University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Code Duplication

Harald Gall

seal.ifi.uzh.ch/evolution

Code is Copied

Small Example from the Mozilla Distribution (Milestone 9)
Extract from /dom/src/base/nsLocation.cpp

```
[432]  NS_IMETHODIMP
[433]  LocationImpl::GetPathname(nsString
[434]  {
[435]    nsAutoString href;
[436]    nsIURI *url;
[437]    nsresult result = NS_OK;
[438]
[439]    result = GetHref(href);
[440]    if (NS_OK == result) {
[441]  #ifndef NECKO
[442]      result = NS_NewURL(&url, href);
[443]  #else
[444]      result = NS_NewURI(&url, href);
[445]  #endif // NECKO
[446]      if (NS_OK == result) {
[447]  #ifdef NECKO
[448]        char* file;
[449]        result = url->GetPath(&file);
[450]  #else
[451]        const char* file;
[452]        result = url->GetFile(&file);
[453]  #endif
[454]        if (result == NS_OK) {
[455]          aPathname.SetString(file);
[456]  #ifdef NECKO
[457]          nsCRT::free(file);
[458]  #endif
[459]        }
[460]        NS_IF_RELEASE(url);
[461]      }
[462]    }
[463]
[464]    return result;
[465]  }
[466]
```

```
[467]  NS_IMETHODIMP
[468]  LocationImpl::SetPathname(const nsString
[469]  {
[470]    nsAutoString href;
[471]    nsIURI *url;
[472]    nsresult result = NS_OK;
[473]
[474]    result = GetHref(href);
[475]    if (NS_OK == result) {
[476]  #ifndef NECKO
[477]      result = NS_NewURL(&url, href);
[478]  #else
[479]      result = NS_NewURI(&url, href);
[480]  #endif // NECKO
[481]      if (NS_OK == result) {
[482]        char *buf = aPathname.ToNewCString();
[483]  #ifdef NECKO
[484]        url->SetPath(buf);
[485]  #else
[486]        url->SetFile(buf);
[487]  #endif
[488]        SetURL(url);
[489]        delete[] buf;
[490]        NS_RELEASE(url);
[491]      }
[492]    }
[493]
[494]    return result;
[495]  }
[496]
```

```
[497]  NS_IMETHODIMP
[498]  LocationImpl::GetPort(nsString& aPort)
[499]  {
[500]    nsAutoString href;
[501]    nsIURI *url;
[502]    nsresult result = NS_OK;
[503]
[504]    result = GetHref(href);
[505]    if (NS_OK == result) {
[506]  #ifndef NECKO
[507]      result = NS_NewURL(&url, href);
[508]  #else
[509]      result = NS_NewURI(&url, href);
[510]  #endif // NECKO
[511]      if (NS_OK == result) {
[512]        aPort.SetLength(0);
[513]  #ifdef NECKO
[514]        PRInt32 port;
[515]        (void)url->GetPort(&port);
[516]  #else
[517]        PRUint32 port;
[518]        (void)url->GetHostPort(&port);
[519]  #endif
[520]        if (-1 != port) {
[521]          aPort.Append(port, 10);
[522]        }
[523]        NS_RELEASE(url);
[524]      }
[525]    }
[526]
[527]    return result;
[528]  }
[529]
```

What is a Code Clone?

a.k.a. Code Duplication, Software Cloning, Copy&Paste Programming

Code Clone = gratuitous copy of source code in a program

Code Clones increasing source code size and potentially increase defects

Code Duplication

Code Duplication

- What is it?
- Why is it harmful?

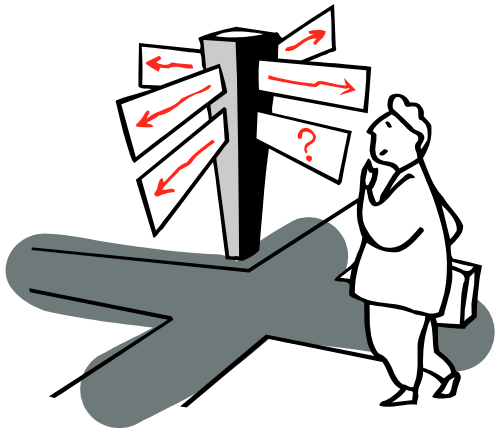
Detecting Code Duplication

Approaches

A Lightweight Approach

Visualization (dotplots)

Duploc



How Much Code is Duplicated?

Usual estimates: 8 to 12% in normal industrial code
15 to 25 % is already a lot!

Case Study	***LOC***	***Duplication without comments***	***with comments***
gcc	460'000	8.7%	5.6%
Database Server	245'000	36.4%	23.3%
Payroll	40'000	59.3%	25.4%
Message Board	6'500	29.4%	17.4%

What is copied code?

Duplicated Code = Source code segments that are found in different places of a system

- in different files
- in the same file but in different functions
- in the same function

The segments must contain some *logic or structure* that can be abstracted, i.e.,

```
...
computeIt(a,b,c,d);
...
```

```
...
computeIt(w,x,y,z);
...
```

is not considered duplicated code.

```
...
getIt(hash(tail(z)));
...
```

```
...
getIt(hash(tail(a)));
...
```

could be abstracted to a new function

Copied artifacts range from expressions, to functions, to data structures, and to entire subsystems.

Definitions

Clone Pair/Group: Set of equivalent Clones

Precision: Percent of reported clones that are genuine

Recall: Percent of genuine clones that are reported

Copied Code Problems

General negative effect:

- Code bloat

Negative effects on Software Maintenance

- Copied Defects
- Changes take double, triple, quadruple, ... Work
- Dead code
- Add to the cognitive load of future maintainers

Copying as additional source of defects

- Errors in the systematic renaming produce unintended aliasing

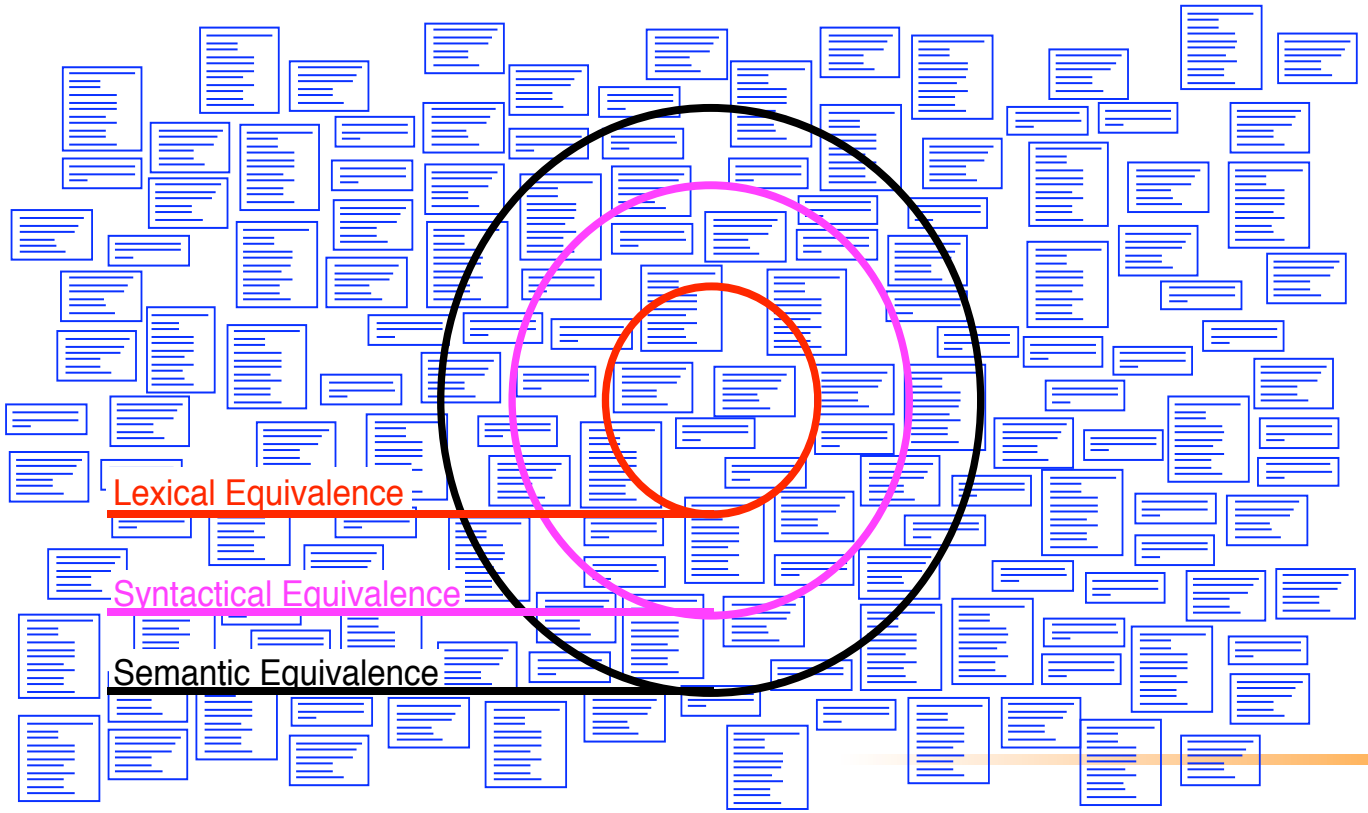
Metaphorically speaking:

