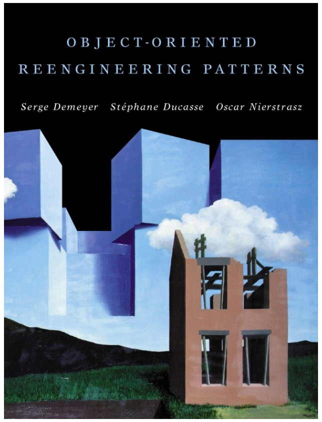
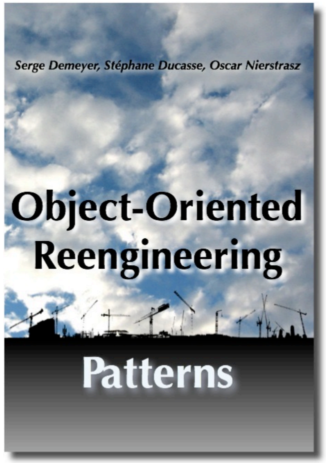
Software Product Design and Development II
CS 473





John Businge

August 2022

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

1ST
TIME

Schedule

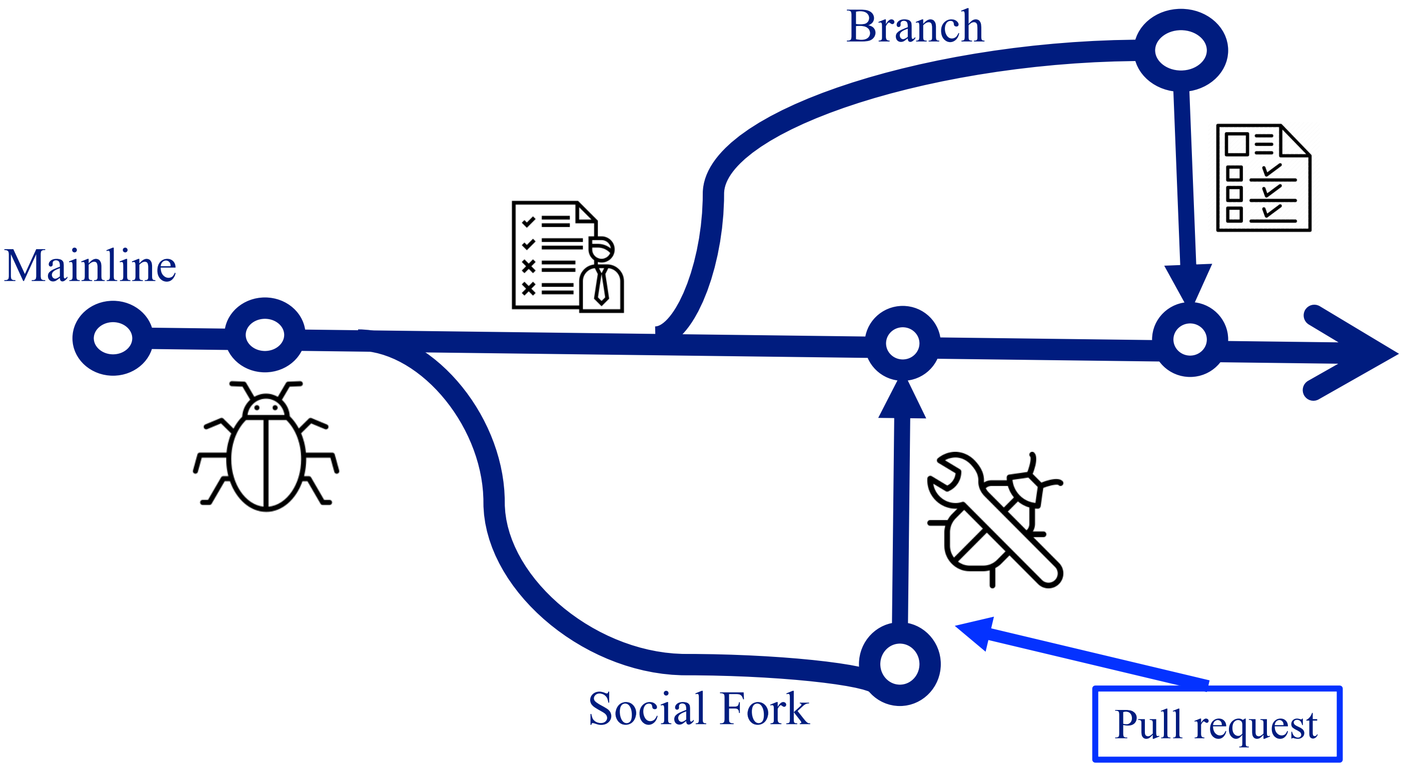
1. **Introduction**
   Software changes and that requires planning
2. **Reverse Engineering**
   How to understand your code
3. **Visualization**
   Scalable approach
4. **Restructuring**
   How to Refactor Your Code
5 . **Code Integration**
   How to resolve conflicts
6. Dynamic Analysis (& Testing)
   To be really certain
7. **Mining Software Repositories**
   Learn from the past
8. **Conclusion**

Goals

We will try to convince you:

- Programs change!
- Reverse engineering, forward 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.

