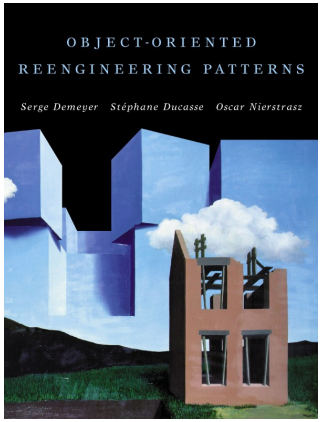
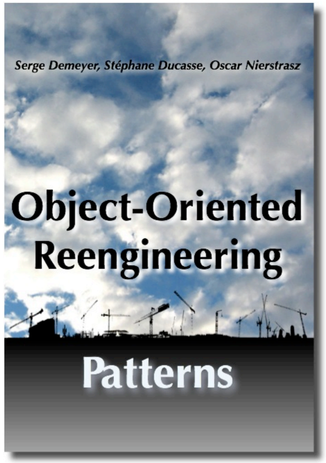
Software Reengineering & Evolution




Serge Demeyer, Stéphane Ducasse, Oscar Nierstrasz
Object-Oriented Reengineering
Patterns

Serge Demeyer

Stéphane Ducasse

Oscar Nierstrasz

January 2019

http://scg.unibe.ch/download/oorp/

Schedule

1. **Introduction**
   There are OO legacy systems too !
2. **Reverse Engineering**
   How to understand your code
3. **Visualization**
   Scalable approach
4. **Dynamic Analysis**
   To be really certain
5. **Restructuring**
   How to Refactor Your Code
6. **Code Duplication**
   The most typical problems
7. **Software Evolution**
   Learn from the past
8. **Going Agile**
   Continuous Integration
9. **Conclusion**

Goals

We will try to convince you:

- Yes, Virginia, there are *object-oriented legacy systems* too!
- Reverse engineering and reengineering are *essential activities* in the lifecycle of any successful software system. (And especially OO ones!)
- There is a large set of *lightweight tools and techniques* to help you with reengineering.
- Despite these tools and techniques, *people must do job* and they represent the most valuable resource.

What is a Legacy System ?

“legacy”

> *A sum of money, or a specified article, given to another by will; anything handed down by an ancestor or predecessor.*
> — *Oxford English Dictionary*

A **legacy system** is a piece of software that:

- you have *inherited*, and
- is *valuable* to you.

Typical **problems** with legacy systems:

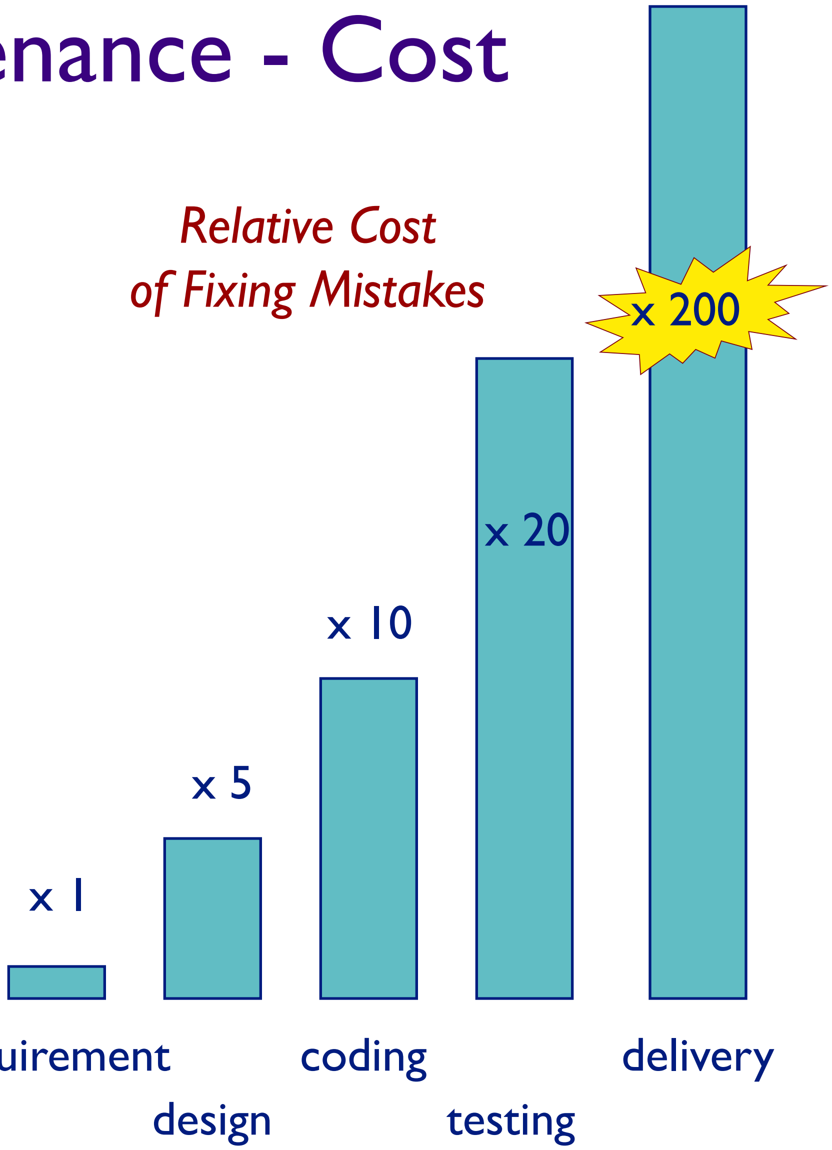
- original developers *not available*
- *outdated* development methods used
- extensive patches and *modifications* have been made
- *missing* or outdated documentation

⇒ *so, further evolution and development may be prohibitively expensive*

Software Maintenance - Cost

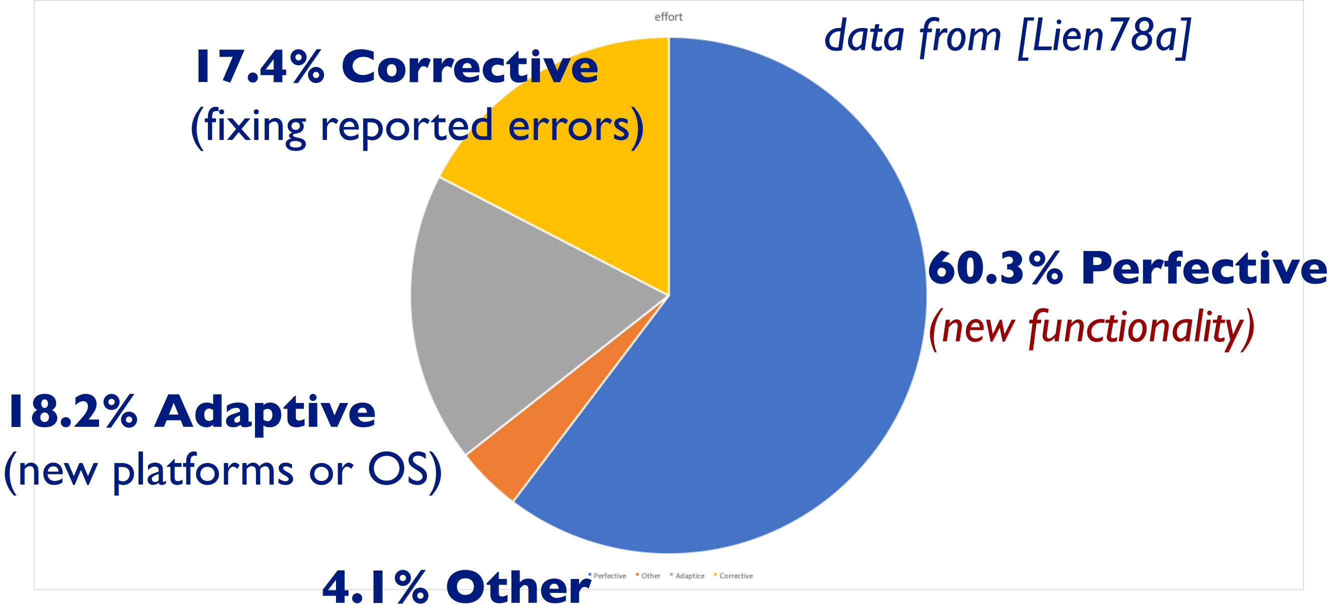
Relative Maintenance Effort
Between 50% and 75% of global effort is spent on "maintenance" !

Relative Cost of Fixing Mistakes



Solution ?
- Better requirements engineering?
- Better software methods & tools (database schemas, CASE-tools, objects, components, …)?

Continuous Development



data from [Lien78a]

The bulk of the maintenance cost is due to *new functionality*
⇒ even with better requirements, it is hard to predict new functions

Modern Methods & Tools ?

[Glas98a] quoting empirical study from Sasa Dekleva (1992)

- Modern methods(*) lead to more reliable software
- Modern methods lead to less frequent software repair
- and ...
- Modern methods lead to more total maintenance time

Contradiction ? *No!*

- modern methods make it easier to change
  ... this capacity is used to enhance functionality!

(*) process-oriented structured methods, information engineering, data-oriented methods, prototyping, CASE-tools – not OO !

Lehman's Laws

A classic study by Lehman and Belady [Lehm85a] identified several "laws" of system change.

Continuing change

- A program that is used in a real-world environment *must change*, or become progressively less useful in that environment.

Increasing complexity

- As a program evolves, it becomes *more complex*, and extra resources are needed to preserve and simplify its structure.

Those laws are still applicable…

What about Objects ?

Object-oriented legacy systems

- = successful OO systems whose architecture and design no longer responds to changing requirements

Compared to traditional legacy systems

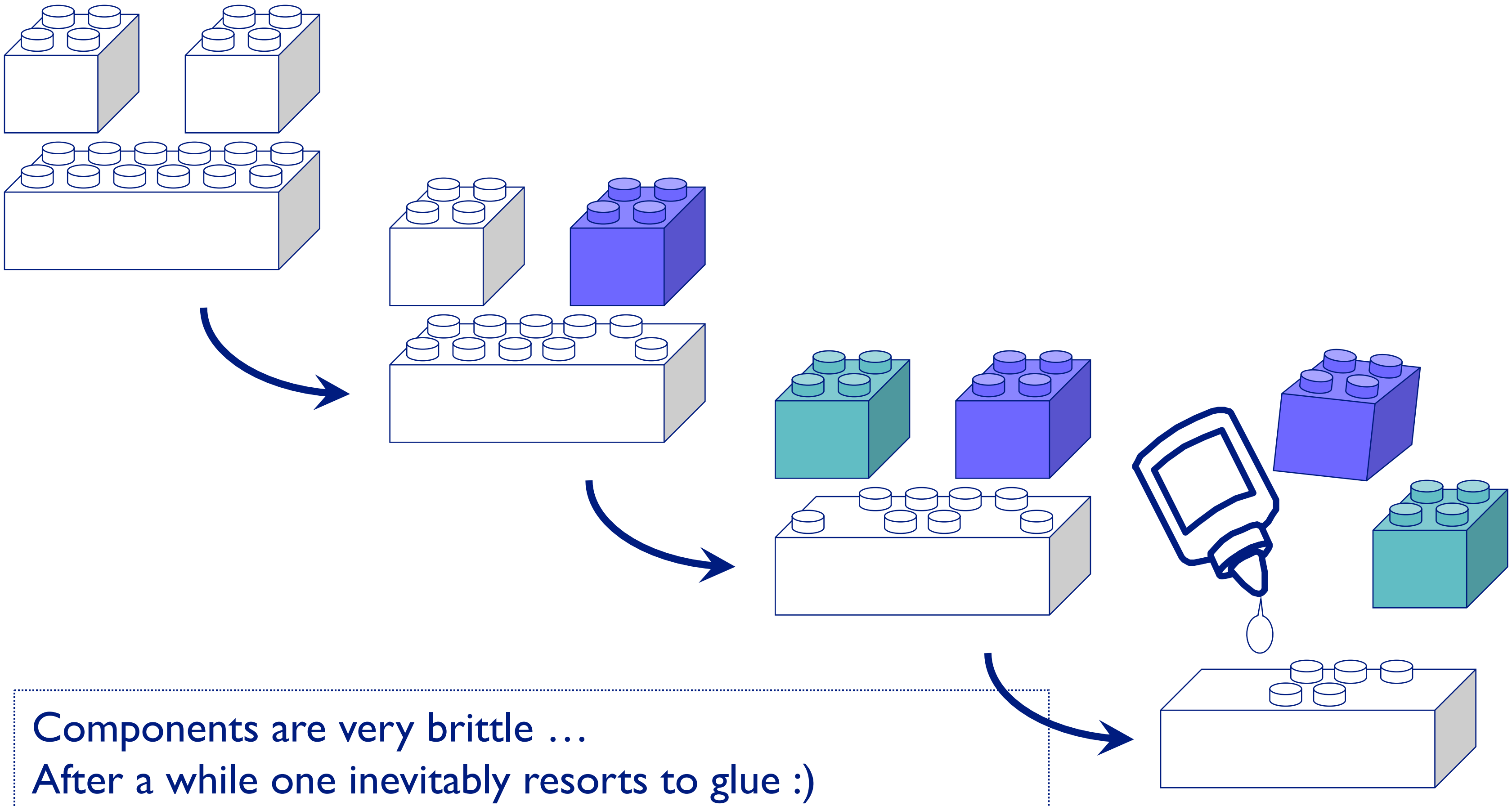
- The *symptoms* and the source of the problems are the *same*
- The *technical details* and solutions may *differ*

OO techniques promise better

- flexibility,
- reusability,
- maintainability
- …

⇒ *they do not come for free*

What about Components ?



Components are very brittle …
After a while one inevitably resorts to glue :)

Soccer Field Metaphor

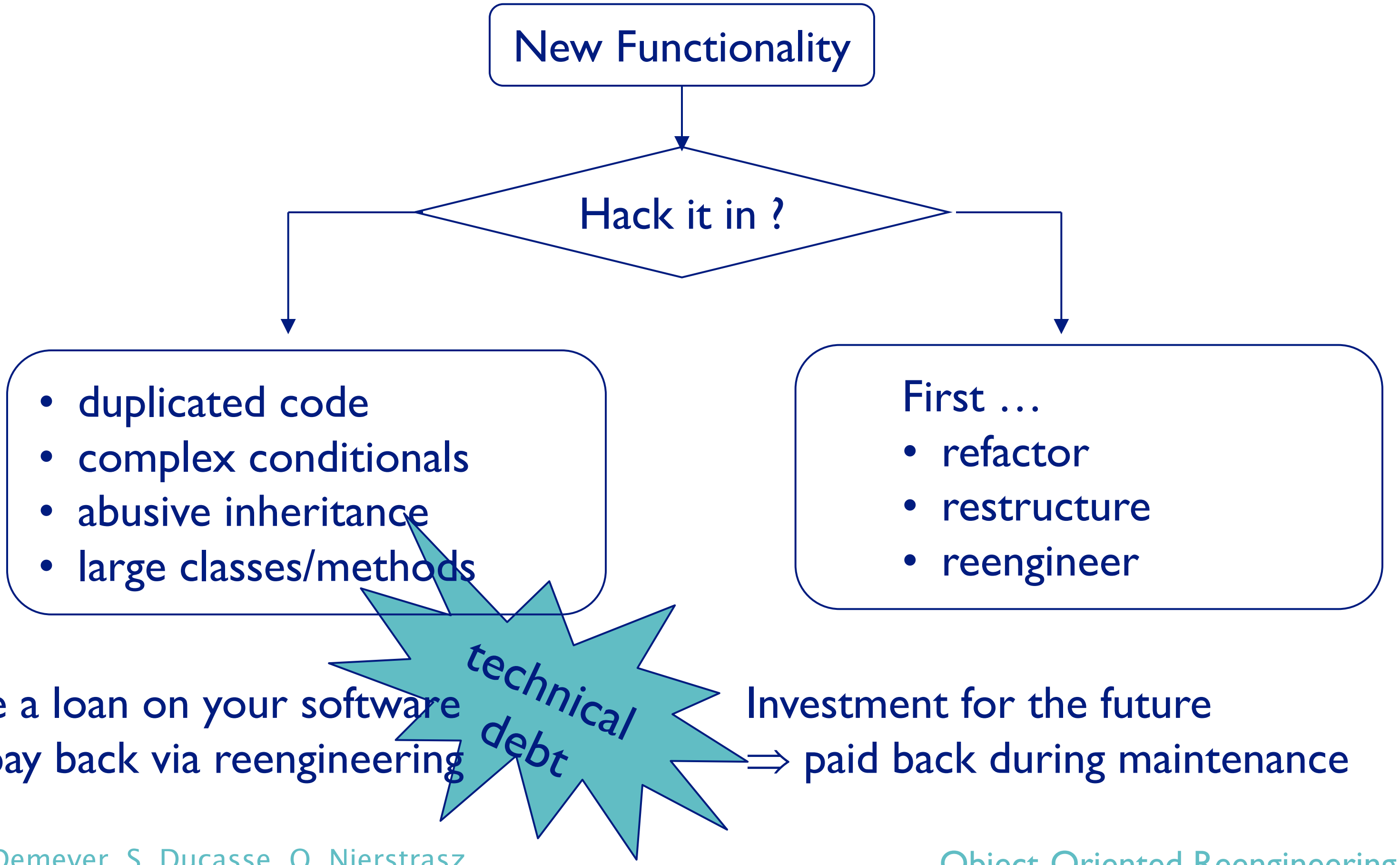
- Assume 10 lines of code
  = 40 tiles of 1 x 1 cm
- 12.5 million lines of code
  ≈ 40 soccer fields

Imagine 400 developers concurrently
moving tiles around on 40 soccer fields
…

A. van Deursen, De software-evolutieparadox
Intreerede TU Delft, 23 feb 2005

How to deal with Legacy ?

New or changing requirements will gradually degrade original design
… unless extra development effort is spent to adapt the structure



New Functionality

Hack it in ?

- duplicated code
- complex conditionals
- abusive inheritance
- large classes/methods

First …

- refactor
- restructure
- reengineer

technical debt

Take a loan on your software
⇒ pay back via reengineering

Investment for the future
⇒ paid back during maintenance

Common Symptoms

Lack of Knowledge

- *obsolete* or no documentation
- *departure* of the original developers or users
- *disappearance of inside knowledge* about the system
- *limited understanding* of entire system

⟹*missing tests*

Process symptoms

- *too long* to turn things over to production
- need for *constant bug fixes*
- *maintenance dependencies*
- *difficulties separating* products

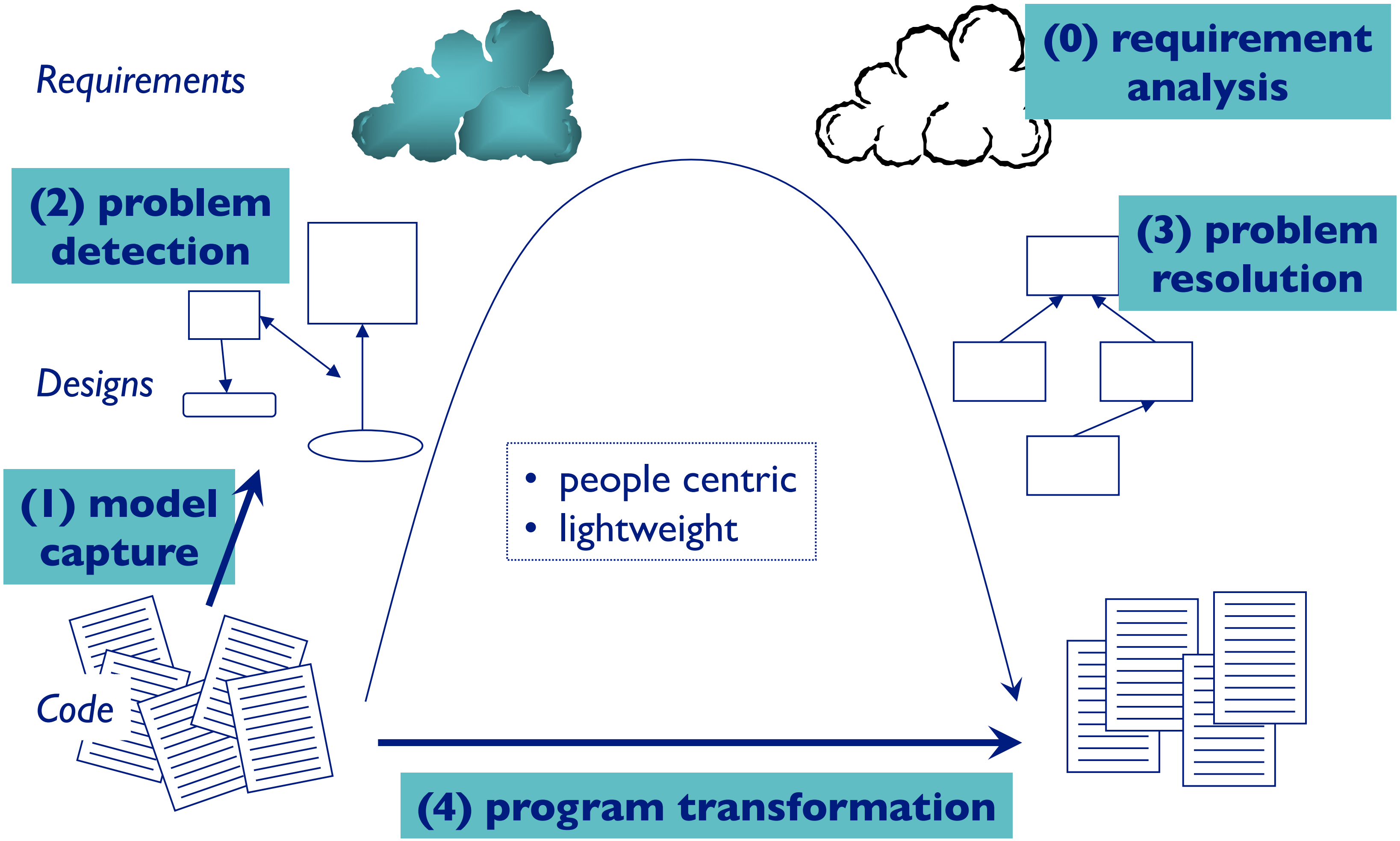
⟹*simple changes take too long*

Code symptoms

- *duplicated* code
- *code smells*

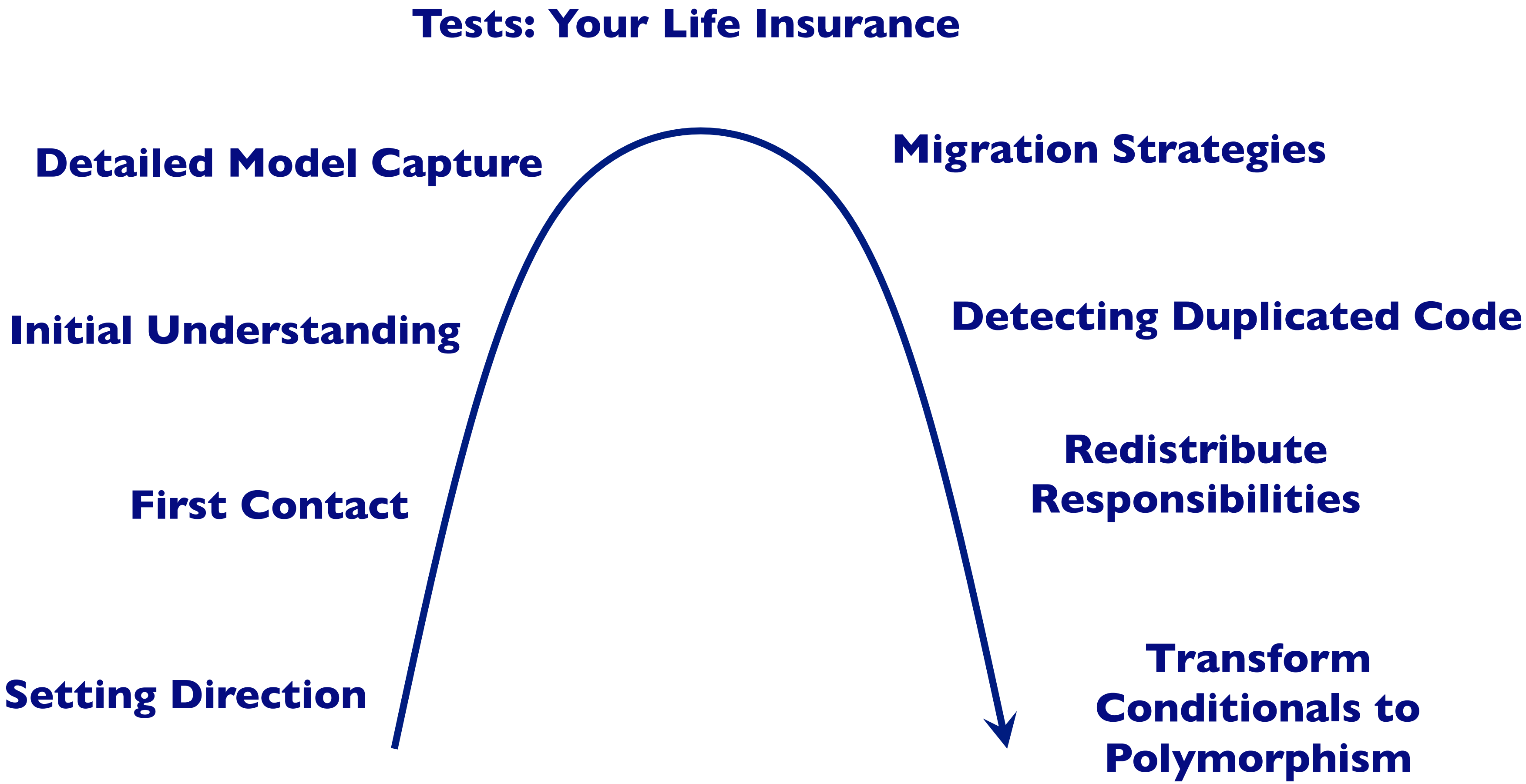
⟹*big build times*

The Reengineering Life-Cycle



- people centric
- lightweight

A Map of Reengineering Patterns



Tests: Your Life Insurance

Setting Direction

First Contact

Initial Understanding

Detailed Model Capture

Migration Strategies

Detecting Duplicated Code

Redistribute Responsibilities

Transform Conditionals to Polymorphism

2. Reverse Engineering

- What and Why
- First Contact
  - ☞ Interview during Demo
- Initial Understanding

What and Why ?

