Software Evolution 2015

Lecture 5: Clone Detection and Management

Status check

- Lab series 1: how did it go?
- Already started on Series 2?

Code duplication, cloning, redundancy

- Important aspect of evolving software
- Active research field with lots of results
- Interesting opportunity for constructing tools

The plan for today

1. Clone detection
   - Finding clones
   - Visualizing clones

2. Break (15 minutes max.)

3. Clone management
   - Preventive
   - Compensative
   - Corrective

4. Lab series 2

Cloning definitions

Clone: fragment of code that is duplicated somewhere else.

Clone pair: two code fragments that are duplicates of each other.

Clone class: any number of code fragments that are all duplicates of each other.

More formally

Consider ‘is clone of’ an *equivalence relation* <->

Reflexivity:

every clone is a clone of itself

Symmetric:

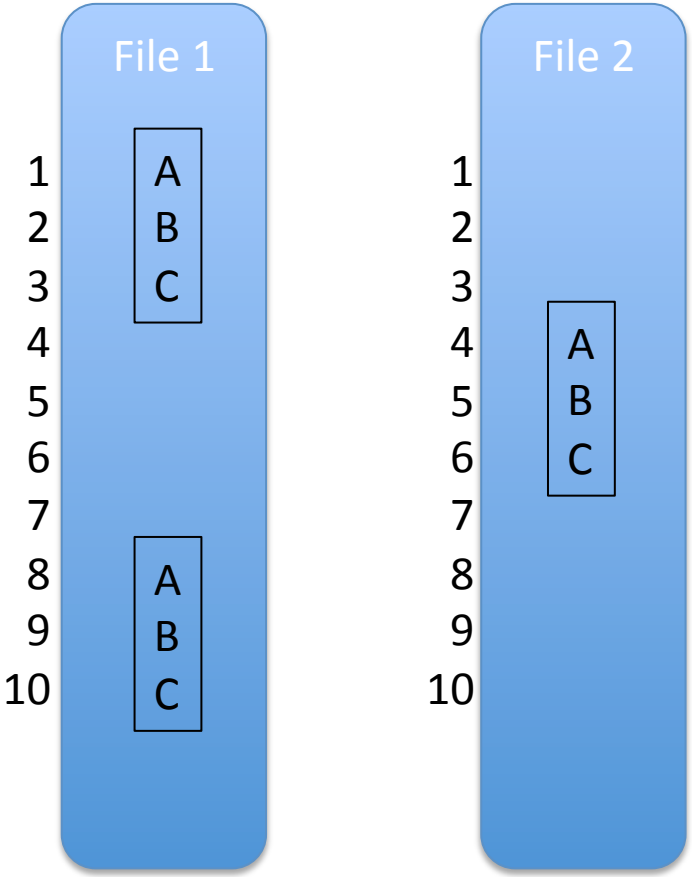
if A <-> B, then B <-> A

Transitivity:

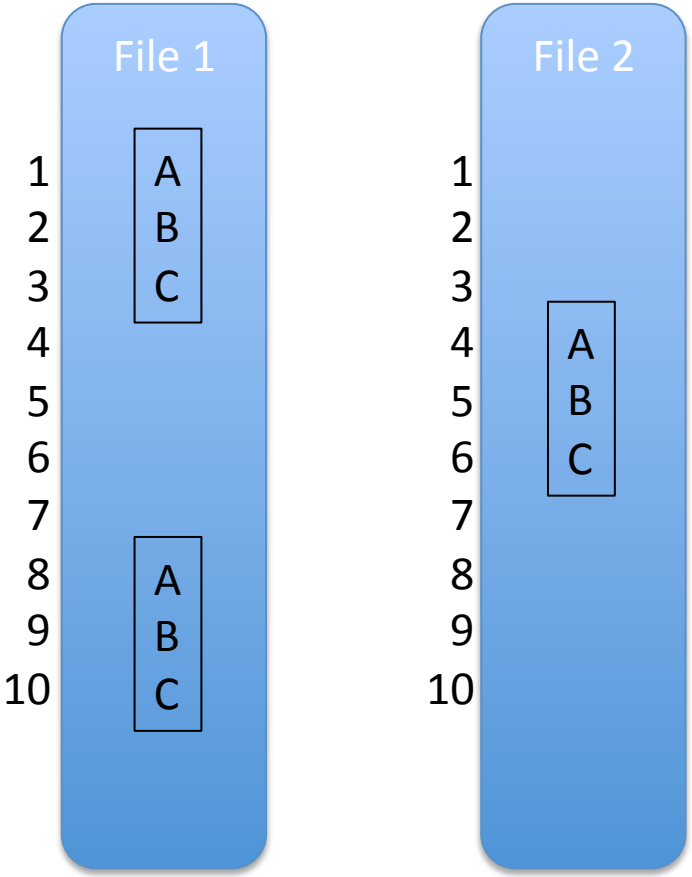
if A <-> B and B <-> C, then A <-> C

Given the above, clone classes are the *equivalence classes* of <->

Cloning definitions



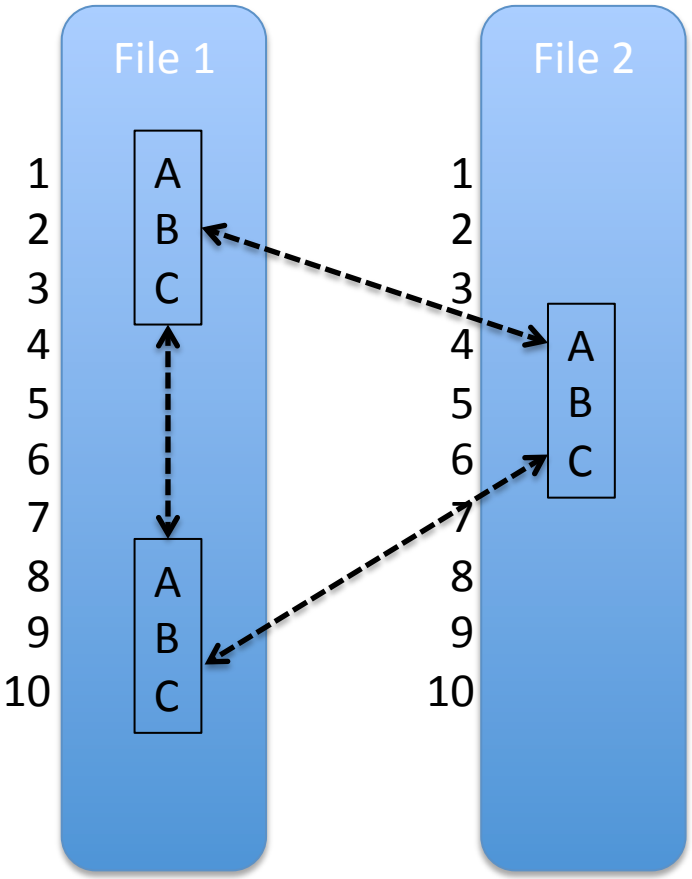
Cloning definitions: clones



Clones:

1. File 1: 1-3
2. File 1: 8-10
3. File 2: 4-6

Cloning definitions: clone pairs



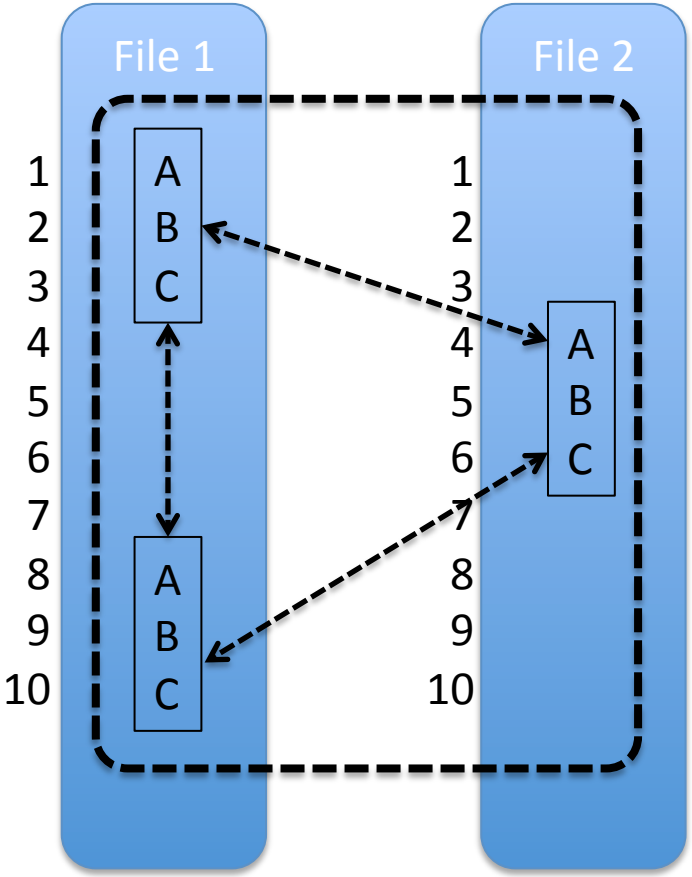
Clones:

1. File 1: 1-3
2. File 1: 8-10
3. File 2: 4-6

Clone pairs:

1. File 1: 1-3 <-> File 1: 8-10
2. File 1: 1-3 <-> File 2: 4-7
3. File 1: 8-10 <-> File 2: 4-6

Cloning definitions: clone classes



Clones:

1. File 1: 1-3
2. File 1: 8-10
3. File 2: 4-6

Clone pairs:

1. File 1: 1-3 <-> File 1: 8-10
2. File 1: 1-3 <-> File 2: 4-7
3. File 1: 8-10 <-> File 2: 4-6

Clone classes:

1. File 1: 1-3, File 1: 8-10, File 2: 4-6

Clone types

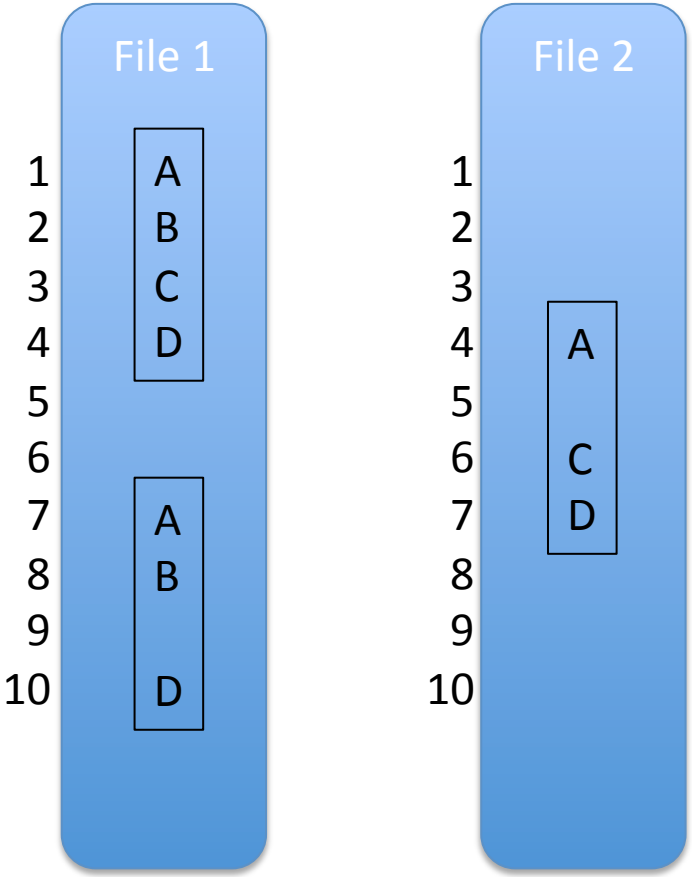
Type 1: exact copy, ignoring whitespace and comments.

Type 2: syntactical copy, changes allowed in variable, type, function identifiers.

Type 3: copy with changed, added, and deleted statements.

