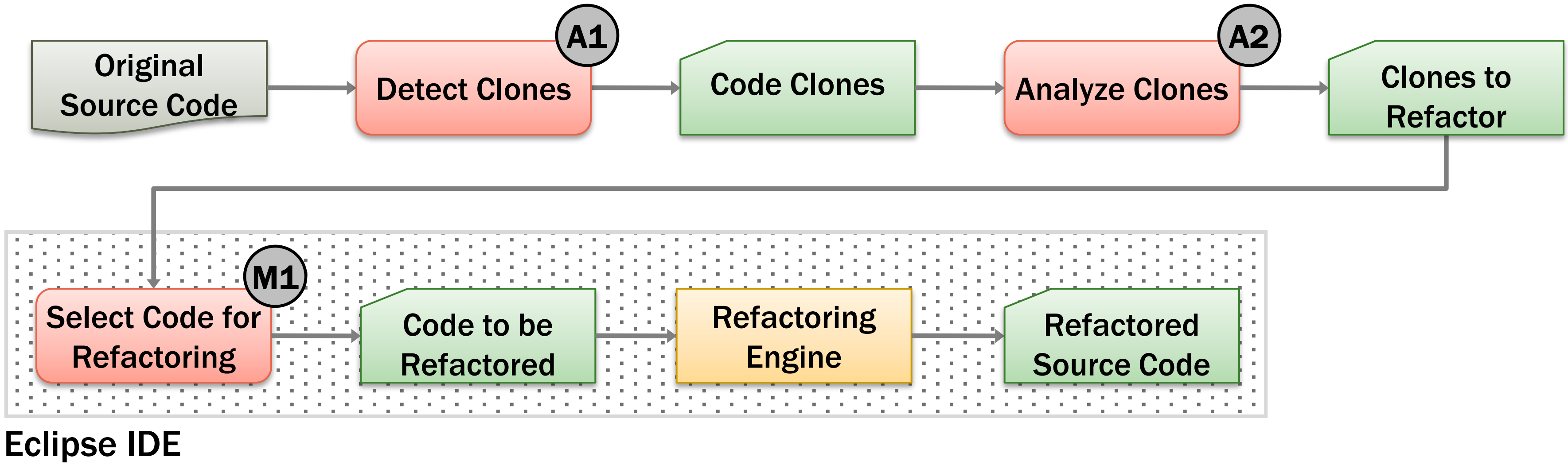
M1 Clones must be detected manually

M2 Clones must be analyzed manually

M3 Each section of code must be manually selected and forwarded to Refactoring Engine

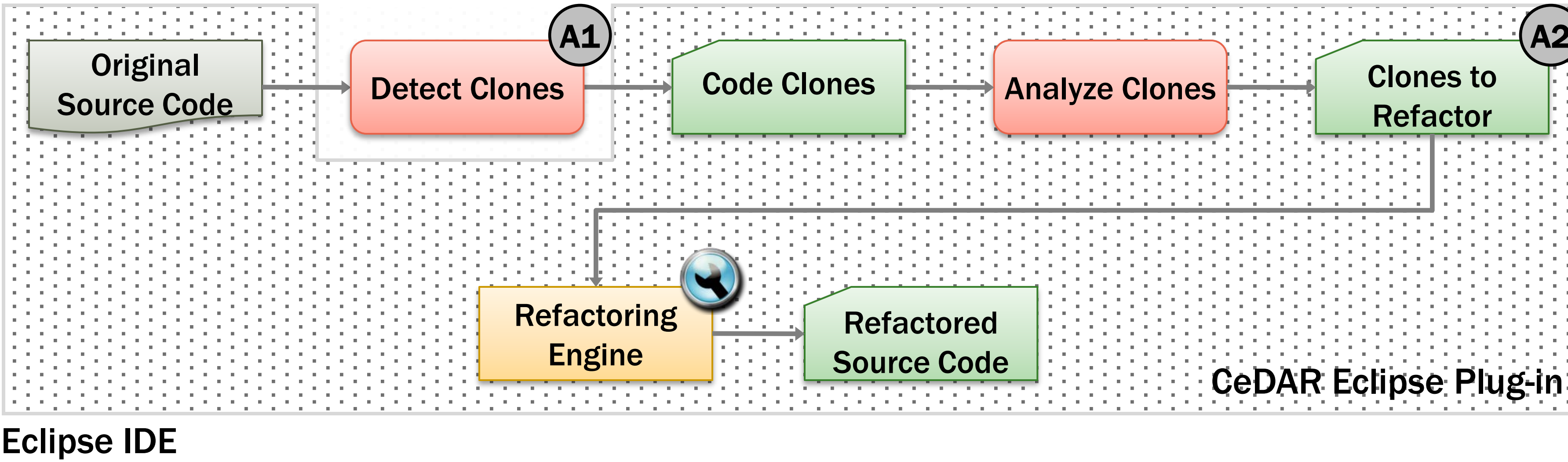
A1
- *Extract Method* refactoring limited to local variable name differences
- Limited to clones in one file
- Clone information only available after selection for refactoring