Program Change



Software Maintenance - Cost

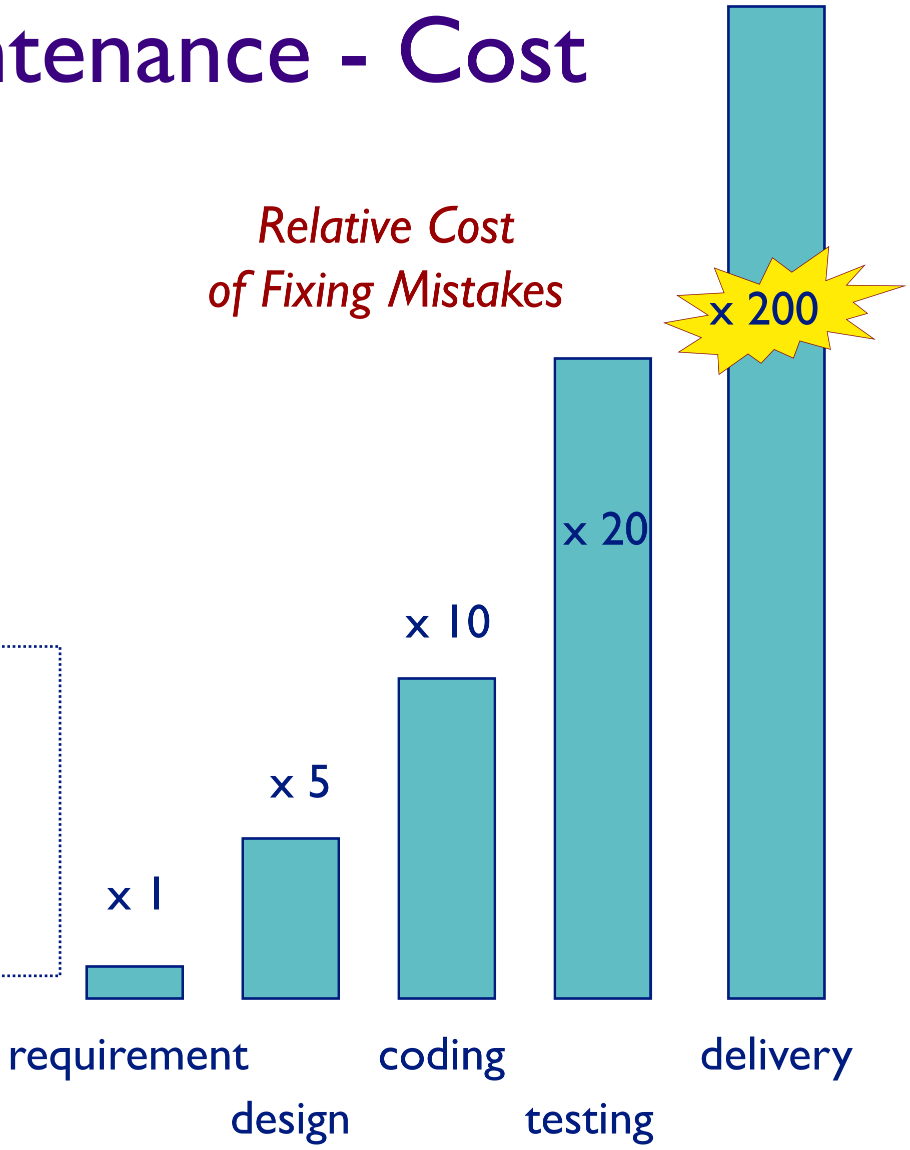
Relative Maintenance Effort

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

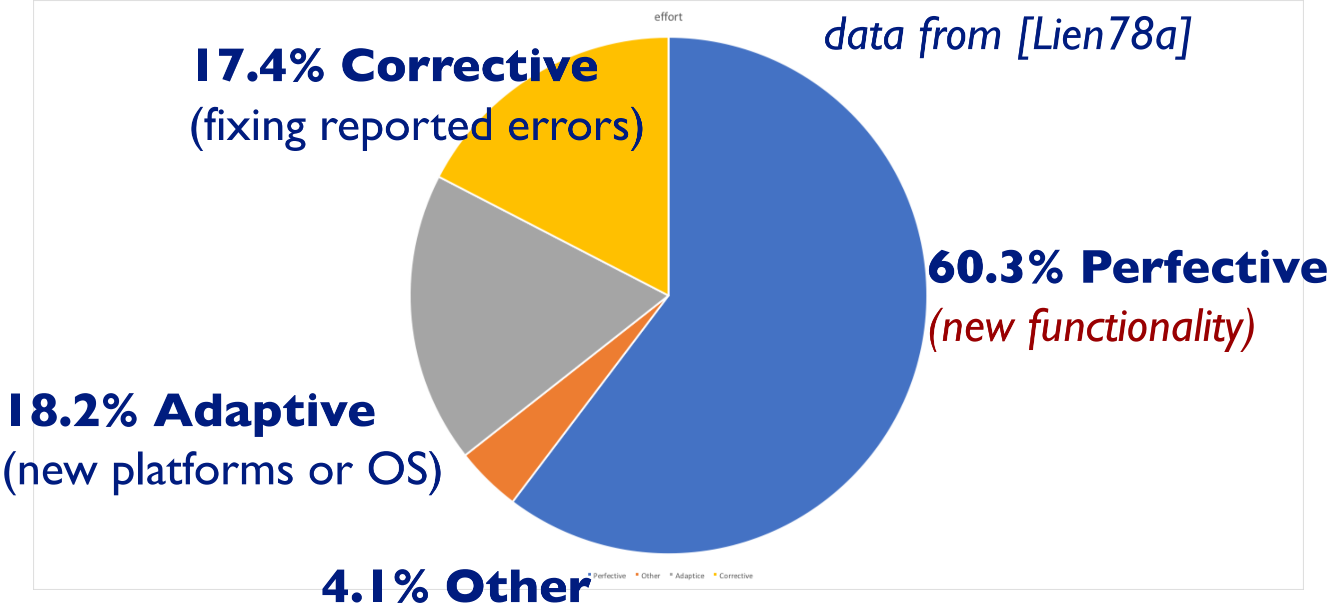
Solution ?

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

Relative Cost of Fixing Mistakes



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

Lehman's Laws

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

Continuing change

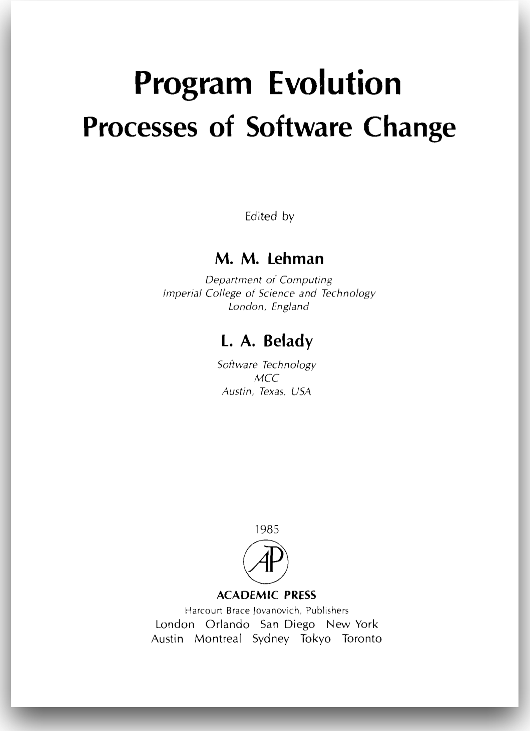
Darwin

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

Increasing complexity

Enthropy

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



Program Evolution
Processes of Software Change

Edited by

M. M. Lehman
Department of Computing
Imperial College of Science and Technology
London, England

L. A. Belady
Software Technology
MCC
Austin, Texas, USA

1985

ACADEMIC PRESS
Harcourt Brace Jovanovich, Publishers
London Orlando San Diego New York
Austin Montreal Sydney Tokyo Toronto

Those laws are still applicable…

Lehman's Laws

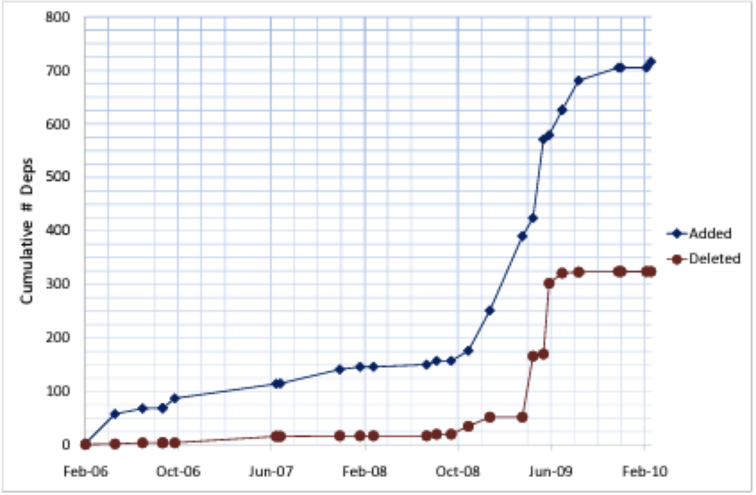


Figure 1: Cumulative Added and Deleted Deps to Eclim

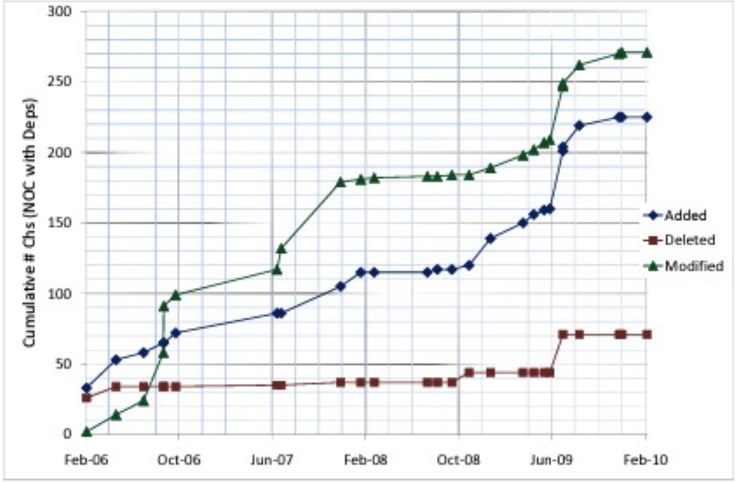
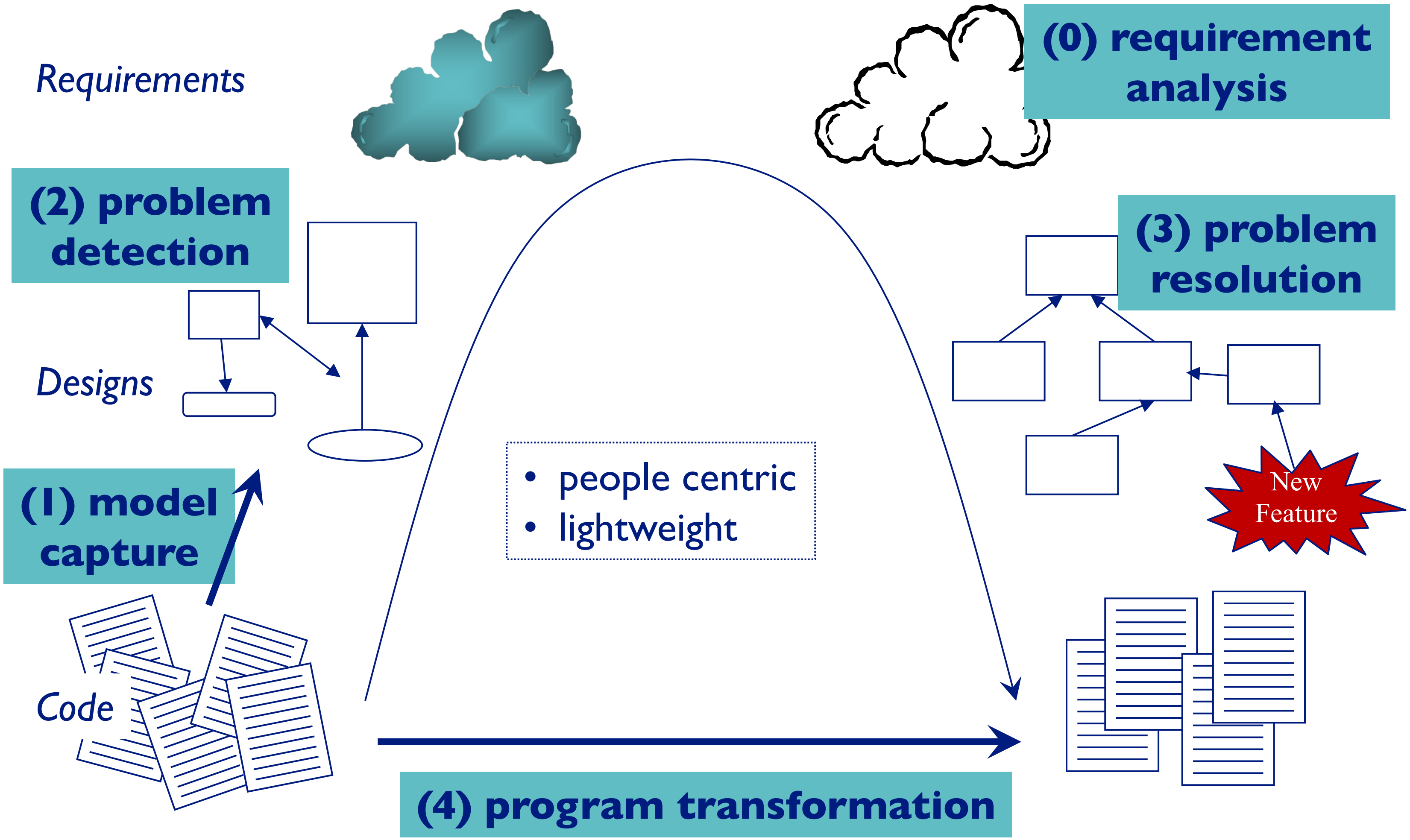