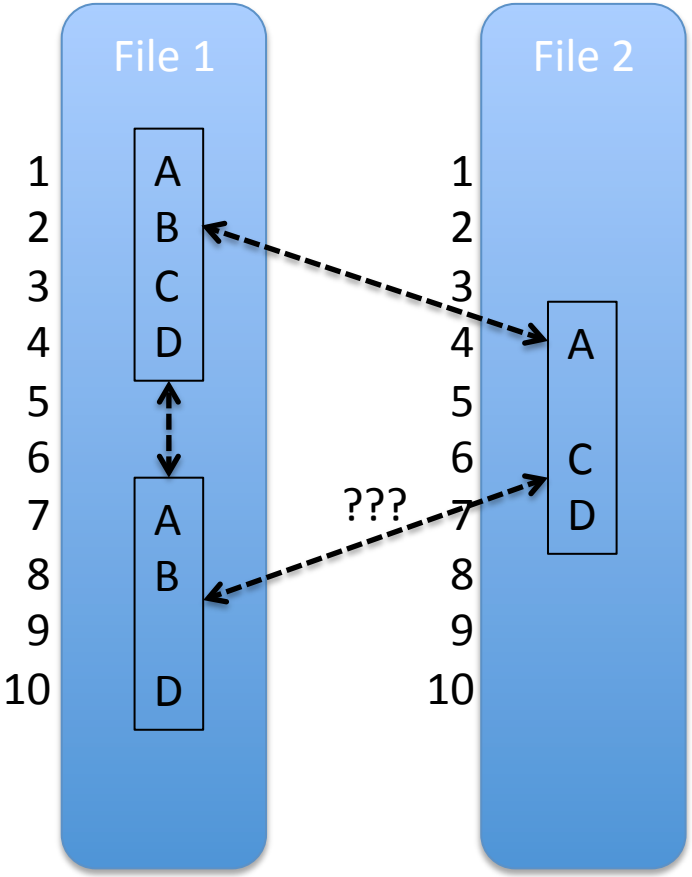
Type 4: functionality is the same, code may be completely different.

Cloning definitions: Type 3?



Example: clones can have 1 line added/deleted.

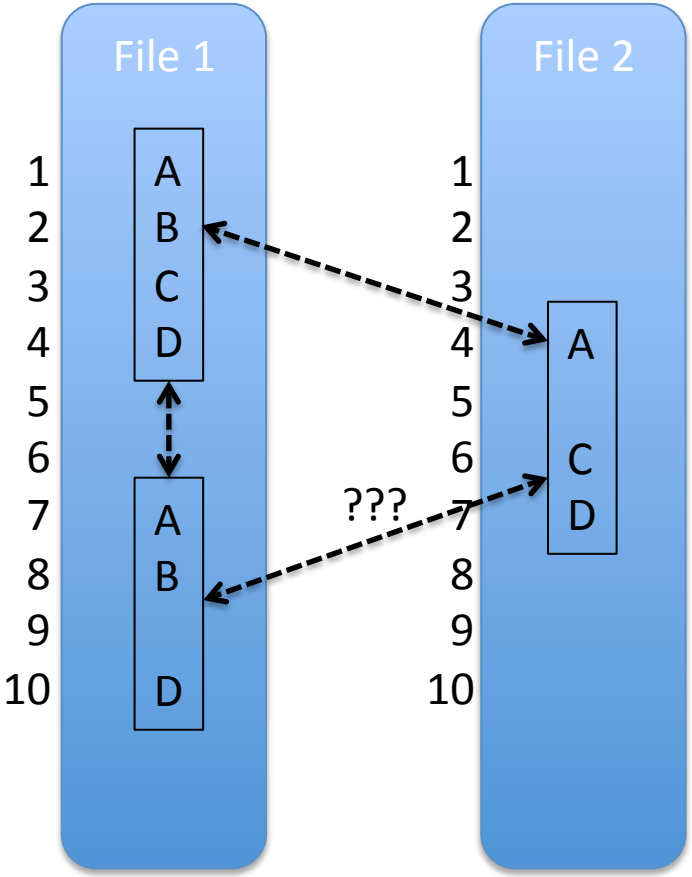
Cloning definitions: Type 3?



Clone pairs

1. File 1: 1-4 <-> File 1: 7-8, 10
2. File 1: 1-4 <-> File 2: 4, 5-7
3. File 1: 7-8, 10 <-> File 2: 4, 5-7 ???

Cloning definitions: Type 3?



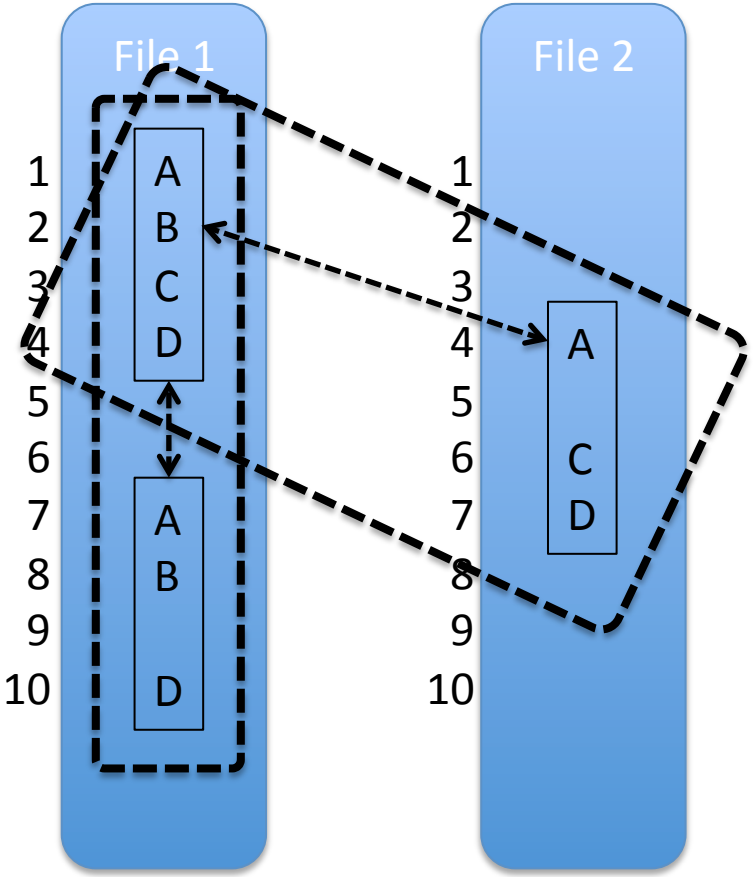
Clone pairs

1. File 1: 1-4 <-> File 1: 7-8, 10
2. File 1: 1-4 <-> File 2: 4, 5-7
3. File 1: 7-8, 10 <-> File 2: 4, 5-7 ???

'is Type 3 clone of' relation is not transitive!

What are the clone classes here?

Cloning definitions: Type 3?



Clone pairs

1. File 1: 1-4 <-> File 1: 7-8, 10
2. File 1: 1-4 <-> File 2: 4, 5-7
3. File 1: 7-8, 10 <-> File 2: 4, 5-7 ???

'is Type 3 clone of' relation is not transitive!

What are the clone classes here?

1. File 1: 1-4, File 1: 7-8, 10
2. File 1: 1-4, File 2: 4, 6-7

Only Type 1 and Type 2 can be considered equivalence relations!

Clone detection algorithms

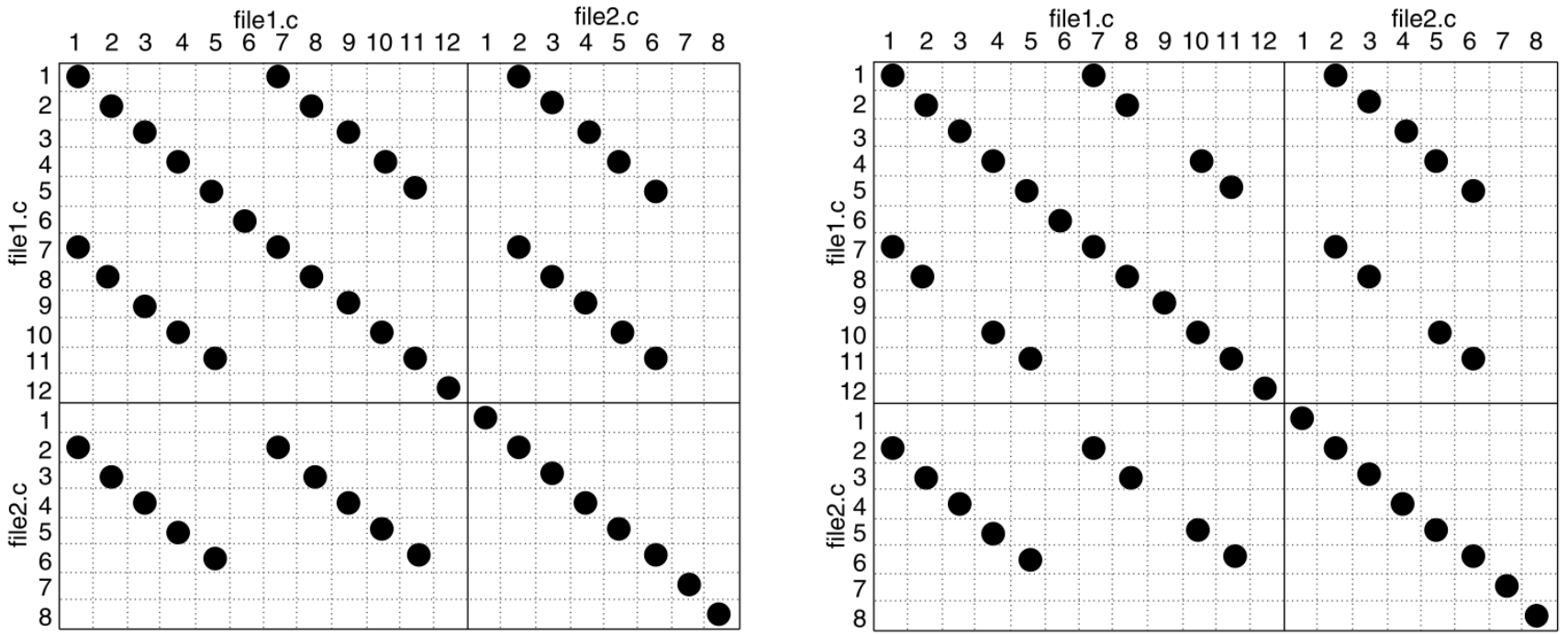
Algorithm can be based on:

- Text
- Tokens
- Metrics
- Abstract Syntax Trees (AST)
- Program Dependency Graphs (PDG)
- Other

Text-based clone detection

- Compare each line to each other textually
- Use hashing to speed up
- Extend cloned lines into bigger fragments

Clone visualization – dot plots



(a) Three clones **(b)** Three clones, one modified line

Fig. 2.3. Dot plots [114, 156, 512]

Source: R. Koschke, *Identifying and Removing Software Clones,* in Software Evolution, Springer, 2008.

Token-based clone detection

- First apply lexical analysis to obtain tokens
- Optionally apply normalization on the tokens
  - e.g. insert missing parentheses:
    if (a) … becomes if (a) {…}
  - Remove some tokens
- Use an efficient algorithm (e.g. based on creating a suffix tree) to detect identical sequences of tokens

Metric-based clone detection

- Split the code into well-defined fragments, for instance methods or functions
- Calculate vector of metrics on each fragment
  - Original approach uses 21 metrics on layout, control flow, and expressions.
- Calculate distance measure between vectors to find similar fragments

AST-based clone detection

- First parse the code into an abstract-syntax tree
- Find identical subtrees
- Optionally ignore the leaves of the tree (e.g., identifiers) to detect Type 2 clones

Finding sequences 1/2

```
void f ()
{
  x=0;
  a=1;
  b=2;
  c=3;
  w=4;
}
```

```
void g ()
{
  y=2;
  a=1;
  b=2;
  c=3;
  i=5;
}
```

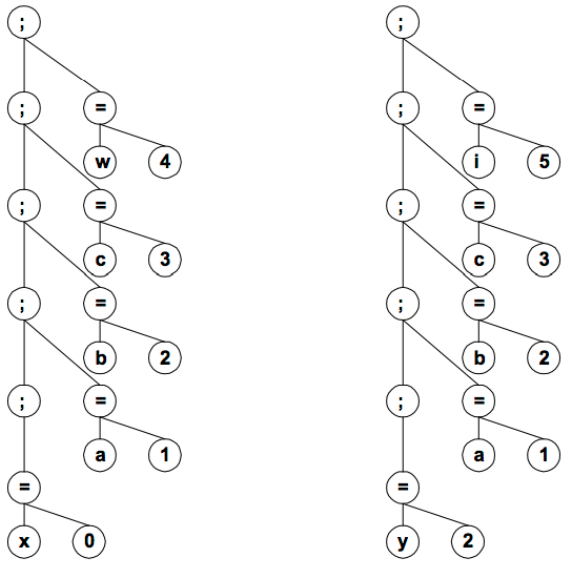


Figure 2 – Example of clone sequence

I. Baxter et al., Clone Detection Using Abstract Syntax Trees, IEEE, 1998.

Finding sequences 2/2

```
1. Build the list structures describing sequences
2. For k = MinimumSequenceLengthThreshold
        to MaximumSequenceLength
3. Place all subsequences of length k
   into buckets according to subsequence hash
4. For each subsequence i and j in same bucket
   If CompareSequences (i,j,k) >
              SimilarityThreshold
   Then { RemoveSequenceSubclonesOf(clones,i,j,k)
          AddSequenceClonePair(Clones,i,j,k)
        }
```

Figure 4 – Sequence detection algorithm

```
void f ()
{
  x=0;
  if (d>1)
  {
    y=1;
    z=2;
  }
  else
  {
    h=2;
    z=1;
    y=3;
  }
}
```

The program has three sequences.