Current Approaches



- A1 Clones detected automatically
- A2 Clones analyzed with automated assistance
- M1 Each section of code must be manually selected and forwarded to Refactoring Engine

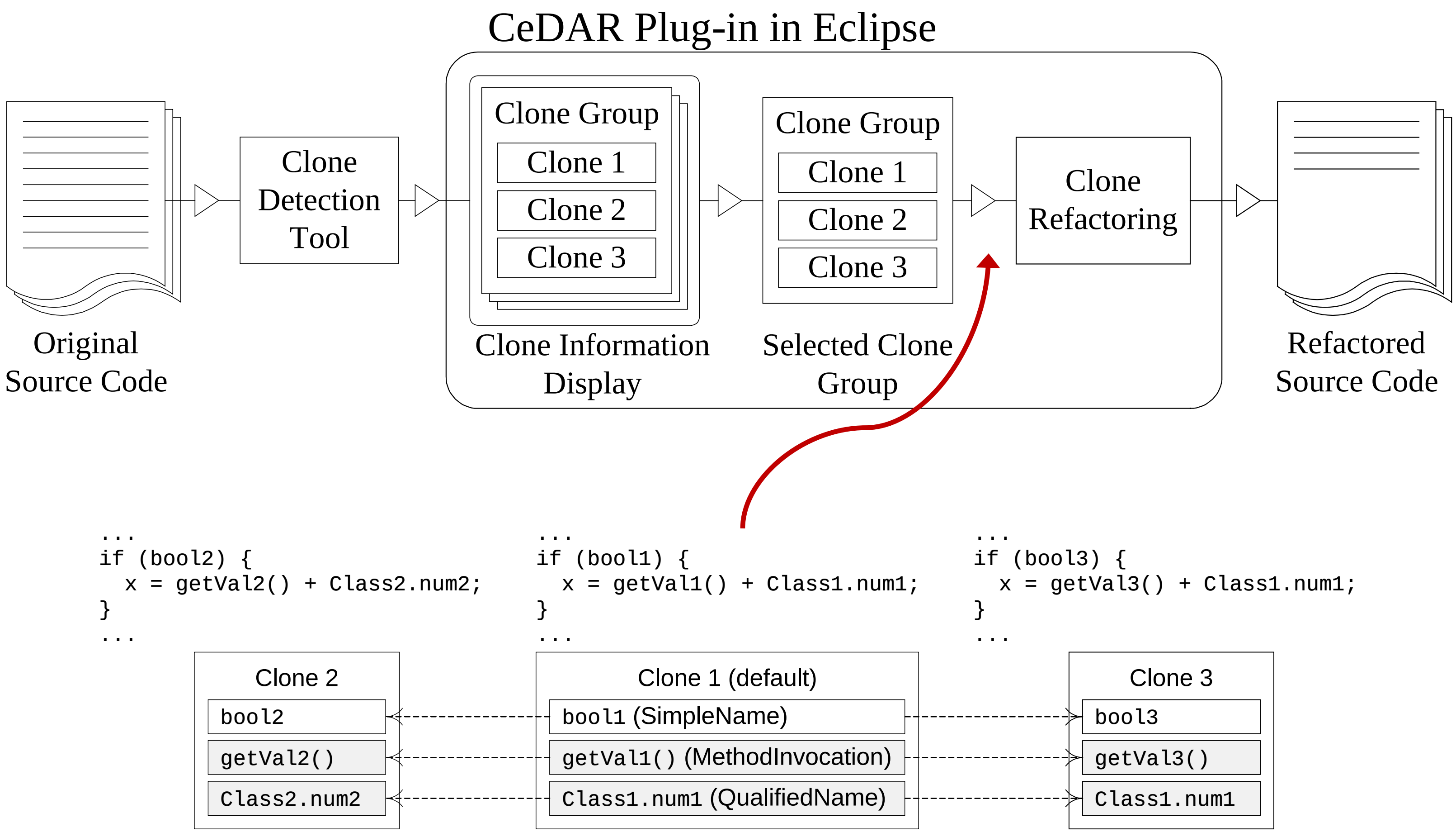
Our Approach: Unified Process



- A1 Automated clone detection remains an external process
- A2 All clone information forwarded to refactoring engine
- Additional parameterized differences such as fields, method calls, and string literals

Parameterized Element Mapping

- Include parameterized values of internal and external fields, method calls, and strings