Figure 2: Cumulative changes of classes with Deps in Eclim

Businge et al. An empirical study of the evolution of Eclipse third-party plug-ins, IWPSE - 2010

The Reengineering Life-Cycle



Requirements

Designs

Code

(0) requirement analysis

(1) model capture

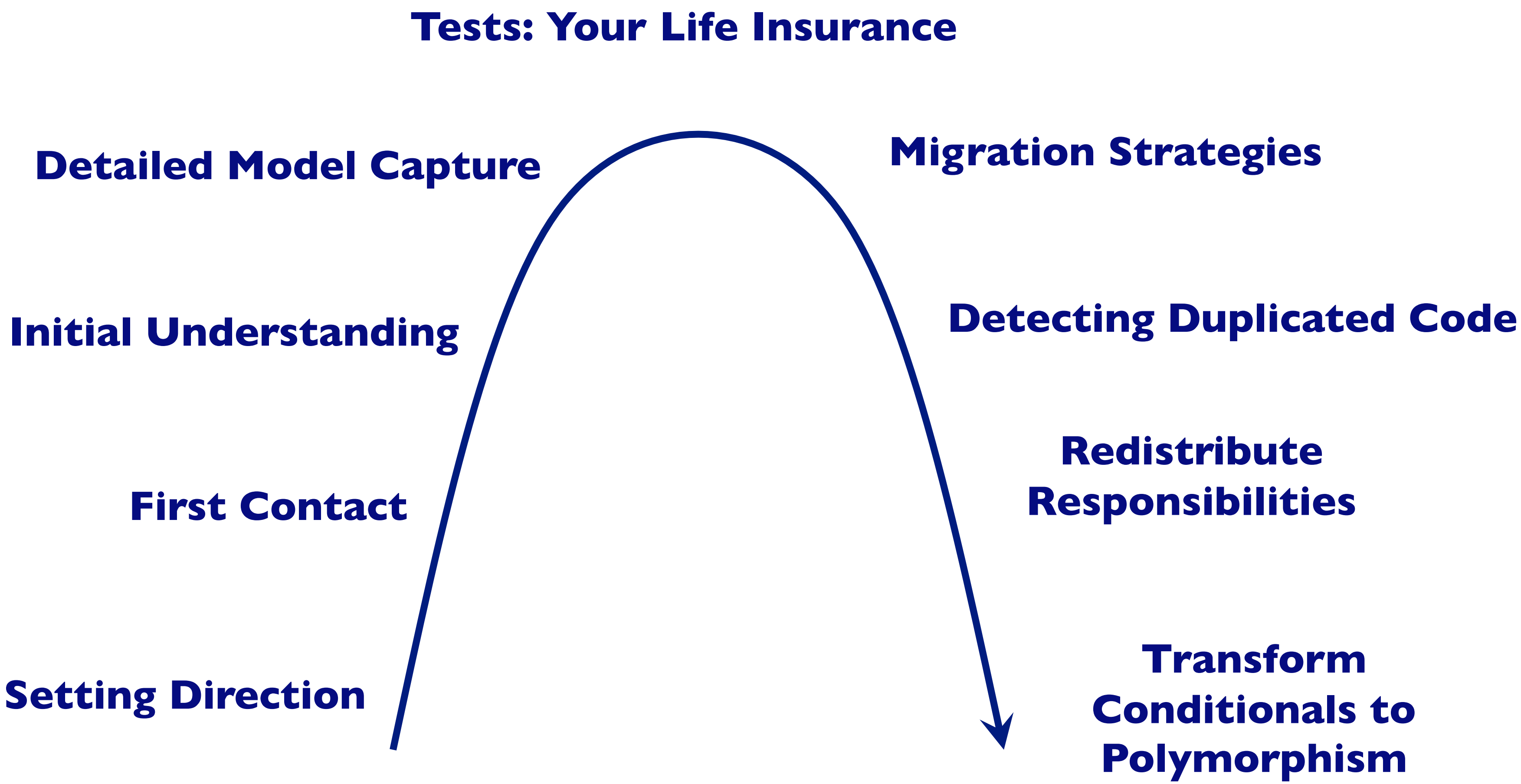
(2) problem detection

(3) problem resolution

(4) program transformation

- people centric
- lightweight

A Map of Reengineering Patterns



Setting Direction

First Contact

Initial Understanding

Detailed Model Capture

Tests: Your Life Insurance