Definition

Reverse Engineering is the *process of analysing* a subject system

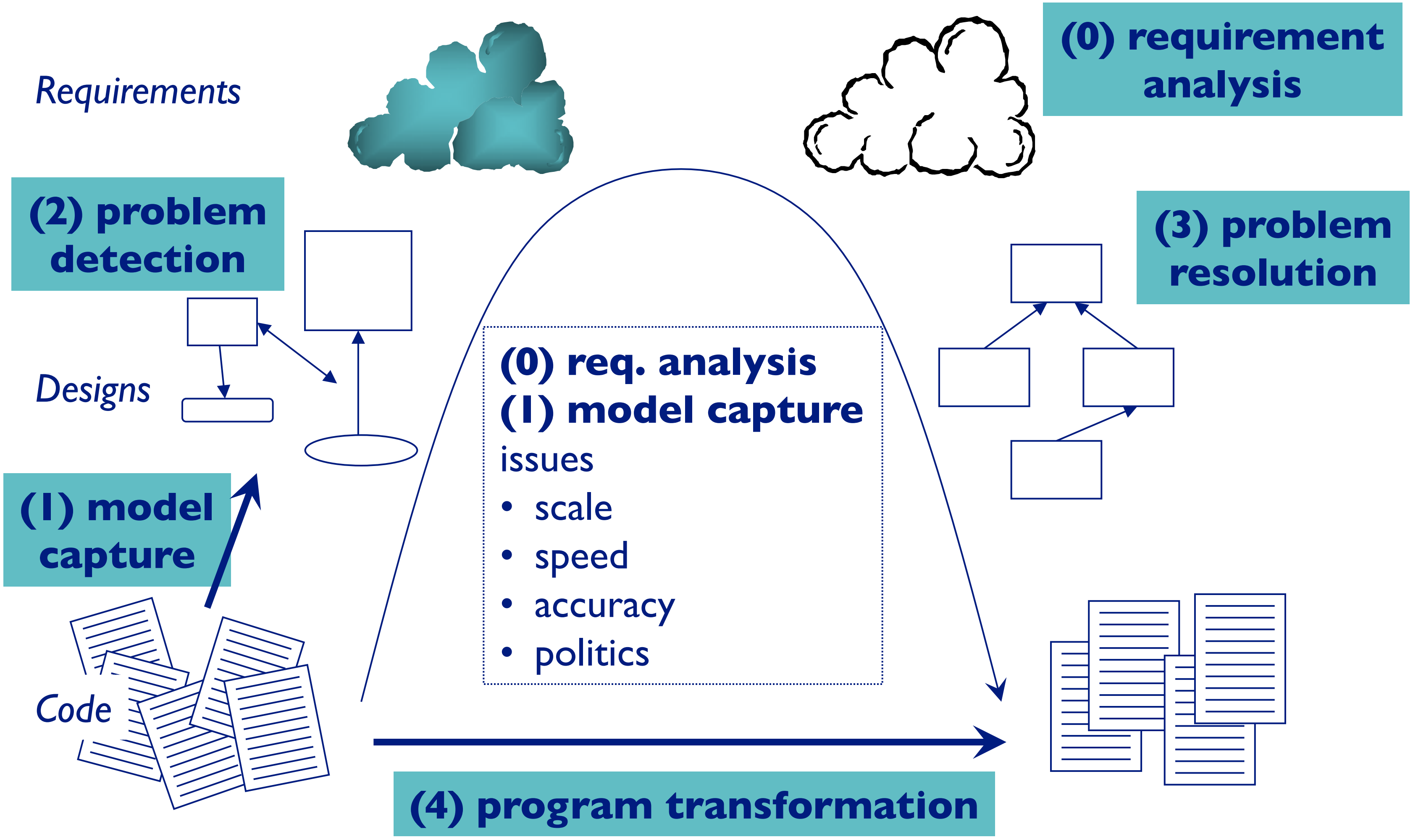
- ☞ to identify the system's components and their interrelationships and
- ☞ create representations of the system in another form or at a higher level of abstraction. — Chikofsky & Cross, '90

Motivation

Understanding other people's code
(cfr. newcomers in the team, code reviewing,
original developers left, ...)

Generating UML diagrams is NOT reverse engineering
... but it is a valuable support tool

The Reengineering Life-Cycle



Requirements

(0) requirement analysis

(2) problem detection

(3) problem resolution

Designs

(0) req. analysis
(1) model capture

issues

- scale
- speed
- accuracy
- politics

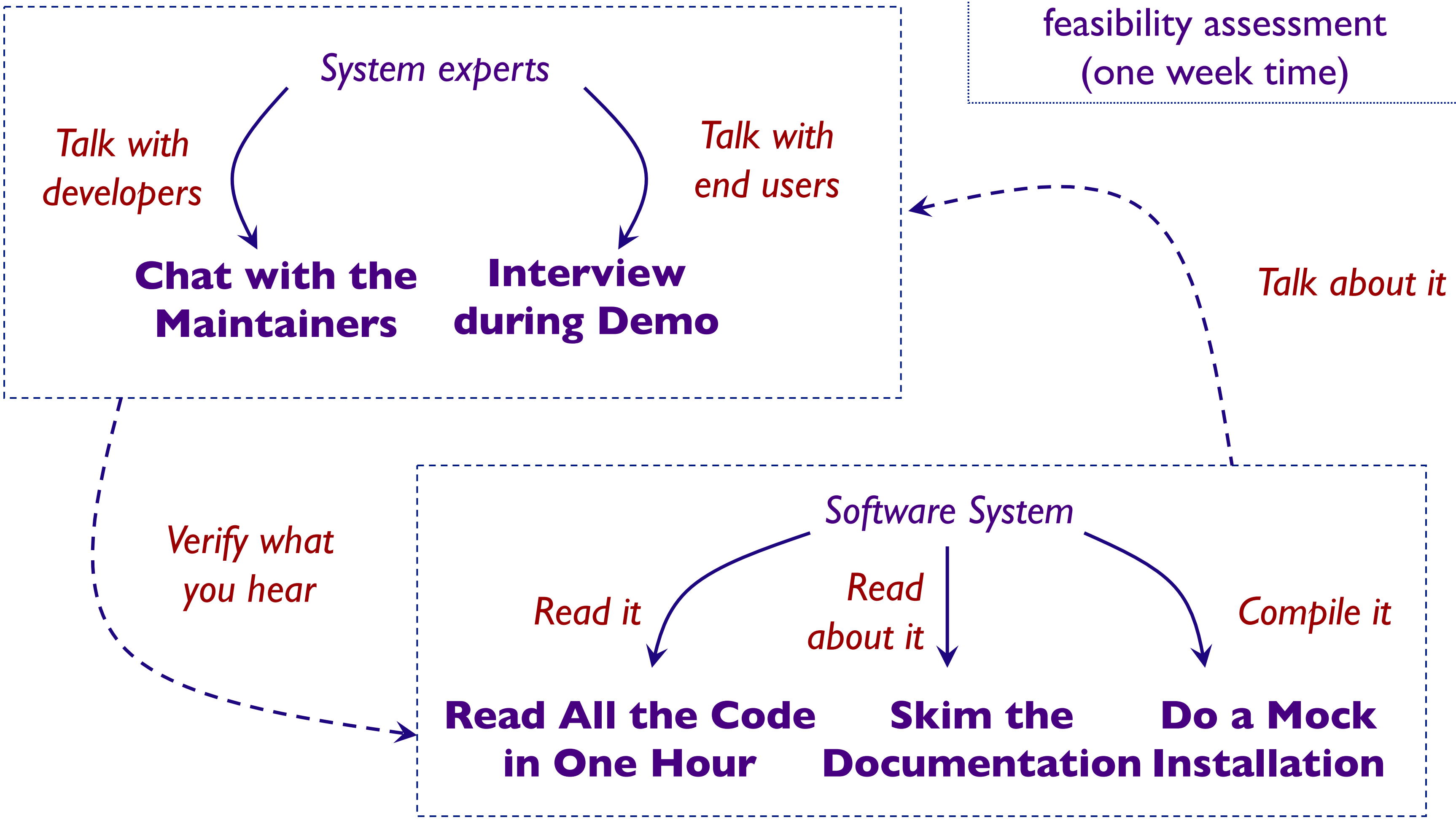
(1) model capture

Code

(4) program transformation

First Contact

feasibility assessment
(one week time)



System experts

Talk with developers

Chat with the Maintainers

Talk with end users

Interview during Demo

Talk about it

Verify what you hear

Software System

Read it

Read All the Code in One Hour

Read about it

Skim the Documentation

Compile it

Do a Mock Installation

First Project Plan

Use *standard templates*, including:

- project scope
  - ☞ see "Setting Direction"
- opportunities
  - ☞ e.g., skilled maintainers, readable source-code, documentation
- risks
  - ☞ e.g., absent test-suites, missing libraries, …
  - ☞ record likelihood (unlikely, possible, likely)
    & impact (high, moderate, low) for causing problems
- go/no-go decision
- activities
  - ☞ fish-eye view

Interview during Demo

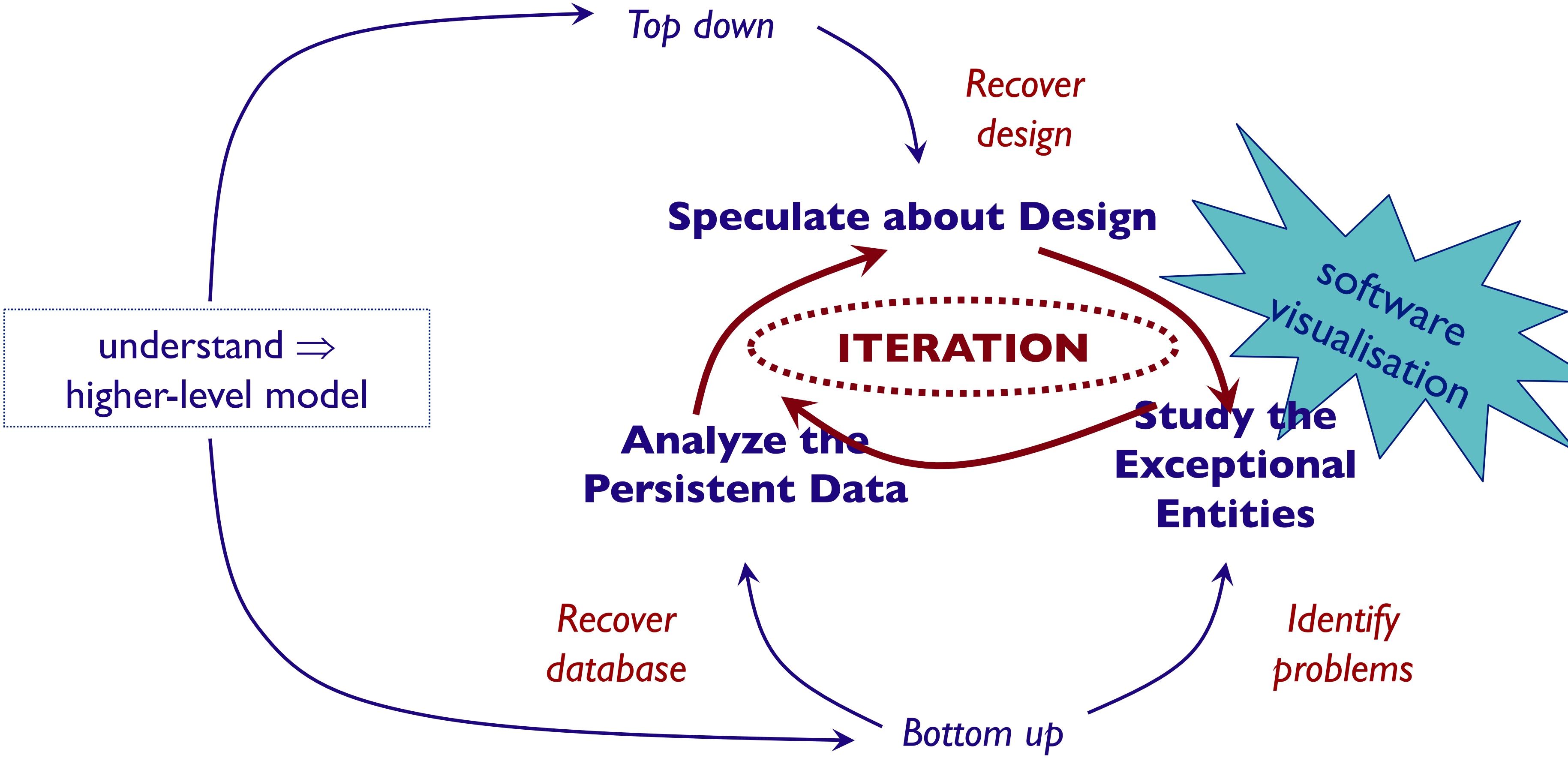
Problem: What are the typical usage scenarios?

Solution: Ask the user!

- Solution: interview during demo
  - select several users
  - demo puts a user in a positive mindset
  - demo steers the interview

- ... however
  - ☞ Which user ?
  - ☞ Users complain
  - ☞ What should you ask ?

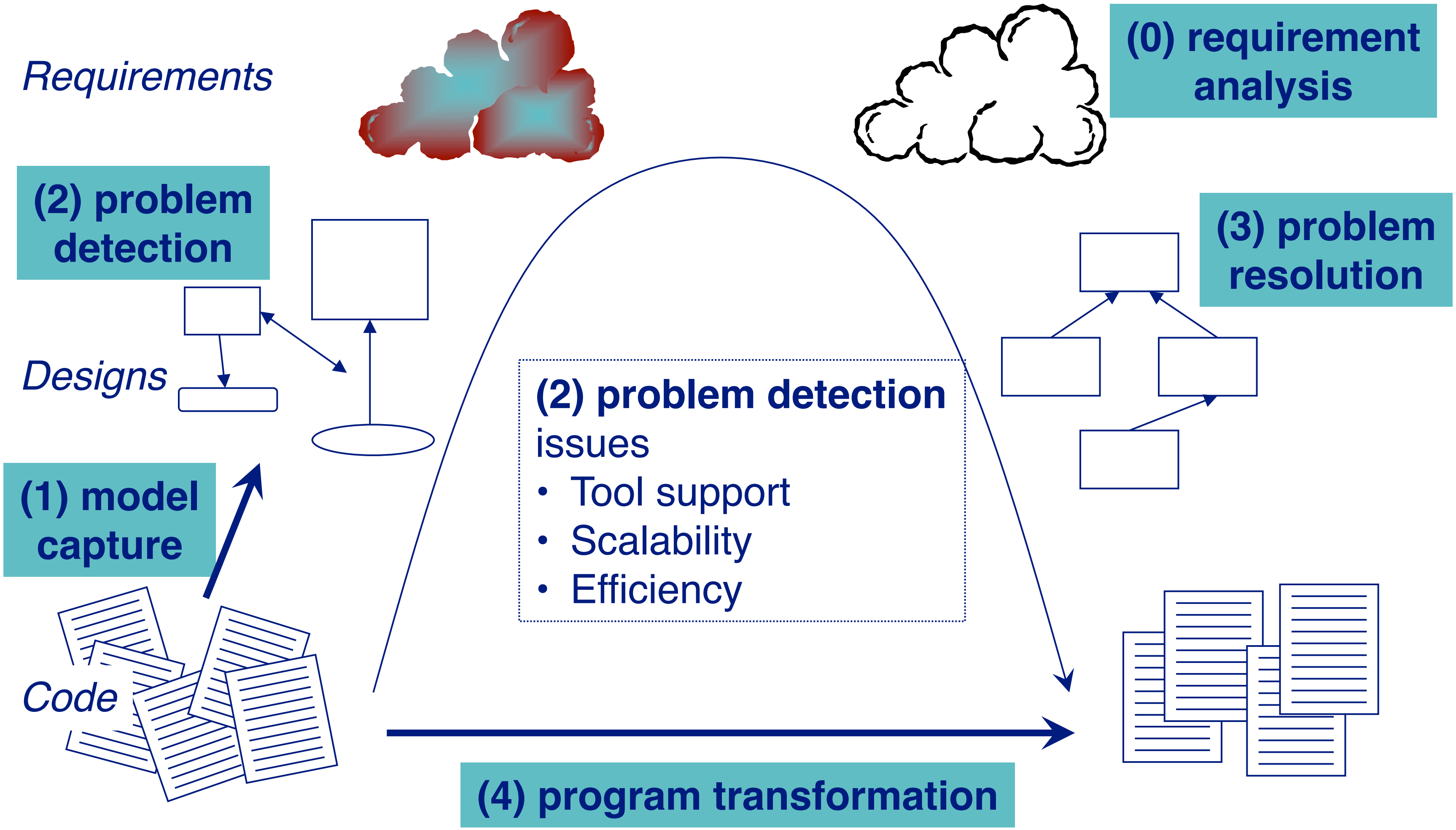
Initial Understanding



3. Software Visualization

- Introduction
  - ☞ The Reengineering life-cycle
- Examples
- Lightweight Approaches
  - ☞ CodeCrawler
- Dynamic Analysis
  - ☞ Key Concept Identification
  - ☞ Feature Location
- Conclusion

The Reengineering Life-cycle



Requirements

(0) requirement analysis

(2) problem detection

Designs

(3) problem resolution

(1) model capture

Code

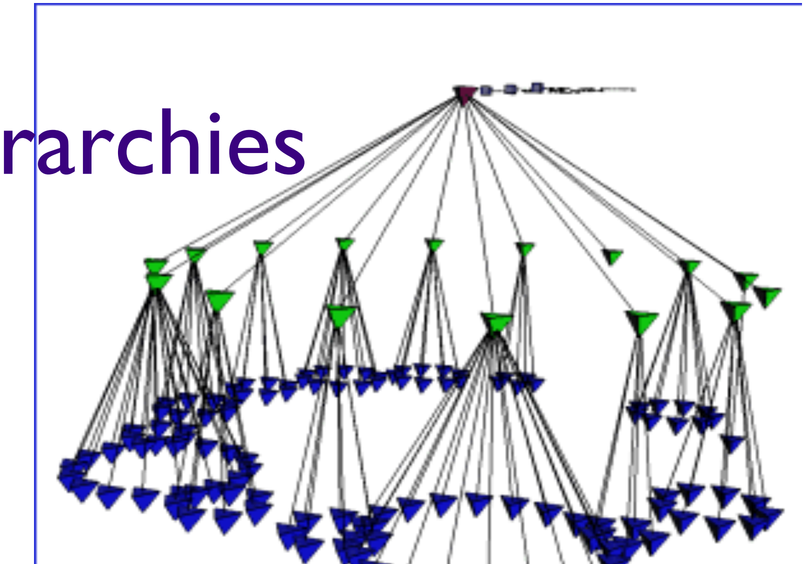
(2) problem detection issues

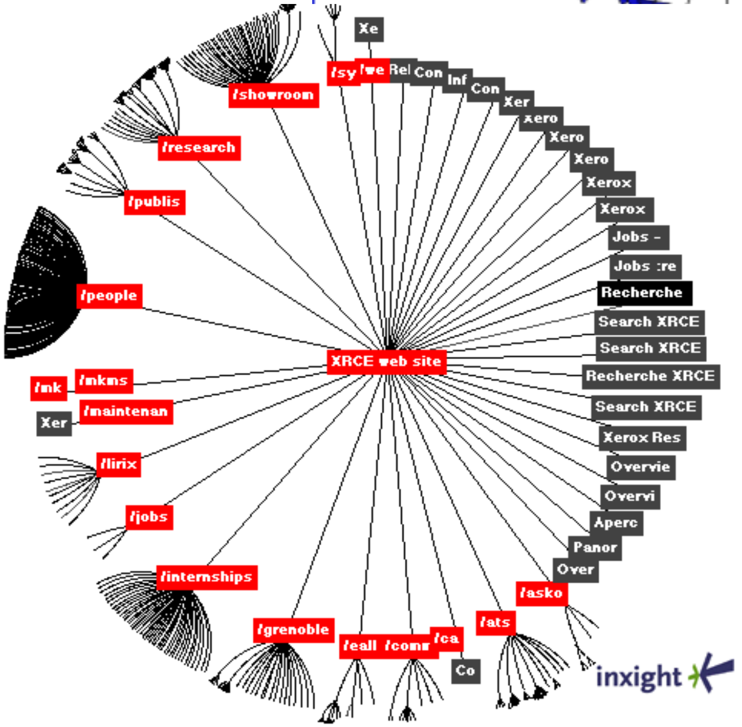
- Tool support
- Scalability
- Efficiency