Type II Clones

- "syntactically identical copy; only variable, type, or function identifiers were changed." [Bellon *et al.*, 2007]
- Fields
  - Include fields that are different between at least two clones
  - Include clones with [field] ←→ [local variable] mappings

```
public class A {
  int field1;
  int field2;

  public void method() {
    {cloned statements}
    {reference to field1}
    {cloned statements}
    ...
    {cloned statements}
    {reference to field2}
    {cloned statements}
  }
}
```

```
public class A {
  int field1;
  int field2;

  public void method() {
    newMethod(field1);
    ...
    newMethod(field2);
  }

  public void newMethod(int field) {
    {cloned statements}
    {reference to field}
    {cloned statements}
  }
}
```

Type II Clones

- Method calls
  - Include methods with no arguments
  - Pass method-related expressions if all clones use same expression
- Strings
  - Include strings with 1-to-1 correspondence

```
public void method() {
  ...
  {reference to p}
  {reference to p.call()}
  ...
  {reference to q}
  {reference to q.call()}
  ...
}
```

```
public void method() {
  ...
  newMethod(p, p.call())
  ...
  newMethod(q, q.call())
  ...
}

public void newMethod
   (Object a, Object b) {
  {reference to a}
  {reference to b}
}
```

```
public void method() {
  ...
  newMethod(p)
  ...
  newMethod(q)
  ...
}

public void newMethod
   (Object a) {
  {reference to a}
  {reference to a.call()}
}
```

Evaluation: Additional Refactorings

- In half of the software artifacts evaluated, the number of refactorings doubled

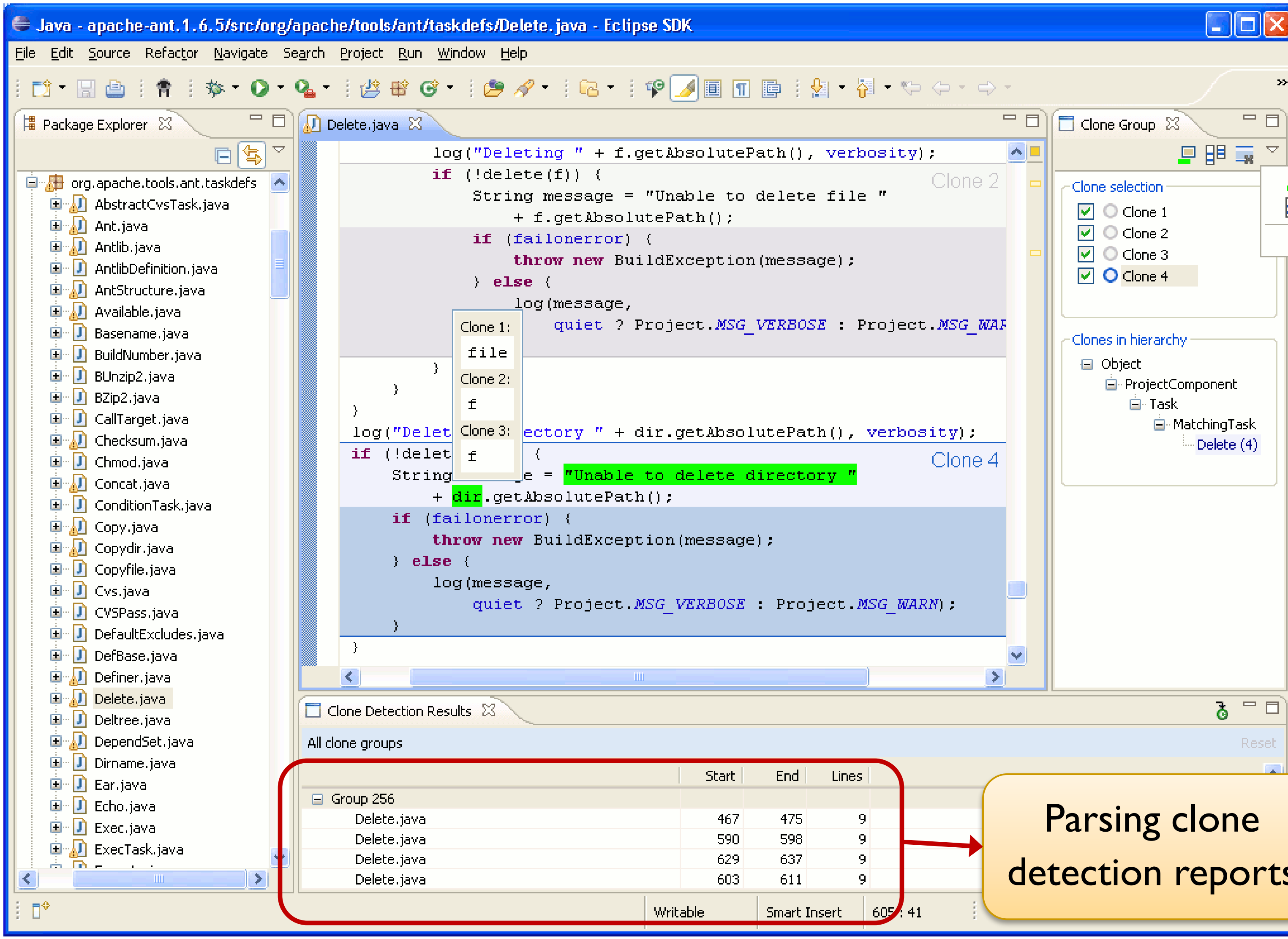
Project	KLoC	CG	Eclipse	CeDAR	Δ
Apache Ant 1.7.0	67	120	14 (12%)	28 (23%)	+14
Columba 1.4	75	88	13 (15%)	30 (34%)	+17
EMF 2.4.1	118	149	8 (5%)	14 (9%)	+6
Hibernate 3.3.2	209	177	15 (8%)	18 (10%)	+3
Jakarta JMeter 2.3.2	54	68	3 (4%)	11 (16%)	+8
JEdit 4.2	51	157	15 (10%)	20 (13%)	+5
JFreeChart 1.0.10	76	291	29 (10%)	62 (21%)	+33
JRuby 1.4.0	101	81	23 (28%)	23 (28%)	0
Squirrel SQL 3.0.3	141	75	8 (11%)	20 (27%)	+12