List Structure:

1. {x=0; if(d>1) … }

   hashcodes = 675, 3004

2. {y=1; z= 2;}

   hashcodes = 1020,755

3. {h=2; z=1; y=3;}
   hashcodes = 786, 756, 704

Figure 3 – List structure example

I. Baxter et al., Clone Detection Using Abstract Syntax Trees, IEEE, 1998.

Alternative AST-based

- First get an AST
- Cut-off or replace leaves you want to ignore
- Serialize the AST back into a string or token stream
- Use a string duplicate search algo to find identical substrings:
  - Suffix tree
  - Suffix array
  - Lab series 1-style hash-and-extend

```
if x + y then a := i; else foo; end if;
if p     then a := j; else foo; end if;
if q     then z := k; else bar; end if;
```

Listing 2. Sequence of if statements in Ada

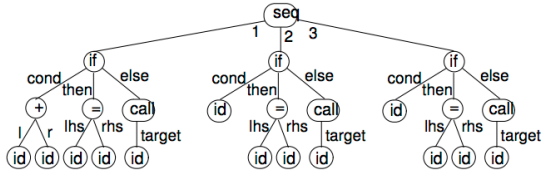


Figure 2. Example AST

Because the token-based clone detection is based on a token stream, we need to serialize the AST nodes. We serialize the AST by a preorder traversal. For each visited AST node N, we emit N as root and associate the number of arguments (number of AST nodes transitively derived from N) with it (in the following presented as subscript).

Note that we assume that we traverse the children of a node from left to right according to their corresponding source locations so that their order corresponds to the textual order.

The serialized AST nodes produced in step (2) for the example are shown in Listing 3.

```
1  seq23
2      if8 +2 id0 id0 =2 id0 id0 call1 id0
3      if6 id0 =2 id0 id0 call1 id0
4      if6 id0 =2 id0 id0 call1 id0
```

Listing 3. Serialized AST nodes

R. Koschke et al., Clone Detection Using Abstract Syntax Suffix Trees, IEEE, 2006.

PDG-based clone detection

- Create a graph that describes control- and data flow dependency (this is hard)
- Find isomorphic subgraphs (this is NP-complete, so you need an approximation)
- You can find (some) Type 3 clones in this way

PDG-based clone detection example

Fragment 1	Fragment 2

`while (isalpha(c) || c == '_' || c == '-') {`
`++ if (p == token_buffer + maxtoken)`
`++ p = grow_token_buffer(p);`
`if (c == '-') c = '_';`
`++ *p++ = c;`
`++ c = getc(finput);`
`}`
	`while (isdigit(c)) {`
	`++ if(p == token_buffer + maxtoken)`
	`++ p = grow_token_buffer(p);`
	`numval = numval*20 + c - '0';`
	`++ *p++ = c;`
	`++ c = getc(finput);`
	`}`

Table 3.1: Duplicated code from bison, taken from the orginal paper by Komondoor and Horwitz [9]

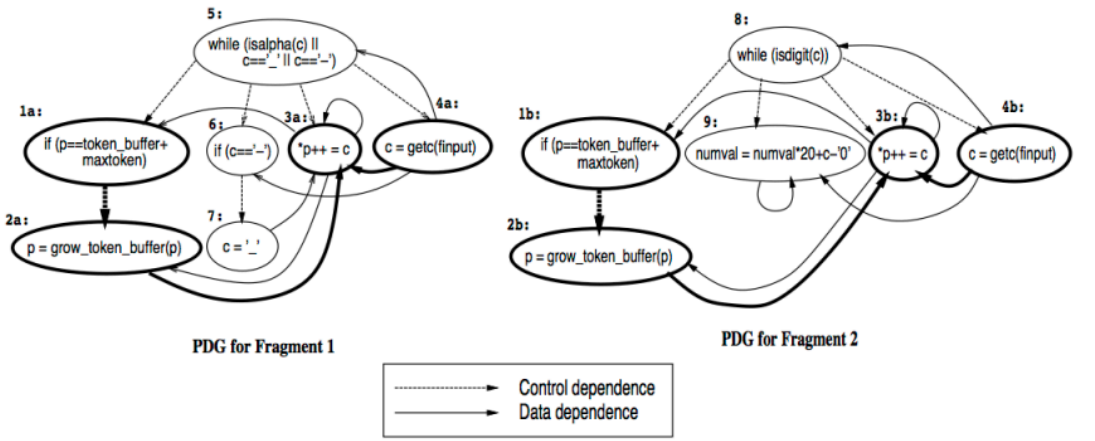


Figure 3.1: The nodes and edges in the partial slices are shown in bold. Taken from the original paper by Komondoor and Horwitz [9]

Graduating on clones

Detecting Refactorable Clones Using PDG and Program Slicing

Ammar Hamid
10463593
ammarhamid84@gmail.com

August 17, 2014, 22 pages

Supervisor: Dr. Vadim Zaytsev
Host organisation: University of Amsterdam

Other clone detection techniques

- Statistical: if fragments do similar things (e.g. calling the same methods) they have more chance of being cloned

- Data mining: frequent item set search

Clone visualization – file view

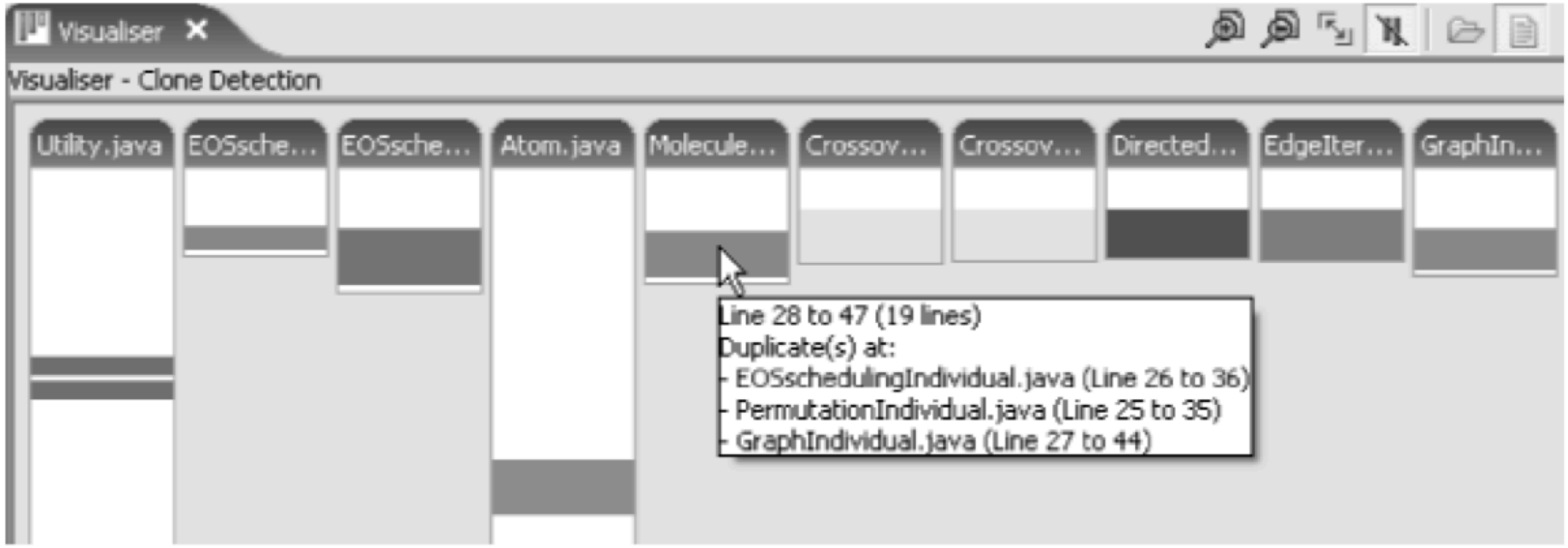
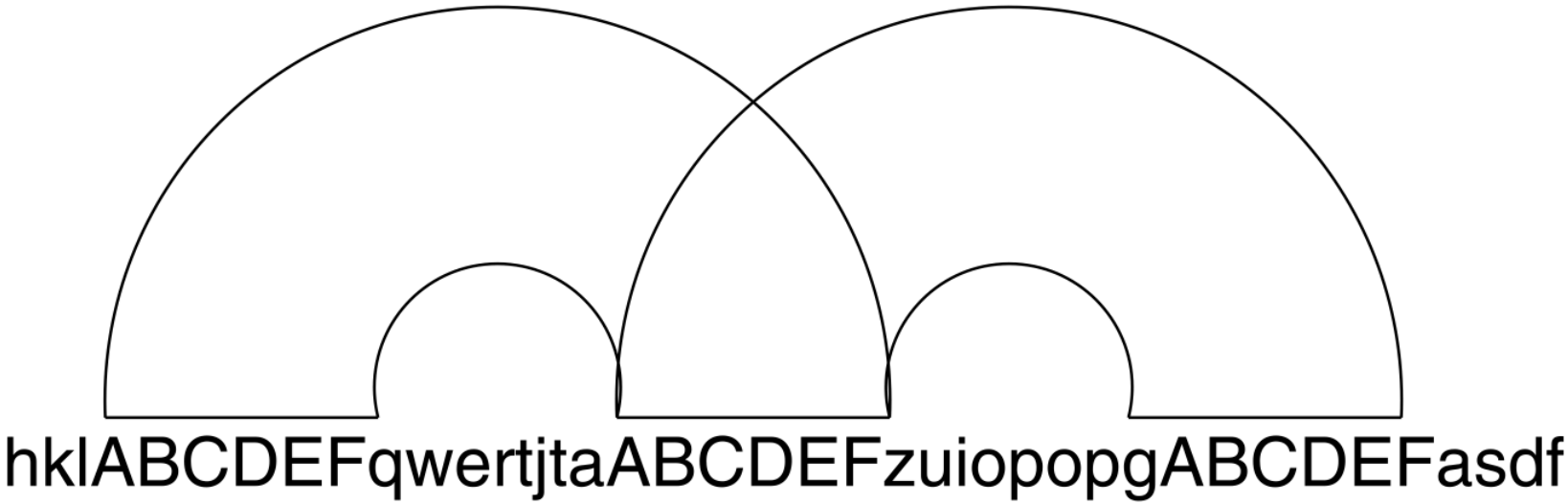


Fig. 2.8. Clones visualizer view in Eclipse adapted from [489]

Source: R. Koschke, *Identifying and Removing Software Clones,* in Software Evolution, Springer, 2008.

Clone visualization – arc diagrams



Source: R. Koschke, *Identifying and Removing Software Clones,* in Software Evolution, Springer, 2008.