(4) program transformation

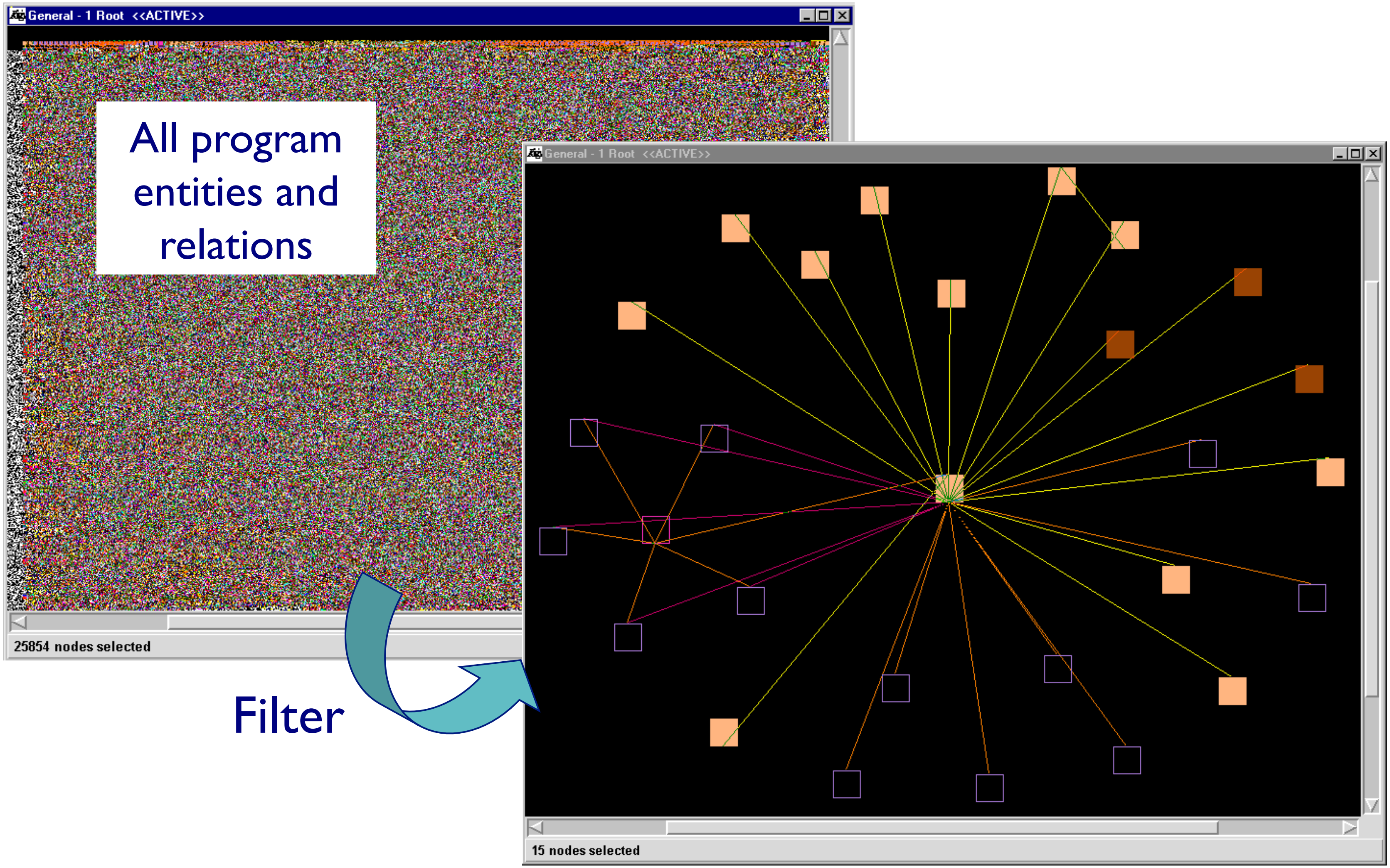
Visualising Hierarchies

- Euclidean cones
  - ☞ Pros:
    - More info than 2D
  - ☞ Cons:
    - Lack of depth
    - Navigation
- Hyperbolic trees
  - ☞ Pros:
    - Good focus
    - Dynamic
  - ☞ Cons:
    - Copyright





Bottom Up Visualisation



All program entities and relations

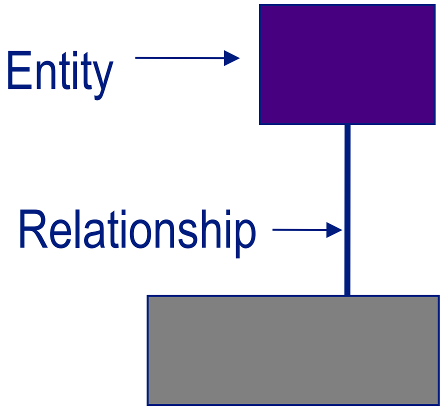
25854 nodes selected

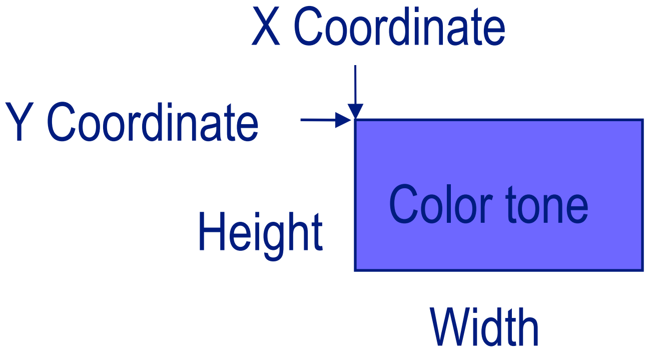
Filter

15 nodes selected

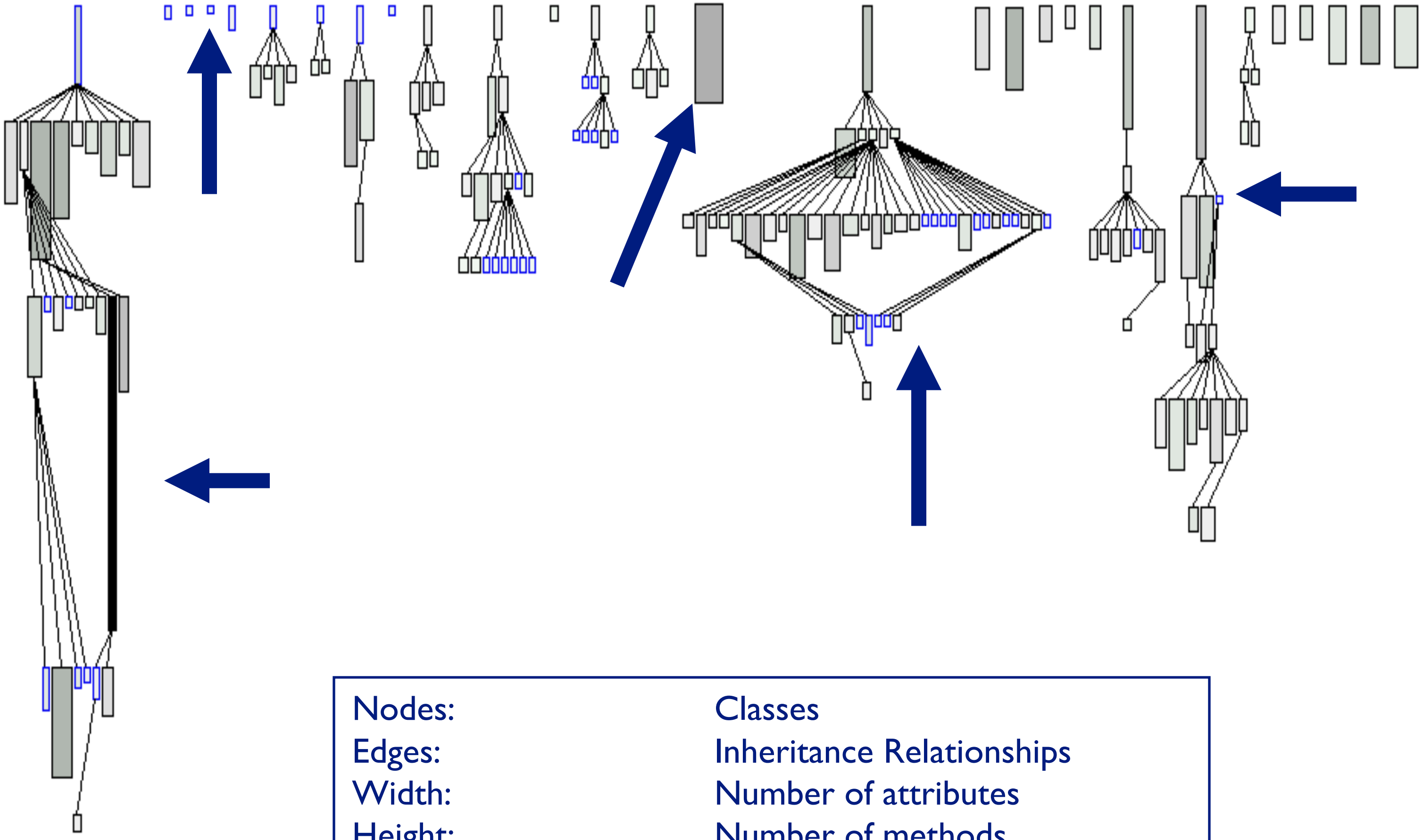
A lightweight approach

- A combination of metrics and software visualization
  - ☞ Visualize software using colored rectangles for the entities and edges for the relationships
  - ☞ Render up to five metrics on one node:
    - Size (1+2)
    - Color (3)
    - Position (4+5)



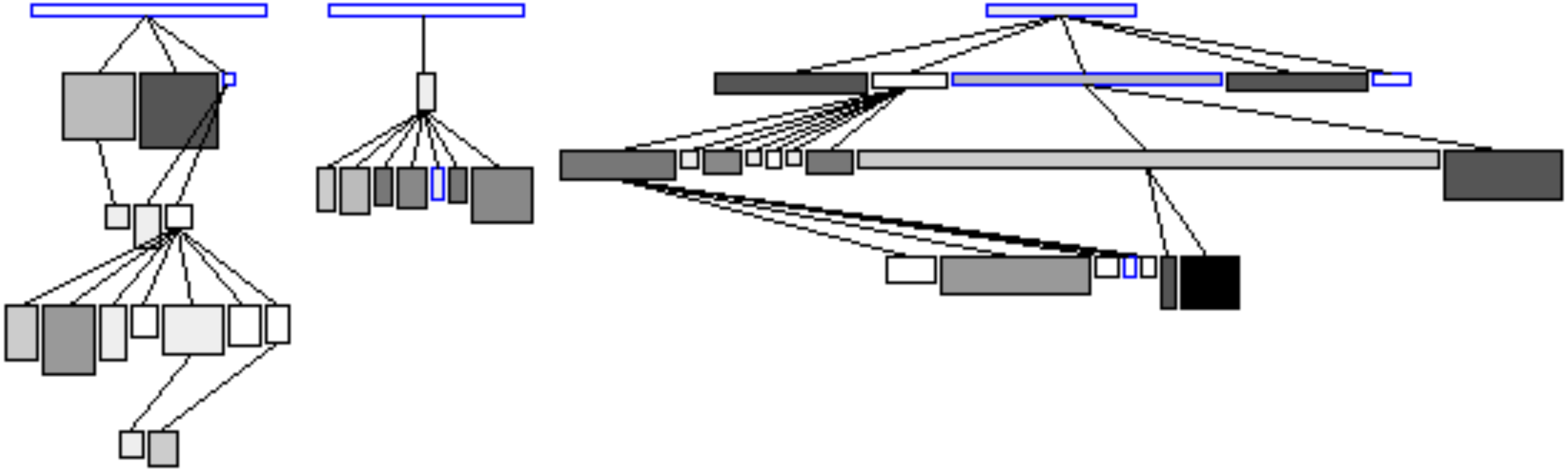


System Complexity View



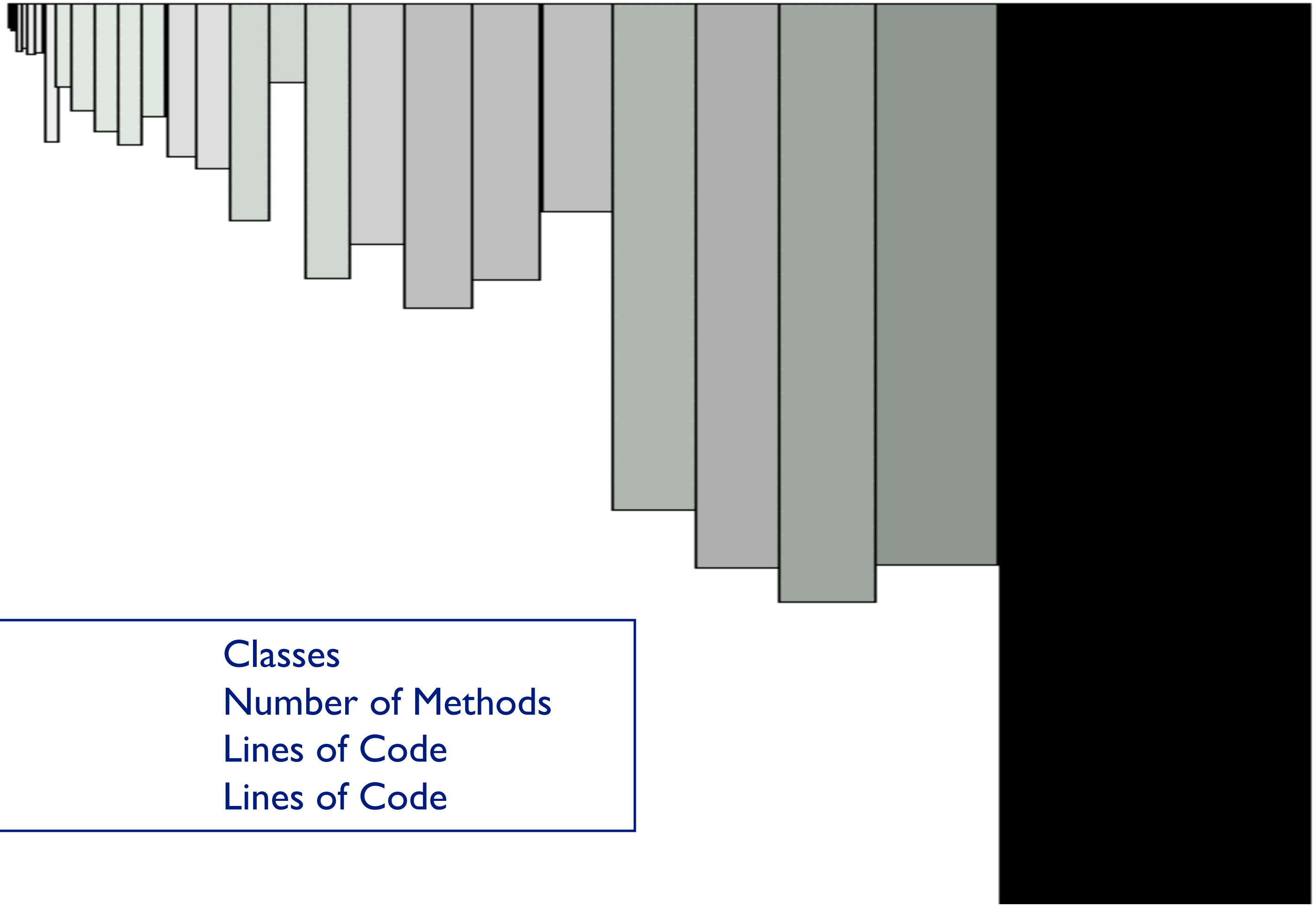
Nodes:	Classes
Edges:	Inheritance Relationships
Width:	Number of attributes
Height:	Number of methods
Color:	Number of lines of code

Inheritance Classification View



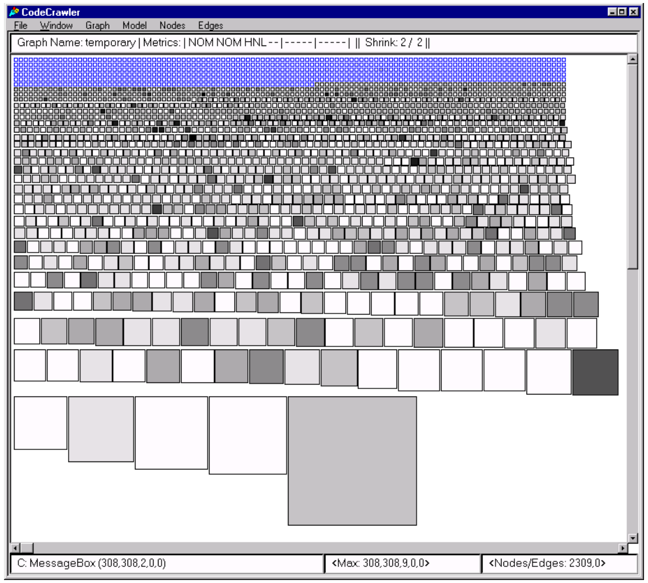
Boxes:	Classes
Edges:	Inheritance
Width:	Number of Methods Added
Height:	Number of Methods Overridden
Color:	Number of Method Extended

Data Storage Class Detection View



Boxes:	Classes
Width:	Number of Methods
Height:	Lines of Code
Color:	Lines of Code

Industrial Validation



Graph Name: temporary | Metrics: | NOM NOM HNL - - | - - - - - | - - - - - | || Shrink: 2 / 2 ||

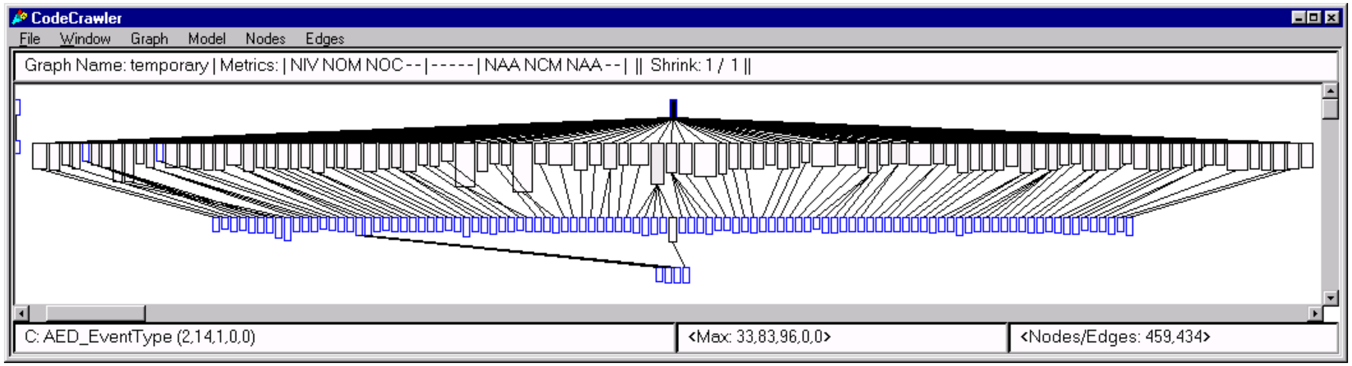
C: MessageBox (308,308,2,0,0) | <Max: 308,308,9,0,0> | <Nodes/Edges: 2309,0>

Personal experience

2-3 days to get something

Nokia (C++ 1.2 MLOC >2300 classes)
Nokia (C++/Java 120 kLOC >400 classes)
MGeniX (Smalltalk 600 kLOC >2100classes)
Bedag (COBOL 40 kLOC)
...

Used by developers + Consultants



Graph Name: temporary | Metrics: | NIV NOM NOC - - | - - - - - | NAA NCM NAA - - | || Shrink: 1 / 1 ||

C: AED_EventType (2,14,1,0,0) | <Max: 33,83,96,0,0> | <Nodes/Edges: 459,434>

State of the Art Tooling

1. source{d}

    https://sourced.tech

    https://github.com/src-d/engine

2. teamscale

    https://www.cqse.eu/

    https://github.com/cqse

3. codescene

    https://codescene.io

    https://github.com/empear-analytics

4. Dynamic Analysis

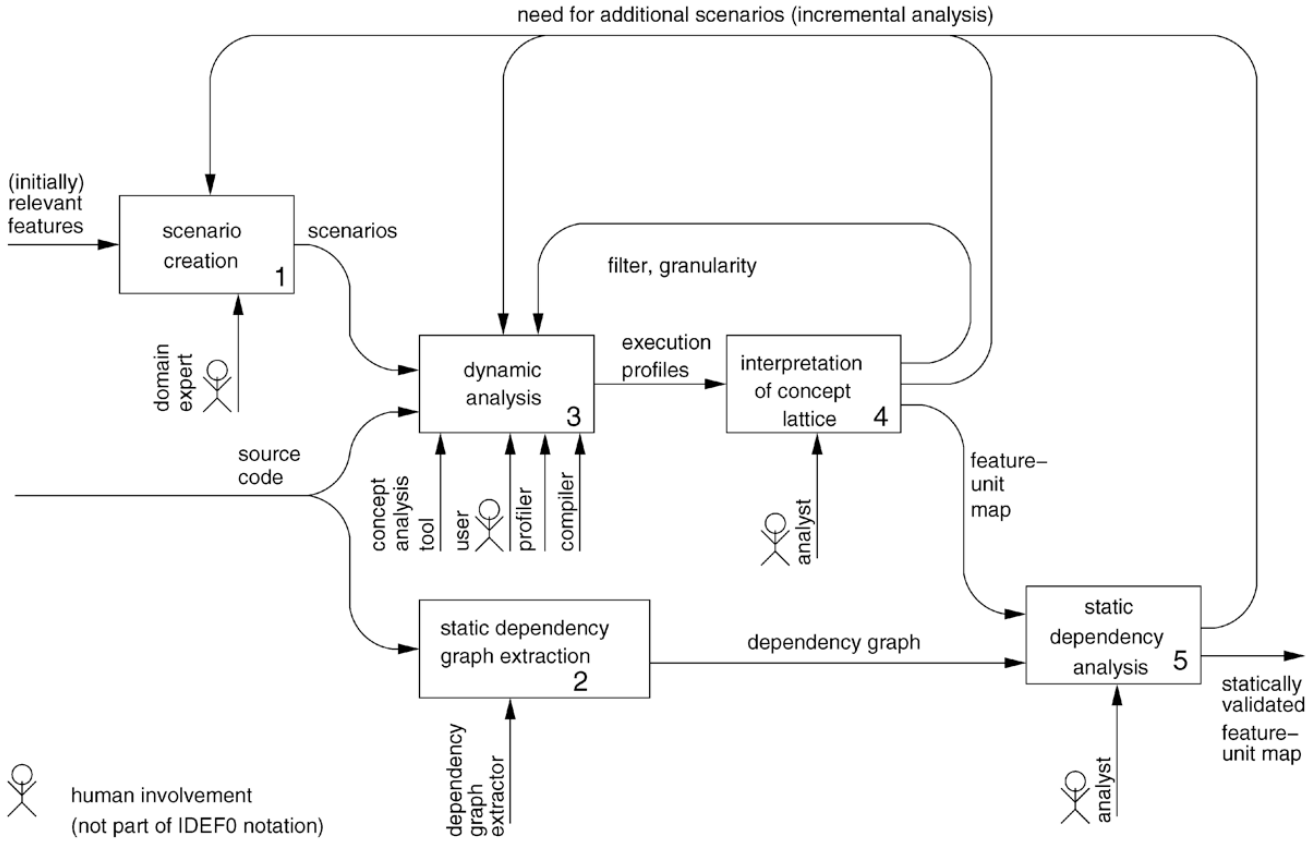
- Key Concept Identification
- Feature Location

Key Concept Identification

Class	IC_CC' + web-mining	Ant docs
Project	√	√
UnknownElement	√	√
Task	√	√
Main	√	√
IntrospectionHelper	√	√
ProjectHelper	√	√
RuntimeConfigurable	√	√
Target	√	√
ElementHandler	√	√
TaskContainer	×	√
Recall (%)	90	-
Precision (%)	60	-

- Extract run-time coupling
- Apply datamining ("google")
- Experiment with documented open-source cases (Ant, JMeter)
  - ☞ recall: +- 90 %
  - ☞ precision: +- 60 %

Feature Location


need for additional scenarios (incremental analysis)
(initially) relevant features
scenario creation 1
scenarios
domain expert
filter, granularity
dynamic analysis 3
execution profiles
interpretation of concept lattice 4
source code
concept analysis tool
user
profiler
compiler
analyst
feature–unit map
static dependency graph extraction 2
dependency graph
static dependency analysis 5
statically validated feature–unit map
analyst
dependency graph extractor
human involvement
(not part of IDEF0 notation)

T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. IEEE Transactions on Software Engineering, 29(3):210–224, March 2003.