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

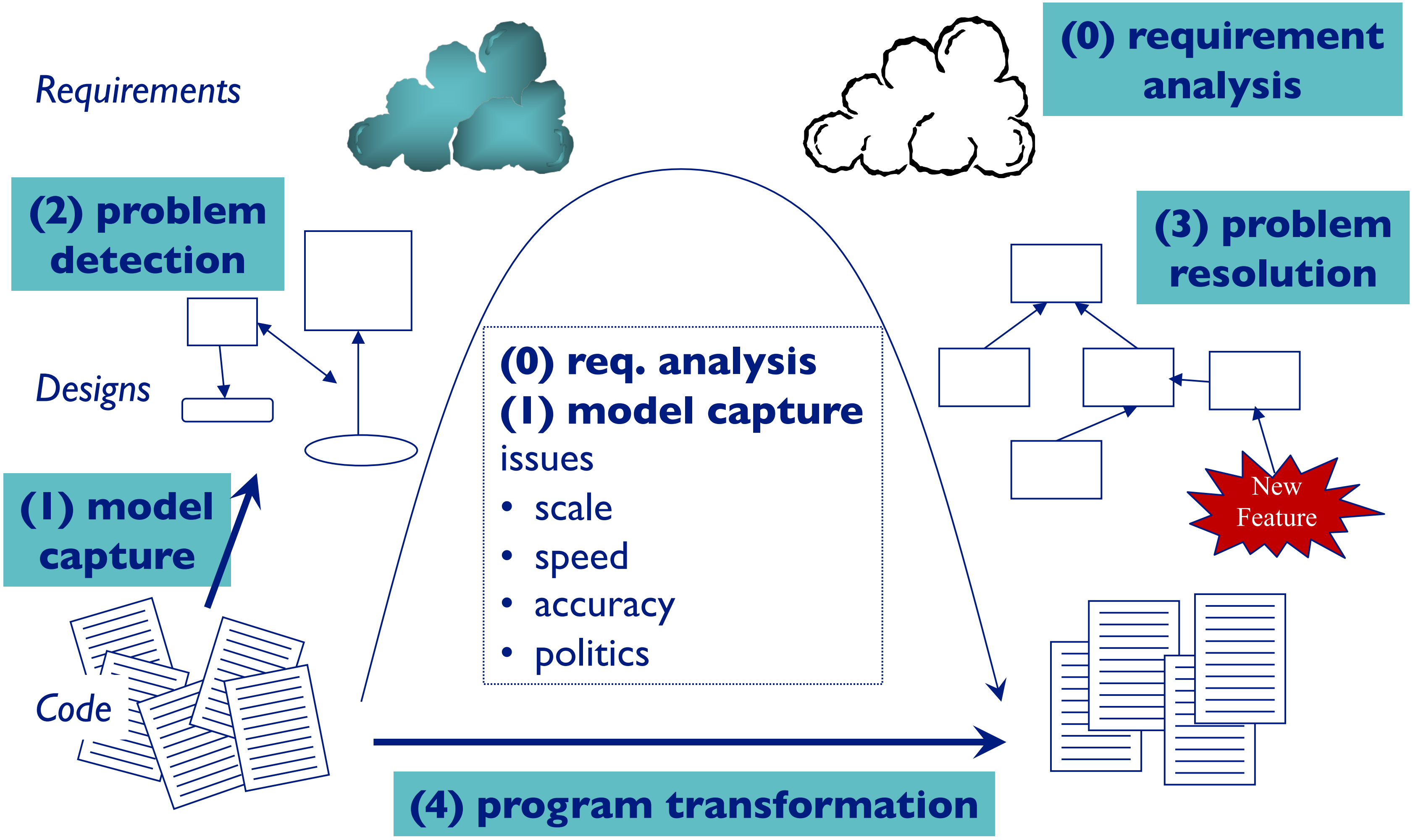
Motivation

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

Designs

Code

(0) requirement analysis

(2) problem detection

(3) problem resolution

(1) model capture

(4) program transformation

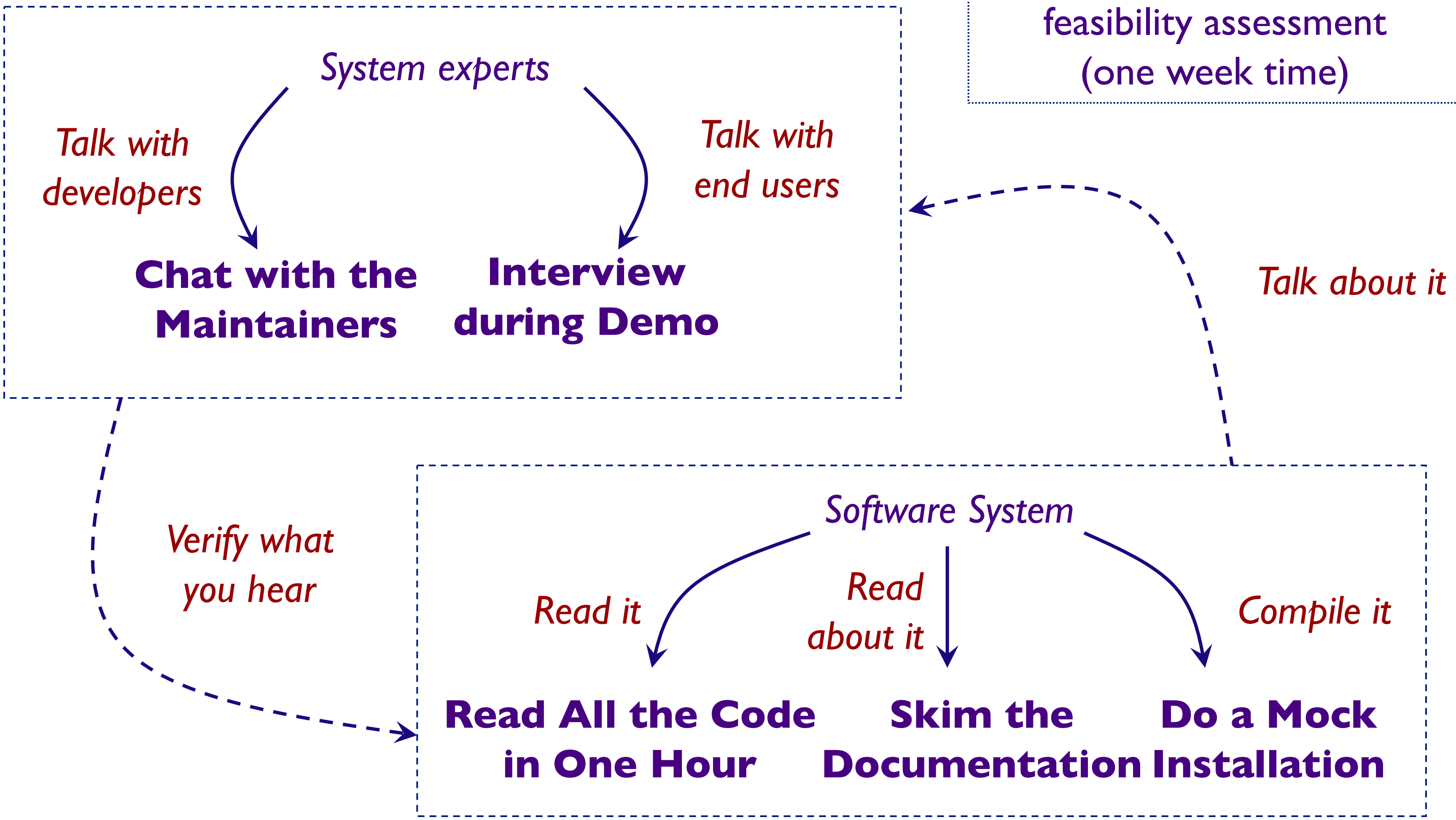
(0) req. analysis
(1) model capture

issues

- scale
- speed
- accuracy
- politics

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

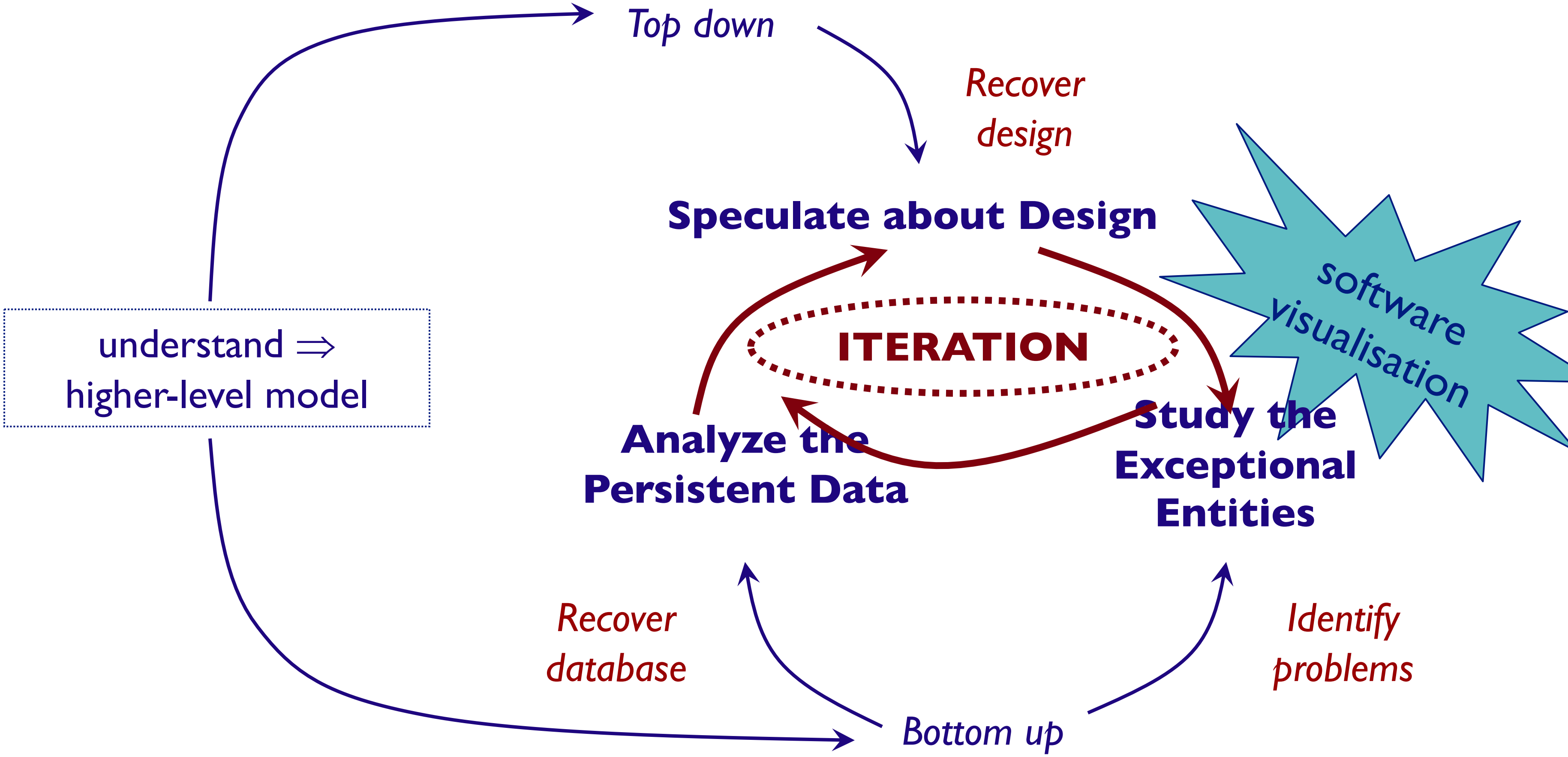
Problem: What are the typical usage scenarios?