Clone visualization – friend wheel

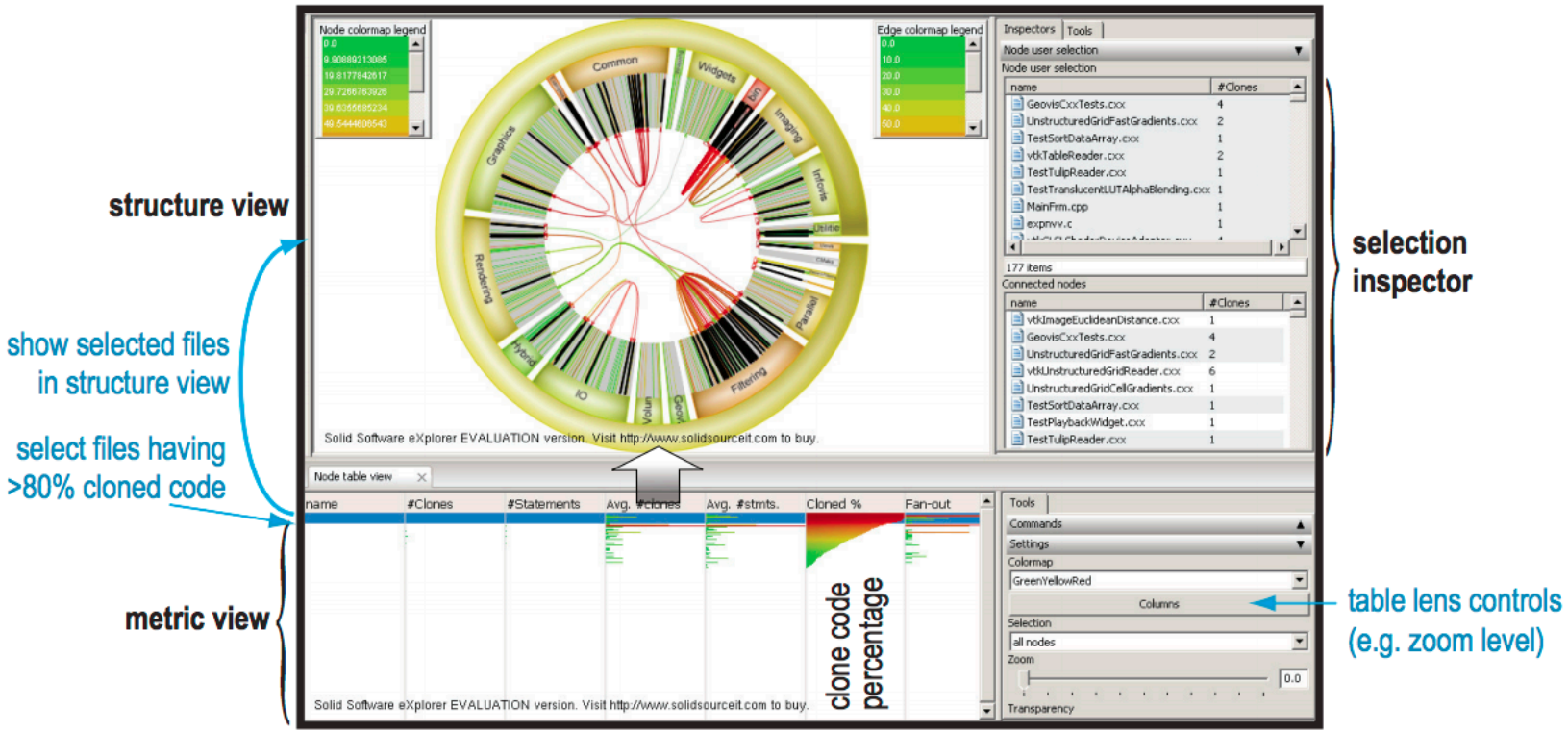


Figure 2. SolidSDD metric view (bottom) and structure view (top) with high-clone percentage files highlighted (see Sec. II).

Source: L. Voinea and A. Telea, *Visual Clone Analysis with SolidSDD,* in Proceedings of IEEE VISSOFT, 2014.

Clone visualization – tree maps

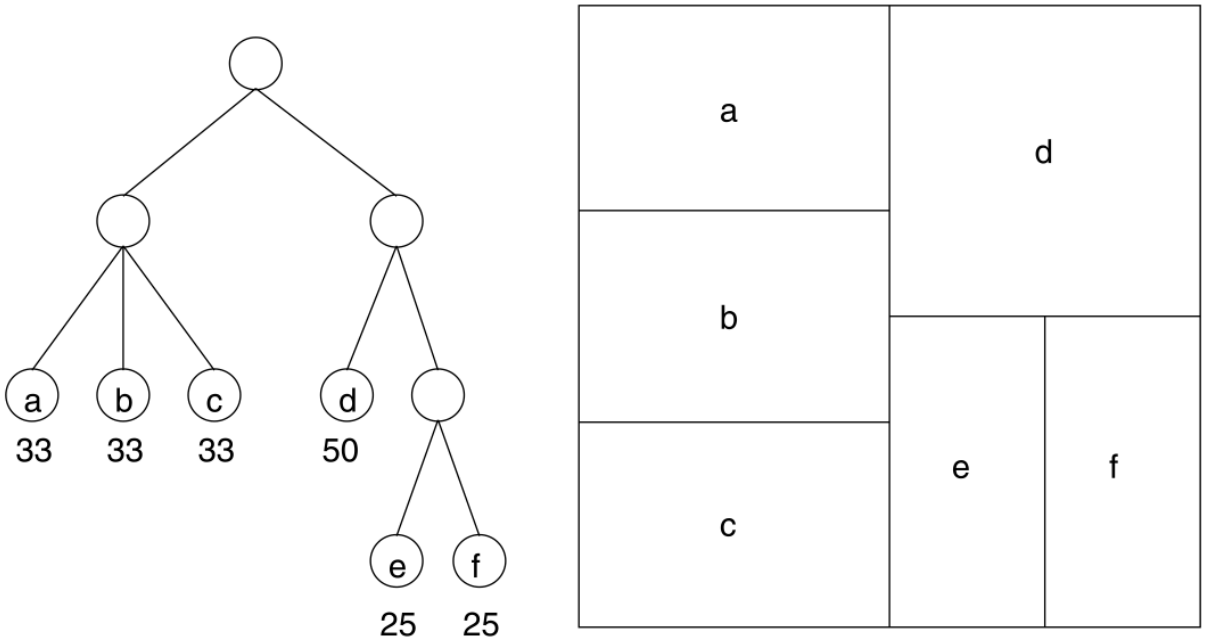


Fig. 2.6. A system decomposition whose leaves are annotated with the number of cloned lines of code and its corresponding tree map

Source: R. Koschke, *Identifying and Removing Software Clones,* in Software Evolution, Springer, 2008.

Clone visualization – polymetric views

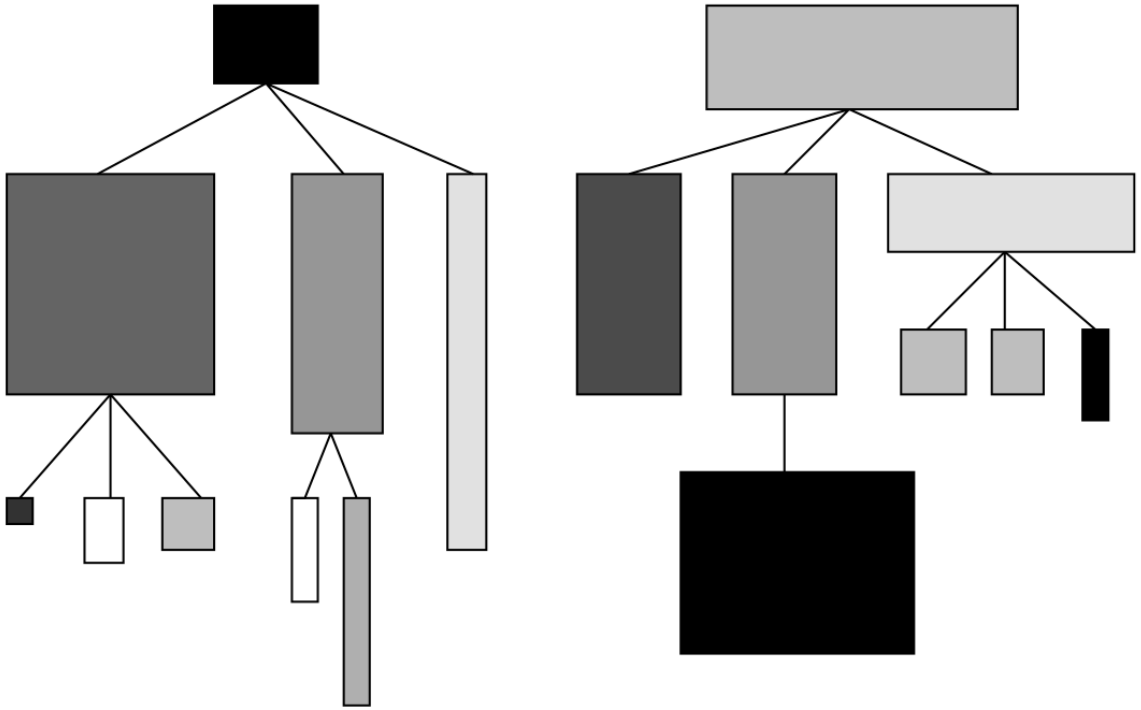
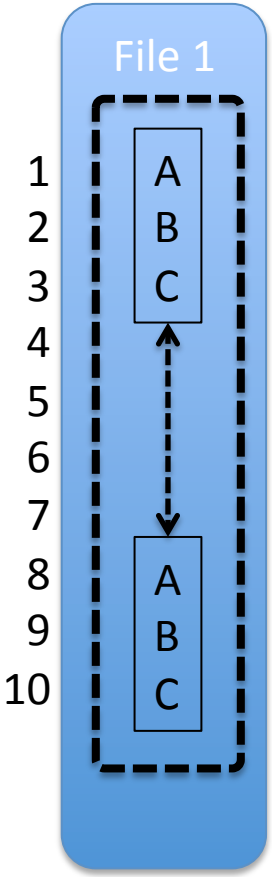


Fig. 2.5. Polymetric view adapted from [437]

Source: R. Koschke, *Identifying and Removing Software Clones,* in Software Evolution, Springer, 2008.

Clone class subsumption



Assuming Type 1 or 2

Clones:

1. File 1: 1-3
2. File 1: 8-10

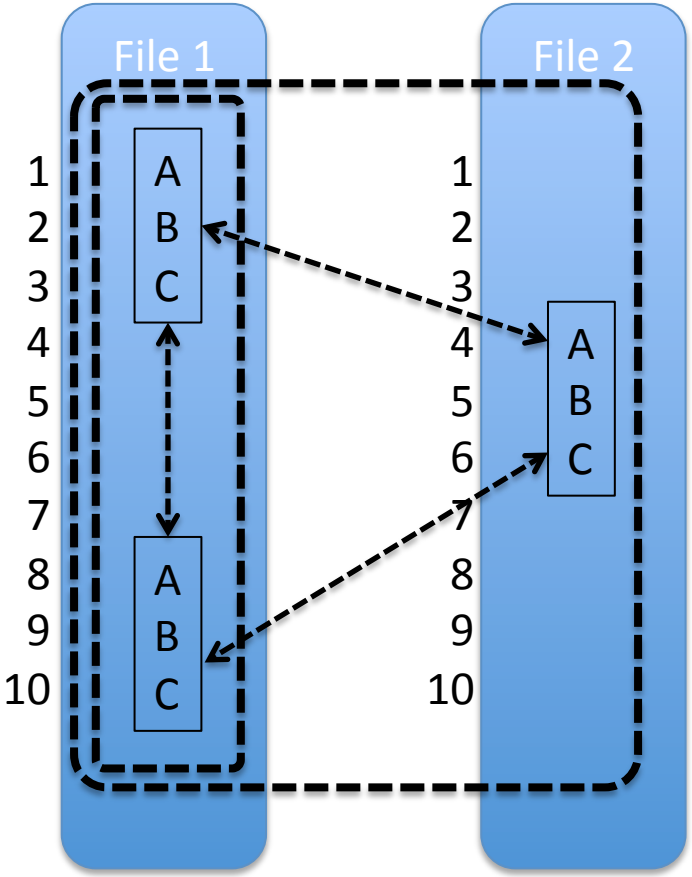
Clone pairs:

1. File 1: 1-3 <-> File 1: 8-10

Clone classes:

1. File 1: 1-3, File 1: 8-10

Clone class subsumption



Assuming Type 1 or 2

Clones:

1. File 1: 1-3
2. File 1: 8-10
3. File 2: 4-6

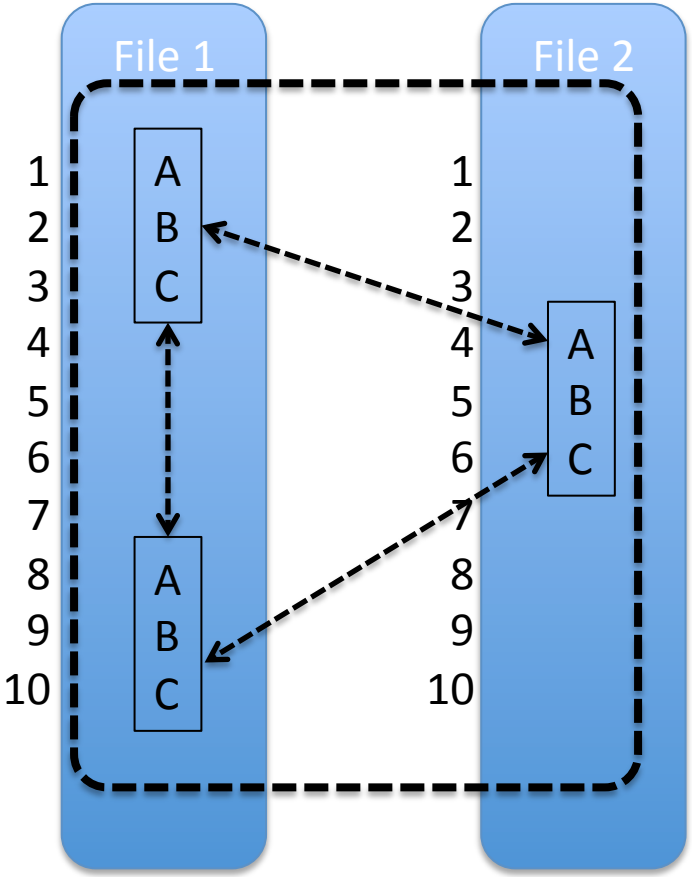
Clone pairs:

1. File 1: 1-3 <-> File 1: 8-10
2. File 1: 1-3 <-> File 2: 4-7
3. File 1: 8-10 <-> File 2: 4-6

Clone classes:

1. File 1: 1-3, File 1: 8-10
2. File 1: 1-3, File 1: 8-10, File 2: 4-6

Clone class subsumption



Assuming Type 1 or 2

Clones:

1. File 1: 1-3
2. File 1: 8-10
3. File 2: 4-6

Clone pairs:

1. File 1: 1-3 <-> File 1: 8-10
2. File 1: 1-3 <-> File 2: 4-7
3. File 1: 8-10 <-> File 2: 4-6

Clone classes:

1. ~~File 1: 1-3, File 1: 8-10~~
2. File 1: 1-3, File 1: 8-10, File 2: 4-6

The plan for today

1. Clone detection
   - Finding clones
   - Visualizing clones
2. Break (15 minutes max.)
3. Clone management
   - Preventive
   - Compensative
   - Corrective
4. Lab series 2

The plan for today

1. Clone detection
   - Finding clones
   - Visualizing clones

2. Break (15 minutes max.)

3. Clone management
   - Preventive
   - Compensative
   - Corrective

4. Lab series 2

Code clones – good, bad, or ugly?

Picture copyright chillyfraco @deviantART

Causes of cloning

?

Causes of cloning

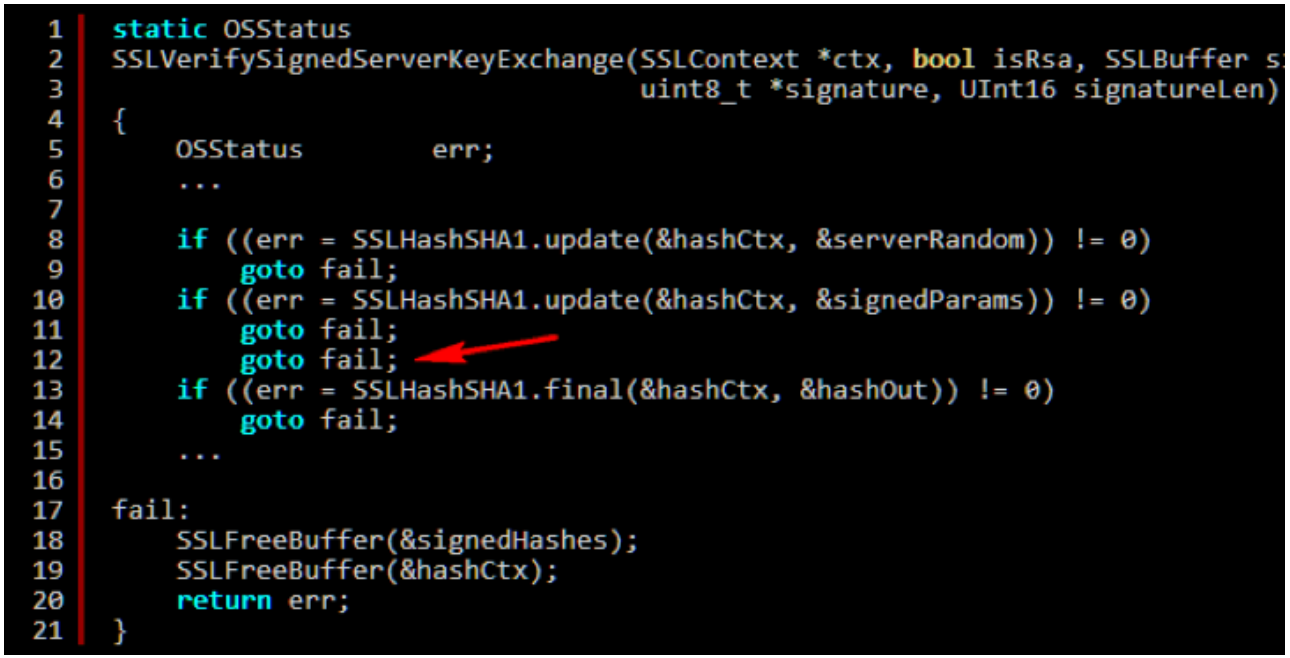
- Forced by programming language
- Forced by software design
- Delayed refactoring (due to uncertainty or time pressure)
- Idiomatic solutions or coding standards
- Templating
- Forking

Consequences of cloning

?

Consequences of cloning

- Increased maintenance effort?
- Increased productivity during development?
- Decreased risk of breaking things?



```
1  static OSStatus
2  SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer s
3                                   uint8_t *signature, UInt16 signatureLen)
4  {
5      OSStatus        err;
6      ...
7
8      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9          goto fail;
10     if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11         goto fail;
12         goto fail;
13     if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14         goto fail;
15     ...
16
17 fail:
18     SSLFreeBuffer(&signedHashes);
19     SSLFreeBuffer(&hashCtx);
20     return err;
21 }
```

Clone management

Preventive – no clones in the first place

Compensative – coping with existing clones

Corrective – fixing existing clones

Corrective clone management

Refactor clones into something “better”

- Example: refactor clone into a method and replace clones by two method calls
- More elaborate: apply aspect-oriented programming

Preventive clone management

"Disable CTRL-C CTRL-V" ;-)

While typing, or at check-in, check for clones

Prerequisites:

- Organizational support needed
- Programming technology must allow non-cloned solutions

Compensative clone management

Maintain knowledge of clones (in the IDE)

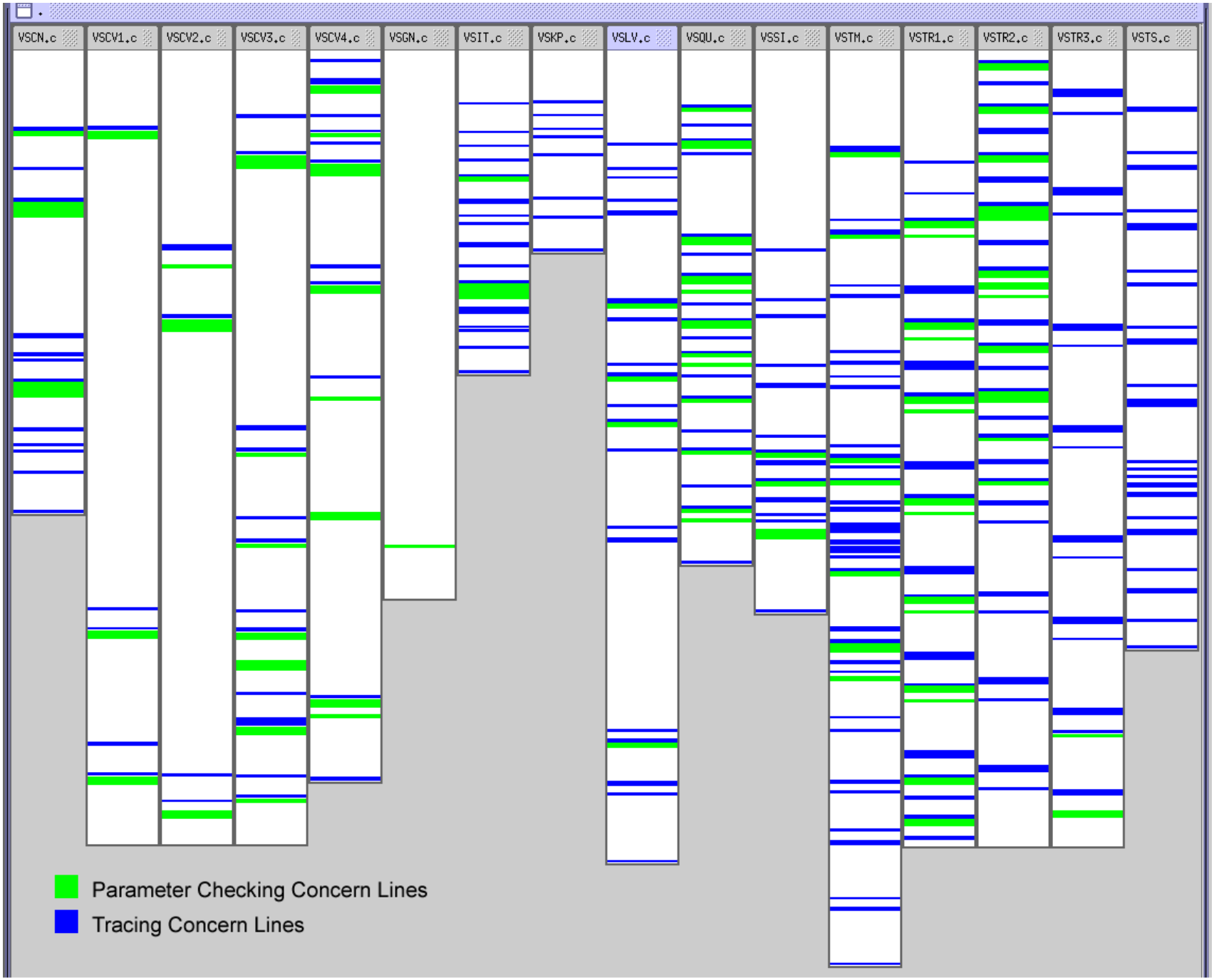
When developers change code, they are alerted that clones exist