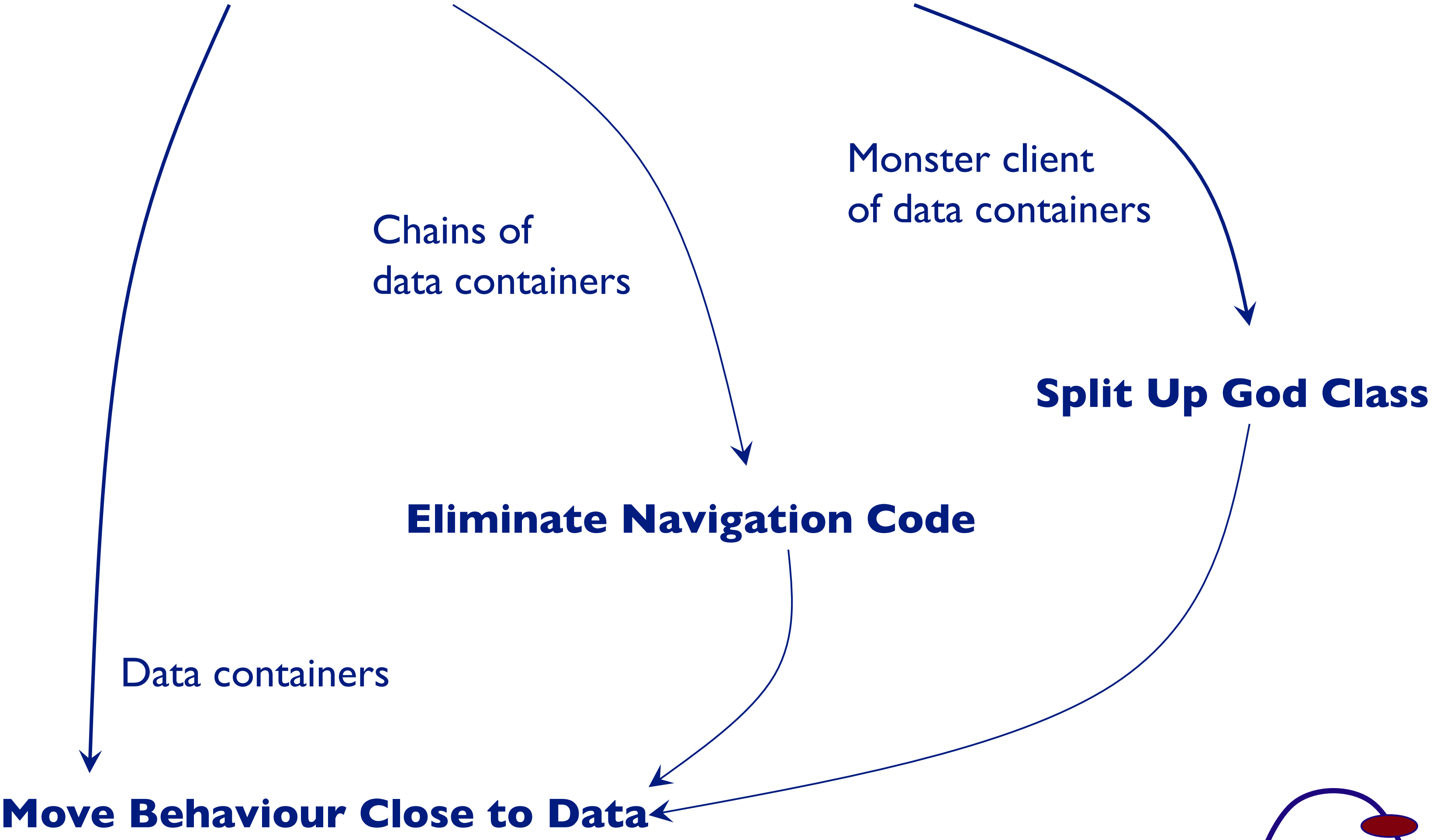
Replication is not supported, industrial cases are rare, …. In order to help the discipline mature, we think that more systematic empirical evaluation is needed. [Tonella et. Al, in Empirical Software Engineering]

5. Restructuring

Redistribute Responsibilities

- ☞ Move Behaviour Close to Data
- ☞ Eliminate Navigation Code
- ☞ Split up God Class
- ☞ Empirical Validation

Redistribute Responsibilities



Chains of data containers

Monster client of data containers

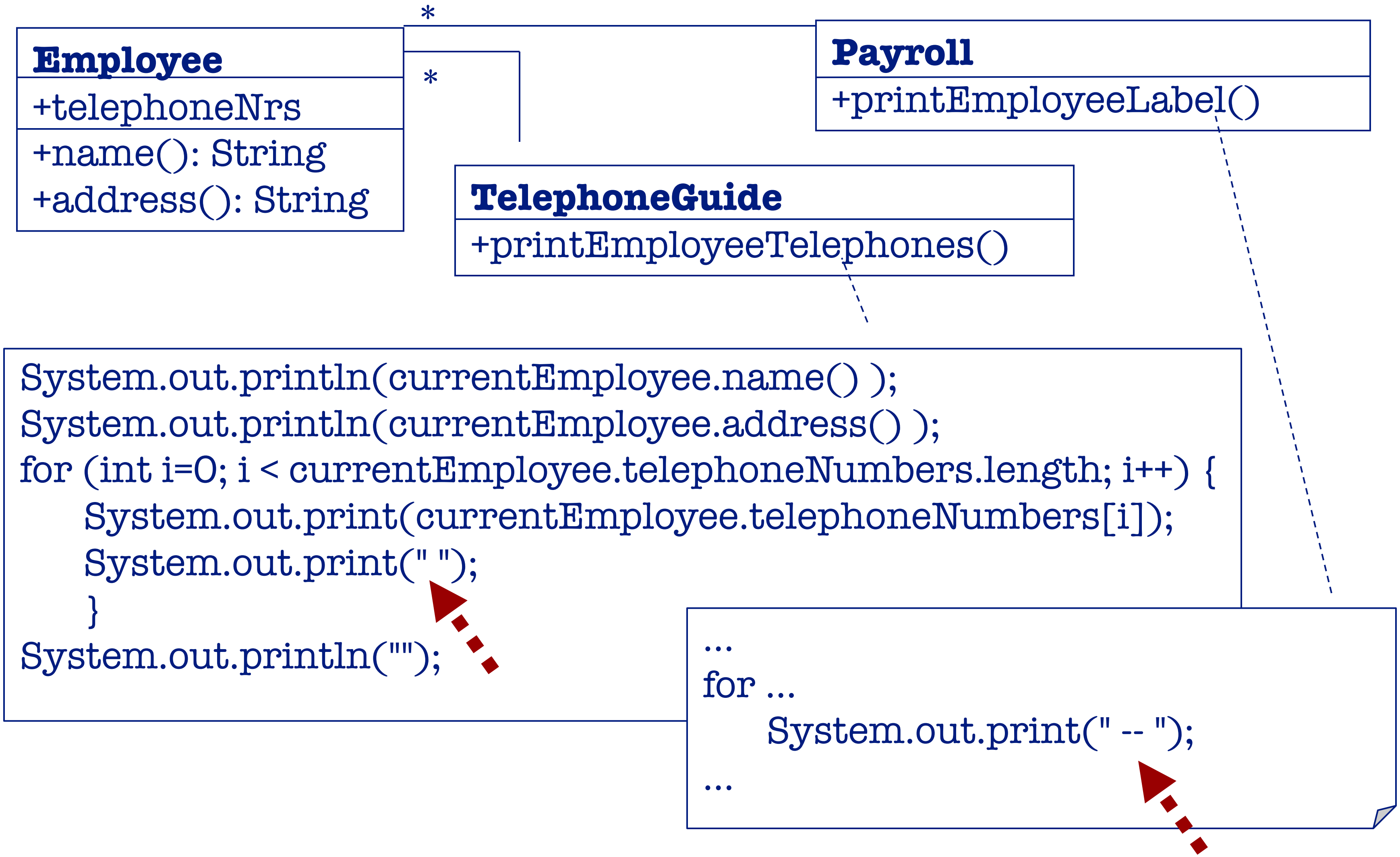
Data containers

Split Up God Class

Eliminate Navigation Code

Move Behaviour Close to Data

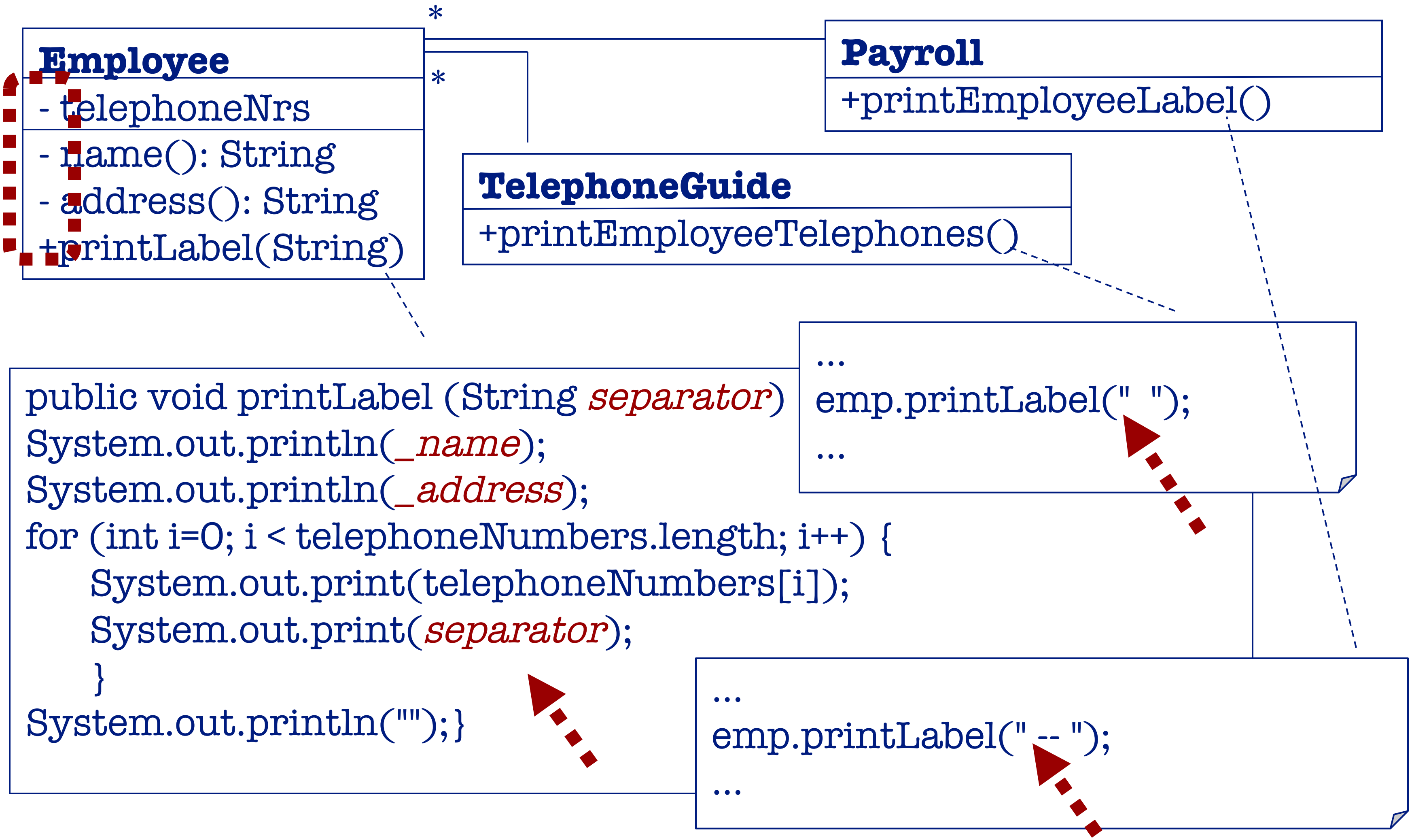
Move Behavior Close to Data (example 1/2)



```
System.out.println(currentEmployee.name() );
System.out.println(currentEmployee.address() );
for (int i=0; i < currentEmployee.telephoneNumbers.length; i++) {
     System.out.print(currentEmployee.telephoneNumbers[i]);
     System.out.print(" ");
     }
System.out.println("");
```

```
...
for ...
     System.out.print(" -- ");
...
```

Move Behavior Close to Data (example 2/2)

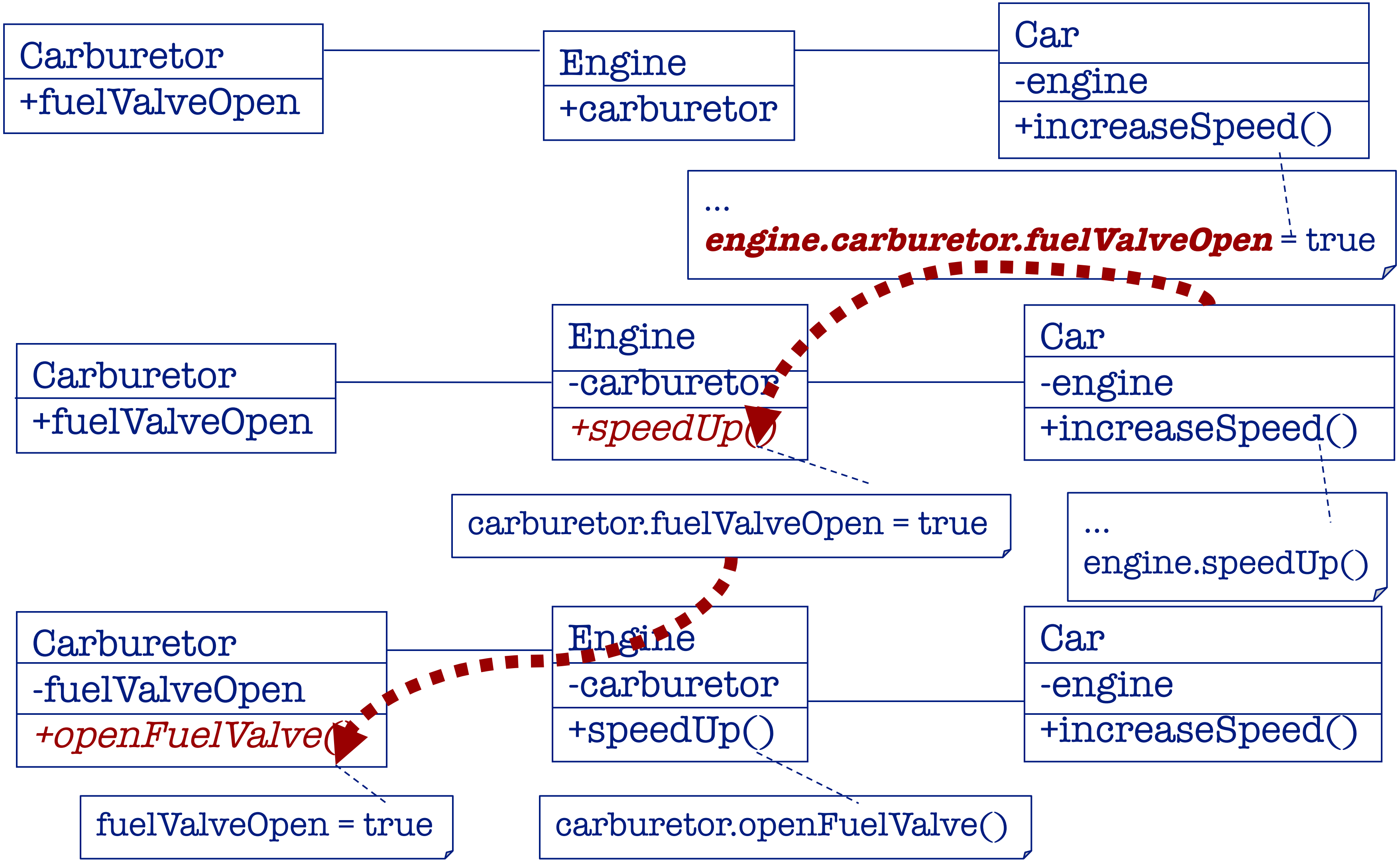


```
public void printLabel (String separator)
System.out.println(_name);
System.out.println(_address);
for (int i=0; i < telephoneNumbers.length; i++) {
      System.out.print(telephoneNumbers[i]);
      System.out.print(separator);
      }
System.out.println("");}
```

```
...
emp.printLabel(" ");
...
```

```
...
emp.printLabel(" -- ");
...
```

Eliminate Navigation Code



Carburetor
+fuelValveOpen

Engine
+carburetor

Car
-engine
+increaseSpeed()

…
engine.carburetor.fuelValveOpen = true

Carburetor
+fuelValveOpen

Engine
-carburetor
+speedUp()

Car
-engine
+increaseSpeed()

carburetor.fuelValveOpen = true

…
engine.speedUp()

Carburetor
-fuelValveOpen
+openFuelValve()

Engine
-carburetor
+speedUp()

Car
-engine
+increaseSpeed()

fuelValveOpen = true

carburetor.openFuelValve()

Split Up God Class

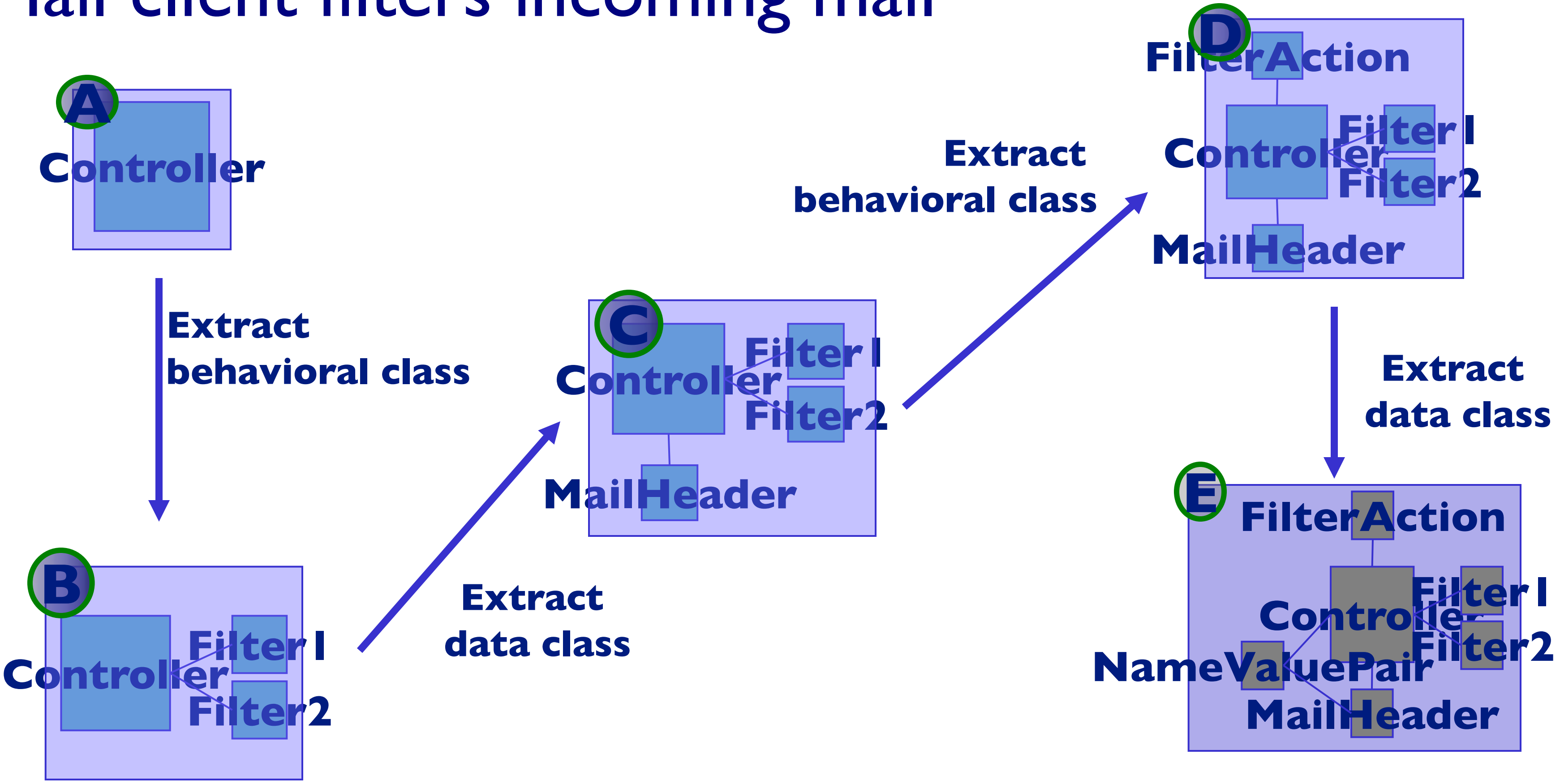
Problem: Break a class which monopolizes control?

Solution: Incrementally eliminate navigation code

- Detection:
  - ☞ measuring size
  - ☞ class names containing Manager, System, Root, Controller
  - ☞ the class that all maintainers are avoiding
- How:
  - ☞ move behaviour close to data + eliminate navigation code
  - ☞ remove or deprecate façade
- However:
  - ☞ If God Class is stable, then don't split
    ⇒ shield client classes from the god class

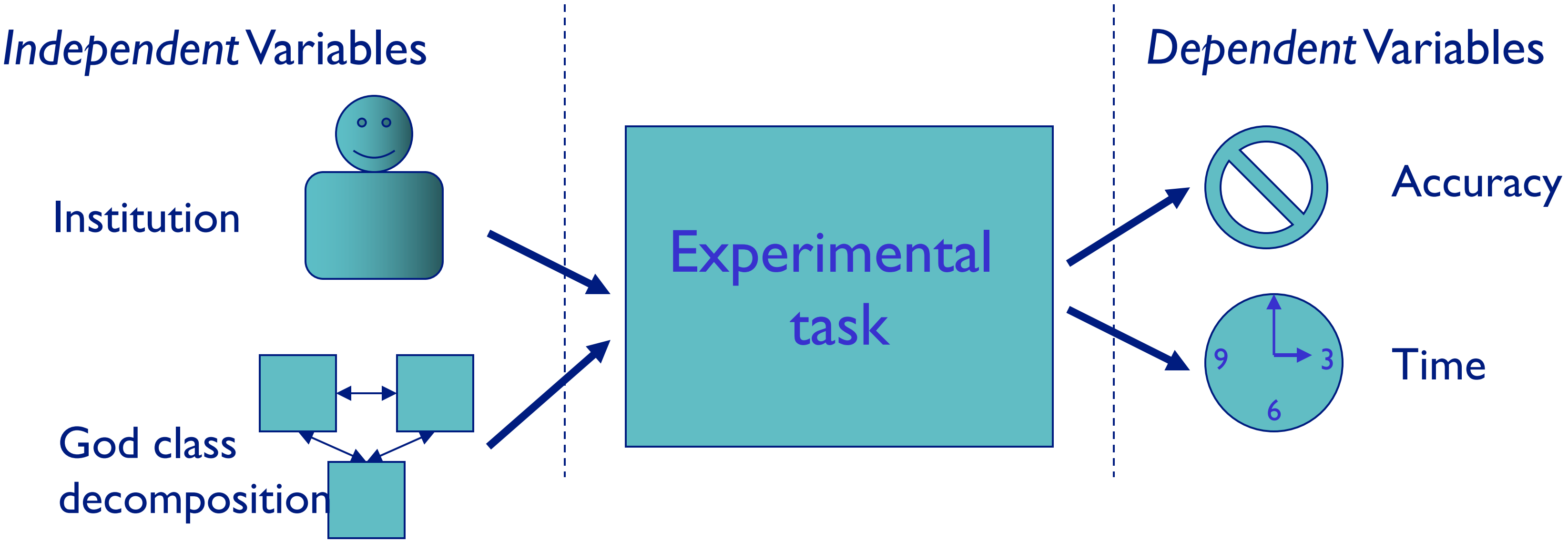
Split Up God Class: 5 variants

Mail client filters incoming mail



Empirical Validation

- **Controlled experiment** with 63 last-year master-level students (CS and ICT)



Independent Variables

Institution

God class decomposition

Experimental task

Dependent Variables

Accuracy

Time

Interpretation of Results

- “*Optimal decomposition*” differs with respect to training
  - ☞ Computer science: preference towards C-E
  - ☞ ICT-electronics: preference towards A-C

- Advanced OO training can induce a preference towards particular styles of decomposition
  - ☞ Consistent with [Arisholm et al. 2004]

