Classification of Java Programs in SPARS-J

Kazuo Kobori, Tetsuo Yamamoto,
Makoto Matsusita and Katsuro Inoue
Osaka University

Contents

- Background
  - SPARS-J
  - Reuse
- Similarity measurement techniques
  - Characteristic metrics method
  - Inclusive relation method
- Examples of its application
- Summaries and Future Works

Background

- SPARS-J is the web-based search engine for support of Software Reuse (for Java)
  - A lot of parts are managed in this system
    - source codes from open-source projects and public access files
    - repository which stores 130,000 classes
  - Components are classified by functions.
    - In order to evaluate use-relation of every function
    - Similar components may have the same functions

Measurement of similarity between Components is needed.

Reuse

Similar components are made by Reuse

Reuse is roughly divided into following two:

1. Reused as it is.
   - Components are copied and used as it is.
   - Some elements may be changed.
2. Reused by changing code.
   - Components are copied and used with additional codes.
   - Some methods and some variables are mainly added.

Similarity measurement technique

- Character string comparison
  - has so far been used for similar comparison of programs
    - the high analysis cost per one comparison
    - Hugeness of the total number of times of comparison

↓

It is unsuitable for SPARS-J

↓

We need much lower cost method

Similarity measurement technique in SPARS-J

- Characteristic metrics method
  - **In order to grasp Reuse as it is**
  - Metrics show the constitution of a component
  - Metric is integer
  - Only comparison of metrics is used for a similarity measurement
    - reduction of calculation cost

- Inclusive relation method
  - **In order to grasp Reuse by minor change**
  - By using the code clone information between components, we analyze inclusive relation
  - It has a scalability which can bear practical analysis.
    - Analysis against millions of lines in practical time.

Characteristic Metrics

- Characteristic metrics is measured from two viewpoints.
  - Complexity
    - number of methods, cyclomatic number, and etc.
    - It shows a structural characteristic.
  - Token-composition
    - number of appearances of each token.
    - Token ＝ Reserved + Symbol + Operator + Identifier
      （96 types） （49） （9） （37）
      （1）
    - It shows a surface characteristic.

Extraction of Characteristic Metrics

```
public class sample {
   int a , b , s ;
   char c ;

   public void main (  ) {
      c = ‘ m ’ ;
      if ( c = = ‘ m ’ ) {
         s = sum ( a , b ) ;
      }
      else {
         s = a + b;
      }

   public void sum ( int p , int q ) {
       return ( p + q ) ;
   }
}
```

Complexity	value
N of Cyclomatic	2
N of method	
⋮	
N of interface	

Token	Value
int	
void	
⋮	
identifer	
T_{total}	

Extraction of Characteristic Metrics

```
public class sample {
   int a , b , s ;
   char c ;

   public void main (  ) {
      c = ‘ m ’ ;
      if ( c = = ‘ m ’ ) {
         s = sum ( a , b ) ;
      }
      else {
         s = a + b;
      }

   public void sum ( int p , int q ) {
       return ( p + q ) ;
   }
}
```

Complexity	value
N of Cyclomatic	2
N of method	2
⋮	⋮
N of interface	0

Complexity
metrics

Token	Value
int	
void	
⋮	
identifer	
T_{total}	

Extraction of Characteristic Metrics

```
public class sample {
   int a , b , s ;
   char c ;

   public void main ( ) {
     c = ‘ m ’ ;
     if ( c = = ‘ m ’ ) {
       s = sum ( a , b ) ;
     }
     else {
       s = a + b;
     }

   public void sum ( int p , int q ) {
      return ( p + q ) ;
   }
}
```

Complexity	value
N of Cyclomatic	2
N of method	2
⋮	⋮
N of interface	0

Complexity metrics

Token	Value
int	3
void	2
⋮	⋮
identifer	23
T_{total}	75

Token composition metrics

Judge Condition -1-

- Step1: We set thresholds of each complexity metrics

Metric	threshold
N of Cyclomatic	0
N of methods	1
N of method calls	2
Nesting depth	1
N of classes	0
N of interfaces	0

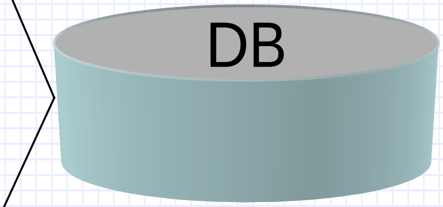
Judge Condition -1-

- We make hash key by Complexity metrics

Hash Key (24bit) =	8bit	8bit	8bit
	metric A	metric B	metric C

We make Hash Table in which Hash Key corresponds to components

[0. 0. 0]= null
⋮
[10. 62.124]= Cp.A
[10. 62.125]= Cp.B, Cp.C
[10. 62.126]= null
⋮
[254.254.254]= Cp.Z



- If we judge new component P
  - **Hash Key of Cp.P**＝[10.62.125]
  - **Thresholds of metric[A,B,C]**＝[0.0.1

[10.62.124]
[10.62.125] **We search these 3 keys**
[10.62.126]

We now similarity components down to Component A , B and C.