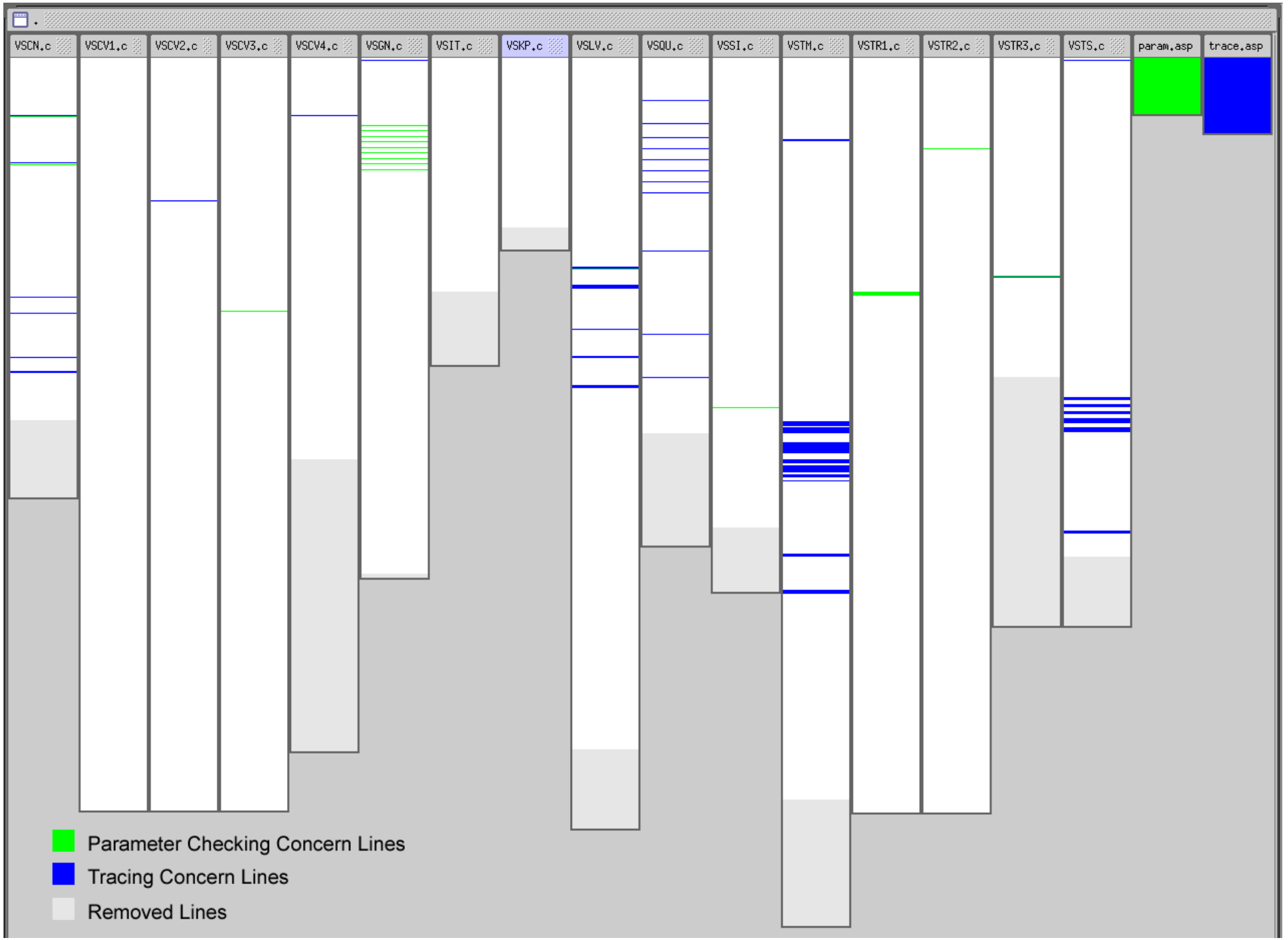
Clones can be annotated with additional knowledge, etc.

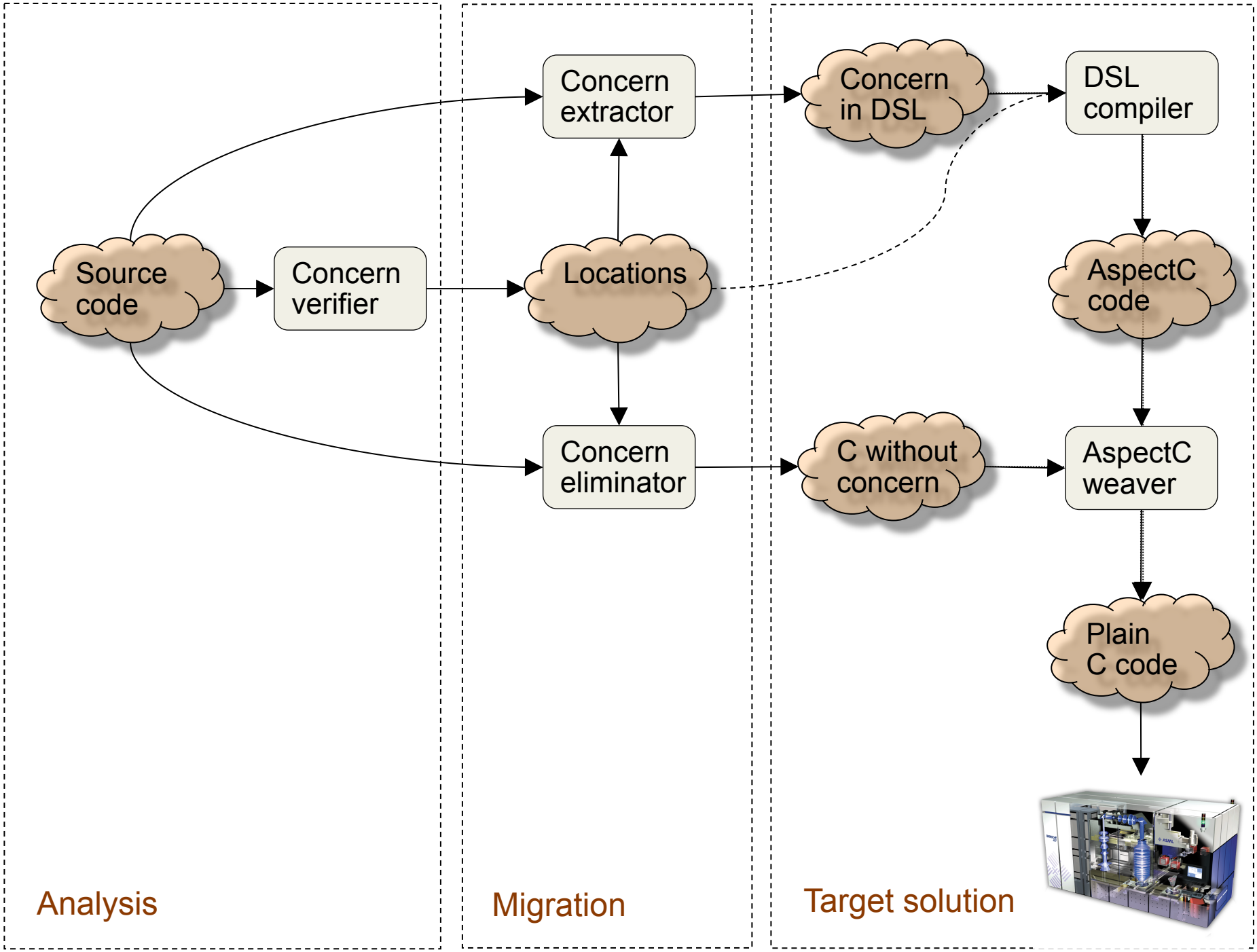
Recommended reading

- R. Koschke, *Identifying and Removing Software Clones,* in *S*oftware Evolution, Springer, 2008.

- C. Kapser, M.W. Godfrey, *"Cloning Considered Harmful" Considered Harmful*, in Proceedings of the 13th Working Conference on Reverse Engineering, 2006.

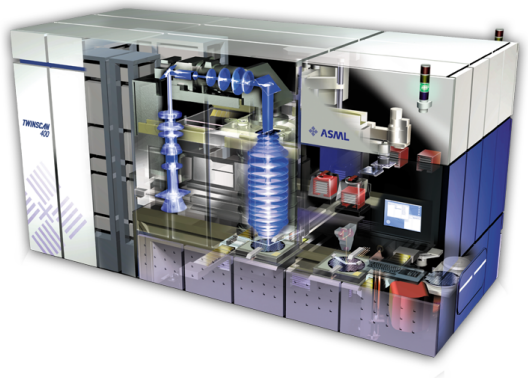

VSCN.c
VSCV1.c
VSCV2.c
VSCV3.c
VSCV4.c
VSGN.c
VSIT.c
VSKP.c
VSLV.c
VSQU.c
VSSI.c
VSTM.c
VSTR1.c
VSTR2.c
VSTR3.c
VSTS.c
Parameter Checking Concern Lines
Tracing Concern Lines


VSCN.c
VSCV1.c
VSCV2.c
VSCV3.c
VSCV4.c
VSGN.c
VSIT.c
VSKP.c
VSLV.c
VSQU.c
VSSI.c
VSTM.c
VSTR1.c
VSTR2.c
VSTR3.c
VSTS.c
param.asp
trace.asp
Parameter Checking Concern Lines
Tracing Concern Lines
Removed Lines


Concern extractor
Concern in DSL
DSL compiler
Source code
Concern verifier
Locations
AspectC code
Concern eliminator
C without concern
AspectC weaver
Plain C code
Analysis
Migration
Target solution

Using Clone Detection for Concern Code Identification

Research results from Ideals project
Cooperation with ASML



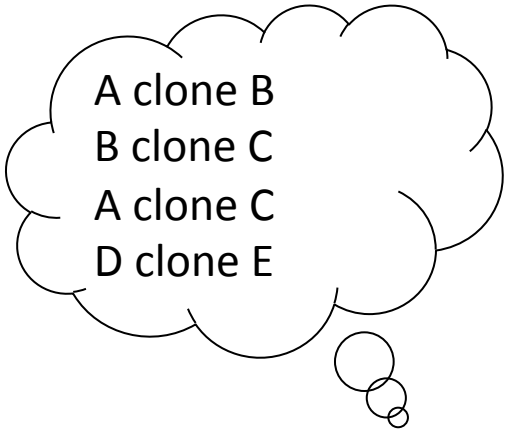
Why Clone Detection?

- Misalignment problem: some concerns do not fit the current decomposition.
- Concerns are implemented repeatedly throughout the system: *scattering* and *tangling*!
- Implementations are likely very similar; copy-paste-adapt, or idiomatic.

Clone Detection

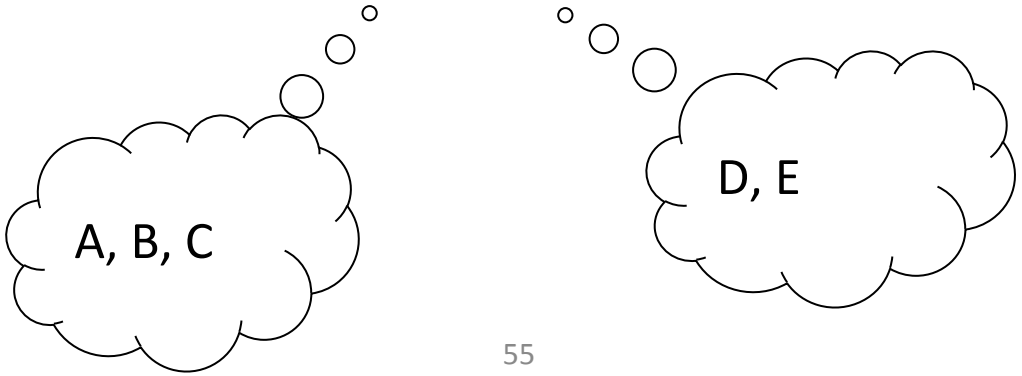
- First experiment using Bauhaus.
- AST-based clone detection.
- Different types of clones:
  - Type 1: Verbatim, modulo layout/comments,
  - Type 2: Identifiers ignored,
  - Type 3: Larger differences allowed.
- We consider type 1 and 2 clones.

Clone Classes



- Clone Pairs → Equivalence Relation.

- Clone Classes: Equivalence Classes.