Parameterized Differences

- Each parameterized difference utilized during *Extract Method* refactoring activity, albeit in varying occurrences

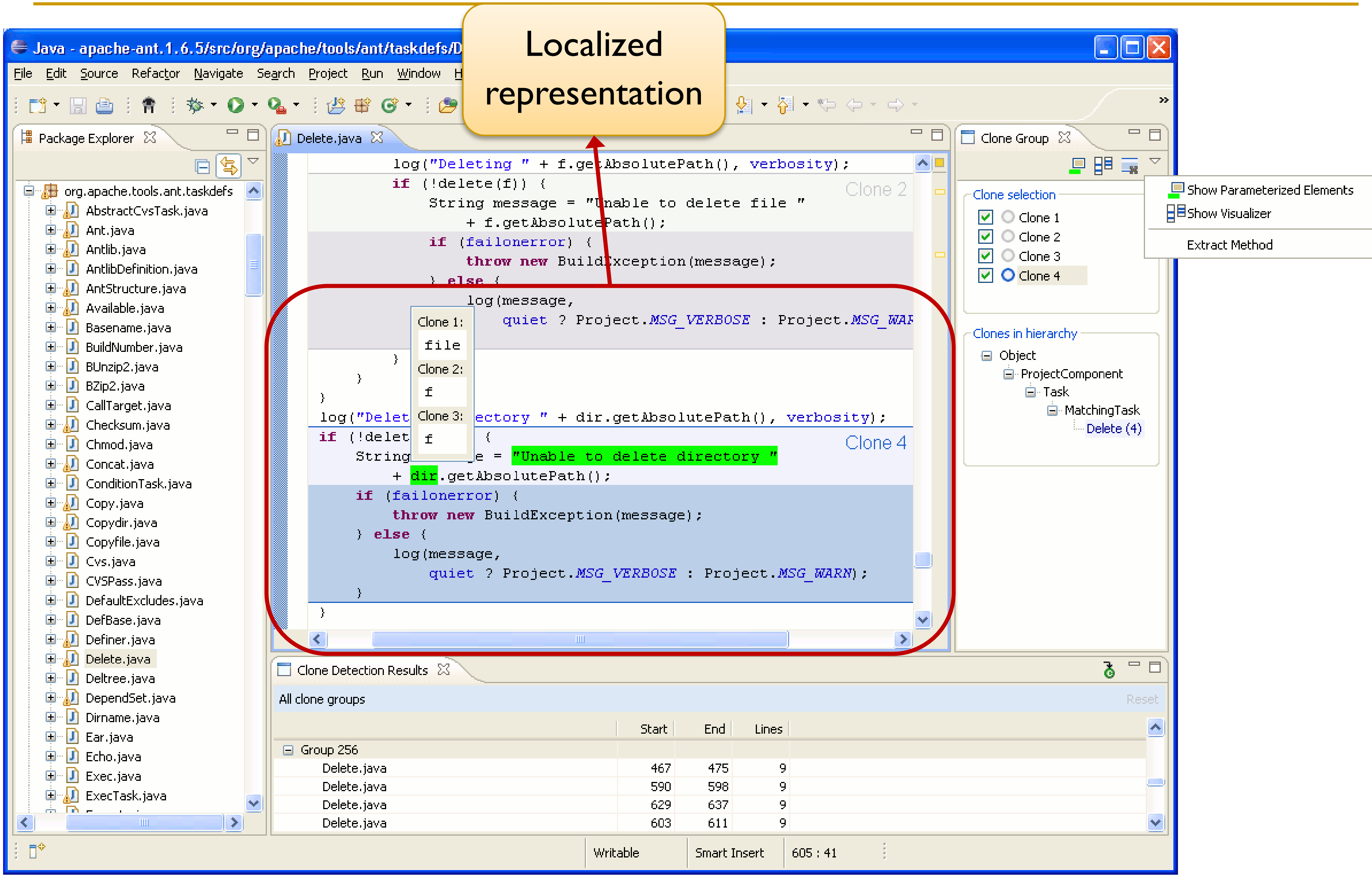
Project	Local Variable	Internal Field	External Field	Method Call	String
Apache Ant 1.7.0	10	8	2	8	6
Columba 1.4	14	7	7	7	5
EMF 2.4.1	6	2	0	2	4
Hibernate 3.3.2	3	0	0	2	2
Jakarta JMeter 2.3.2	8	1	1	2	7
JEdit 4.2	4	1	1	1	2
JFreeChart 1.0.10	34	19	11	13	5
Squirrel SQL 3.0.3	12	6	3	9	4