- Software Aging, “hardening of the arteries”,
- “Software Entropy” increases even small design changes become very difficult to effect

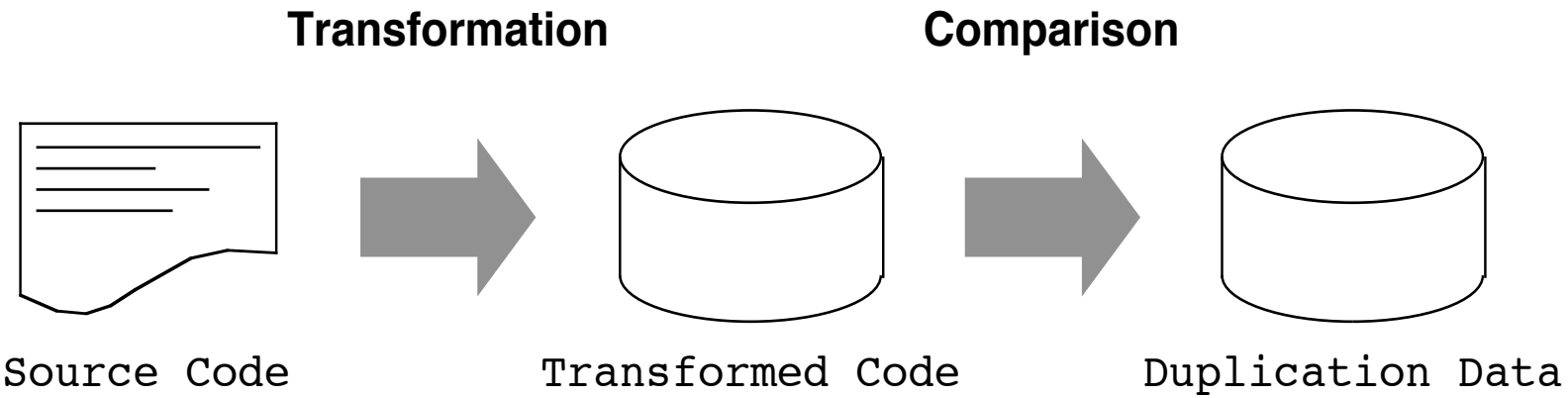
Code Duplication Detection

Nontrivial problem:

- No a priori knowledge about which code has been copied
- How to find all clone pairs among all possible pairs of segments?



General Schema of Detection Process



Author	*Level*	*Transformed Code*	*Comparison Technique*
[John94a]	Lexical	Substrings	String-Matching
[Duca99a]	Lexical	Normalized Strings	String-Matching
[Bake95a]	Syntactical	Parameterized Strings	String-Matching
[Mayr96a]	Syntactical	Metric Tuples	Discrete comparison
[Kont97a]	Syntactical	Metric Tuples	Euclidean distance
[Baxt98a]	Syntactical	AST	Tree-Matching

Detection

String Matching – Represents and evaluates code using string comparisons

Token Parsing – Code transformation into tokens for comparison

Graph Matching – Pattern matching on graph representations of code

Detection Strategies

Older, Studied Extensively — Newer, Bleeding Edge

Less Complex — More Complex

Syntactic Understanding — Semantic Understanding

String Matching Techniques

Exact String Matching

Parameterized Matching

Substring Matching

Parameterized Matching

Employs exact string matching for comparison

1. Normalization
2. Concatenation
3. Hashing
4. Extract longest matches

Matching Algorithm

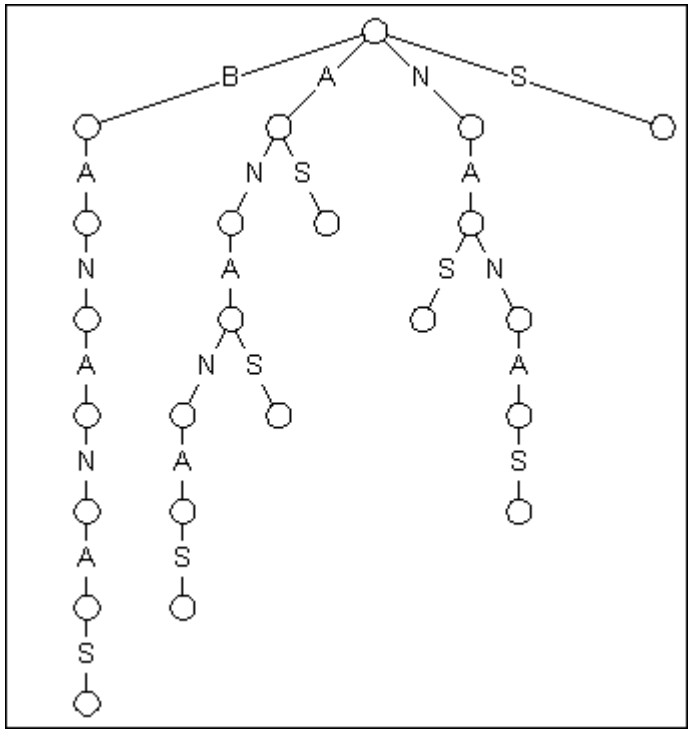
No algorithm can avoid worst case running time of $O(n^2)$

Using a suffix tree we can improve running time complexity to O(n+m). Where m is the number of matches

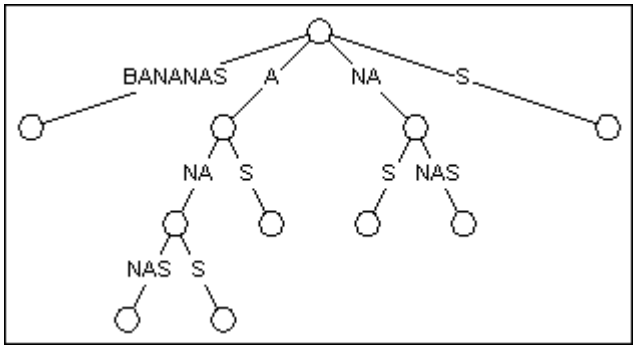
The input size n, is reduced by hashing

Suffix Tree Example

Suffix Tree



Suffix Tree



Substring Matching

Substring Matching provides a faster search algorithm.