Concern Code

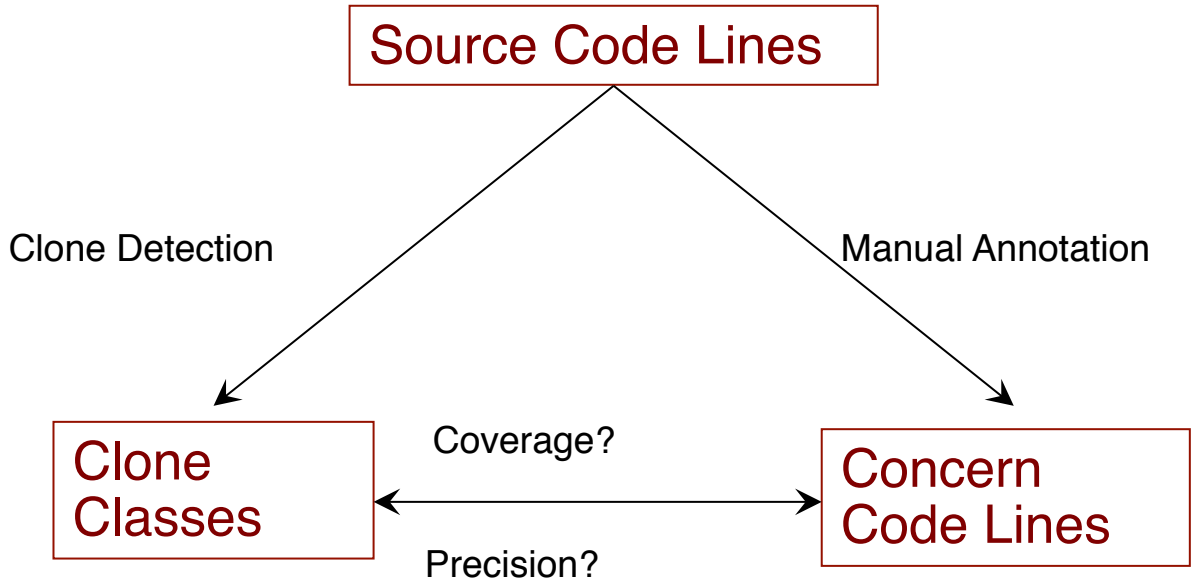
- VS : component for data conversion
- 19 KLOC, 11 files.

Concern	LOC	%
Tracing	1539	8%
Error Handling	1716	9%
Memory Error Handling	965	5%
Parameter Checking	1441	7%
Total	5561	29%

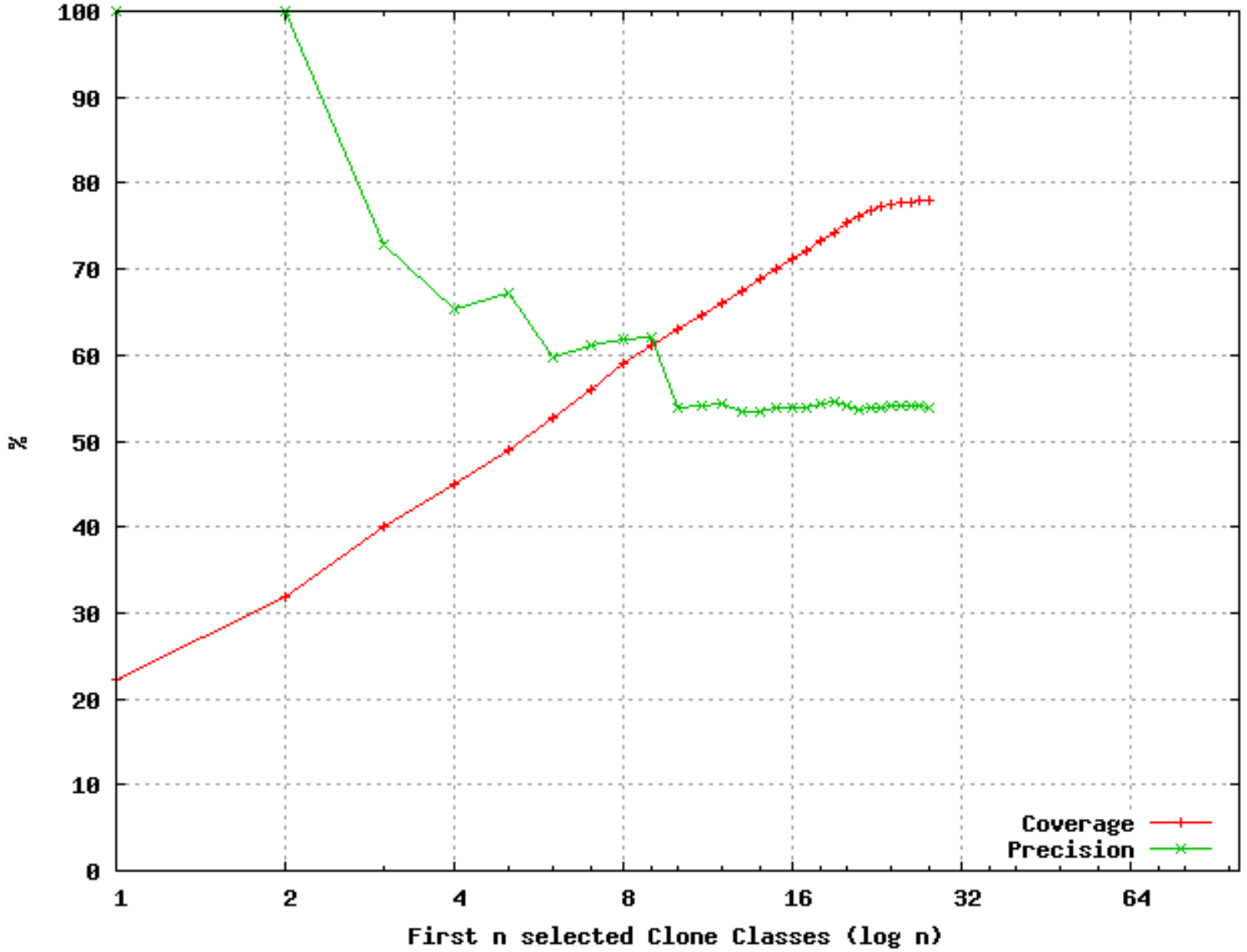
Clone Detection Results

- 6244 clones.
- 12231 LOC involved in clones.
- 703 clone classes.
- Biggest clone class: 523 lines (79 clones)
- Average size of clone class: 65,5 lines.

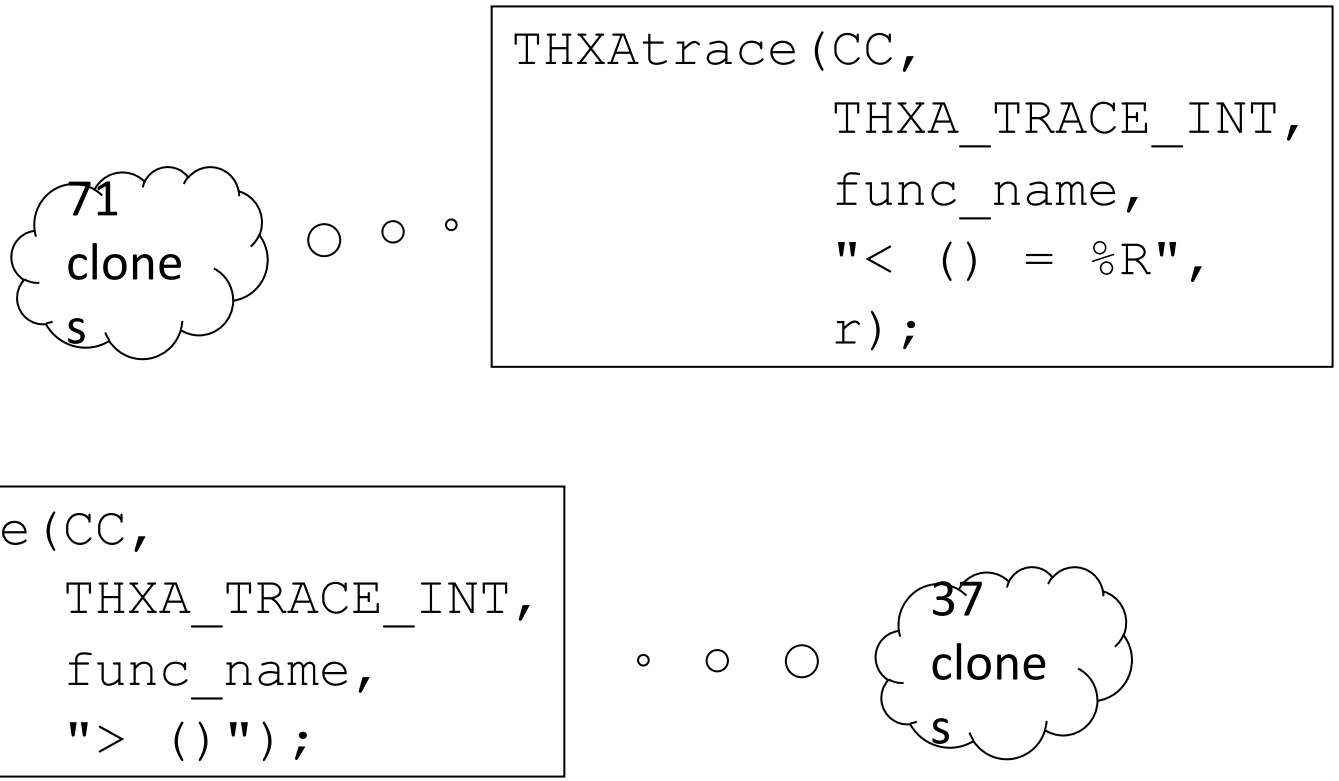
Concern Code Coverage



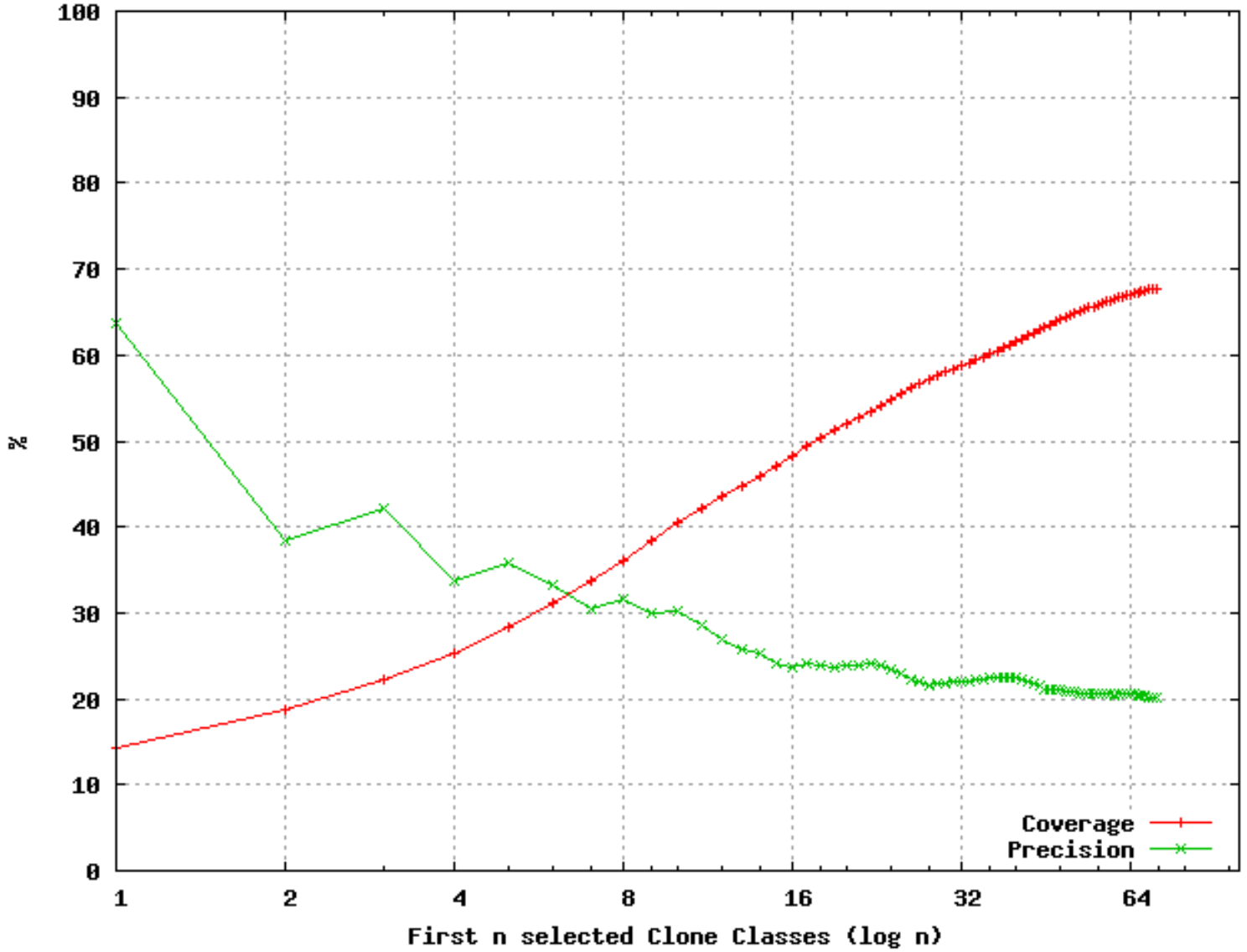
Tracing (1539 LOC)