CeDAR in Eclipse

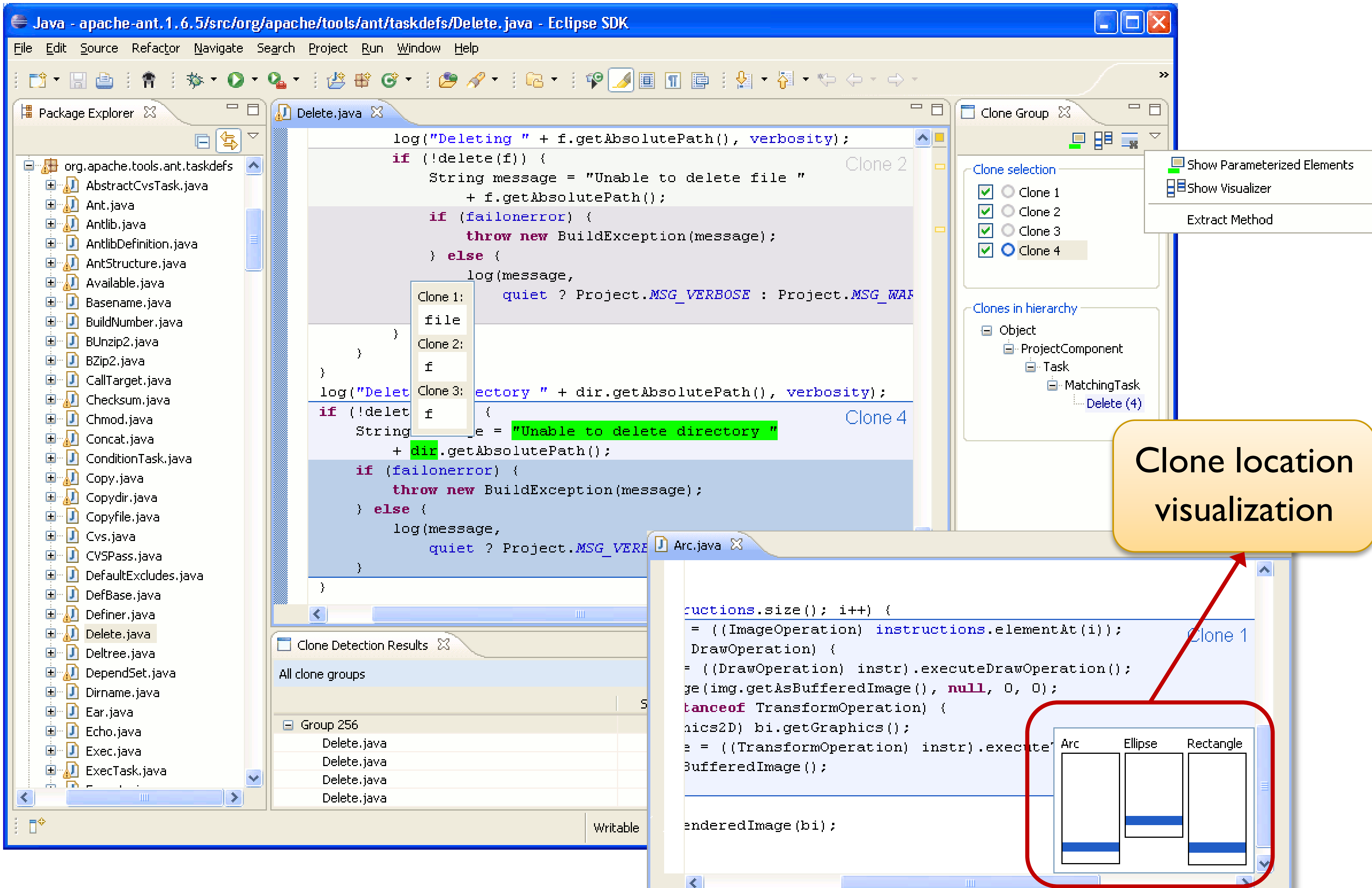


Parsing clone detection reports

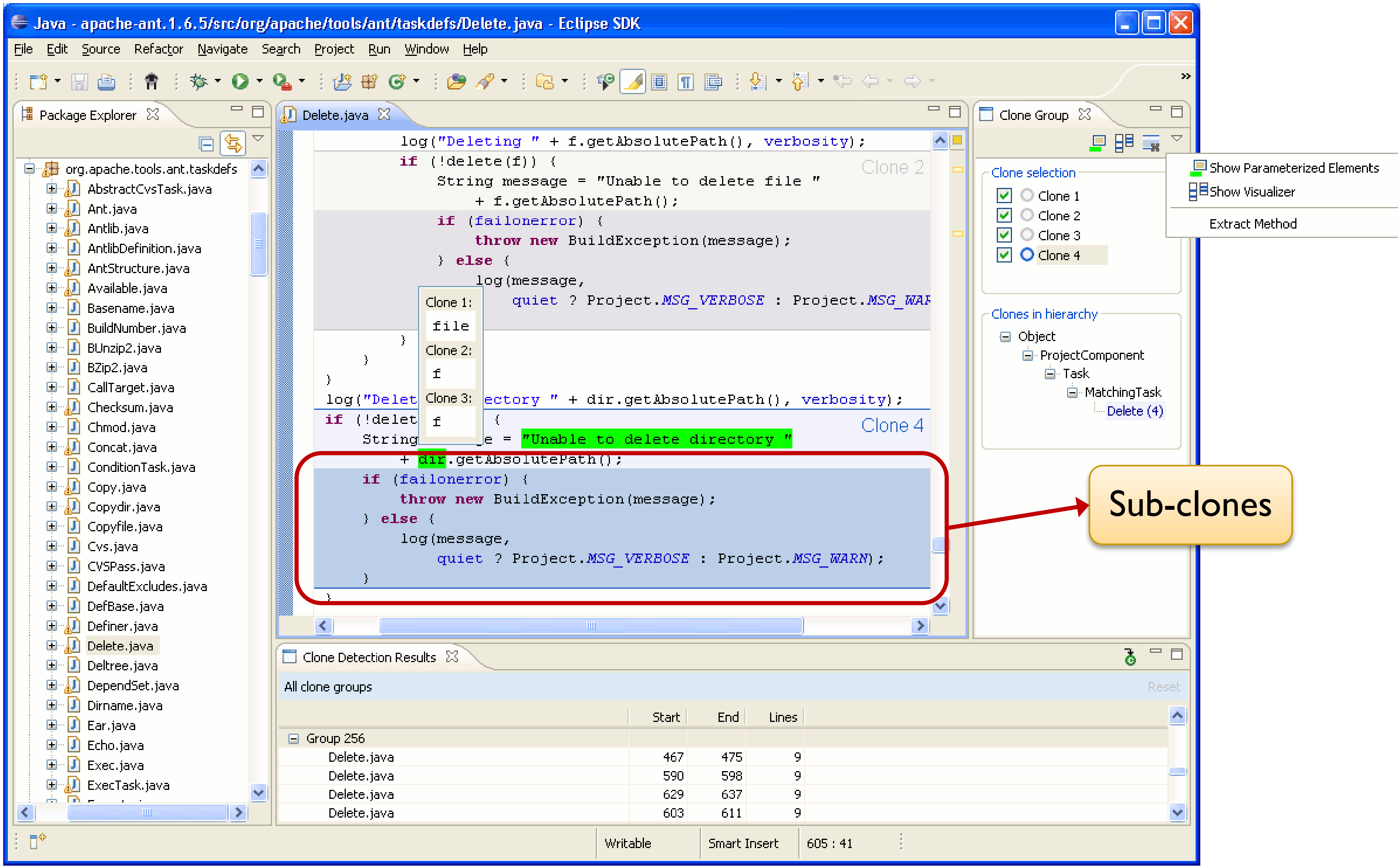
CeDAR in Eclipse



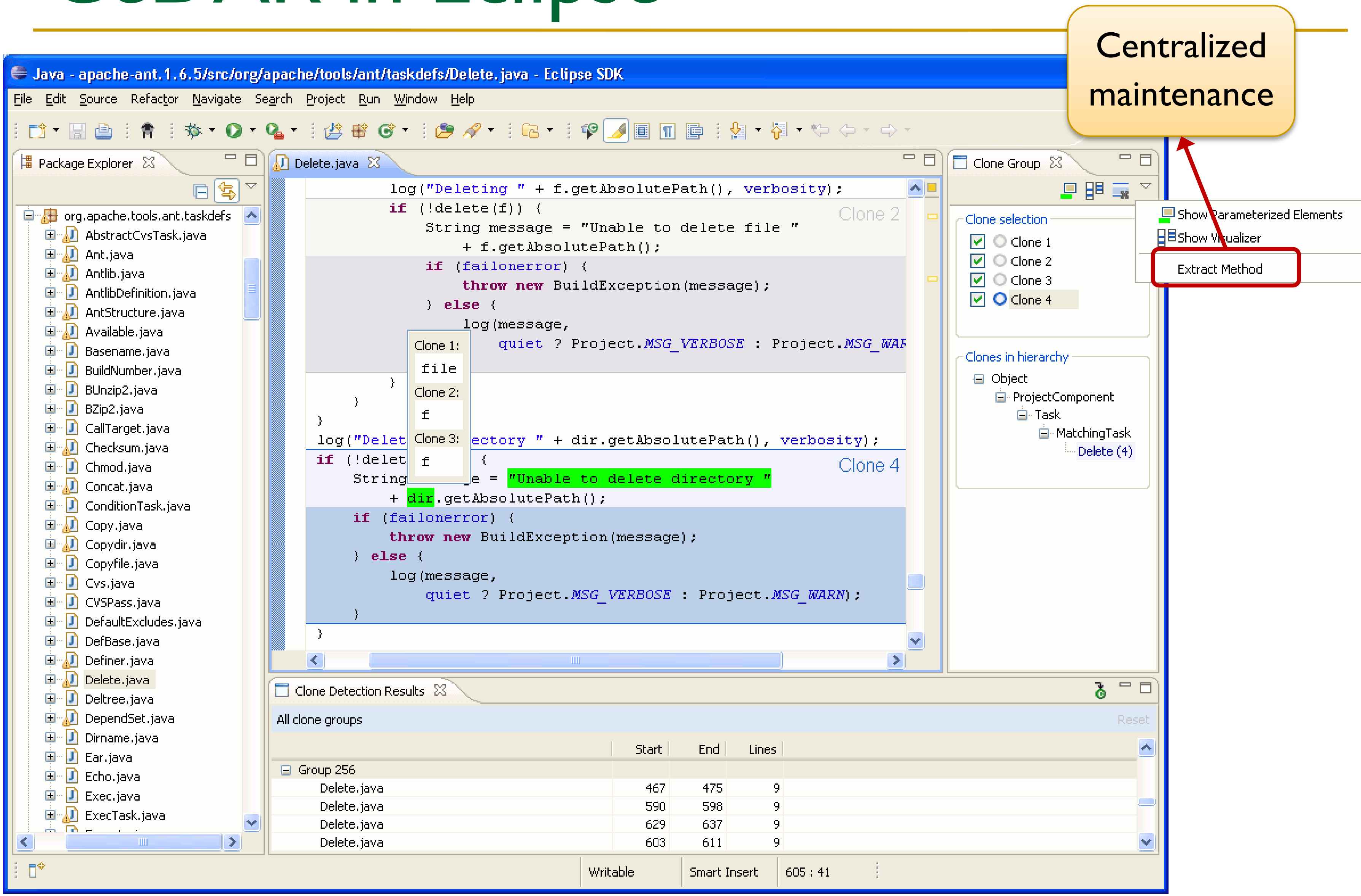
CeDAR in Eclipse



CeDAR in Eclipse



CeDAR in Eclipse



Summary

- Clone maintenance process (detection, analysis, and refactoring) unified within Eclipse through CeDAR
- Extensions incorporate more parameterized differences among clones to enable additional accepted refactorings
- Instances of clone refactoring doubled in many of the evaluated software artifacts

Contributions

- Representation
  - Visualization and representation of clones at the clone group level and a transformation-based clone analysis approach
- Analysis
  - The discovery of additional clone properties related to the semantic relationships of clone groups, and refactoring of partial clones
- Refactoring
  - A unified clone maintenance process that reduces the manual steps required for refactoring and increases support for refactoring of different clone types

Future Research Plan

- Continued Focus on Clone Maintenance
  - Increasing refactoring capabilities
  - Incorporating visualizations in the refactoring task
  - Clone models via model weaving
- Broader Application of Work
  - Additional clone property analysis (e.g., outlier clones)
  - Information retrieval and model analysis

Publications

Journals

R. Tairas, J. Gray, Extending an IDE's Refactoring Engine for Additional Clone Refactoring Opportunities, *Information and Software Technology*, *in preparation*.