Judge Condition -2-

- Step2 : Components are judged by characteristic metrics

Token
Composition
Metrics

Component	A	B
int	3	4
void	2	2
⋮	⋮	⋮
identifer	23	25
T_{total}	75	76

■ ***D(A,B)*: Non-similarity between Component A and B**

The sum of the difference of TCM

$$\boxed{D(A,B) \equiv \frac{\mathrm{diff}(A,B)}{\min(T_{total}(A)\ ,\ T_{total}(B))}} < \text{threshold}$$

Pattern of Reuse

1. Reused as it is.

   It can be extracted by judging similar components.

2. Reused by changing code.

   It can be extracted not by judging similar components,
   but by detecting inclusive relation.

Pattern of Reuse

1. Reused as it is.

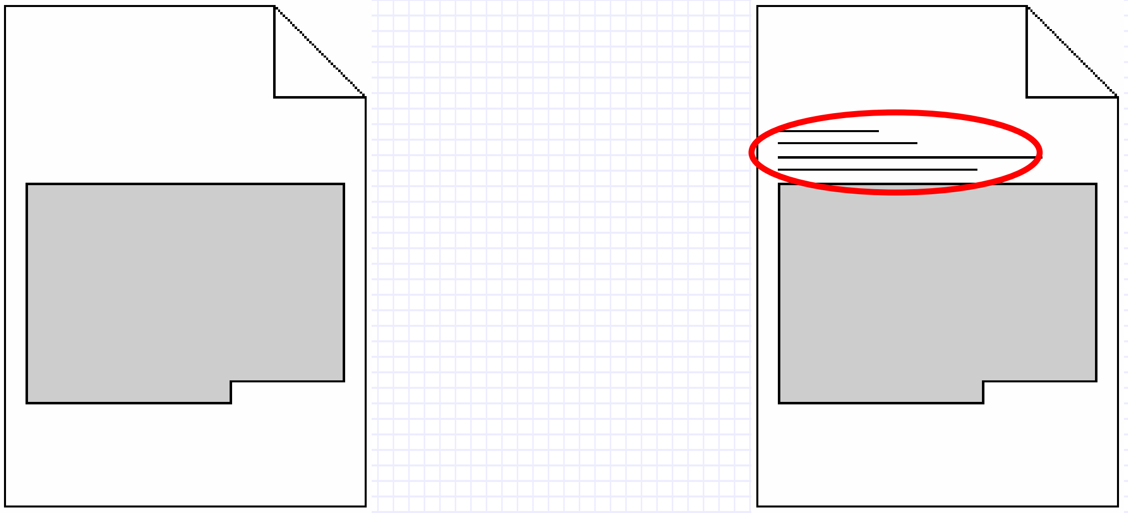
   It can be extracted by judging similar components.

2. Reused by changing code.

   It can be extracted not by judging similar components,
   but by detecting inclusive relations.

Inclusive relation

- In characteristic metrics method
  - One component contains another component completely.
  - However, If the difference of size is more than the threshold.
  - In this case, these two components can’t be judged to be similar.



Inclusive relation method

- In order to grasp reuse with code addition
- By using the code clone information between components, we analyze inclusive relation
  - Use of a code clone detection tool：「***CCFinder***」*
    - It has a scalability which can bear practical analysis.
      - Analysis against millions of lines in practical time.