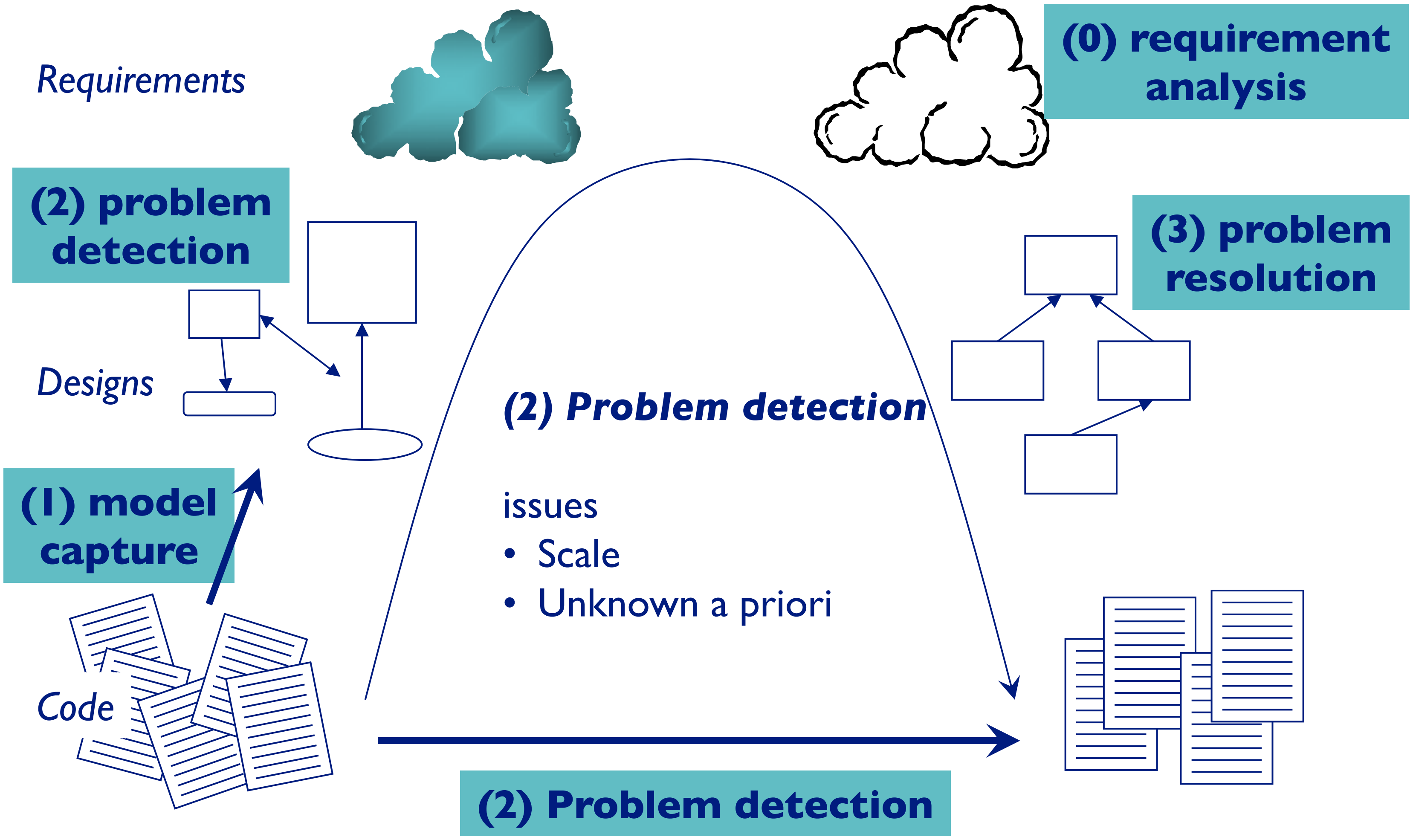
“Good” design is in the eye of the beholder

6. Code Duplication

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

- ***Code Duplication***
  - ☞ What is it?
  - ☞ Why is it harmful?
- Detecting Code Duplication
- Approaches
- A Lightweight Approach
- Visualization (dotplots)
- Duploc
- Recent trends

The Reengineering Life-Cycle



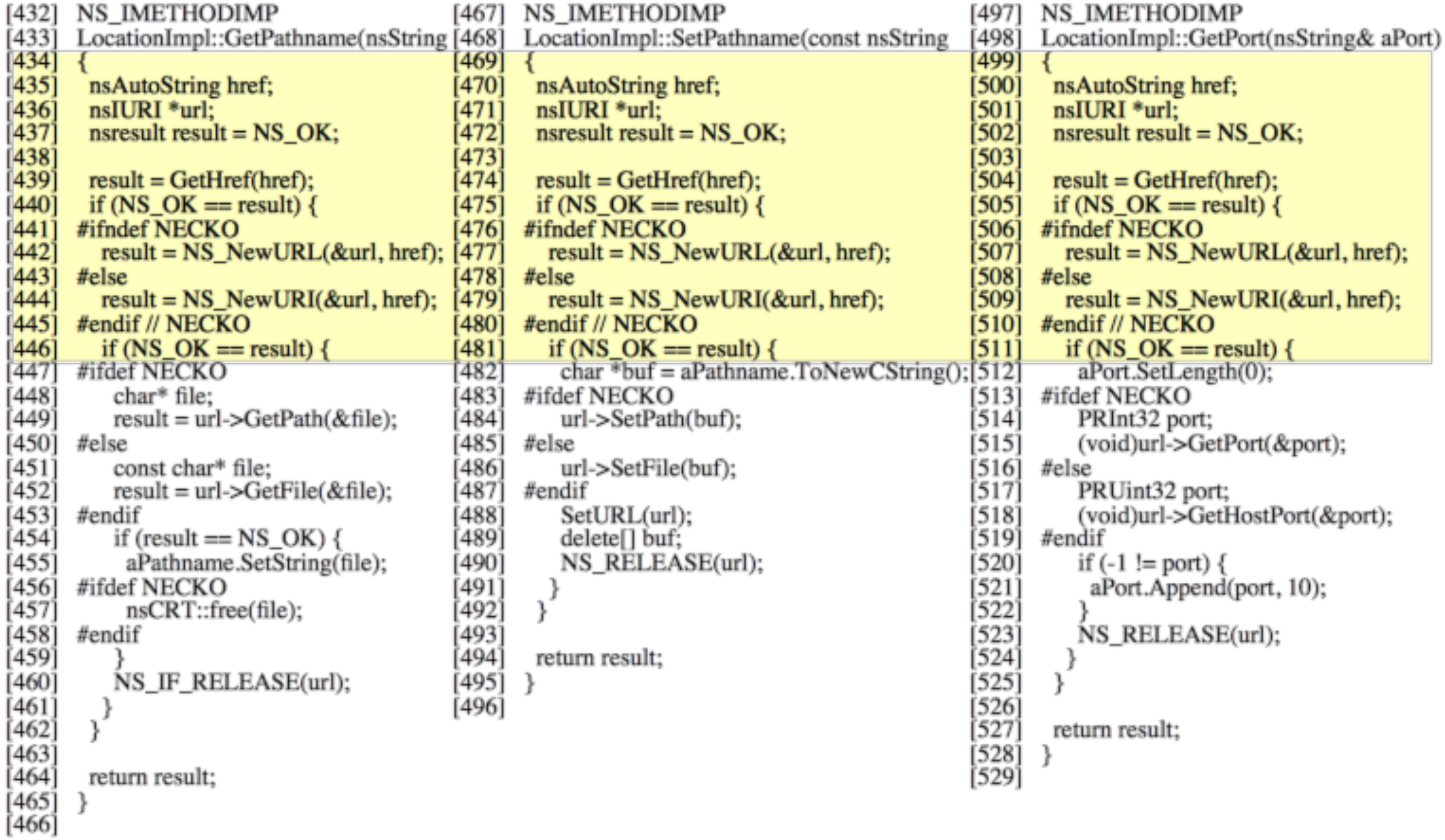
(2) Problem detection

issues

- Scale
- Unknown a priori

Code is Copied

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



```
NS_IMETHODIMP
LocationImpl::GetPathname(nsString
{
  nsAutoString href;
  nsIURI *url;
  nsresult result = NS_OK;

  result = GetHref(href);
  if (NS_OK == result) {
#ifndef NECKO
    result = NS_NewURL(&url, href);
#else
    result = NS_NewURI(&url, href);
#endif // NECKO
    if (NS_OK == result) {
#ifdef NECKO
      char* file;
      result = url->GetPath(&file);
#else
      const char* file;
      result = url->GetFile(&file);
#endif
      if (result == NS_OK) {
        aPathname.SetString(file);
#ifdef NECKO
        nsCRT::free(file);
#endif
      }
      NS_IF_RELEASE(url);
    }
  }

  return result;
}
```

```
NS_IMETHODIMP
LocationImpl::SetPathname(const nsString
{
  nsAutoString href;
  nsIURI *url;
  nsresult result = NS_OK;

  result = GetHref(href);
  if (NS_OK == result) {
#ifndef NECKO
    result = NS_NewURL(&url, href);
#else
    result = NS_NewURI(&url, href);
#endif // NECKO
    if (NS_OK == result) {
      char *buf = aPathname.ToNewCString();
#ifdef NECKO
      url->SetPath(buf);
#else
      url->SetFile(buf);
#endif
      SetURL(url);
      delete[] buf;
      NS_RELEASE(url);
    }
  }

  return result;
}
```

```
NS_IMETHODIMP
LocationImpl::GetPort(nsString& aPort)
{
  nsAutoString href;
  nsIURI *url;
  nsresult result = NS_OK;

  result = GetHref(href);
  if (NS_OK == result) {
#ifndef NECKO
    result = NS_NewURL(&url, href);
#else
    result = NS_NewURI(&url, href);
#endif // NECKO
    if (NS_OK == result) {
      aPort.SetLength(0);
#ifdef NECKO
      PRInt32 port;
      (void)url->GetPort(&port);
#else
      PRUint32 port;
      (void)url->GetHostPort(&port);
#endif
      if (-1 != port) {
        aPort.Append(port, 10);
      }
      NS_RELEASE(url);
    }
  }

  return result;
}
```

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%

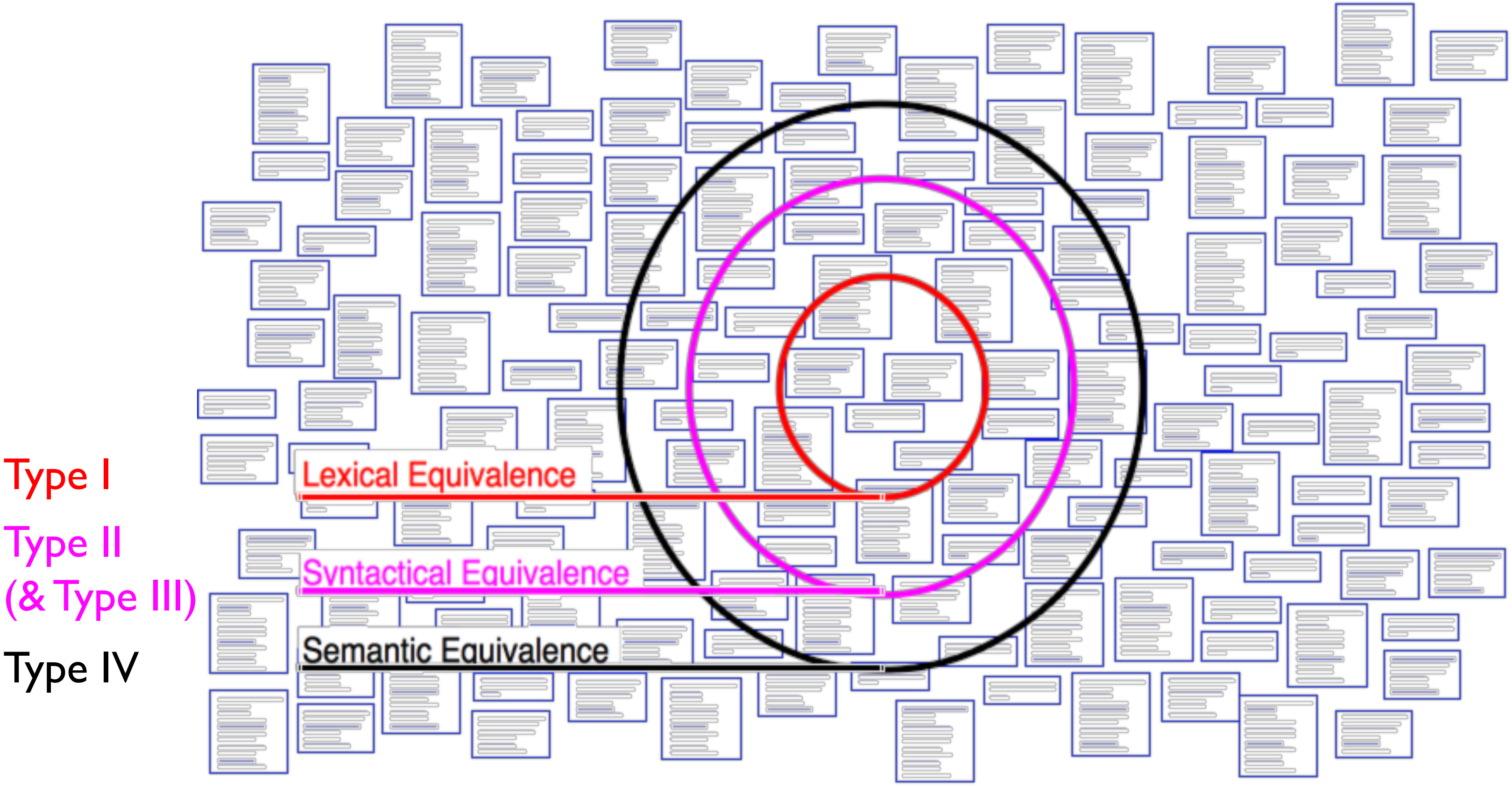
Copied Code Problems

- 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

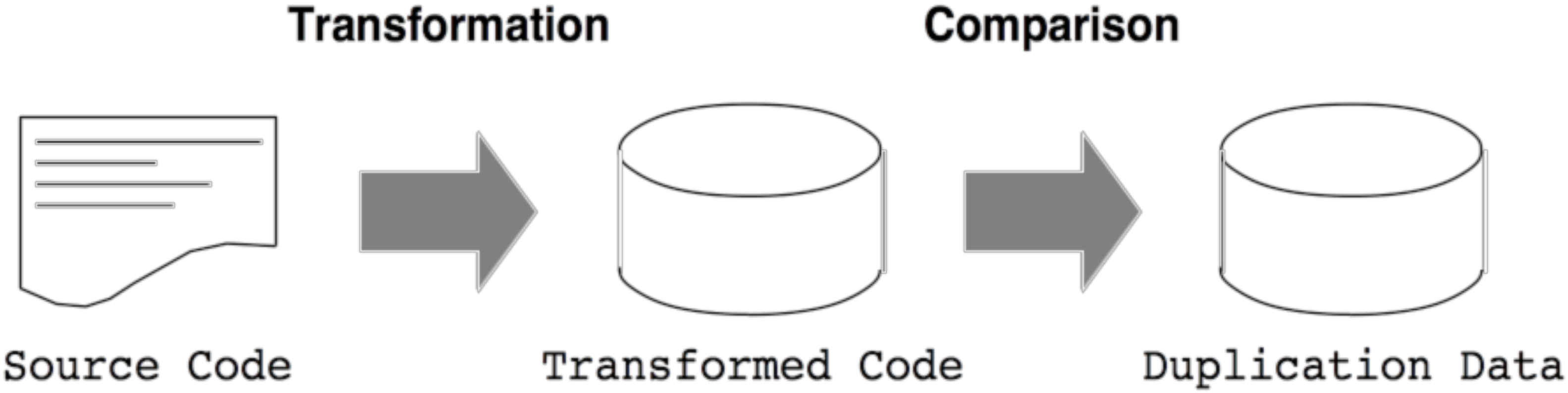
Code Duplication Detection

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

Simple Detection Approach (i)

- **Assumption**:
  - Code segments are just copied and changed at 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+]
```

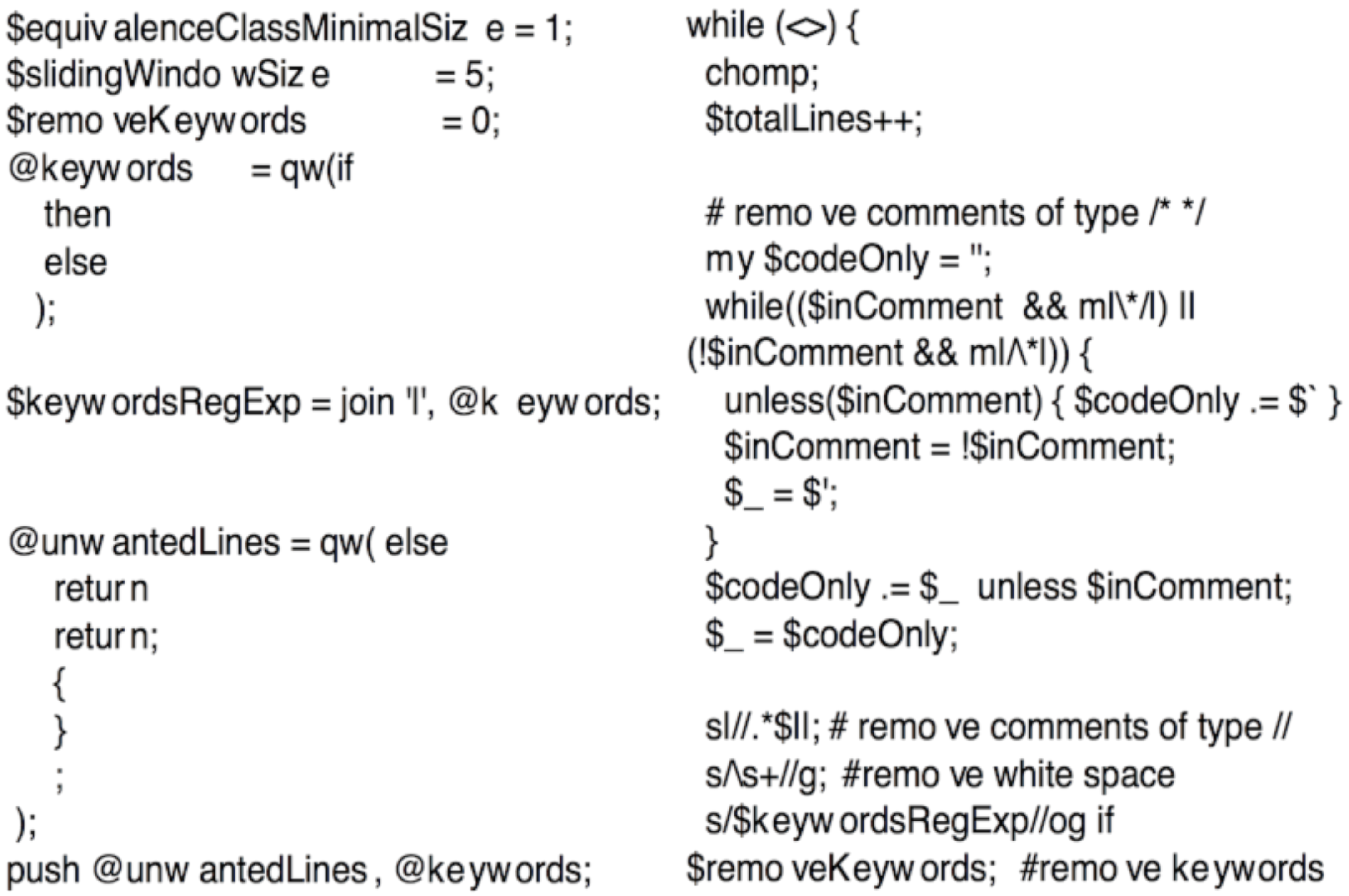
Simple Detection Approach (ii)

- **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:

    1. Preprocessing: Compute the hash value for each line

    2. Actual Comparison: Compare all lines in the same hash bucket
- **Evaluation of the Approach**
  - ☞ Advantages: Simple, language independent
  - ☞ Disadvantages: Difficult interpretation

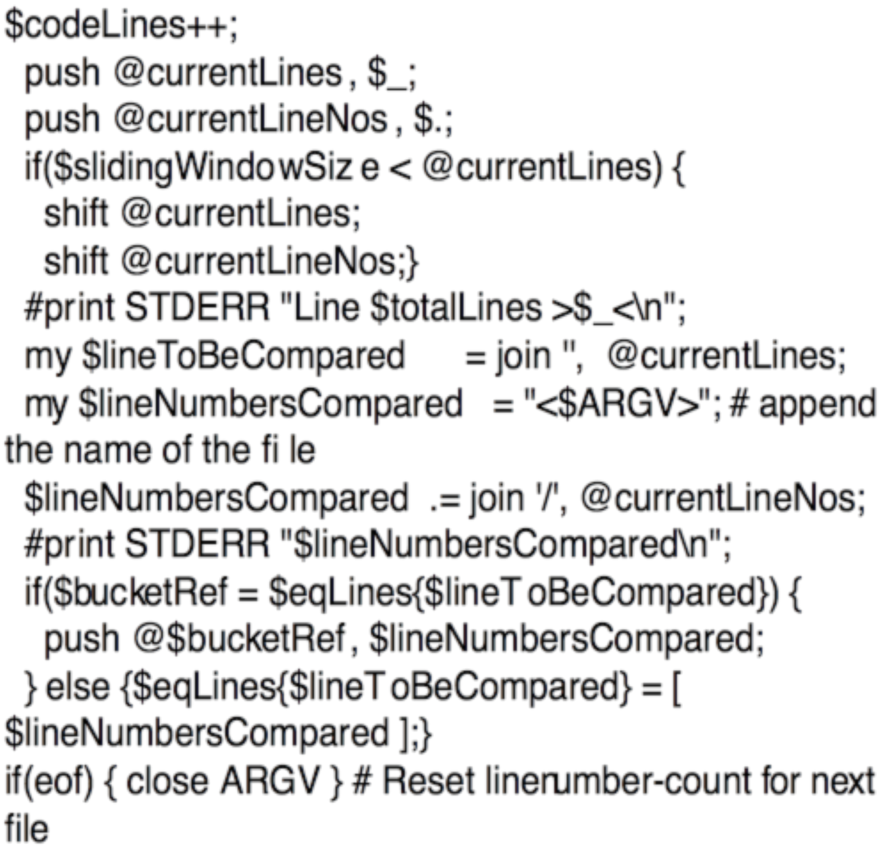
A Perl script for C++ (1/2)



```
$equiv alenceClassMinimalSiz  e = 1;
$slidingWindo wSiz e          = 5;
$remo veK eyw ords             = 0;
@keyw ords      = qw(if
   then
   else
  );

$keyw ordsRegExp = join 'l', @k  eyw ords;


@unw antedLines = qw( else
    retur n
    retur n;
    {
    }
    ;
 );
push @unw antedLines , @ke yw ords;
```

```
while (<>) {
  chomp;
  $totalLines++;

  # remo ve comments of type /* */
  my $codeOnly = '';
  while(($inComment  && ml\*/l) ll
(!$inComment && ml/\*l)) {
    unless($inComment) { $codeOnly .= $` }
    $inComment = !$inComment;
    $_ = $';
  }
  $codeOnly .= $_  unless $inComment;
  $_ = $codeOnly;

  sl//.*$ll; # remo ve comments of type //
  s/\s+//g;  #remo ve white space
  s/$k eyw ordsRegExp//og if
$remo veKeyw ords;   #remo ve ke ywords
```

A Perl script for C++ (2/2)



```
$codeLines++;
  push @currentLines , $_;
  push @currentLineNos , $.;
  if($slidingWindowSiz e < @currentLines) {
    shift @currentLines;
    shift @currentLineNos;}
  #print STDERR "Line $totalLines >$_<\n";
  my $lineToBeCompared      = join '',  @currentLines;
  my $lineNumbersCompared   = "<$ARGV>"; # append
the name of the fi le
  $lineNumbersCompared  .= join '/', @currentLineNos;
  #print STDERR "$lineNumbersCompared\n";
  if($bucketRef = $eqLines{$lineT oBeCompared}) {
    push @$bucketRef , $lineNumbersCompared;
  } else {$eqLines{$lineT oBeCompared} = [
$lineNumbersCompared ];}
if(eof) { close ARGV } # Reset linenumber-count for next
file
```

- Handles multiple files
- Removes comments
  and white spaces
- Controls noise (if, {,)
- Granularity (number of lines)
- Possible to remove keywords

Output Sample