Tracing (1539 LOC)



Error Handling (1716 LOC)



Error Handling (1716 LOC)

```
int  r = OK;
char *func_name = "VSCN_WriteMsg";
```

314 clones

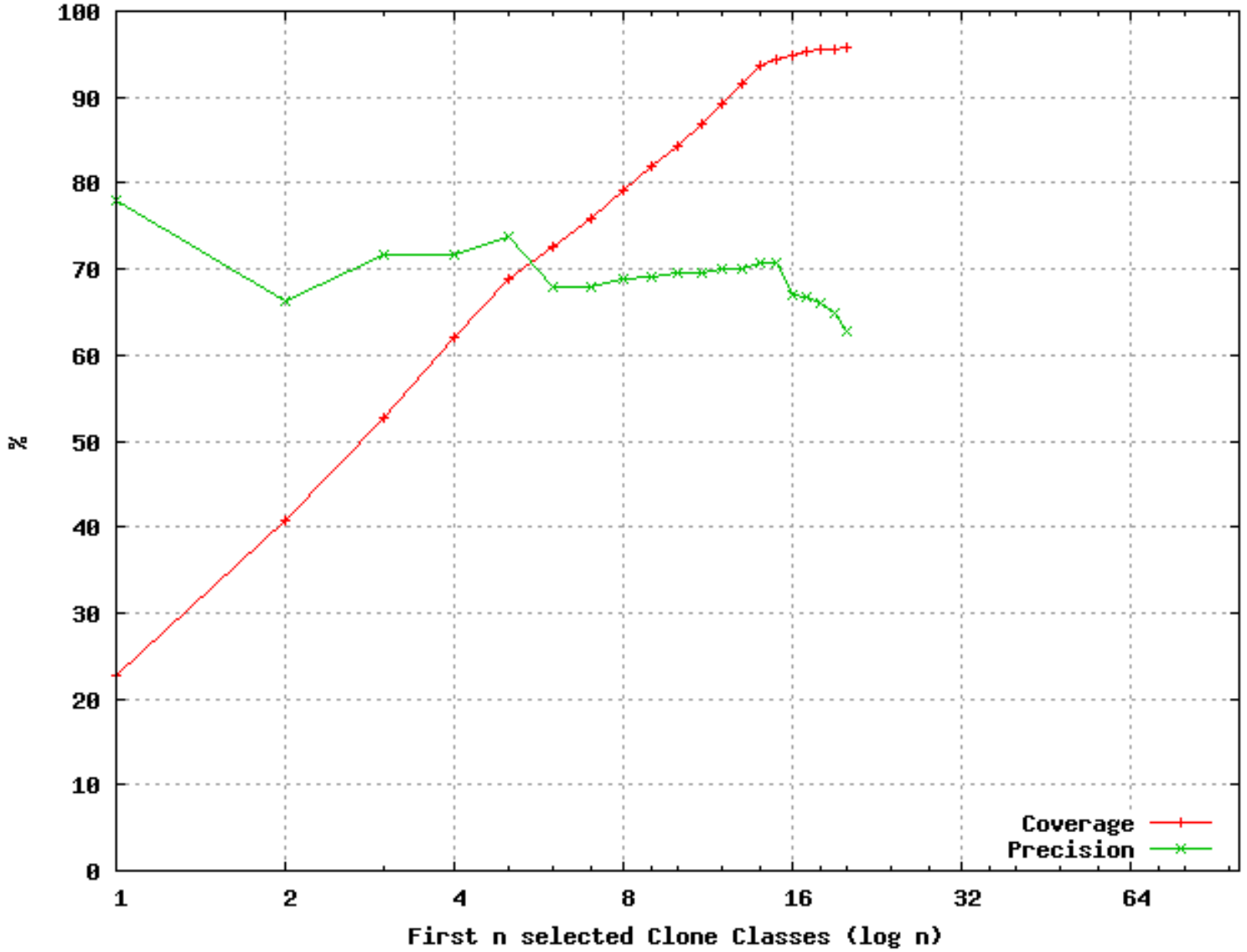
```
if (r == OK)
{
    r = VSTR_initialise();
}
```

17 clones

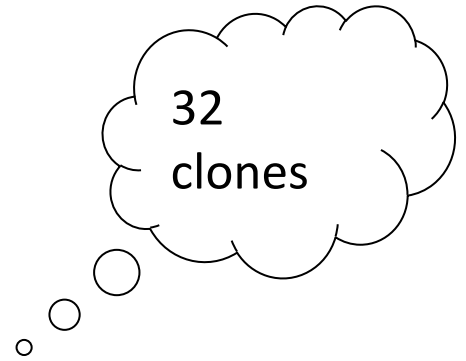
```
{
    r = VSXA_CN_OS_ERR_WRITE_ERR;
    ERXA_LOG(r, 0,
             ("%s: write() failed: %s",
             func_name,
             strerror(os_error)));
}
```

```
return r;
```

Memory Error Handling (965 LOC)

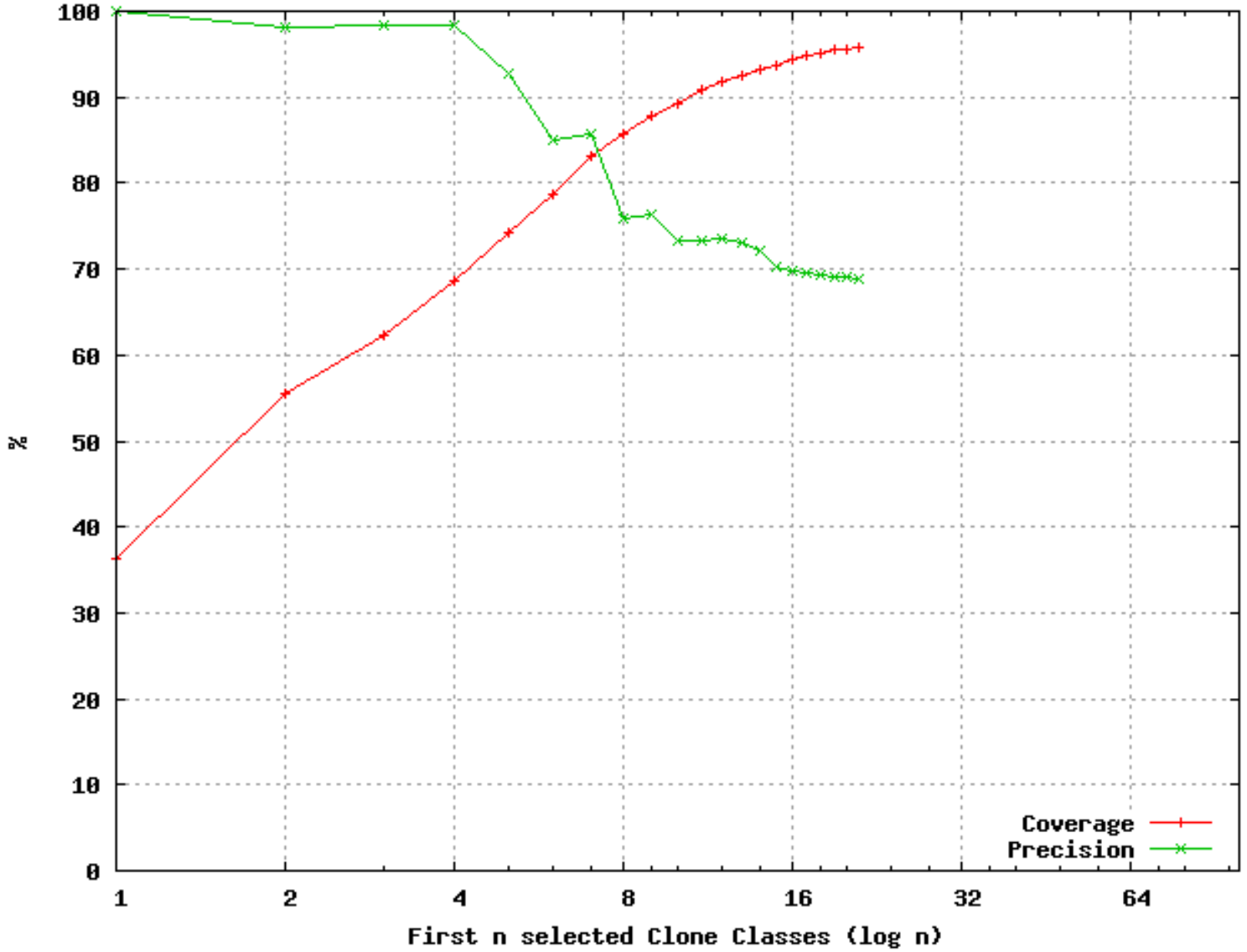


Memory Error Handling (965 LOC)

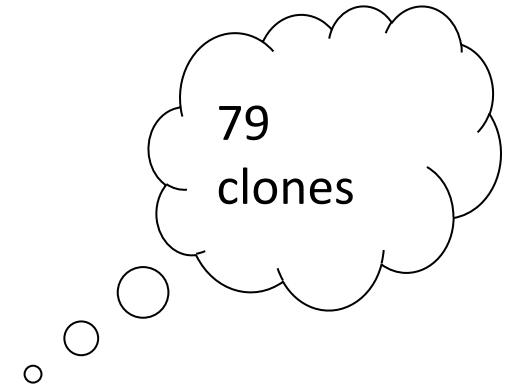



```
if (r != OK)
{
    ERXA_LOG(r, 0, ("PLXAmem_free failed"));
    ERXA_LOG(VSXA_MEMORY_ERR, r,
        ("%s: Memory failure with debug buffer ",
        "(ignored).",
        func_name));
    r = VSXA_MEMORY_ERR;
}
```

Parameter Checking (1441 LOC)



Parameter Checking (1441 LOC)




```
if ((r == OK) && (msg == (void *) NULL))
{
    r = VSXA_PARAMETER_ERR;
    ERXA_LOG(r, 0,
        ("%s: input parameter '%s' is NULL.",
        func_name,
        "msg"));
}
```

Summary

- Parameter checking and memory handling code is covered well.
- Identification of (general) error handling and tracing code is more limited by the clone detection technique.

The plan for today

1. Clone detection
   - Finding clones
   - Visualizing clones

2. Break (15 minutes max.)

3. Clone management
   - Preventive
   - Compensative
   - Corrective

4. Lab series 2

Assignment: Lab series 2

Objective:

- Design and build a clone management tool in Rascal

Collaboration:

- in pairs (e.g. same as Lab 1) or individual

Deliverables:

- Working clone detector in Rascal
- Visualization of detected clones

Grading is performed in an interactive session

You need to self-study the Rascal visualization library (see BB) and some literature on building tools for maintainers/developers

Assignment: Lab series 2 (1/2)

Working prototype implementation:

1. Clone detector that detects at least Type 1 clones in a Java project:
   - The detected clone classes are written to file in a textual representation.
   - Clone classes that are strictly included in others are dropped from the results (subsumption).

2. Report of cloning statistics showing at least:
   - % of duplicated lines, number of clones, number of clone classes, biggest clone (in lines), biggest clone class, and example clones.

3. At least one insightful vizualization of the cloning in a project.

4. Convincing testing harnass that ensures your clone detector works.

Assignment: Lab series 2 (2/2)

To score a higher grade than the base grade (7), additionally do:

- Also detect Type 2, Type 3, and Type 4 clone classes.

See BB for full grading details.