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



Top down

Recover design

Speculate about Design

ITERATION

Analyze the Persistent Data

Study the Exceptional Entities

software visualisation

understand ⇒ higher-level model

Recover database

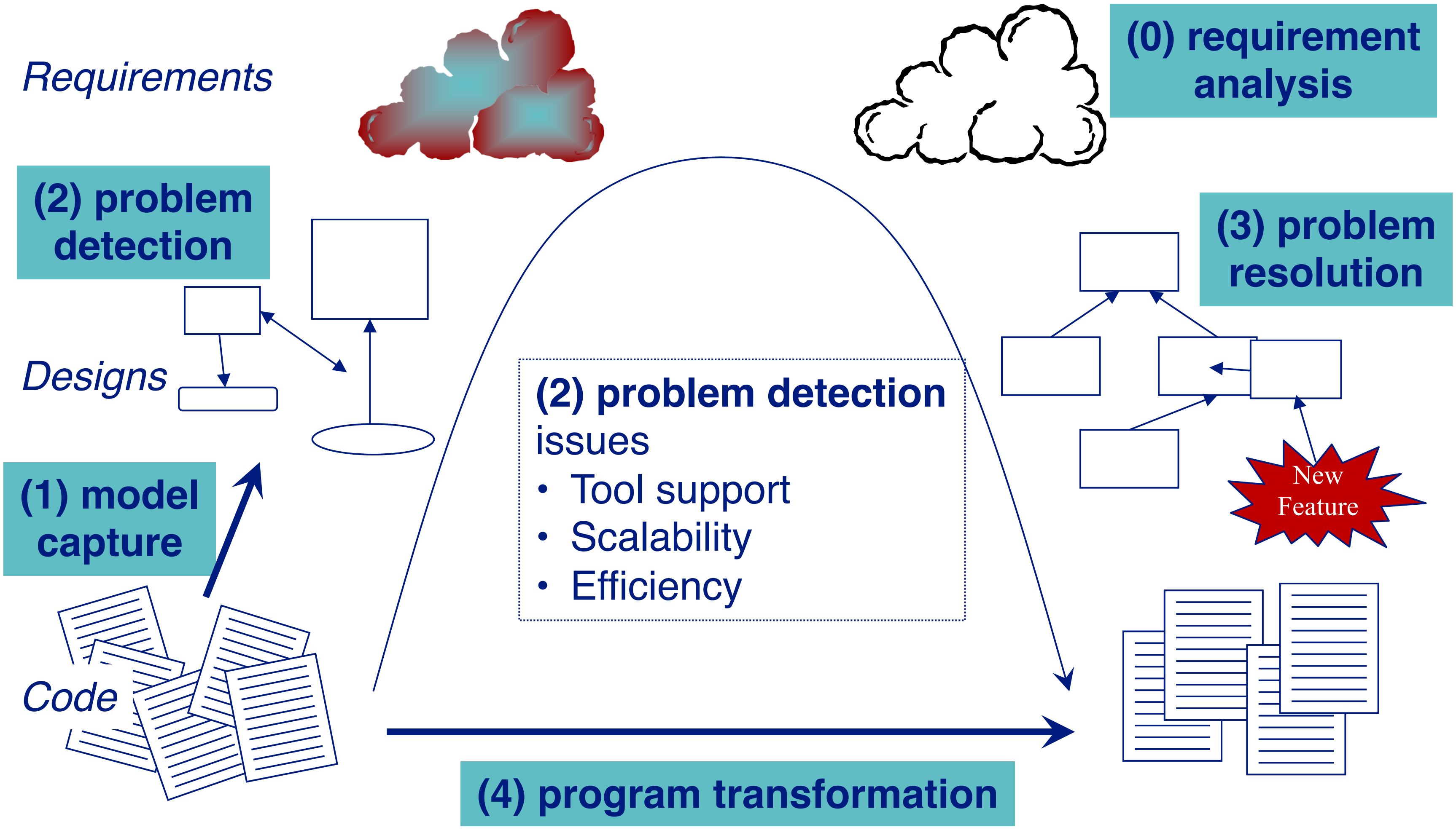
Identify problems

Bottom up

3. Software Visualization

- Introduction
  - The Reengineering life-cycle
- Examples
- Lightweight Approaches
  - tooling

The Reengineering Life-cycle



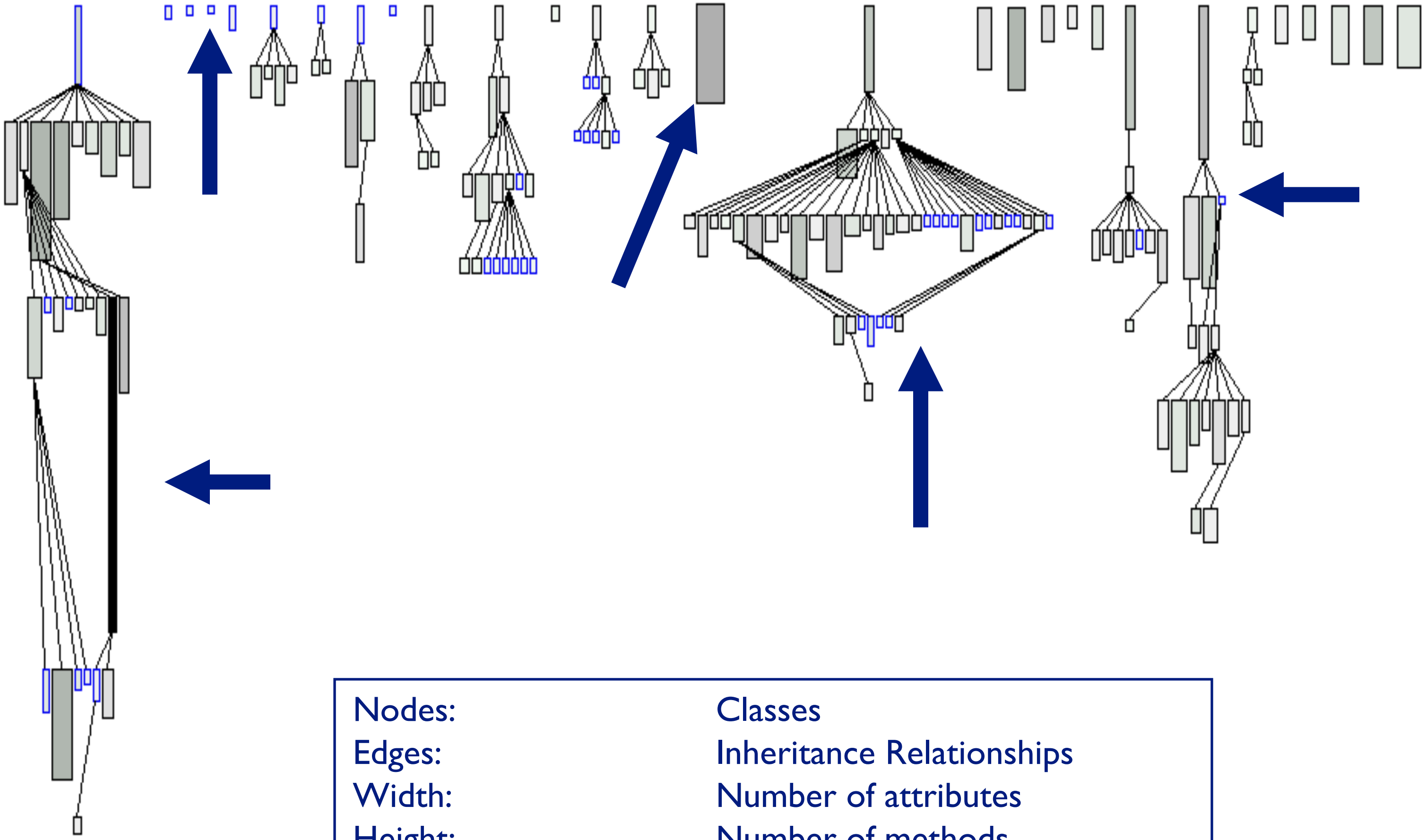
(2) problem detection issues

- Tool support
- Scalability
- Efficiency

UML Diagrams

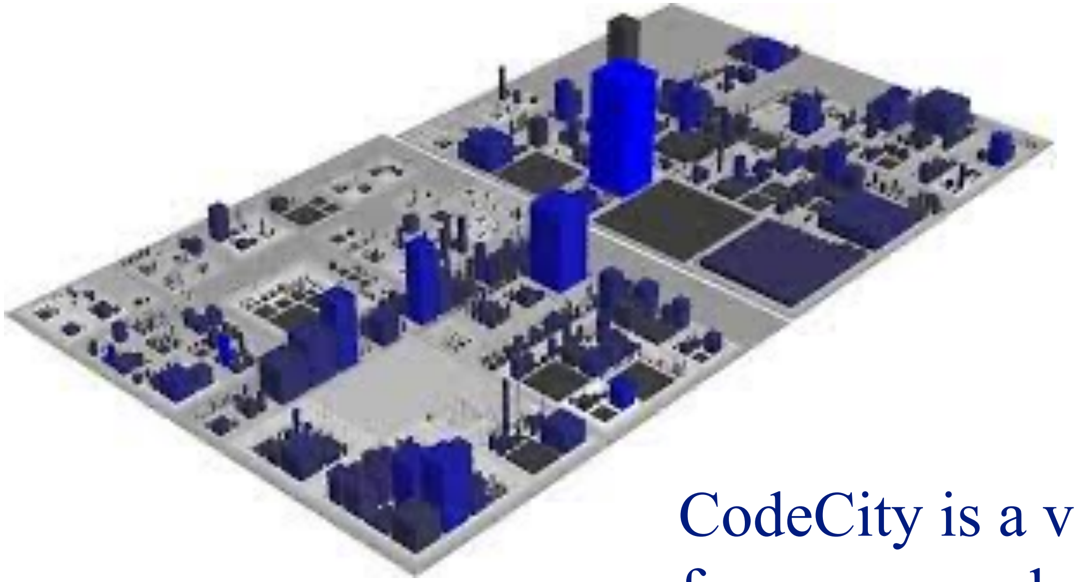
- (Mostly) Simple and Standard Way to present an abstract visualization of a system
- UML defines 14 diagrams
- Useful to plan and design the reengineering project
- You will be using UML diagrams to show the system before and after the change