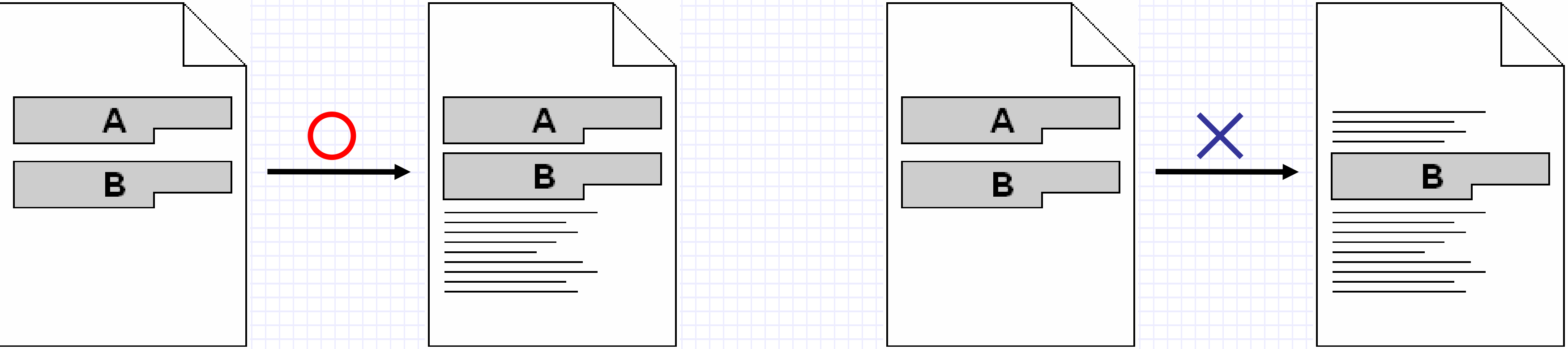
***Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, "CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code," IEEE Trans. Software Engineering, vol. 28, no. 7, pp. 654-670, (2002-7).**

The Inclusive Relations in Software Components

- About Component x:
  - Total Line of Codes of x ＝ LOC(x)
  - The Number of Lines of x which is also contained in component y as a code clone ＝ Cy(x)