```
Lines:
create_property(pd,pnImplObjects,stReference,false,*iImplObjects);
create_property(pd,pnElttype,stReference,true,*iEltType);
create_property(pd,pnMinelt,stInteger,true,*iMinelt);
create_property(pd,pnMaxelt,stInteger,true,*iMaxelt);
create_property(pd,pnOwnership,stBool,true,*iOwnership);
Locations: </face/typesystem/SCTypesystem.C>6178/6179/6180/6181/6182
</face/typesystem/SCTypesystem.C>6198/6199/6200/6201/6202
Lines:
create_property(pd,pnSupertype,stReference,true,*iSupertype);
create_property(pd,pnImplObjects,stReference,false,*iImplObjects);
create_property(pd,pnElttype,stReference,true,*iEltType);
create_property(pd,pMinelt,stInteger,true,*iMinelt);
create_property(pd,pnMaxelt,stInteger,true,*iMaxelt);
Locations: </face/typesystem/SCTypesystem.C>6177/6178
</face/typesystem/SCTypesystem.C>6229/6230
```

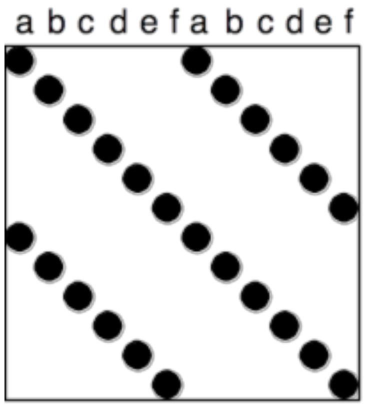
Lines = duplicated lines

Locations = file names and line number

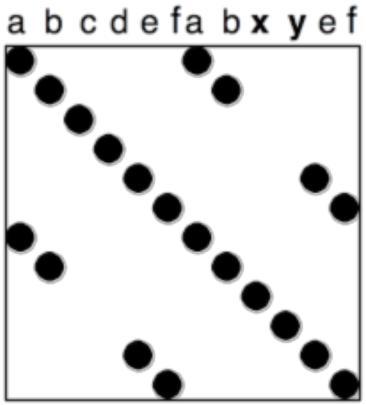
Visualization of Duplicated Code

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

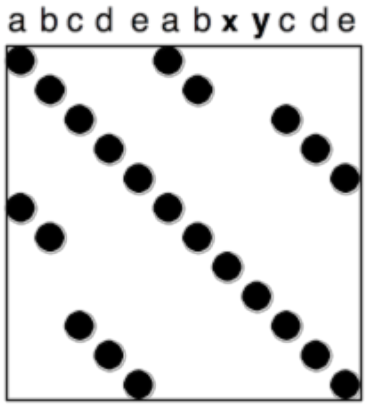
- Dotplots — Technique from DNA Analysis
  - Code is put on vertical as well as horizontal axis
  - A match between two elements is a dot in the matrix



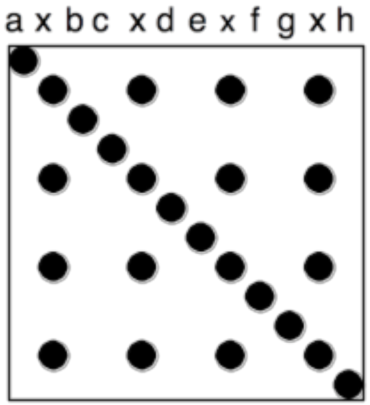
Exact Copies



Copies with Variations

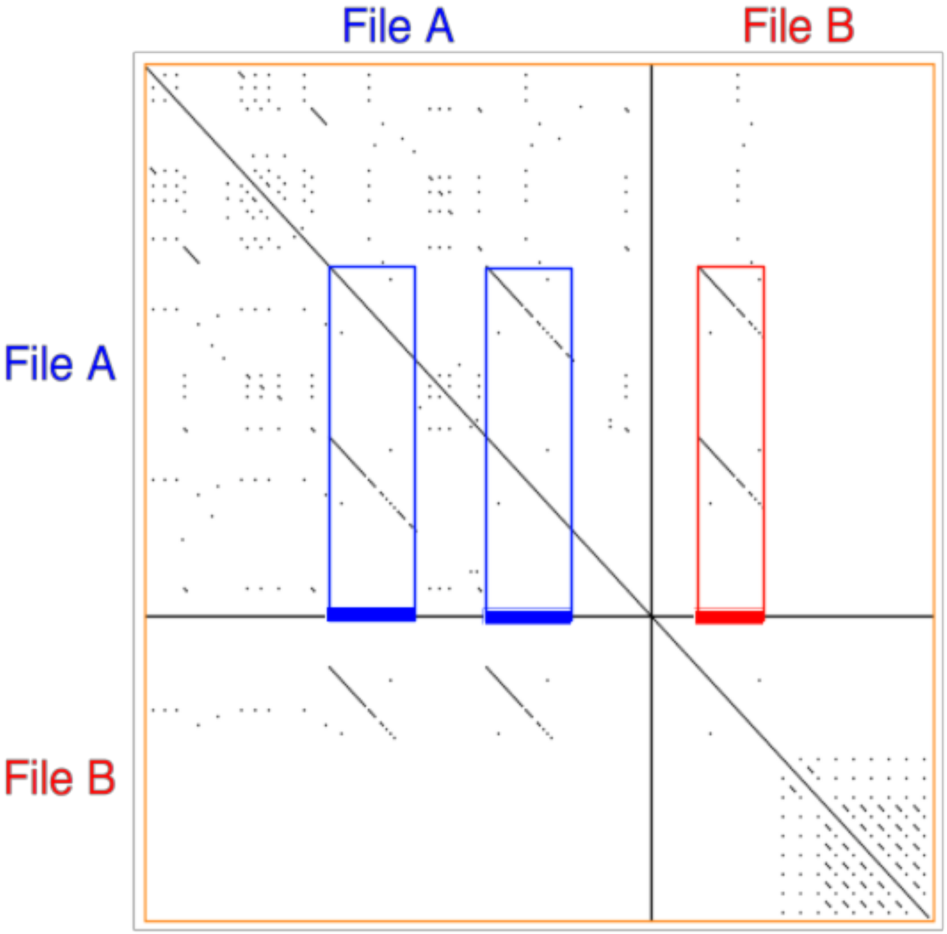


Inserts/Deletes



Repetitive Code Elements

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)

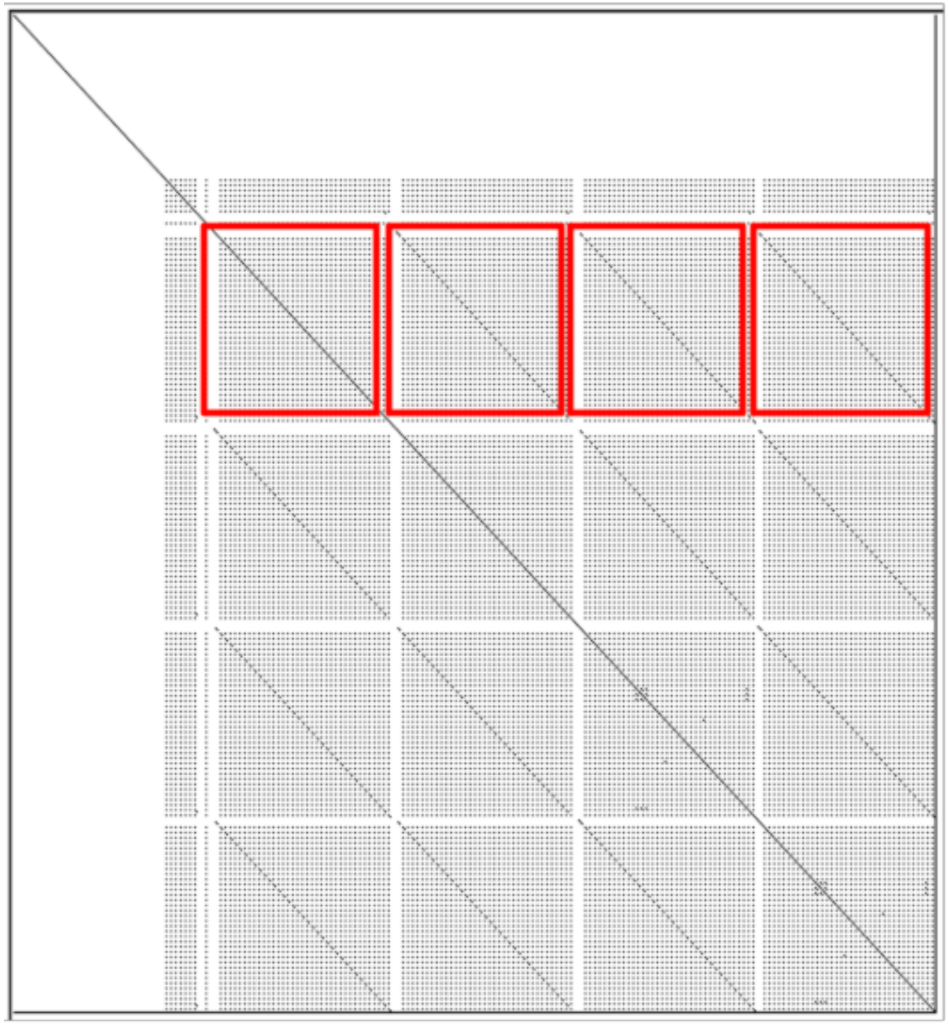
Visualization of Repetitive Structures

Detected Problem

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

Possible Solution

Strategy Method



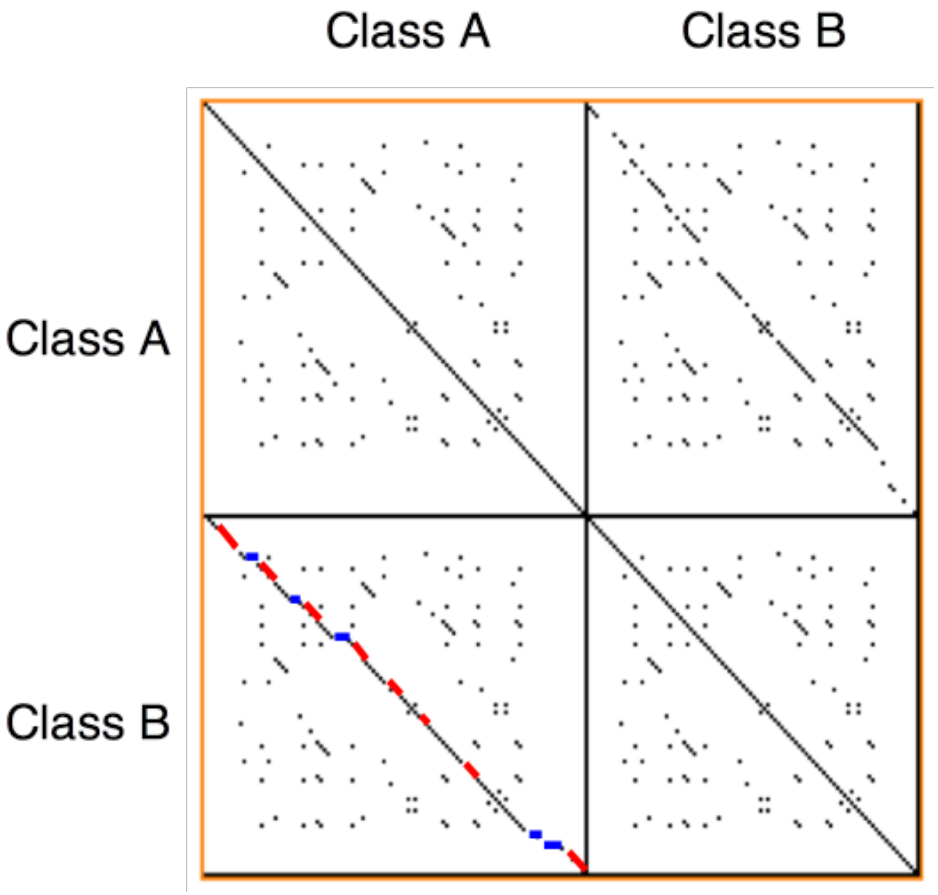
Visualization of Cloned Classes

Detected Problem:

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

Possible Solution

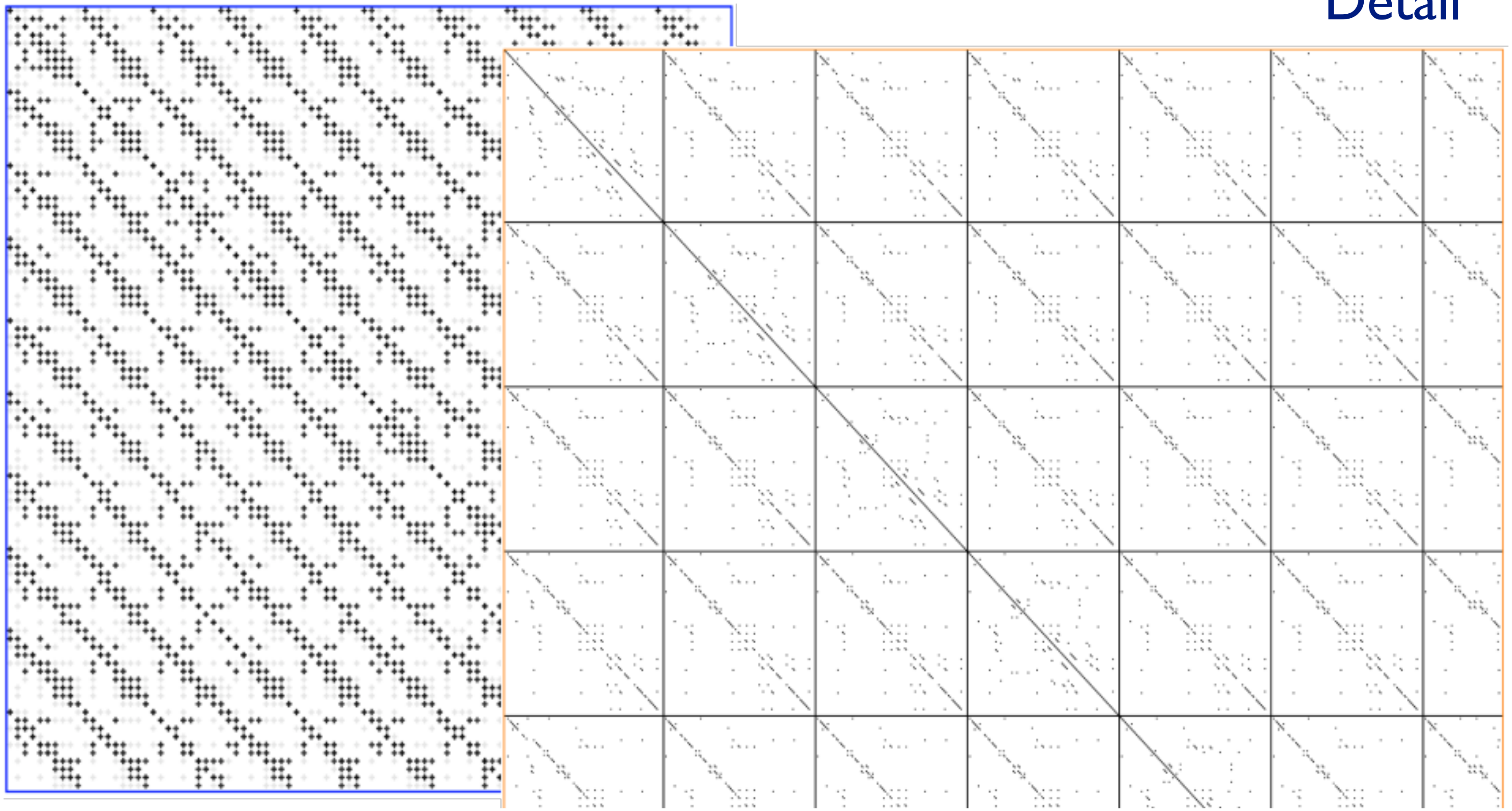
Subclassing …



Visualization of Clone Families

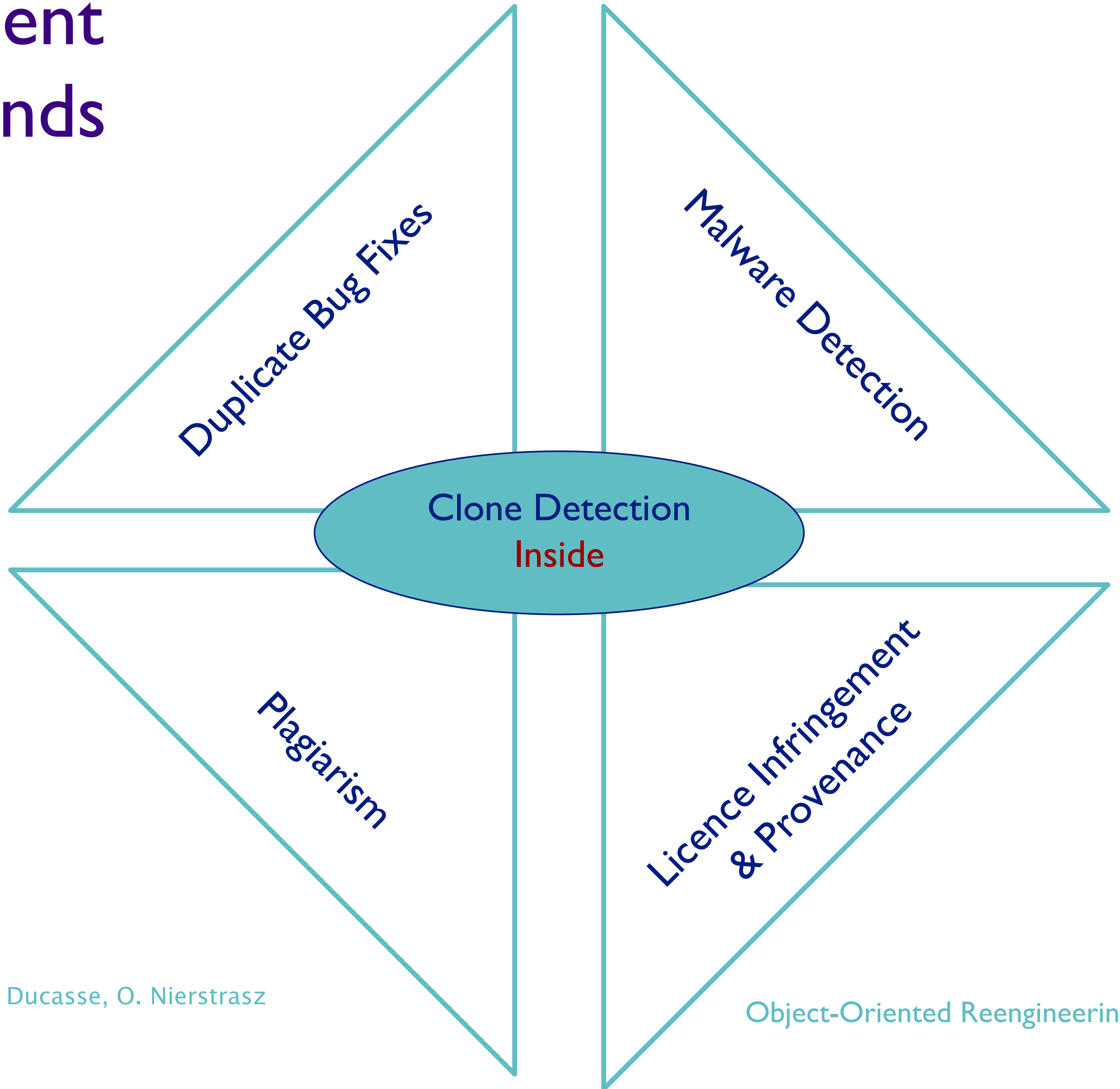
Overview

Detail



20 Classes implementing lists for different data types

Recent Trends



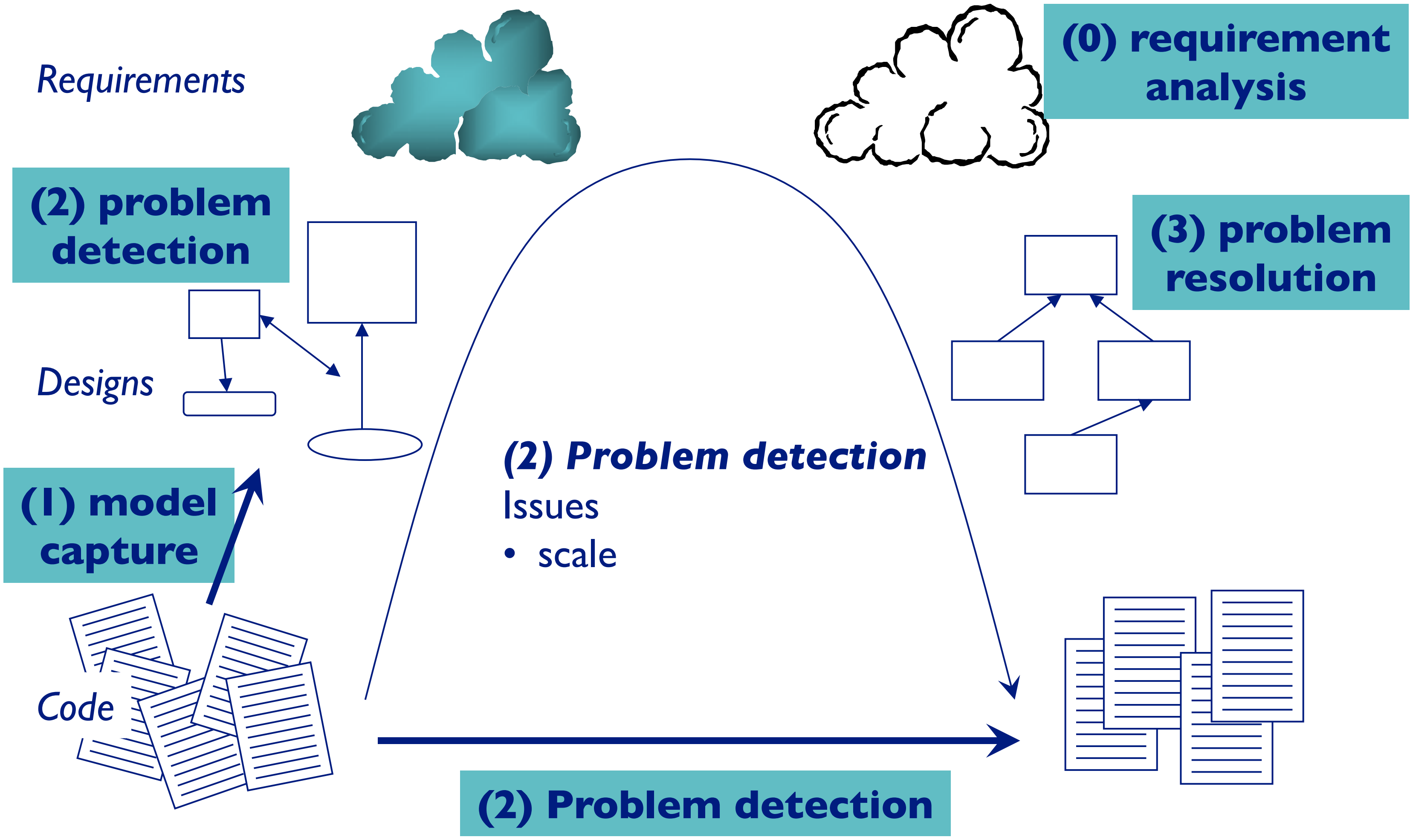
7. Software Evolution

- Exploiting the Version Control System
  - ☞ Visualizing CVS changes
- The Evolution Matrix
- Test History

It is not *age* that turns a piece of software into a legacy system, but the *rate* at which it has been developed and adapted without being reengineered.

[Demeyer, Ducasse and Nierstrasz: Object-Oriented Reengineering Patterns]

The Reengineering Life-Cycle

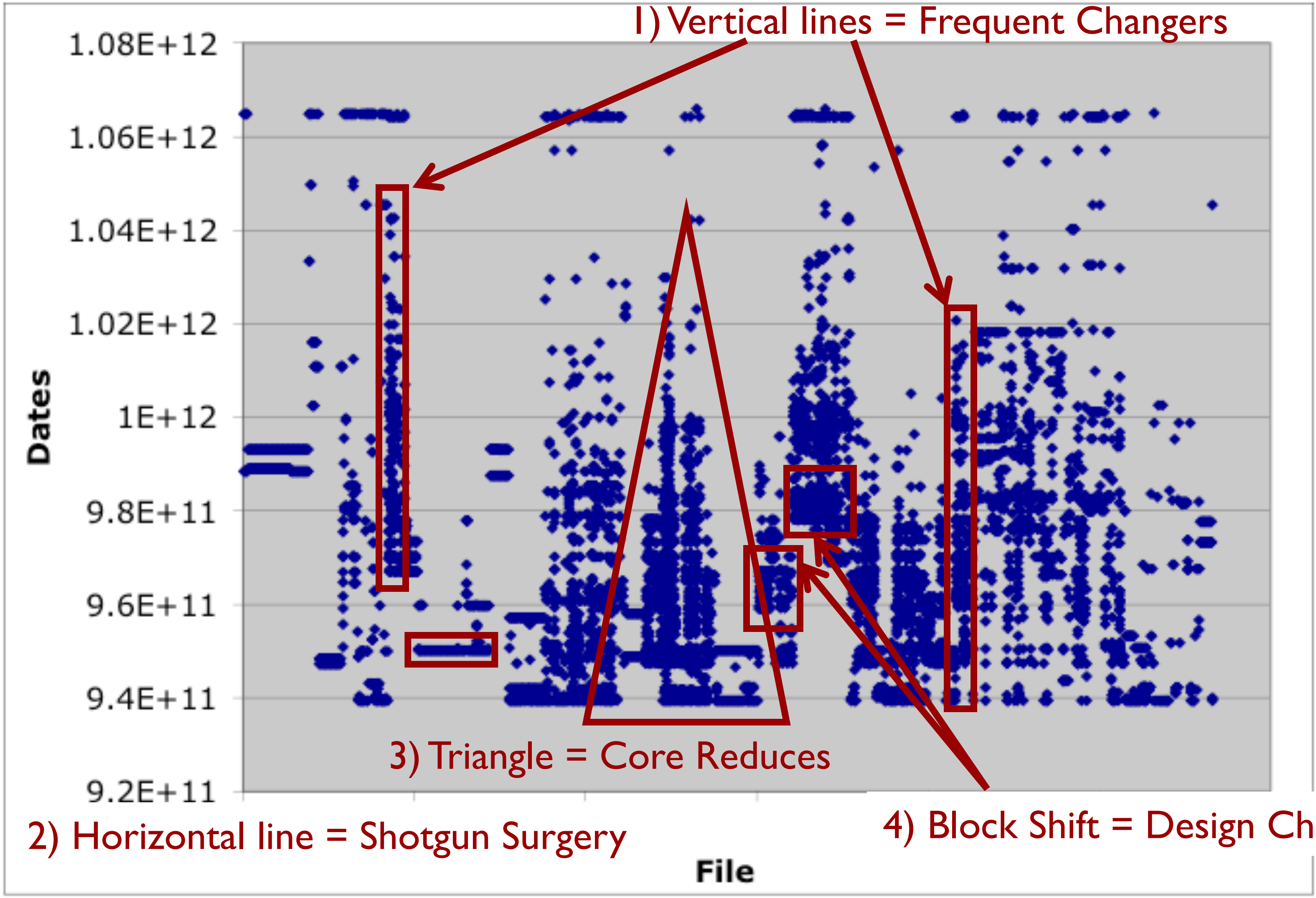


(2) Problem detection

Issues

- scale

Analyse CVS changes



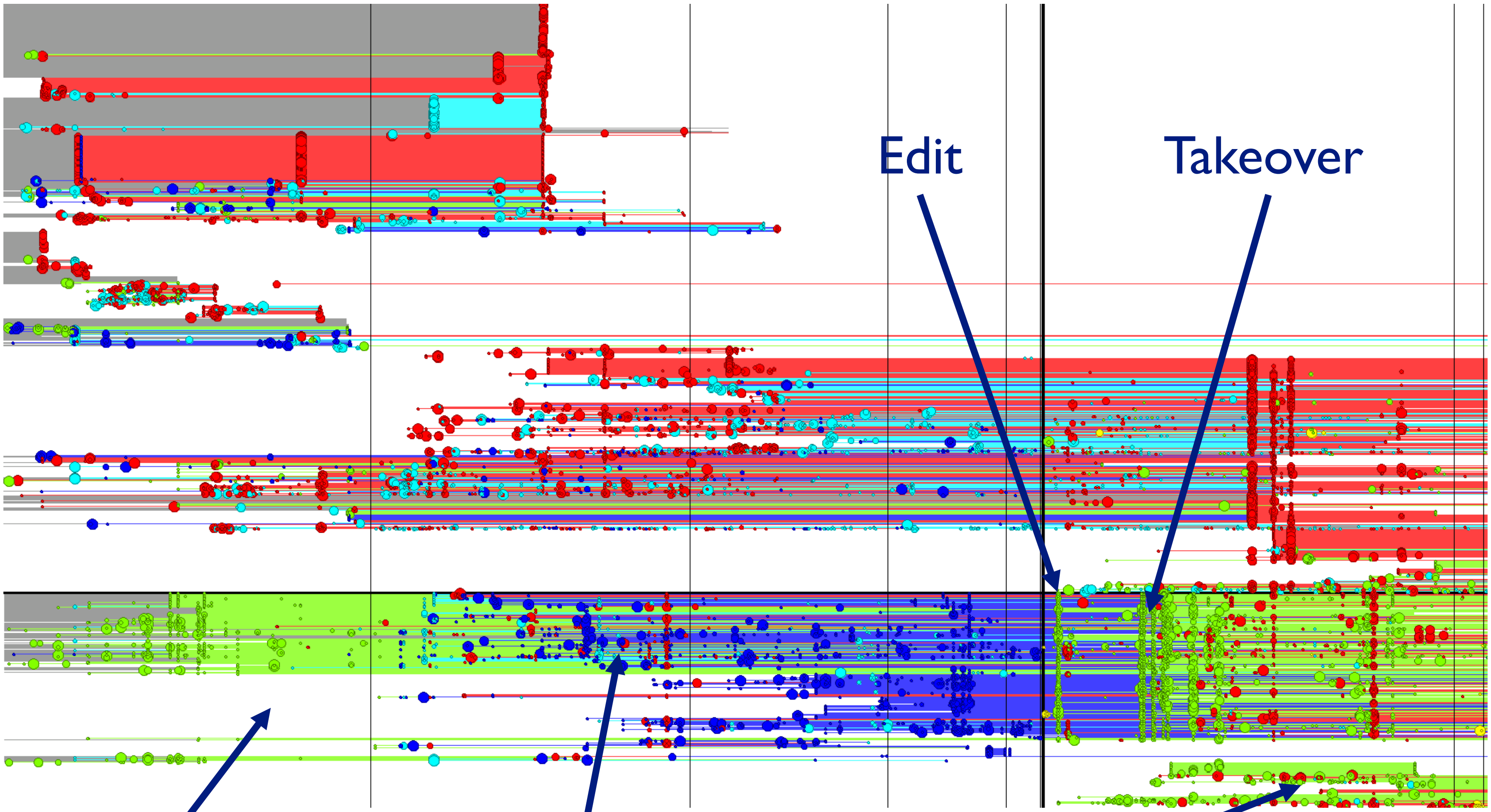
1) Vertical lines = Frequent Changers

2) Horizontal line = Shotgun Surgery

3) Triangle = Core Reduces

4) Block Shift = Design Change

Ownership Map: Developer Activity



Edit

Takeover

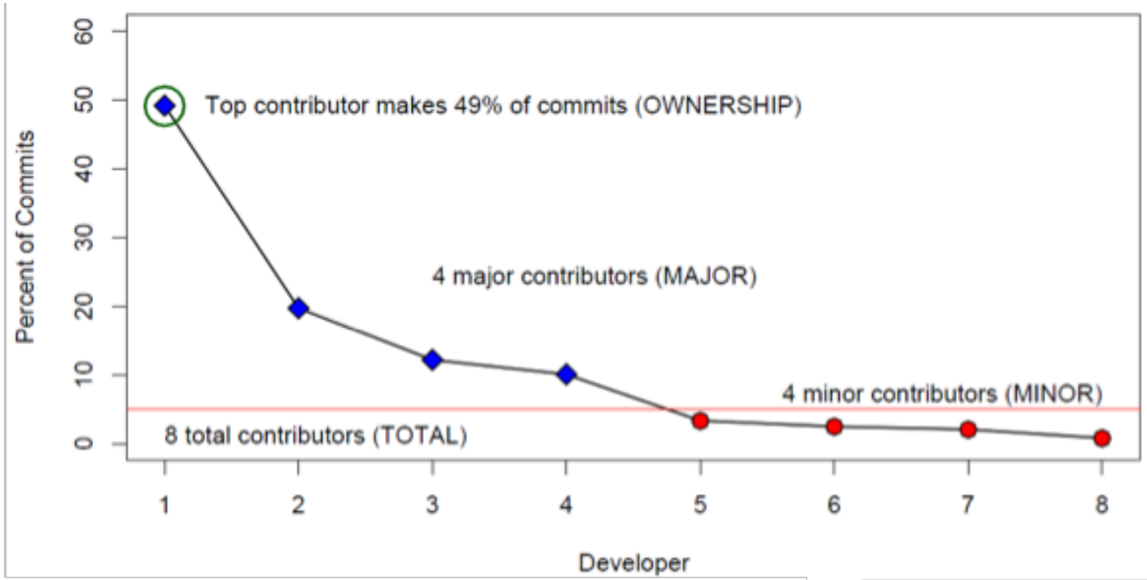
Monologue

Familiarization

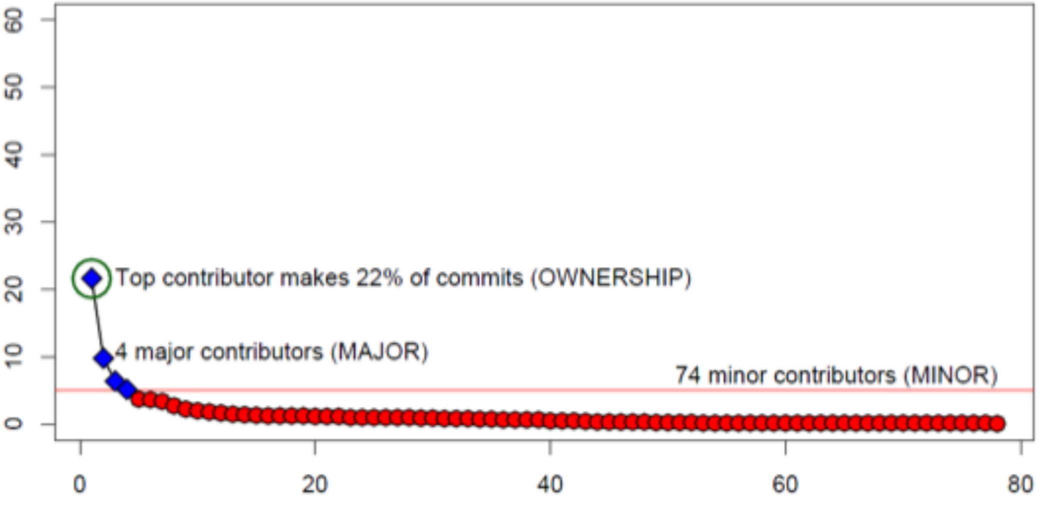
Dialogue