System Complexity View



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

Code City



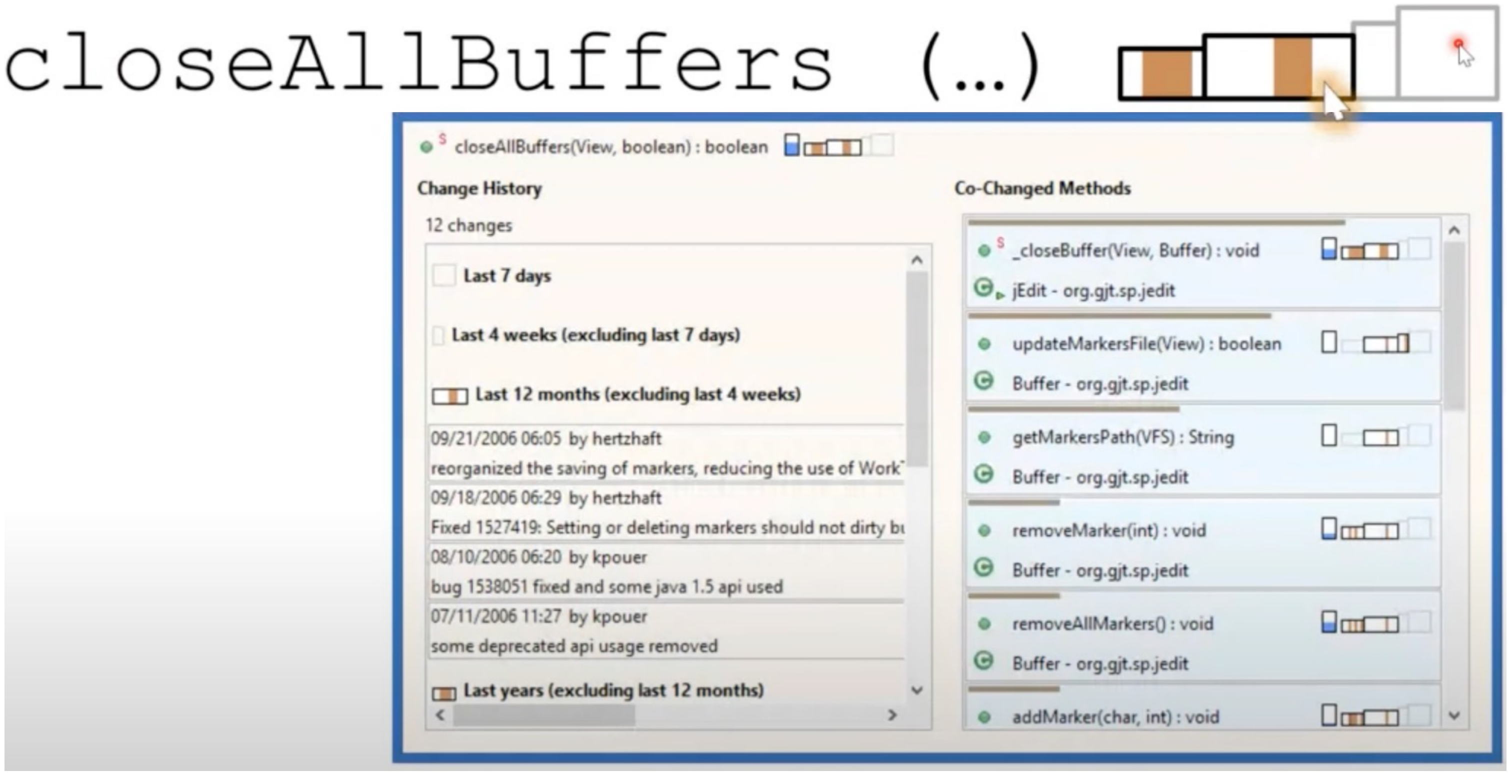
CodeCity is a visualization concept for source code.

The source code is shown as an interactive 3D city.

Code City

- Packages are “districts”, “neighborhoods,” or “city blocks”
- Each “building” represents a class \
- Width = Number of Attributes
- Height = Number of Methods
- Antennas => Classes with many methods and no attributes
- Parking lot => Classes with many attributes and no methods
- Skyscraper => Classes with a large number of methods and has many attributes

Method change visialization



closeAllBuffers (...)

closeAllBuffers(View, boolean) : boolean

Change History

12 changes

- Last 7 days
- Last 4 weeks (excluding last 7 days)
- Last 12 months (excluding last 4 weeks)

09/21/2006 06:05 by hertzhaft
reorganized the saving of markers, reducing the use of Work

09/18/2006 06:29 by hertzhaft
Fixed 1527419: Setting or deleting markers should not dirty bu

08/10/2006 06:20 by kpouer
bug 1538051 fixed and some java 1.5 api used

07/11/2006 11:27 by kpouer
some deprecated api usage removed

- Last years (excluding last 12 months)

Co-Changed Methods

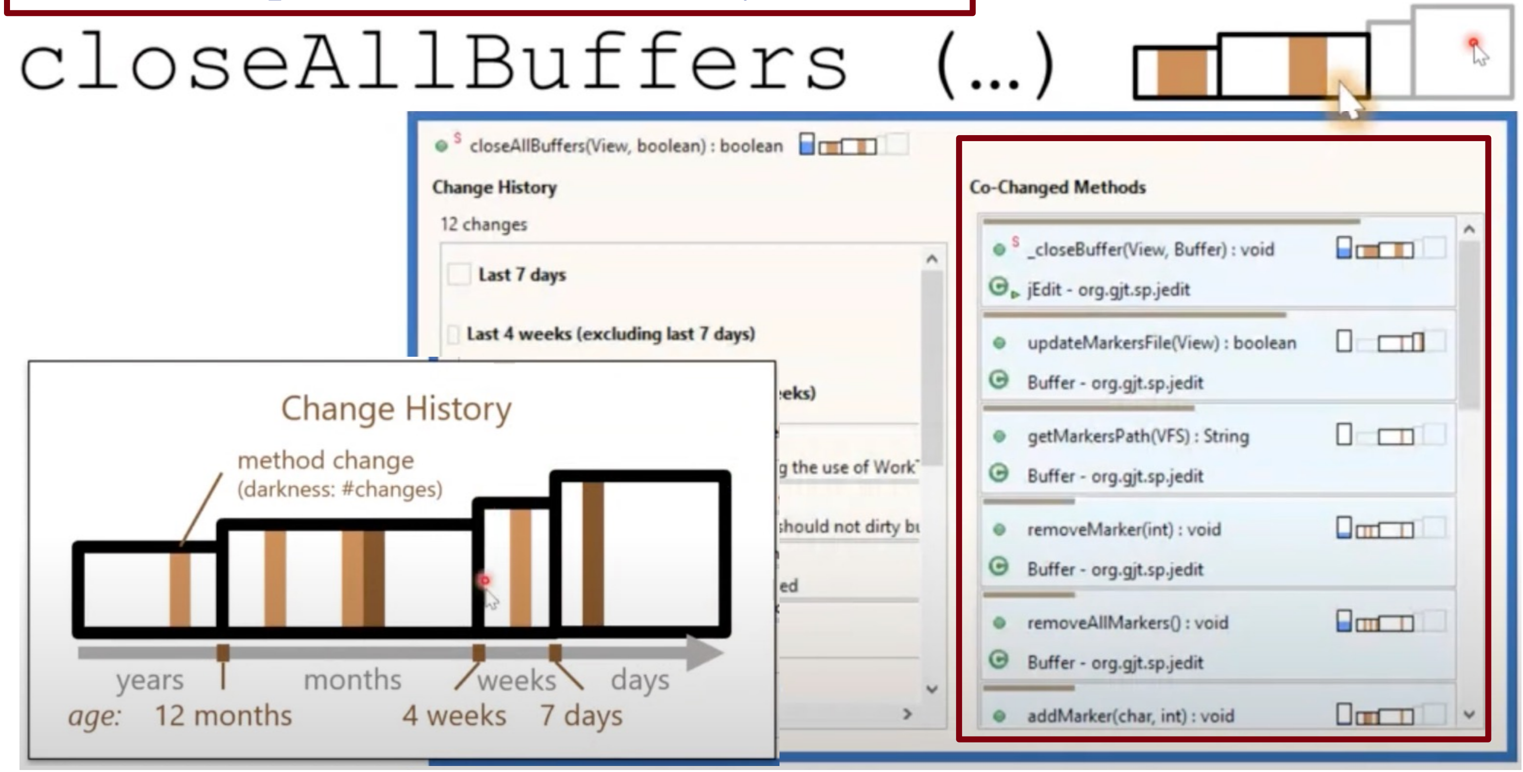
- _closeBuffer(View, Buffer) : void
  jEdit - org.gjt.sp.jedit
- updateMarkersFile(View) : boolean
  Buffer - org.gjt.sp.jedit
- getMarkersPath(VFS) : String
  Buffer - org.gjt.sp.jedit
- removeMarker(int) : void
  Buffer - org.gjt.sp.jedit
- removeAllMarkers() : void
  Buffer - org.gjt.sp.jedit
- addMarker(char, int) : void

Embedding Evolutionary Context

Beck et al. Rethinking User Interfaces for Feature Location. ICPC 2015

Method change visialization

Observation: Recent history is often important than old history



closeAllBuffers(View, boolean) : boolean

Change History

12 changes

Last 7 days

Last 4 weeks (excluding last 7 days)

eks)