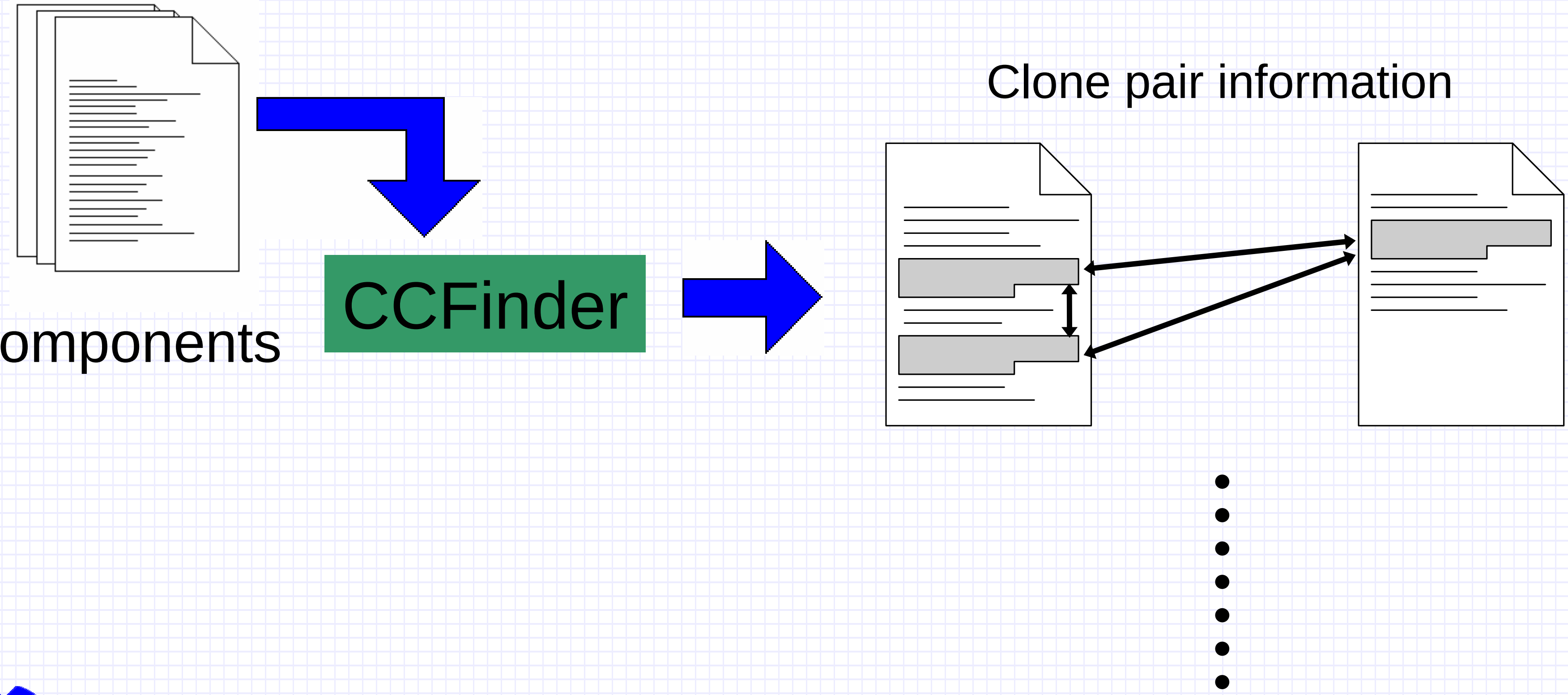
Δ←threshold

$$x \subseteq y \;\Leftrightarrow\; \boxed{\mathrm{LOC}(x) \times \Delta \leqq \mathrm{Cy}(x)}$$



The Extraction Method of Inclusive Relation -1-

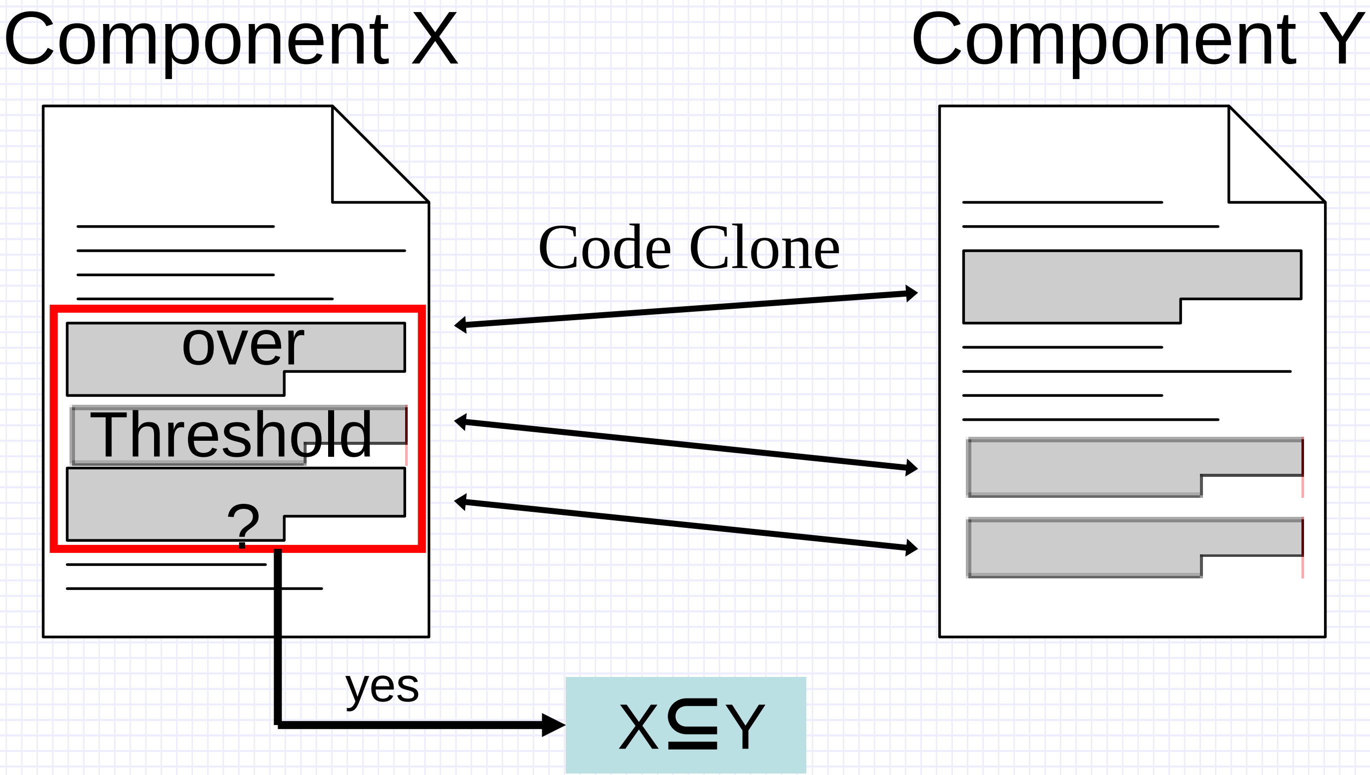
- Step 1: Code clone pair information is calculated through analysis of CCFinder.