1. Normalization
2. Substring Generation
3. Matching
4. Consolidation
5. Reporting

Caveat

Exact string matching does not find clones with trivial alterations that don't change the semantics

Normalization has the risk of false positives

```
x+y=z;    !=   z+x=y; ->   p+p=p
for(i=0; i<k; i++)   ->   for(p=p; p<p; p++)
```

Token Parsing Techniques

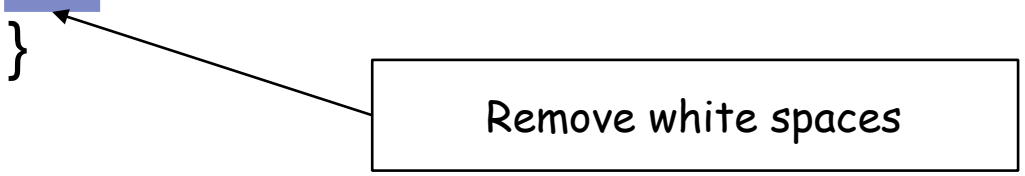
Transforms code into tokens by using language specific constructs into a single token string

Find similarities within this token string

Transform token clones back into code clones for presentation

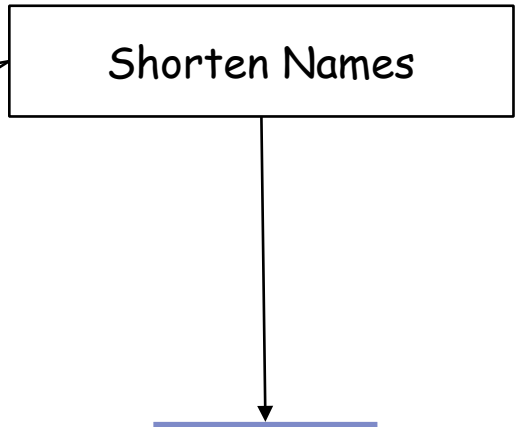
Token Parsing Example

```
int main(){
    int i = 0;
    static int j=5;
    while(i<20){
        i=i+j;
    }
    std::cout<<"Hello World"<<i<<std::endl;
    return 0;
}
```



Token Parsing Example

```
int main(){
int i = 0;
static int j=5;
while(i<20){
i=i+j;
}
std::cout<<"Hello World"<<i<<std::endl;
return 0;
}
```



Token Parsing Example

```
int main (){
int i = 0;
int j = 5;
while (i < 20){
i  =  i + j;
}
cout << "Hello World" << i << endl;
return 0;
}
```

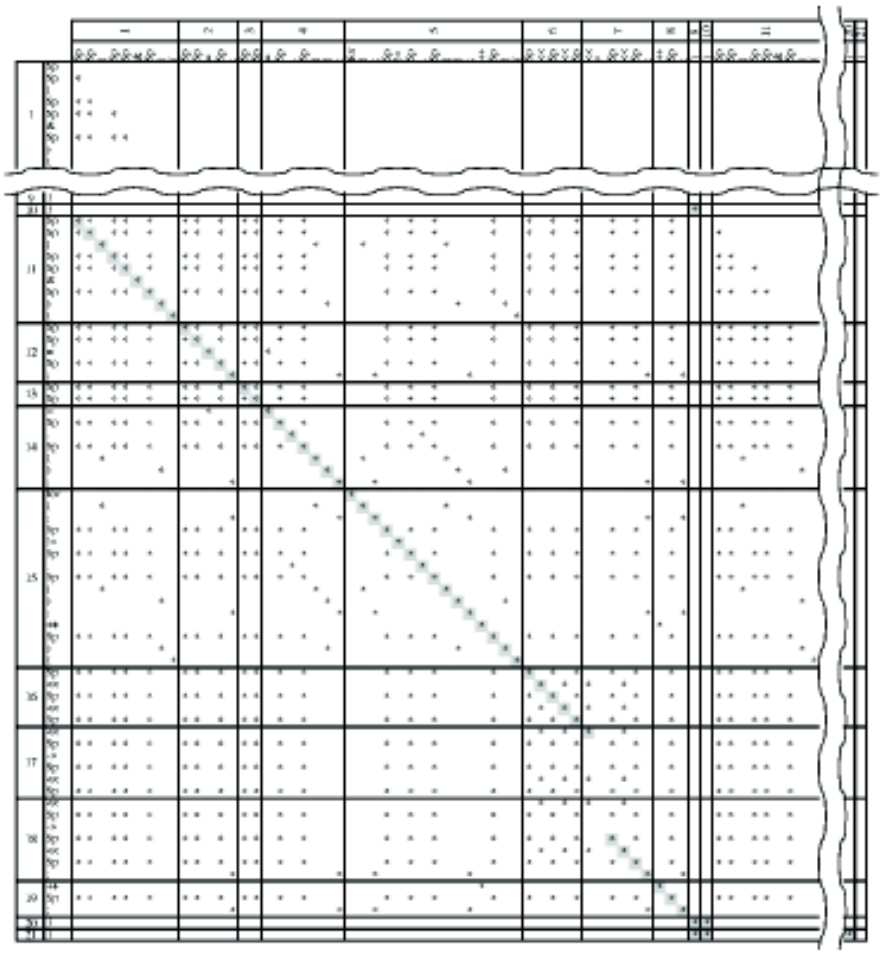
Tokenize everything, except language constructs

Token Parsing Example

```
$p $p(){
$p $p = $p;
$p $p = $p;
while($p < $p ){
$p = $p + $p;
}
$p << $p << $p << $p;;
return $p;
}
```

Clone relations with all the transformation rules are compared to clone relations with a subset of the transformation rules