R. Tairas, J. Gray, An Information Retrieval Process to Aid in the Analysis of Clones, *Empirical Software Engineering*, 14(1): 33-56, 02/09.

J. Zhang, Y. Lin, J. Gray, R. Tairas, Aspect Mining from a Modeling Perspective, *Int. J. of Computer Applications in Technology*, 31(1/2): 74-82, '08.

Conferences and Workshops

R. Tairas, F. Jacob, J. Gray, Visualizing Code Clones in a Localized Manner, *ACM Symposium on Software Visualization*, Salt Lake City, UT, 10/10, *under review*.

R. Tairas, J. Gray, Sub-clones: Considering the Part Rather than the Whole, *Int. Conf. on Software Engineering Research and Practice* (SERP), Las Vegas, NV, 07/10, *to appear*.

F. Jacob, R. Tairas, Code Template Inference Using Language Models, *ACM Southeast Conf.*, Oxford, MS, April 2010.

R. Tairas, J. Gray, Sub-clone Refactoring in Open Source Software Artifacts, *Symp. on Applied Computing* (SAC), Sierre, Switzerland, 03/10: 2364-2365.

R. Tairas, Centralizing Clone Group Representation and Maintenance, *Student Research Competition*, *Int. Conf. on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA), Orlando, FL, 10/09: 781-782.

R. Tairas, M. Mernik, J. Gray, Using Ontologies in the Domain Analysis of Domain-Specific Languages, *Workshop on Transformation and Weaving Ontologies in Model-Driven Engineering* (TWOMDE), *Int. Conf. on Model Driven Engineering, Languages, and Systems* (MoDELS), LNCS 5421, Toulouse, France, 09/08: 332-342. (Best Paper Award)

Y. Sun, Z. Demirezen, F. Jouault, R. Tairas, J. Gray, Tool Interoperability through Model Transformations, *Int. Conf. on Software Language Engineering* (SLE), LNCS 5452, Toulouse, France, 09/08: 178-187.

R. Tairas, A. Liu, F. Jouault, J. Gray, CoCloRep: A DSL for Code Clones, *Int. Workshop on Software Language Engineering* (ATEM), *Int. Conf. on Model Driven Engineering, Languages, and Systems* (MoDELS), Nashville, TN, 10/07: 91-99.

R. Tairas, J. Gray, I. Baxter, Visualization of Clone Detection Results, *Eclipse Technology Exchange Workshop* (ETX), *Int. Conf. on Object-Oriented Programming, Systems, Languages and Applications* (OOPSLA), Portland, OR, 10/06: 50-54.

R. Tairas, J. Gray, Phoenix-Based Clone Detection using Suffix Trees, *ACM Southeast Conf.*, Melbourne, FL, 03/06: 679-684.

Doctoral Symposium

R. Tairas, Clone Maintenance through Analysis and Refactoring, *Int. Symp. on the Foundations of Software Engineering* (FSE), Atlanta, GA, 11/08: 29-32.

R. Tairas, Clone Detection and Refactoring, *Int. Conf. on Object-Oriented Programming, Systems, Languages and Applications* (OOPSLA), Portland, Oregon, 10/06: 50-54.

Tool Demonstrations

R. Tairas, J. Gray, Get to Know Your Clones with CeDAR, *Int. Conf. on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA), Orlando, FL, 10/09: 817-818.

R. Tairas, J. Gray, I. Baxter, Visualizing Clone Detection Results, *Int. Conf. on Automated Software Engineering* (ASE), Atlanta, GA, 11/07: 549-550.

Code Clones Literature

- http://www.cis.uab.edu/tairasr/clones/literature/
  - Containing 185 research citations (as of June 2010)
  - Includes web sites of tools, events, and research groups
  - Has been cited by several research publications

“I regard your clone detection literature page as the most up-to-date and condensed source of new clone detection papers.”

“…I often visit and make use of (it).”

“This site was very useful for me when I was studying the clone detection problem. I think, it is the most useful site concerning clone detection on the Internet.”

“Your clone bibliography page … has been a very useful resource for our work.”

Thank you

- Personal:
  - http://www.cis.uab.edu/tairasr
- Code Clones Literature:
  - http://www.cis.uab.edu/tairasr/clones/literature
- SoftCom Laboratory:
  - http://www.cis.uab.edu/softcom