Data from Windows Vista and Windows 7

What to (re)test ?

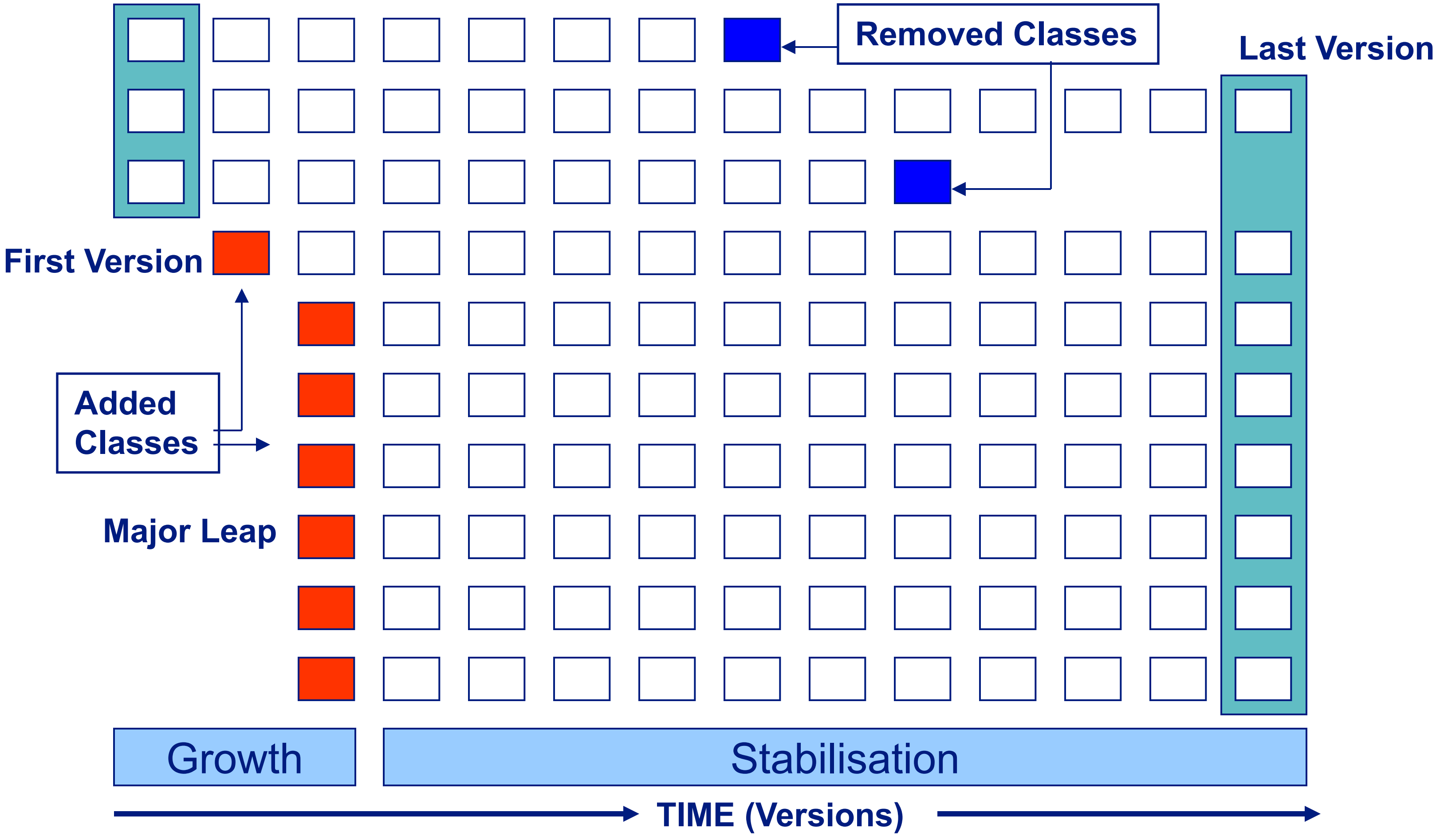


Software components with a high level of ownership will have fewer failures than components with lower top ownership levels.

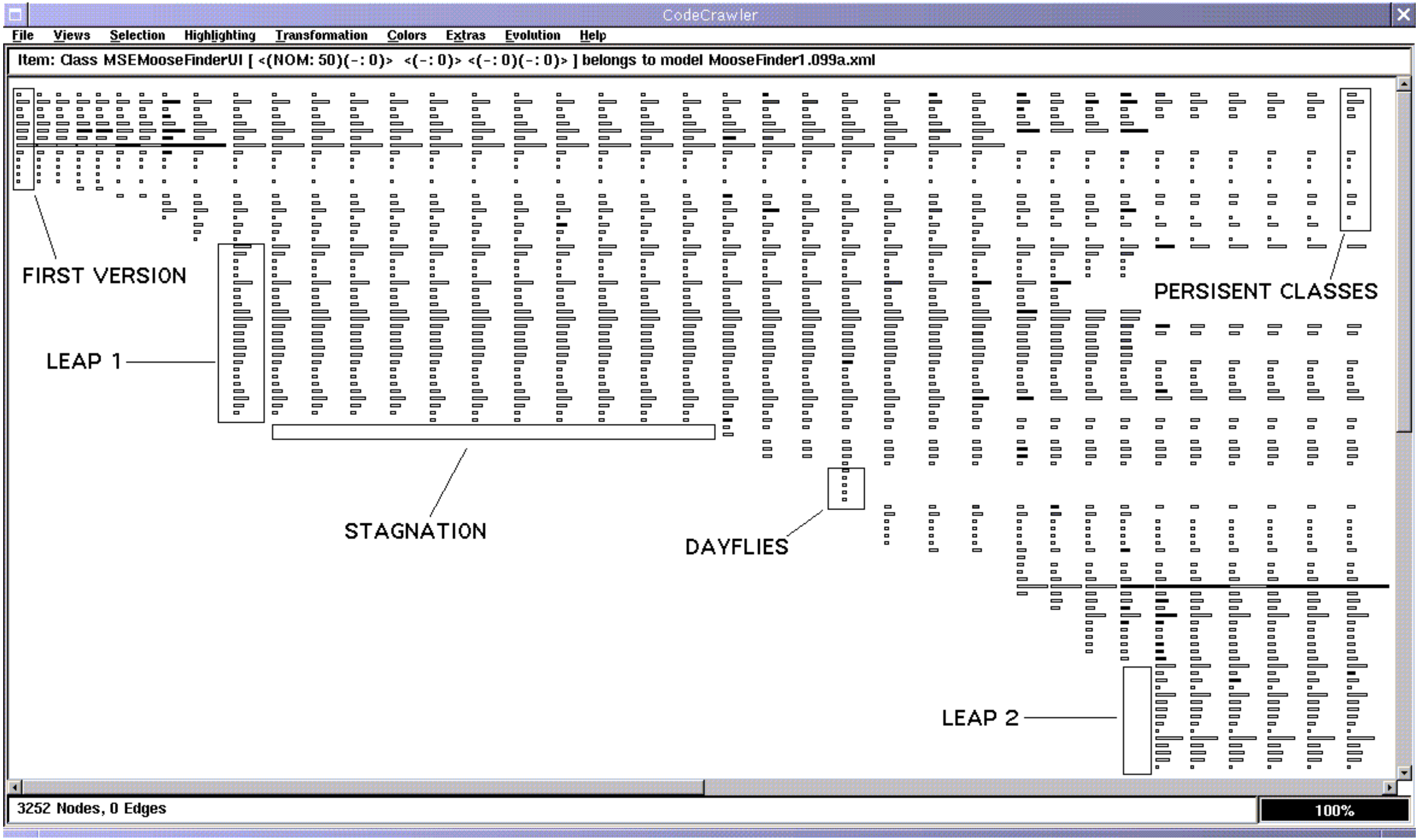


Software components with many minor contributors will have more failures than software components that have fewer.

The Evolution Matrix



Example: MooseFinder (38 Versions)

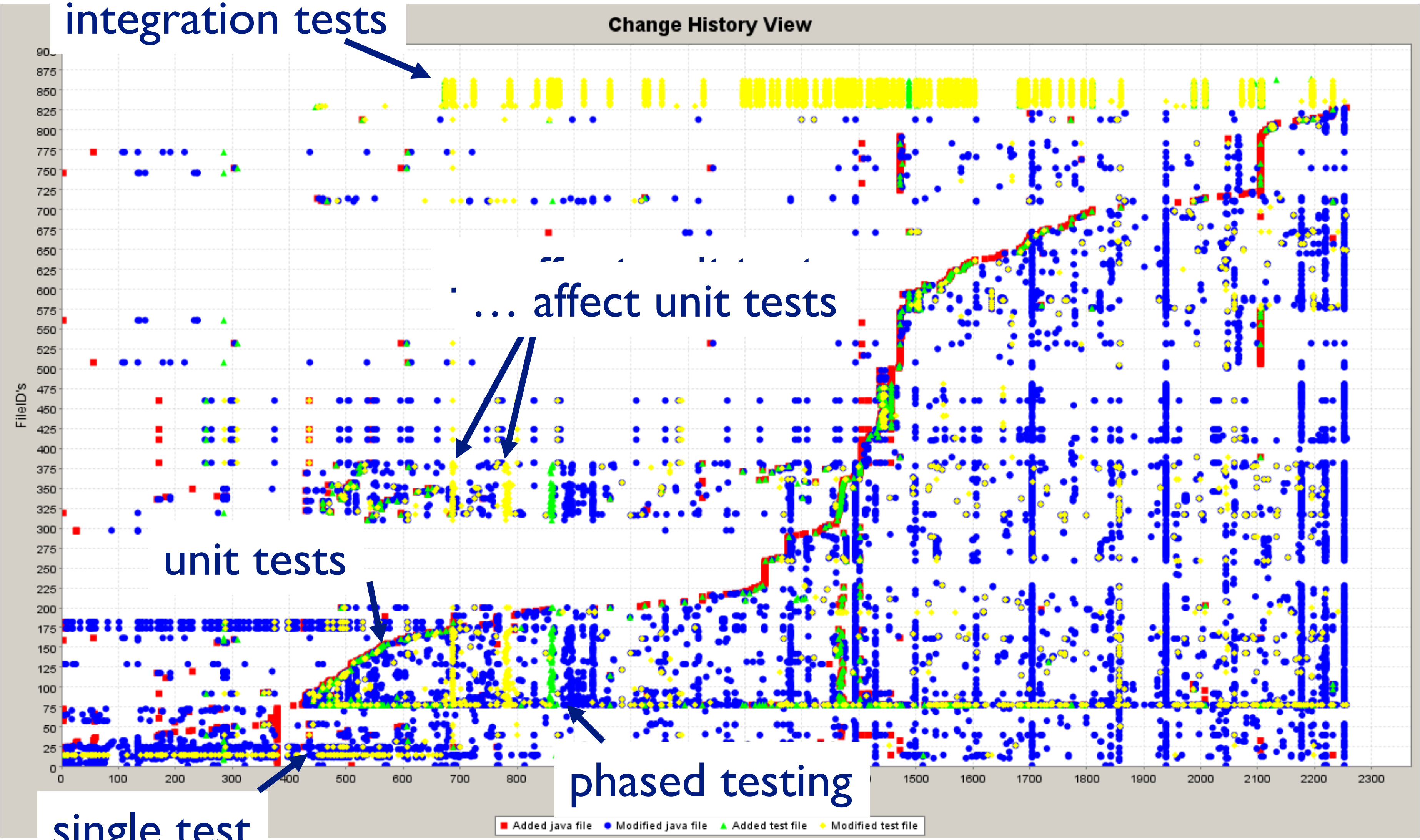


Item: Class MSEMooseFinderUI [<(NOM: 50)(-: 0)> <(-: 0)> <(-: 0)(-: 0)>] belongs to model MooseFinder1.099a.xml

3252 Nodes, 0 Edges

System under study = checkstyle

Test history



integration tests

… affect unit tests

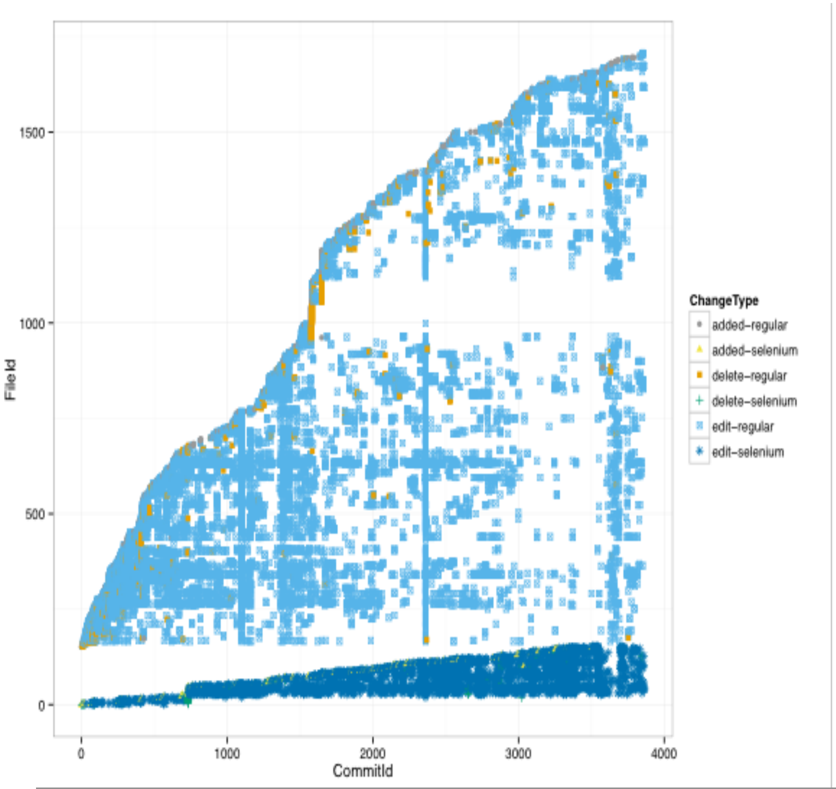
unit tests

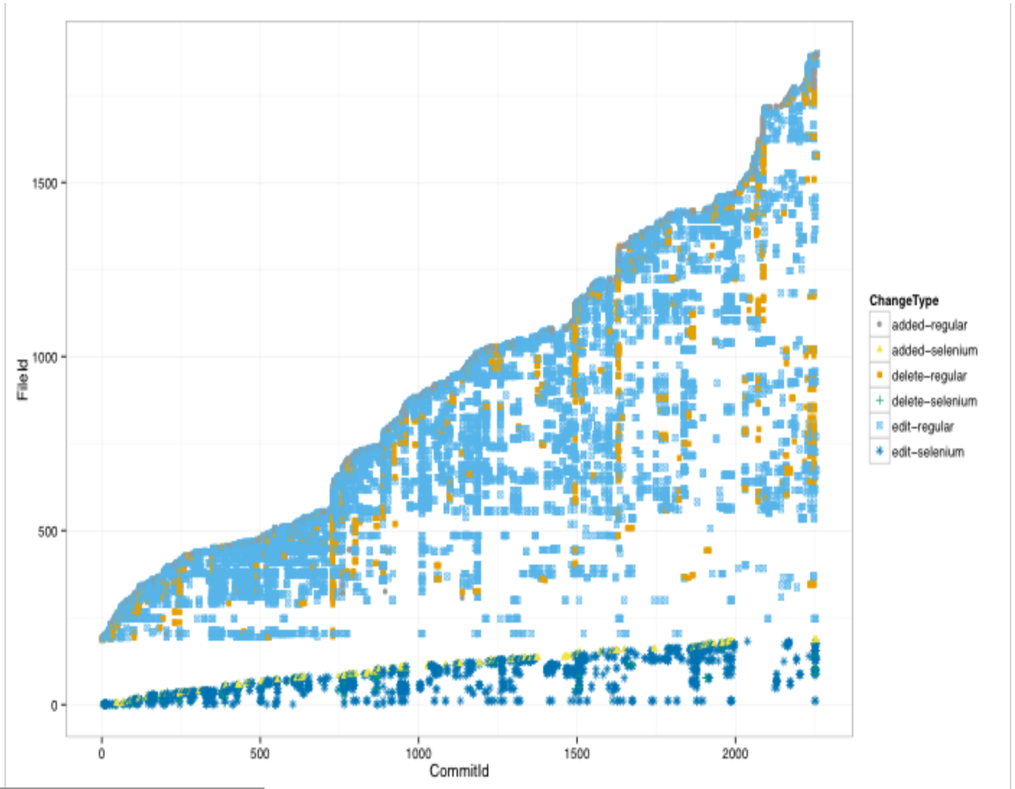
phased testing

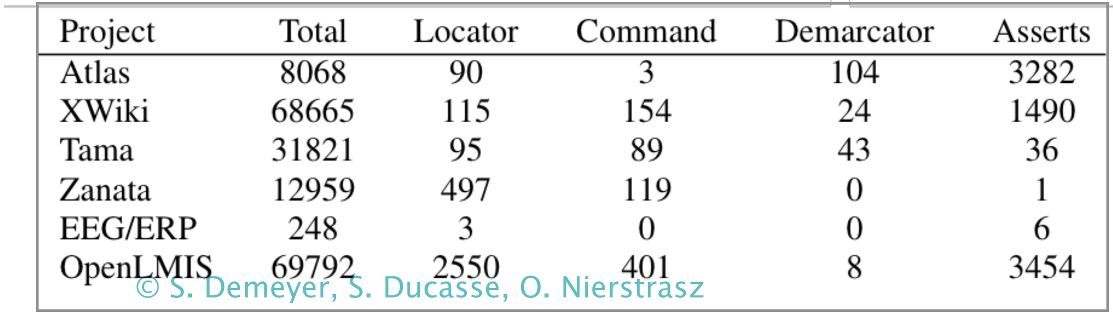
single test

Selenium Tests

Git repositories of the XWiki, OpenLMIS and Atlas
© Laurent Christophe (Vrije Universiteit Brussel)





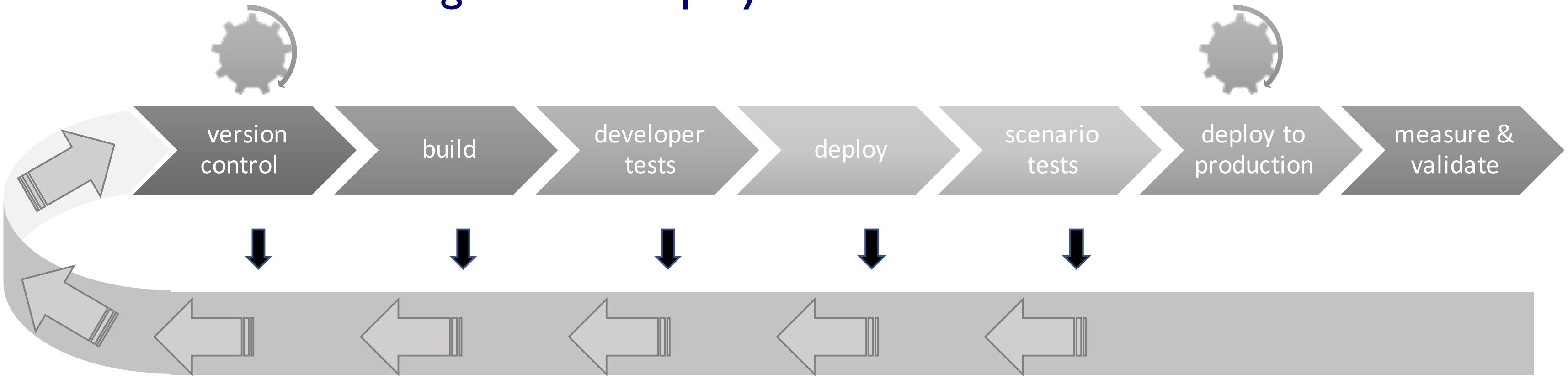


Project	Total	Locator	Command	Demarcator	Asserts
Atlas	8068	90	3	104	3282
XWiki	68665	115	154	24	1490
Tama	31821	95	89	43	36
Zanata	12959	497	119	0	1
EEG/ERP	248	3	0	0	6
OpenLMIS	69792	2550	401	8	3454

Avoid Magic Constants !!

8. Going Agile

- Continuous Integration / Deployment



<<Breaking the Build>>

Mining Software Repositories

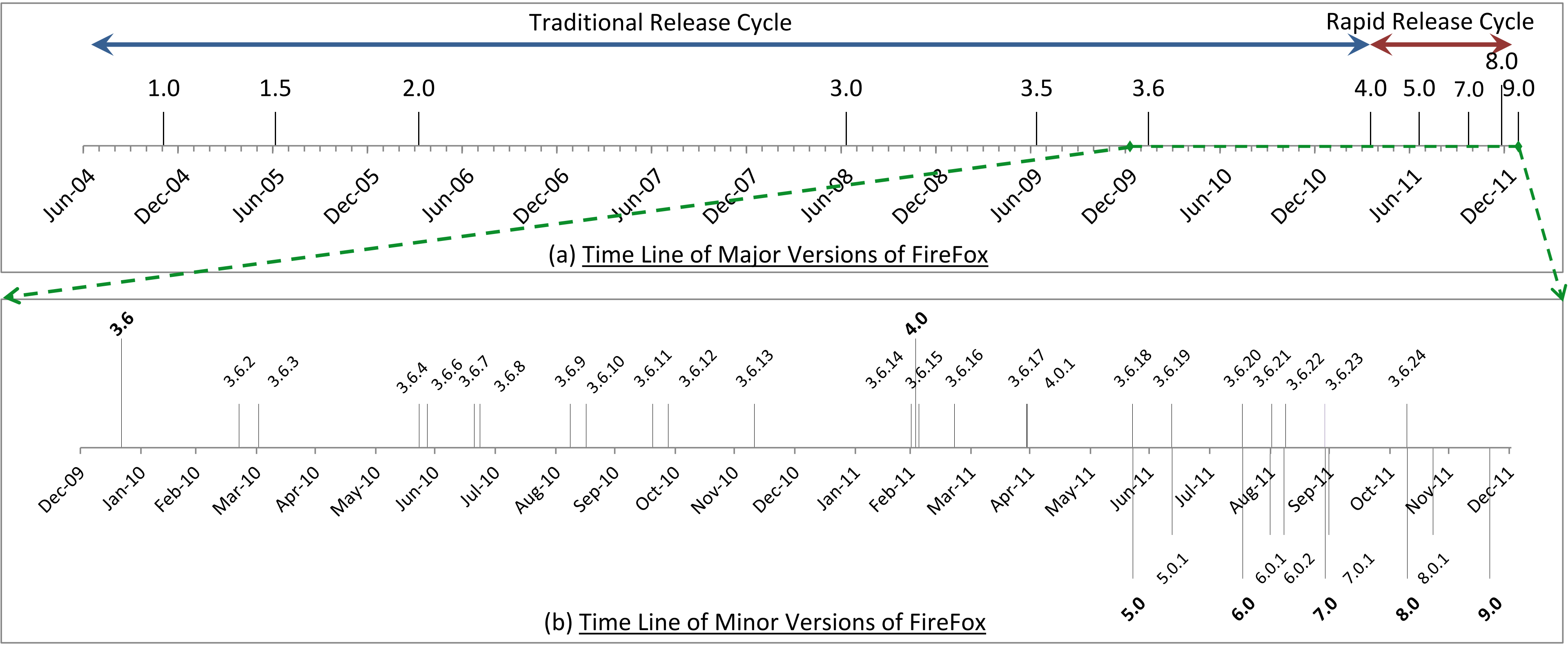
The Mining Repositories (MSR) field analyzes the rich data available in software repositories to uncover interesting and actionable information about software systems and projects.

Conferences

2018—15th edition, Gothenburg, Sweden
2017—14th edition, Buenos Aires, Argentina
2016—13th edition, Austin, Texas
2015—12th edition, Florence, Italy
2014—11th edition, Hyderabad, India
2013—10th edition, San Francisco, USA
2012—9th edition, Zürich, CH
2011—8th edition, Honolulu, HI, USA
2010—7th edition, Cape Town, ZAF
2009—6th edition, Vancouver, CAN
2008—5th edition, Leipzig, DEU
2007—4th edition, Minneapolis, MN, USA
2006—3rd edition, Shanghai, CHN
2005—2nd edition, Saint Luis, MO, USA
2004—1st edition, Edinburgh, UK

Hall of Fame — Mining Challenge

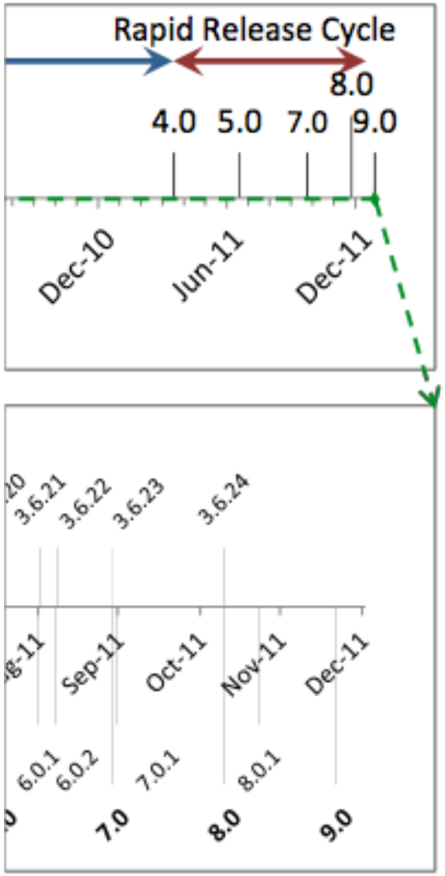
2018 — IDE Event Stream (JetBrains)
2017 — TravisTorrent (Github)
2016 — BOA (SourceForge & Github)
2015 — StackOverflow
2014 — GitHub
2013 — StackOverflow
2012 — Android
2011 — Netbeans+Eclipse
2010 — GNOME Projects
2009 — GNOME project
2008 — Eclipse
2007 — Eclipse Developer
2006 — PostgreSQL

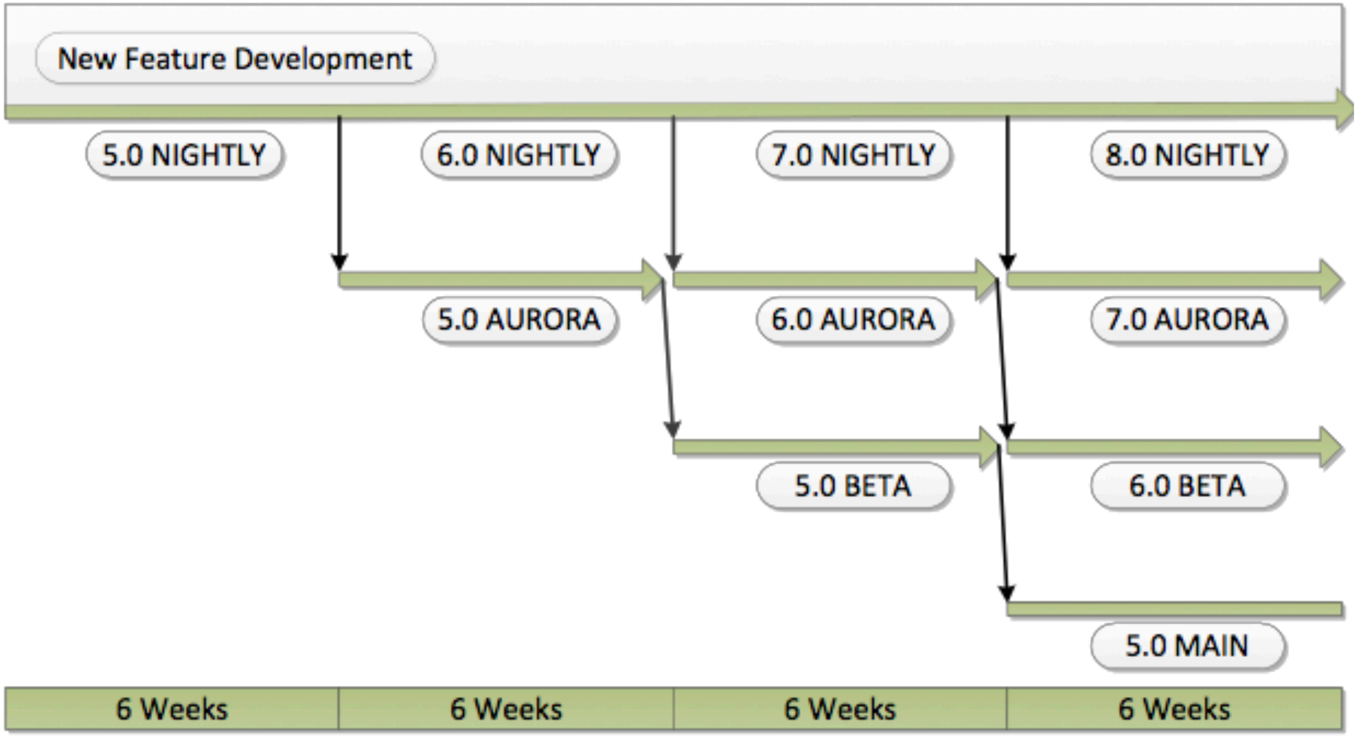


(a) Time Line of Major Versions of FireFox

(b) Time Line of Minor Versions of FireFox

[Khom2014] Khomh, F. Adams, B, Dhaliwal, T and Zou, Y
Understanding the Impact of Rapid Releases on Software Quality: The Case of Firefox, Empirical Software Engineering, Springer.
http://link.springer.com/article/10.1007/s10664-014-9308-x





New Feature Development

5.0 NIGHTLY

6.0 NIGHTLY

7.0 NIGHTLY

8.0 NIGHTLY

5.0 AURORA