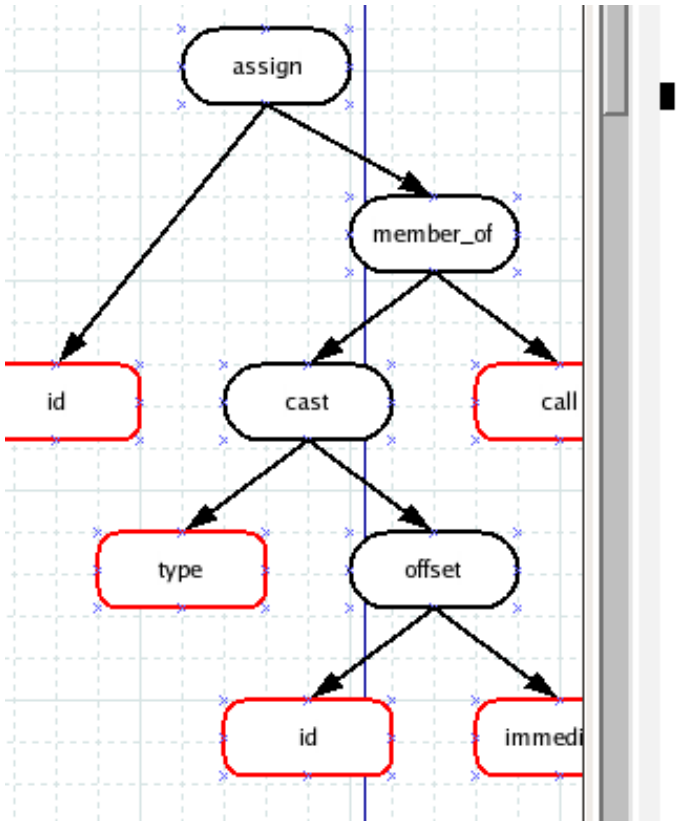
CCFinder – A Code Clone Finder Tool



Graph Matching Techniques

Form machine representation of code

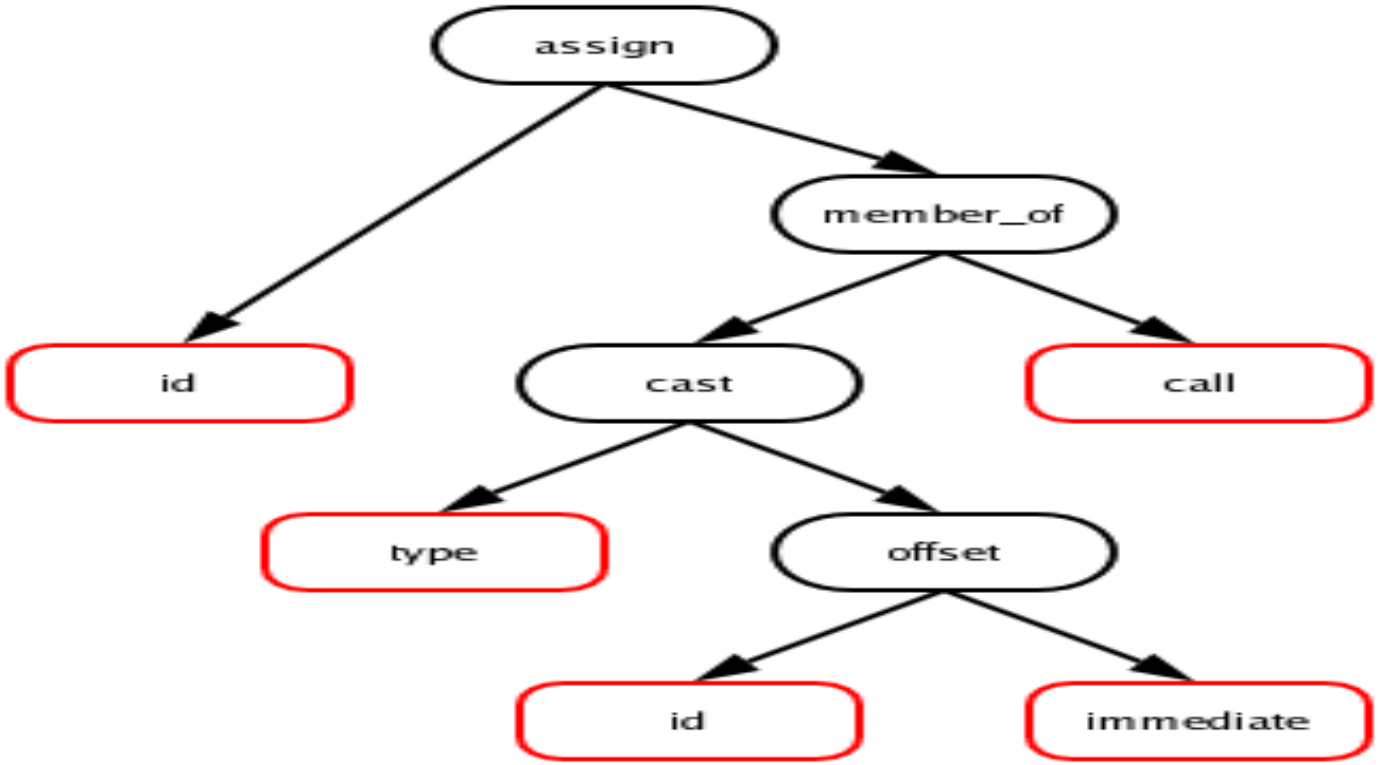
Identify clones as identical subgraphs




```
    interfaces =
  if(attributes =
    this.attribut
  if(fields == nu
    fields = new
  if(methods == r
    methods = new

  this.class_name
  this.superclass
  this.file_name
  this.major
  this.minor
  this.access_fla
  this.constant_p
  this.interfaces
  this.fields
  this.methods
  this.attributes
  this.source

  // Get source f
  for(int i=0; i
    if(attributes
source_file_name
break;
    }
  }

  /* According to
```

Abstract Syntax Subtree Matching

source_file_name = ((SourceFile) attributes[i]).getSourceFileName();



Abstract Syntax Subtree Matching

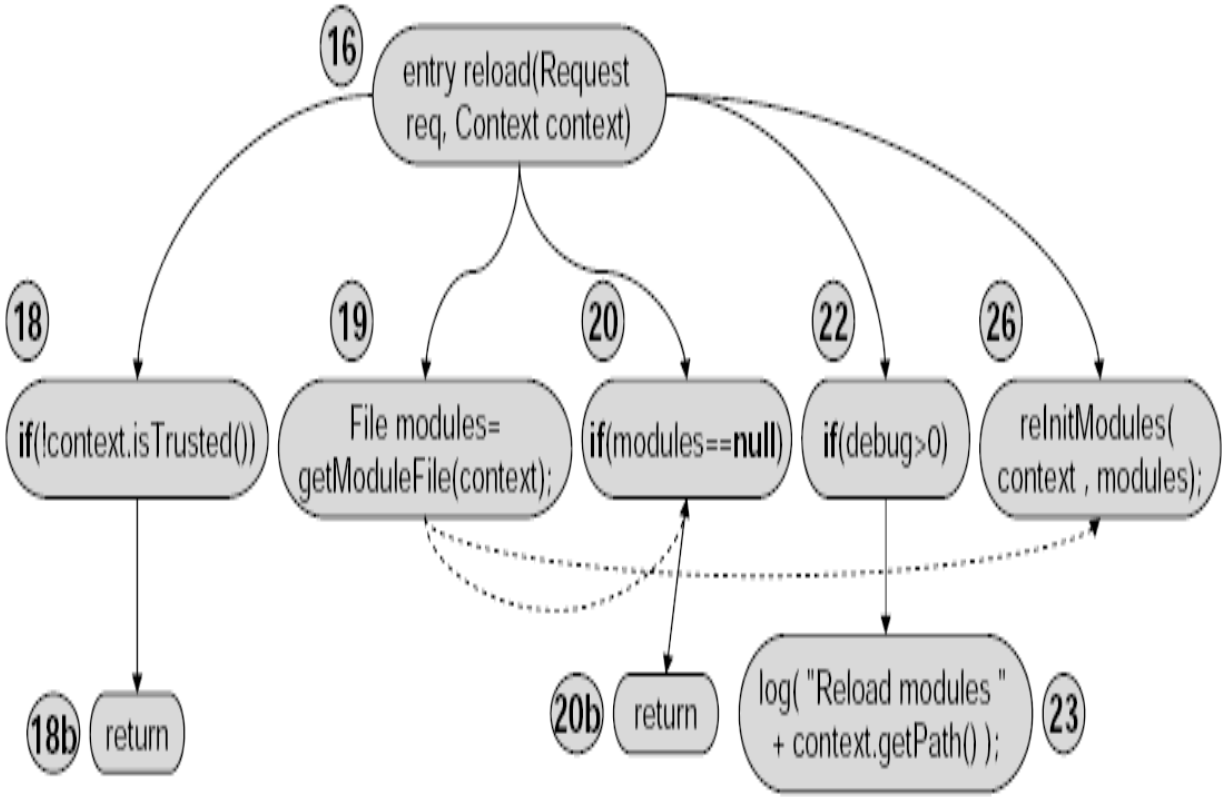
Hash subgraphs

Identify maximal identical or similar subgraphs

Identify sequences of subgraphs

Program Dependency Graph Matching

```
public class FooBar {
  public void contextInit(Context ctx) {
    if (!ctx.isTrusted()) { return; }
    if (debug > 0) {
      log("contextInit " + ctx + " " + cm.getState());
    }
    File modules = getModuleFile(ctx);
    if (modules == null) { return; }
    reInitModules(ctx, modules);
  }

  public void reload(Context context) {
    if (!context.isTrusted()) { return; }
    File modules = getModuleFile(context);
    if (modules == null) { return; }
    if (debug > 0) {
      log("Reload modules " + context.getPath());
    }
    reInitModules(context, modules);
  }
}
```