The Extraction Method of Inclusive Relation -2-

- Step2: For each component X, check this formula

$$x \subseteq y \iff \boxed{LOC(x) \times \Delta \leqq Cy(x)}$$



The Extraction Method of Inclusive Relation -3-

- Step３: By comparing metrics, this judges whether the extracted pair is an inclusive relation.

metric	Cp.X	Cp.Y
int	3 <	4
void	2 <	2
⋮	⋮ <	⋮
identifer	23 <	40
T_{total}	75 <	102

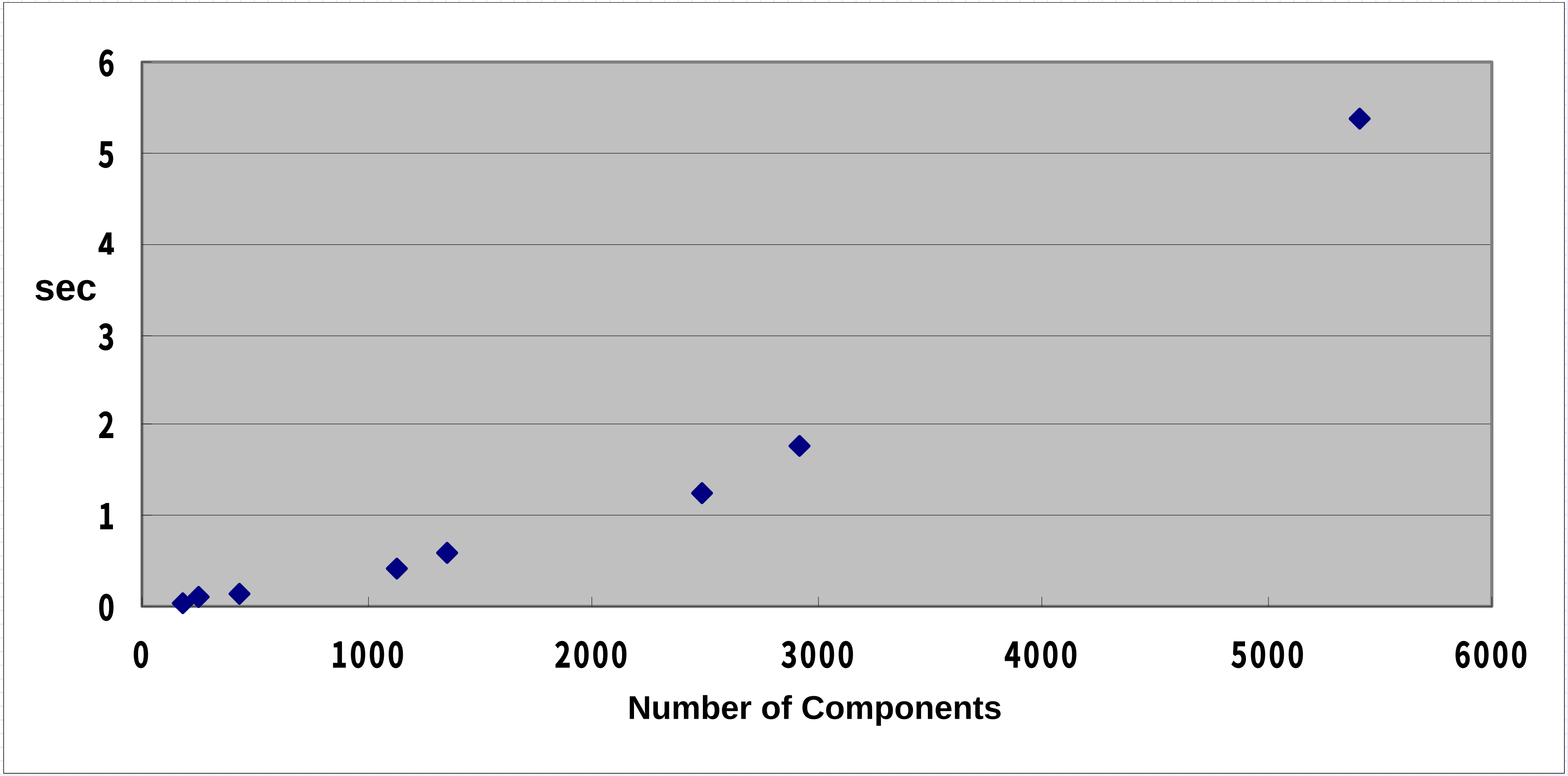
■Cp.X ⊆ Cp.Y

Application Result

- Characteristic metrics method
  - We show the cost scale figure
- Inclusive relation method
  - We show some examples which are in inclusive relation