g the use of Work

should not dirty bu

ed

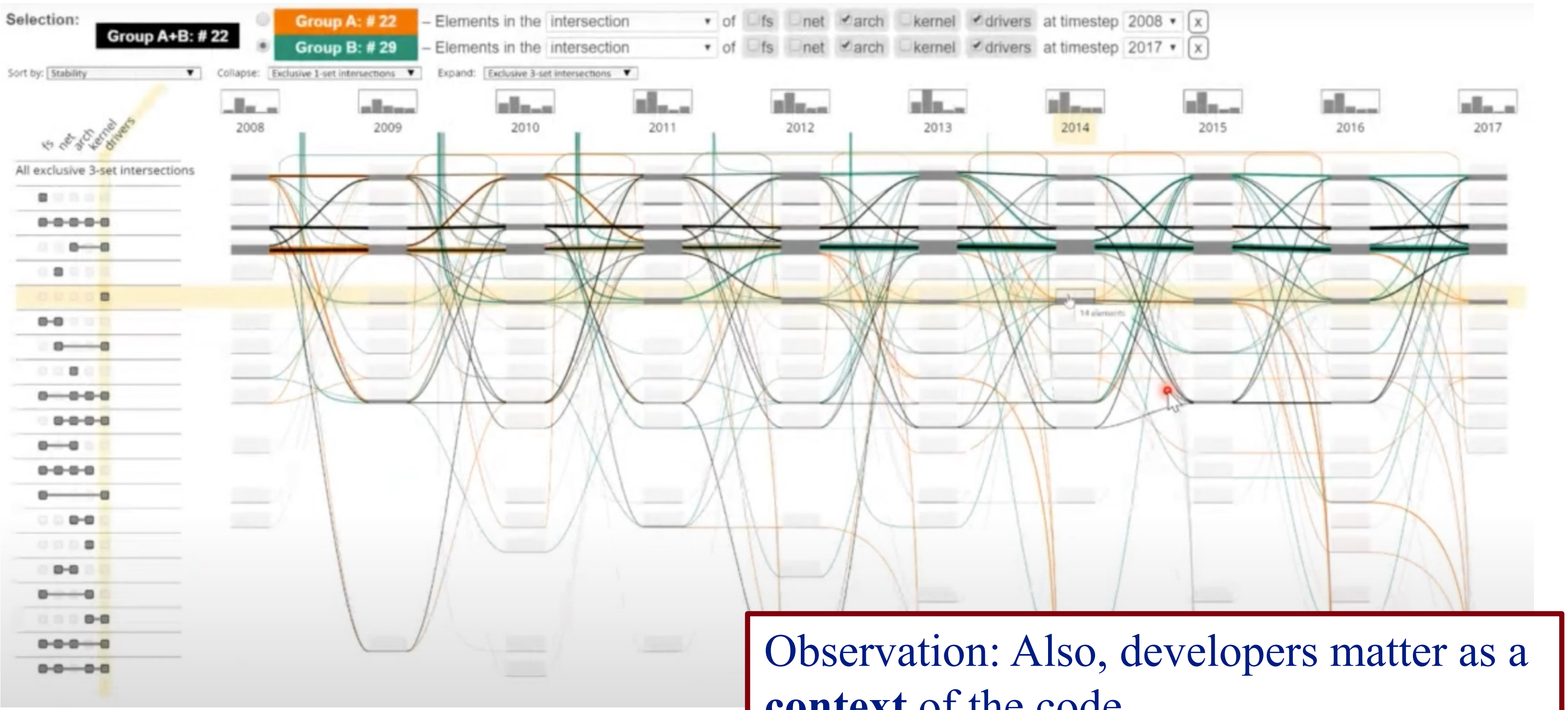
Co-Changed Methods

- _closeBuffer(View, Buffer) : void — jEdit - org.gjt.sp.jedit
- updateMarkersFile(View) : boolean — Buffer - org.gjt.sp.jedit
- getMarkersPath(VFS) : String — Buffer - org.gjt.sp.jedit
- removeMarker(int) : void — Buffer - org.gjt.sp.jedit
- removeAllMarkers() : void — Buffer - org.gjt.sp.jedit
- addMarker(char, int) : void

Embedding Evolutionary Context

Beck et al. Rethinking User Interfaces for Feature Location. ICPC 2015

Software Developers



Selection: **Group A+B: # 22**

○ **Group A: # 22** – Elements in the intersection ▾ of □fs □net ☑arch □kernel ☑drivers at timestep 2008 ▾ x

● **Group B: # 29** – Elements in the intersection ▾ of □fs □net ☑arch □kernel ☑drivers at timestep 2017 ▾ x

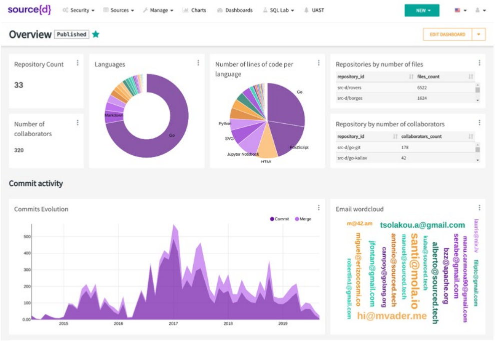
Sort by: Stability ▾ Collapse: Exclusive 1-set intersections ▾ Expand: Exclusive 3-set intersections ▾

Observation: Also, developers matter as a **context** of the code.

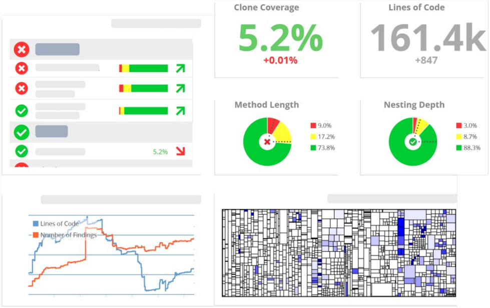
Agarwal, S.; Beck, F.,: *Set Streams: Visual Exploration of Dynamic Overlapping Sets*.
In: Computer Graphics Forum, Jg. 39 (2020) Nr. 3, S. 383-391. doi:10.1111/cgf.13988

State of the Art Tooling

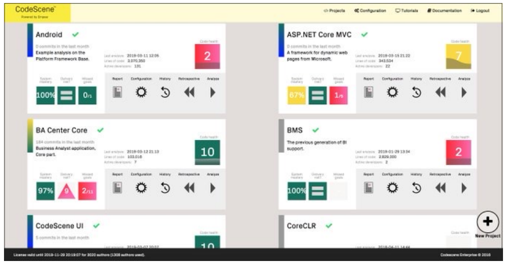
1. source{d}
   https://sourced.tech
   https://github.com/src-d/engine