Program Dependence Graph Matching

Vertices are lines of code

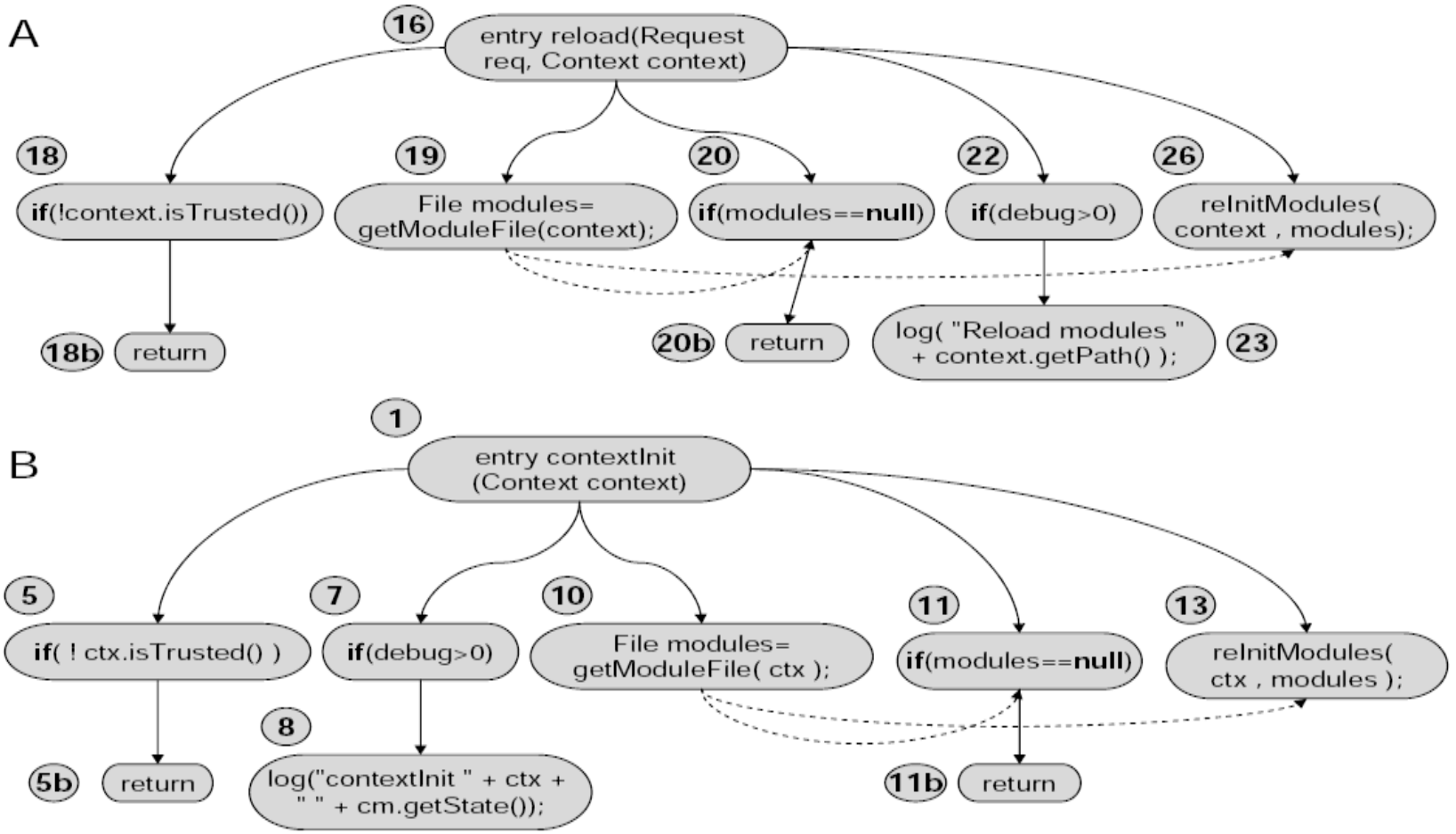
Edges are attributed with different types of dependencies (control flow, data flow, etc.)

NP complete in general, k-cutoff in maximal graph size used to limit runtime

- Experiments determine k=20 as best

$O(|V|^2)$ possible graph starting points, reduced via heuristic

Two Clones Found by fg-PDG



Metrics?

Need to evaluate different clone detection techniques

Hard to know the real number of clones in a non-trivial application

How to compare different types of clones?

Basic Metrics

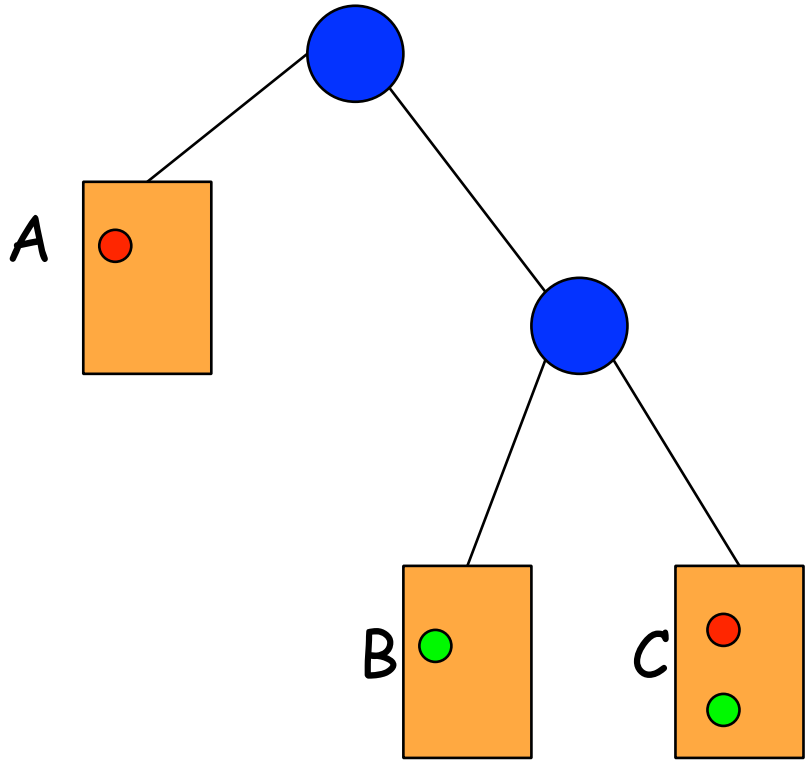
LOC: Line number count

SLOC: Line number count after the removal of blanks

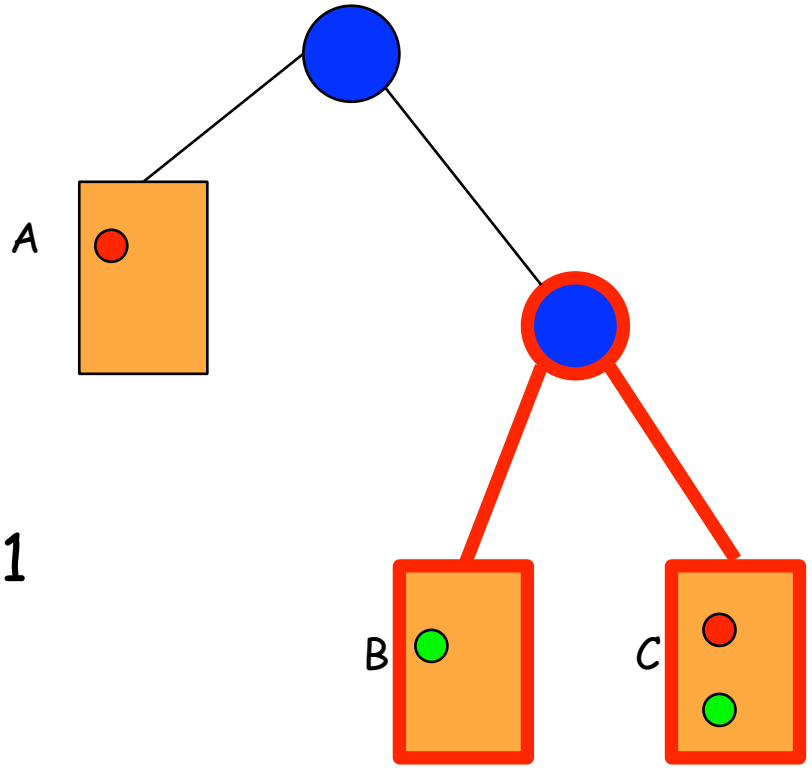
%LOC: Percent of lines with clones in them