Application Result -1-

■ calculation time of Characteristic Metrics Method

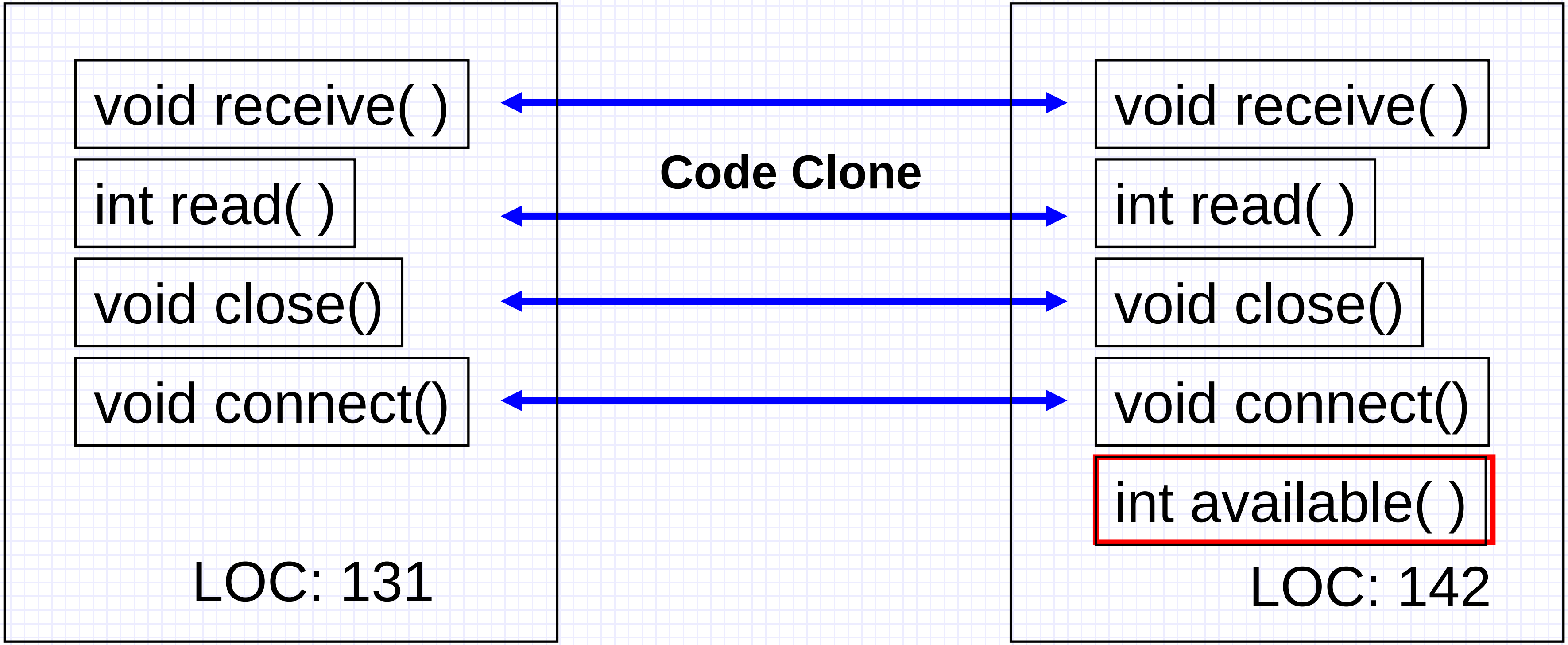


※calculation time of characteristic string method = 24.3 sec (at 500 component

Application Result -2-

- Example of a extracted inclusive relation

PipedReader ⊆ PipedInputStream



void receive( )

int read( )

void close()

void connect()

LOC: 131

Code Clone

void receive( )

int read( )

void close()

void connect()

int available( )

LOC: 142

Application Result -2-

- other Examples of a extracted inclusive relation

PropertyPermission
LOC:135
⊆
SocketPermission
LOC:457
FilePermmission
LOC:249

Format
LOC:25
⊆
NumberFormat
LOC:207

Summary And Future Work

- Summary
  - We have suggested similarity measurements
    - Characteristic metrics method
    - Inclusive relations method
- Future Work
  - Evaluation of system performance
  - Adjustment of a threshold