6.0 AURORA

7.0 AURORA

5.0 BETA

6.0 BETA

5.0 MAIN

6 Weeks | 6 Weeks | 6 Weeks | 6 Weeks

✓ bugs are fixed faster

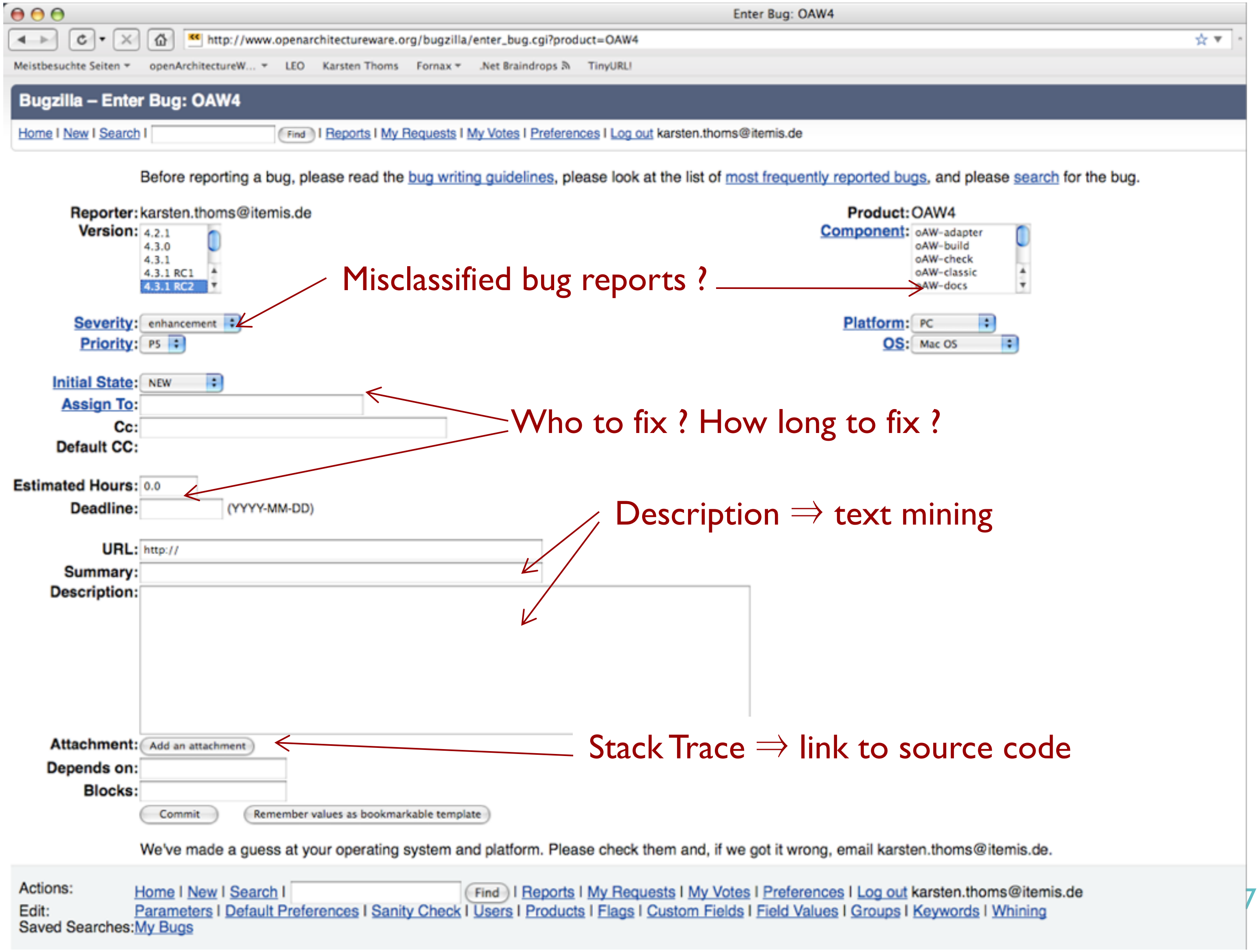
(but … harder bugs propagated to later releases)

✓ amount of pre- & post-release bugs ± the same

✓ the program crashes earlier

(perhaps due to recent features)

Recommender Systems



Enter Bug: OAW4

http://www.openarchitectureware.org/bugzilla/enter_bug.cgi?product=OAW4

Meistbesuchte Seiten | openArchitectureW... | LEO | Karsten Thoms | Fornax | .Net Braindrops | TinyURL!

Bugzilla – Enter Bug: OAW4

Home | New | Search | [ ] Find | Reports | My Requests | My Votes | Preferences | Log out karsten.thoms@itemis.de

Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently reported bugs, and please search for the bug.

Reporter: karsten.thoms@itemis.de

Product: OAW4

Version: 4.2.1, 4.3.0, 4.3.1, 4.3.1 RC1, 4.3.1 RC2

Component: oAW-adapter, oAW-build, oAW-check, oAW-classic, oAW-docs

Misclassified bug reports ?

Severity: enhancement

Priority: P5

Platform: PC

OS: Mac OS

Initial State: NEW

Assign To:

Cc:

Default CC:

Who to fix ? How long to fix ?

Estimated Hours: 0.0

Deadline: (YYYY-MM-DD)

Description ⇒ text mining

URL: http://

Summary:

Description:

Attachment: Add an attachment

Stack Trace ⇒ link to source code

Depends on:

Blocks:

Commit | Remember values as bookmarkable template

We've made a guess at your operating system and platform. Please check them and, if we got it wrong, email karsten.thoms@itemis.de.

Actions: Home | New | Search | [ ] Find | Reports | My Requests | My Votes | Preferences | Log out karsten.thoms@itemis.de

Edit: Parameters | Default Preferences | Sanity Check | Users | Products | Flags | Custom Fields | Field Values | Groups | Keywords | Whining

Saved Searches: My Bugs

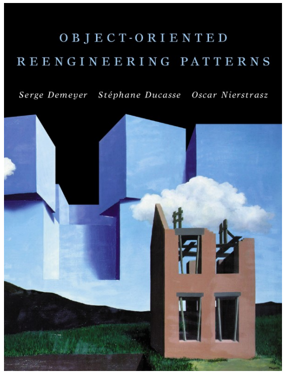
9. Conclusion

1. **Introduction**
   There are OO legacy systems too !
2. **Reverse Engineering**
   How to understand your code
3. **Visualization**
   Scalable approach
4. **Dynamic Analysis**
   To be really certain
5. **Restructuring**
   How to Refactor Your Code
6. **Code Duplication**
   The most typical problems
7. **Software Evolution**
   Learn from the past
8. **Going Agile**
   Continuous Integration
9. **Conclusion**

Goals

We will try to convince you:

- Yes, Virginia, there are *object-oriented legacy systems* too!
  - ☞ … actually, that's a sign of health
- Reverse engineering and reengineering are *essential activities* in the lifecycle of any successful software system. (And especially OO ones!)
  - ☞ … consequently, do not consider it second class work
- There is a large set of *lightweight tools and techniques* to help you with reengineering.
  - ☞ … check our book, but remember the list is growing
- Despite these tools and techniques, *people must do job* and represent the most valuable resource.
  - ☞ … pick them carefully and reward them properly



⇒ Did we convince you ?