%FILE: Percent of files with clones in them

Interesting Metrics: Radius

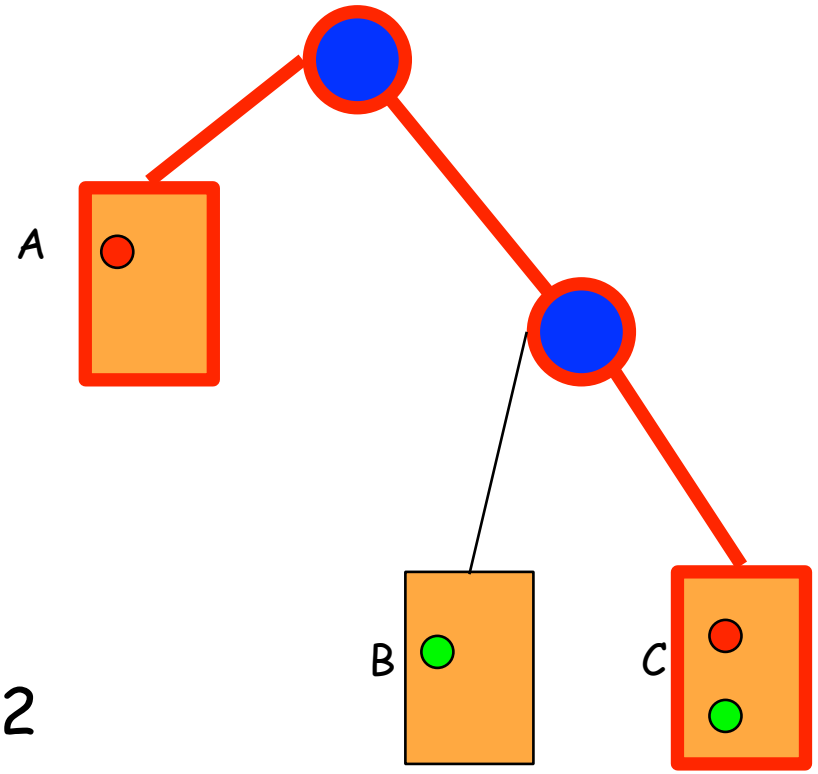


Interesting Metrics: Radius



RAD(B,C)= 1

Interesting Metrics: Radius



RAD(A,C)= 2

Comparison of Clone Detectors

	CCFinder Token (1128)	CloneDr AST (84)	Cavet Metric (278)	Jplag Token (131)	Moss Unknown (120)
CCFinder		1090/38	1089/27	989/87	1025/101
CloneDr	43		265/13	120/11	111/9
Cavel	251	70		120/15	109/10
Jplag	44	73	273		67/50
Moss	19	76	268	81	

Comparison of Clone Detectors

Frequency	CCFinder	CloneDr	Cavet	JPlag	Moss
1	569	66	40	95	104
2	98	6	34	10	8
3	33	2	13	4	0
4	14	0	6	1	0
5	16	0	5	0	0
6	19	0	5	0	0
7	2	0	1	0	0

In addition Cavet found clones with frequencies: 8,12, and 13

Comparison of Clone Detectors

	CCFinder	CloneDr	Cavet	JPlag	Moss
Recall	72	9	19	12	10
Precision	72	100	63	82	73

•Different code clone detectors find different clones

•**String** based find direct clones
•**Token** based find polymorphism issues and may be difficult to fix
•**Graph** based find clones that can be automatically refactored

Code Clone Refactoring

Use standard Refactoring methods

- “Extract” - Make a procedure
- “Pull Up” - Make an superclass

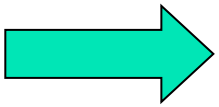
Aspect Oriented Programming

- Advanced technique for clones that are too tough for procedural or OO solutions

Duploc: A Lightweight Approach (1)

- Assumption
  - Code segments are just copied and changed in a few places
- Code Transformation Step
  - remove white space, comments
  - remove lines that contain uninteresting code elements (e.g., just 'else' or '}')

```
...
//assign same fastid as container
fastid = NULL;
const char* fidptr = get_fastid();
if(fidptr != NULL) {
 int l = strlen(fidptr);
 fastid = newchar[ l + 1 ];
```



```
...
fastid=NULL;
constchar*fidptr=get_fastid();
if(fidptr!=NULL)
intl=strlen(fidptr)
fastid = newchar[l+1]
```

A Lightweight Approach (2)

Code Comparison Step

- Line based comparison (Assumption: Layout did not change during copying)
- Compare each line with each other line.
- Reduce search space by hashing:
  - Preprocessing: Compute the hash value for each line
  - Actual Comparison: Compare all lines in the same hash bucket

Evaluation of the Approach

- Advantages: Simple, language independent
- Disadvantages: Difficult interpretation

Enhanced Simple Detection Approach

Code Comparison Step

- Same as before +
  - Collect consecutive matching lines into match sequences
  - Allow holes in the match sequence

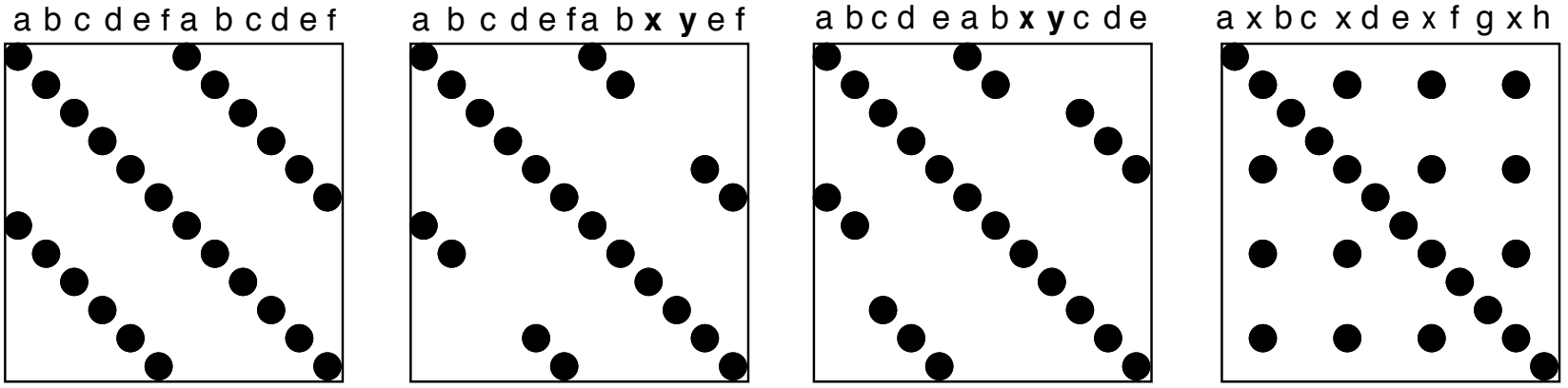
Evaluation of the Approach

- Advantages
  - Identifies more real duplication, language independent
- Disadvantages
  - Less simple
  - Misses copies with (small) changes on every line

Visualization of Duplicated Code

- Visualization provides insights into the duplication situation
- A simple version can be implemented in three days
- Scalability issue

- Dotplots — Technique from DNA Analysis
  - Code is put on vertical as well as horizontal axis
  - A match between two elements is a dot in the matrix



Visualization of Copied Code Sequences

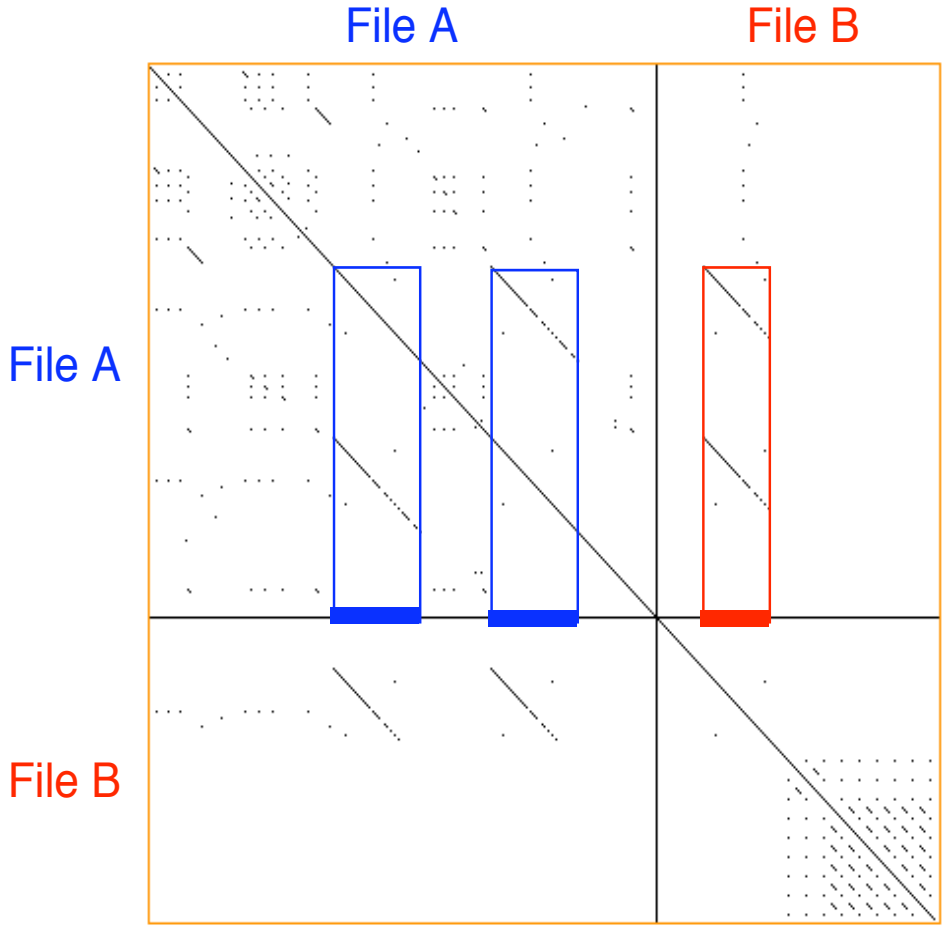
Detected Problem

File A contains two copies of a piece of code

File B contains another copy of this code

Possible Solution

Extract Method



All examples are made using Duploc from an industrial case study (1 Mio LOC C++ System)

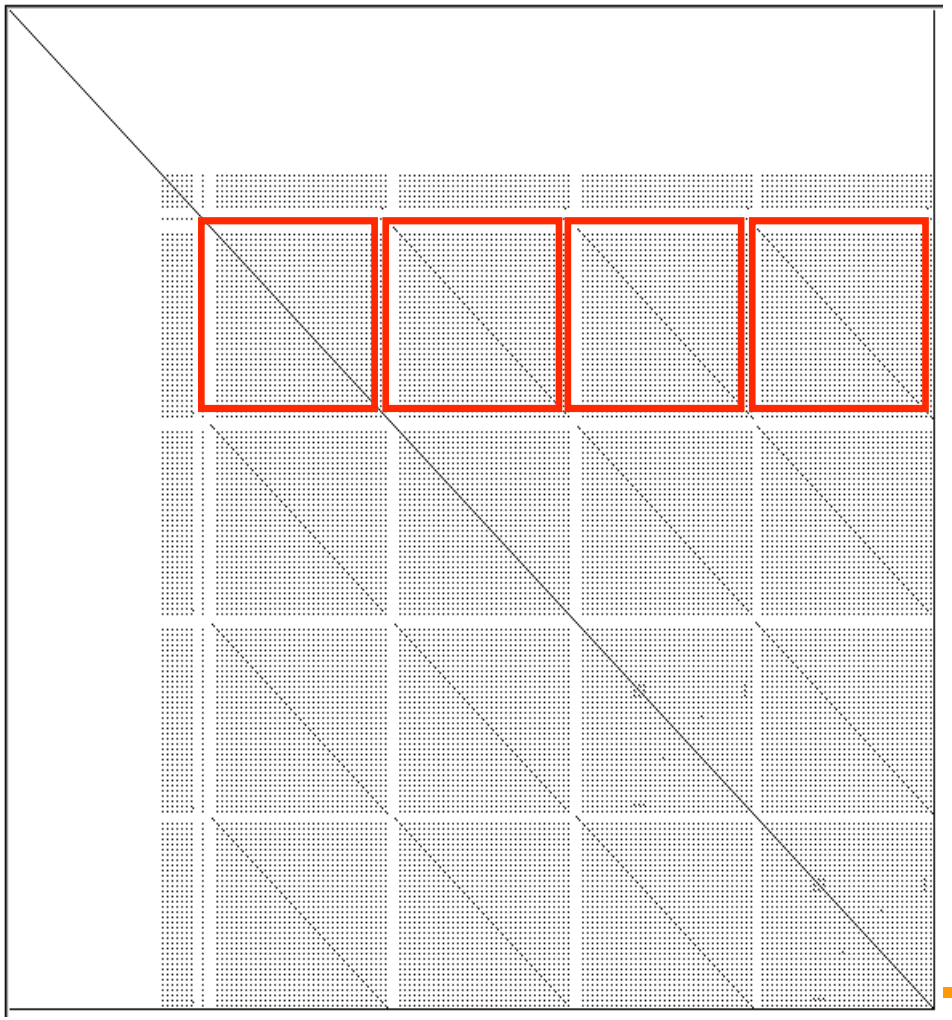
Visualization of Repetitive Structures

Detected Problem

4 Object factory clones: a switch statement over a type variable is used to call individual construction code

Possible Solution

Strategy Method



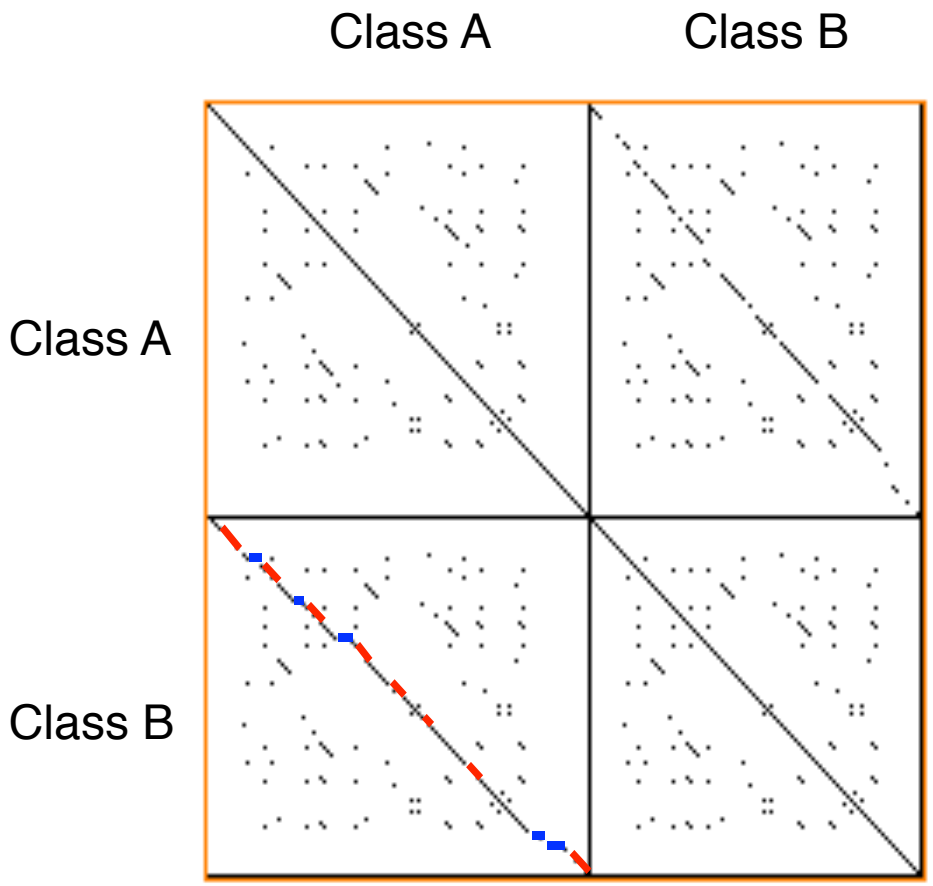
Visualization of Cloned Classes

Detected Problem

Class A is an edited copy
of class B. Editing & Insertion

Possible Solution

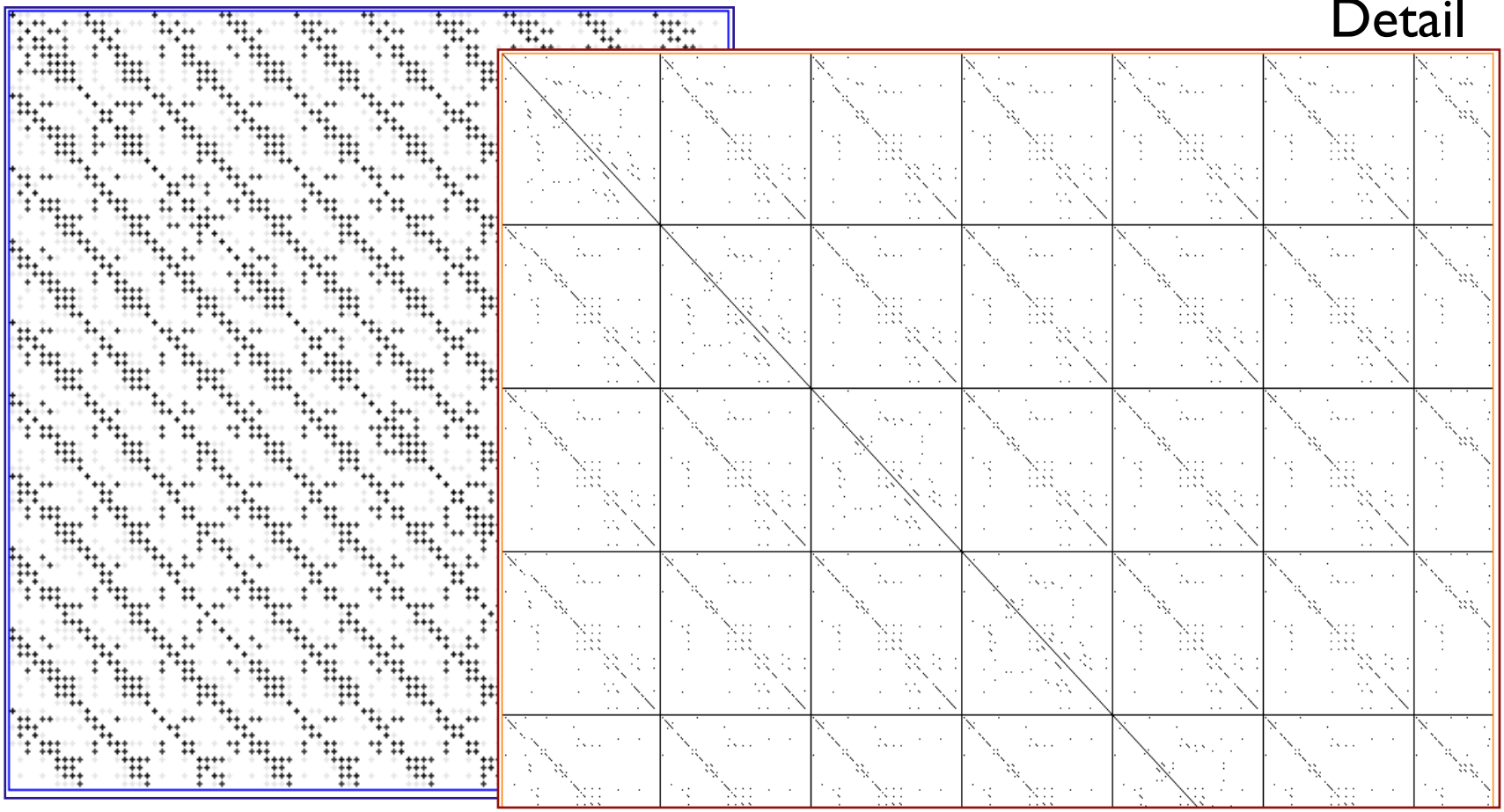
Subclassing …



Visualization of Clone Families

Overview

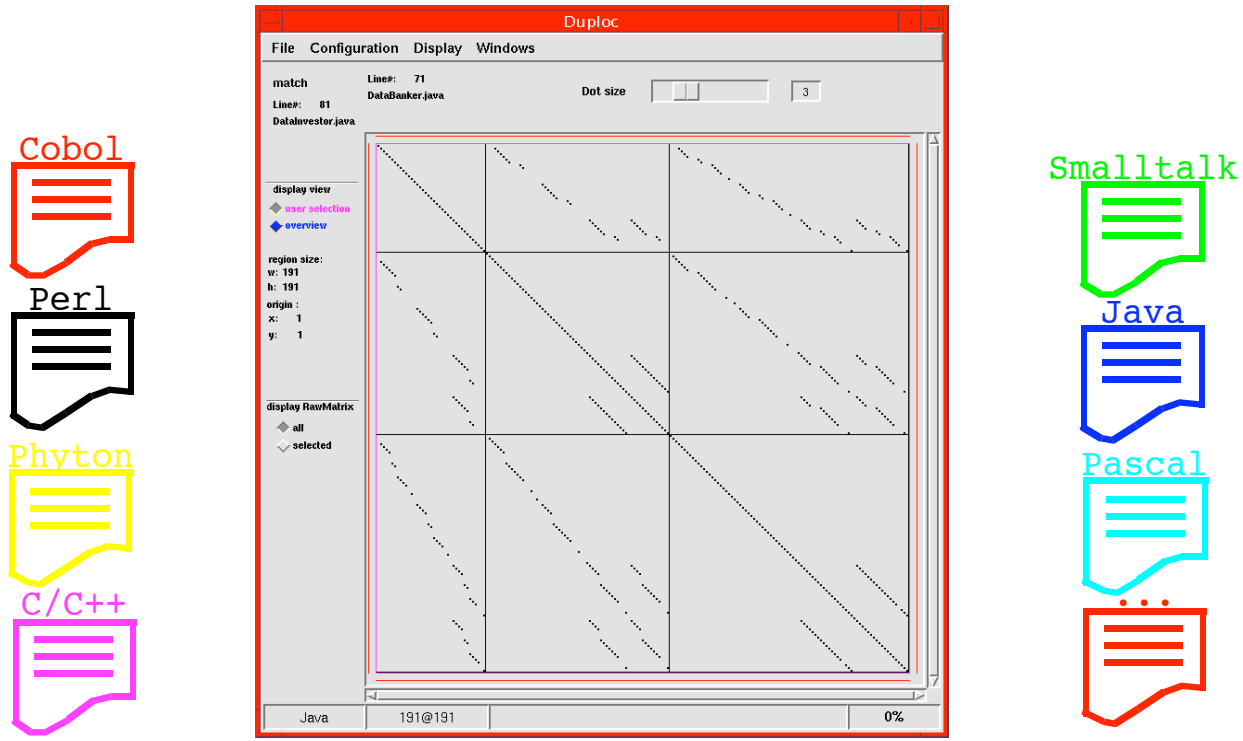
Detail



20 Classes implementing lists for different data types

Lightweight is sometimes not enough

Duploc is scalable, integrates detection and visualization



It runs really everywhere (Smalltalk inside)

More Clone Detection

Tool	Author	Supported Languages	Domain	Approach Category	Background
CCFinder	T.Kamiya	C, C++, COBOL, Java, Emacs Lisp, Plain Text	Clone Detection	Transformation followed by token matching	Academic
CloneDr	I. Baxter	C, C++, COBOL, Java, Progress	Clone Detection	Abstract Syntax Tree comparison	Commercial
Covet	J. Bailey J. Mayrand	Java	Clone Detection	Comparison of Function Metrics	Academic
JPlag	G. Malpohl	C, C++, Java, Scheme	Plagiarism Detection	Transformation followed by token matching	Academic
Moss	A. Aiken	Ada, C, C++, Java, Lisp, ML, Pascal, Scheme	Plagiarism Detection	Unpublished	Academic

[Burd02]

Résumé

Duplicated code is a real problem

- makes a system progressively harder to change

Detecting duplicated code is a hard problem

- some simple technique can help
- tool support is needed

Visualization of code duplication is useful

- some basic support are easy to build
- one student build a simple visualization tool in three days

Curing duplicated code is an active research area