2. teamscale
   https://www.cqse.eu/
   https://github.com/cqse



3. codescene
   https://codescene.io
   https://github.com/empear-analytics



4. Restructuring

Identifying refactoring targets

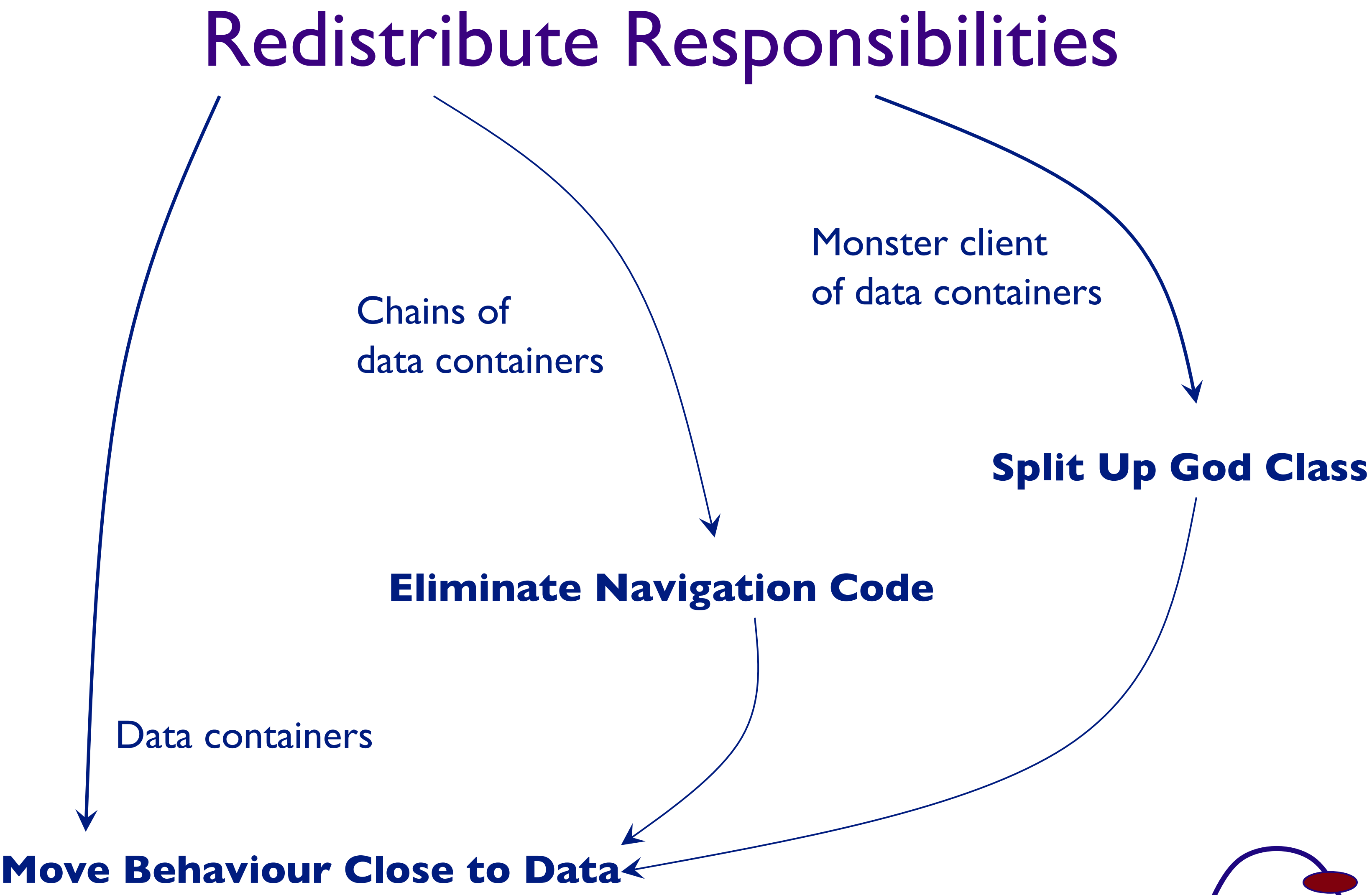
Redistribute Responsibilities

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

Identifying refactorings in code

Refactoring-aware techniques

Identifying Refactoring Targets



Redistribute Responsibilities

Data containers

Chains of data containers

Monster client of data containers

Split Up God Class

Eliminate Navigation Code

Move Behaviour Close to Data

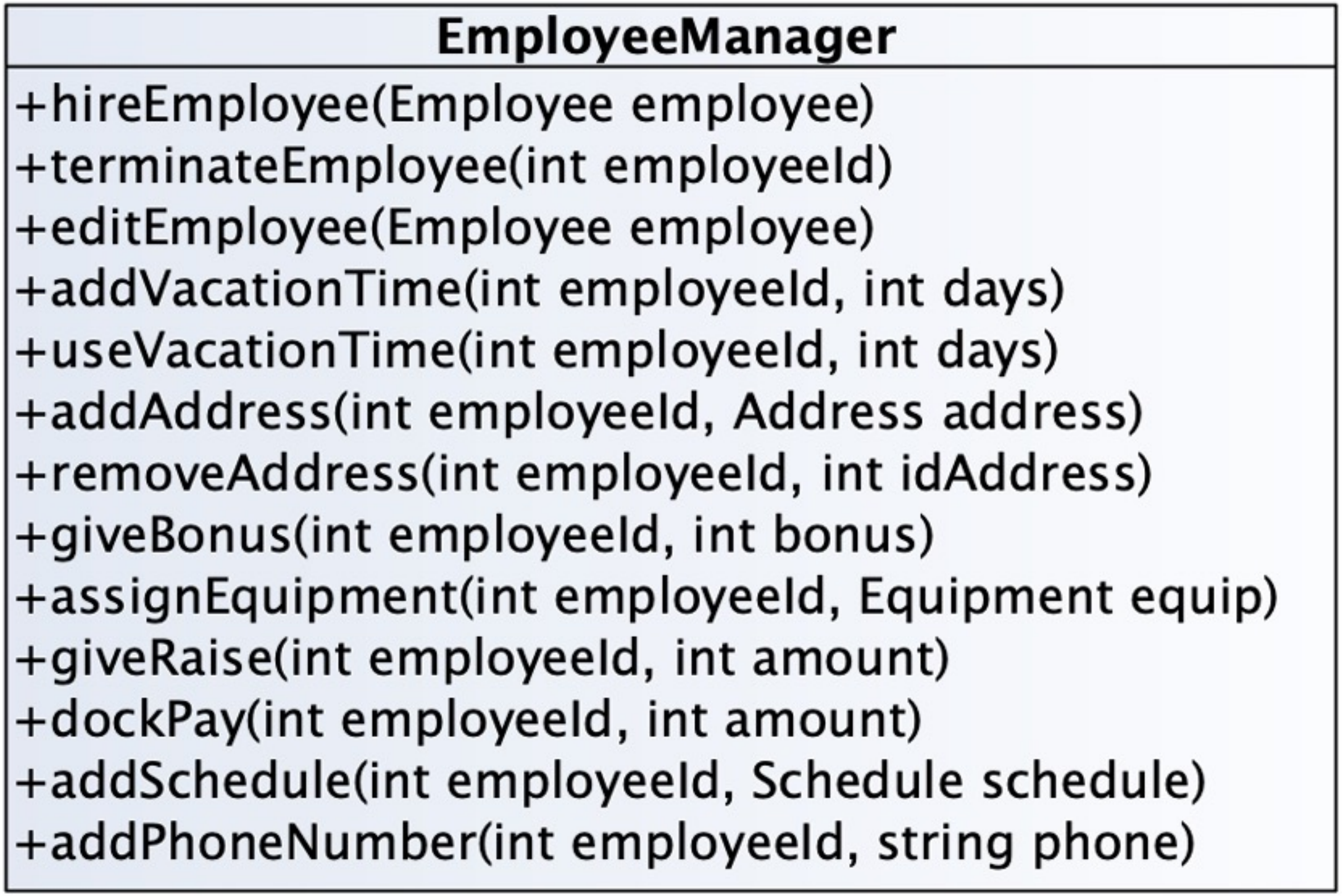
Split Up God Class

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

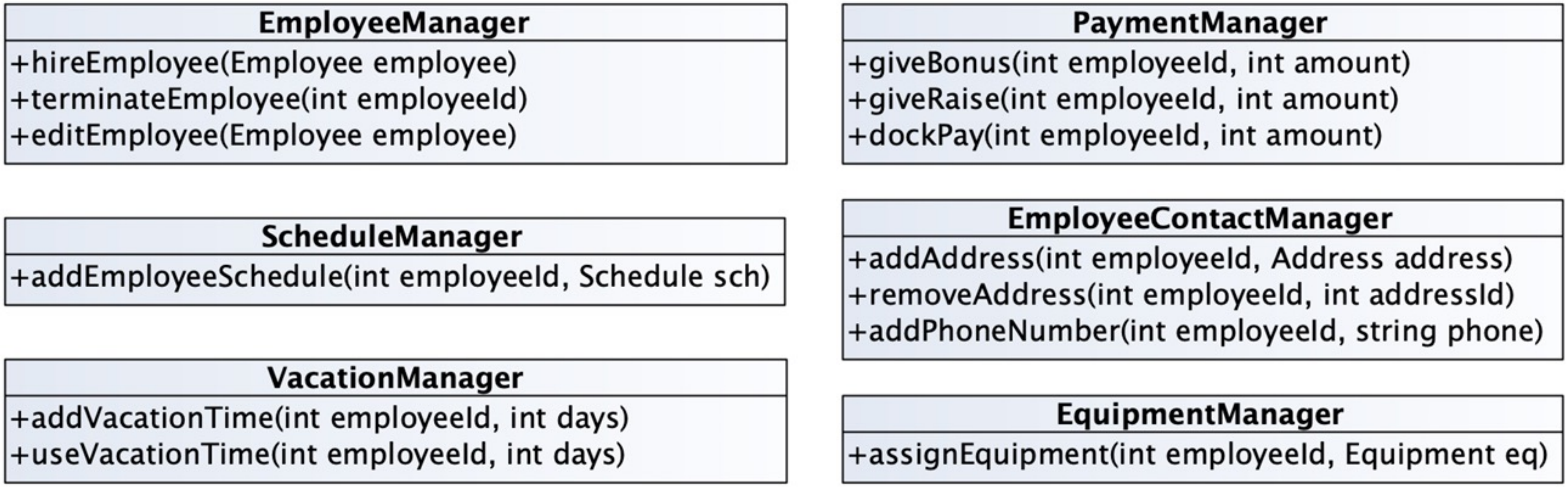
Split Up God Class



EmployeeManager

```
+hireEmployee(Employee employee)
+terminateEmployee(int employeeId)
+editEmployee(Employee employee)
+addVacationTime(int employeeId, int days)
+useVacationTime(int employeeId, int days)
+addAddress(int employeeId, Address address)
+removeAddress(int employeeId, int idAddress)
+giveBonus(int employeeId, int bonus)
+assignEquipment(int employeeId, Equipment equip)
+giveRaise(int employeeId, int amount)
+dockPay(int employeeId, int amount)
+addSchedule(int employeeId, Schedule schedule)
+addPhoneNumber(int employeeId, string phone)
```

Split Up God Class



EmployeeManager

+hireEmployee(Employee employee)
+terminateEmployee(int employeeId)
+editEmployee(Employee employee)

ScheduleManager

+addEmployeeSchedule(int employeeId, Schedule sch)

VacationManager

+addVacationTime(int employeeId, int days)
+useVacationTime(int employeeId, int days)

PaymentManager

+giveBonus(int employeeId, int amount)
+giveRaise(int employeeId, int amount)
+dockPay(int employeeId, int amount)

EmployeeContactManager

+addAddress(int employeeId, Address address)
+removeAddress(int employeeId, int addressId)
+addPhoneNumber(int employeeId, string phone)

EquipmentManager

+assignEquipment(int employeeId, Equipment eq)

Identifying Refactorings in code

Refactoring is noise in evolution analysis

- **Merge conflicts**: when merging development branches
- **Bug-inducing analysis** (SZZ): flag refactoring edits as bug-introducing changes
- **Tracing requirements to code:** miss traceability links due to refactoring
- **Regression testing:** unnecessary execution of tests for refactored code with no behavioral changes
- **Code review/merging:** refactoring edits tangled with the actual changes intended by developers
- **Dependency analysis:** cause breaking changes to clients of libraries and frameworks

Refactoring-Aware Techniques

Many refactoring-aware techniques:

- IntelliMerge & Refmerge – merge branches
- Neto et al. (ESEM ‘19) – detect bug inducing changes
- APIDiff – adapt client software to library and framework updates
- Wang et al. (ICSE ‘19) - select regression tests
- RefDistiller: assist code review

All developed in in the presence of refactoring operations.

Refactoring-Aware Techniques

- Accurate refactoring detection is required for the tools to be efficient
- RefactoringMiner (SOA tool) [Tsantalis et al. TSE'20]
- RefactoringMiner has the highest average precision (99.6%) and recall (94%) among all competitive tools
- The tool takes an input two revisions (e.g. commits) and returns a list of refactorings

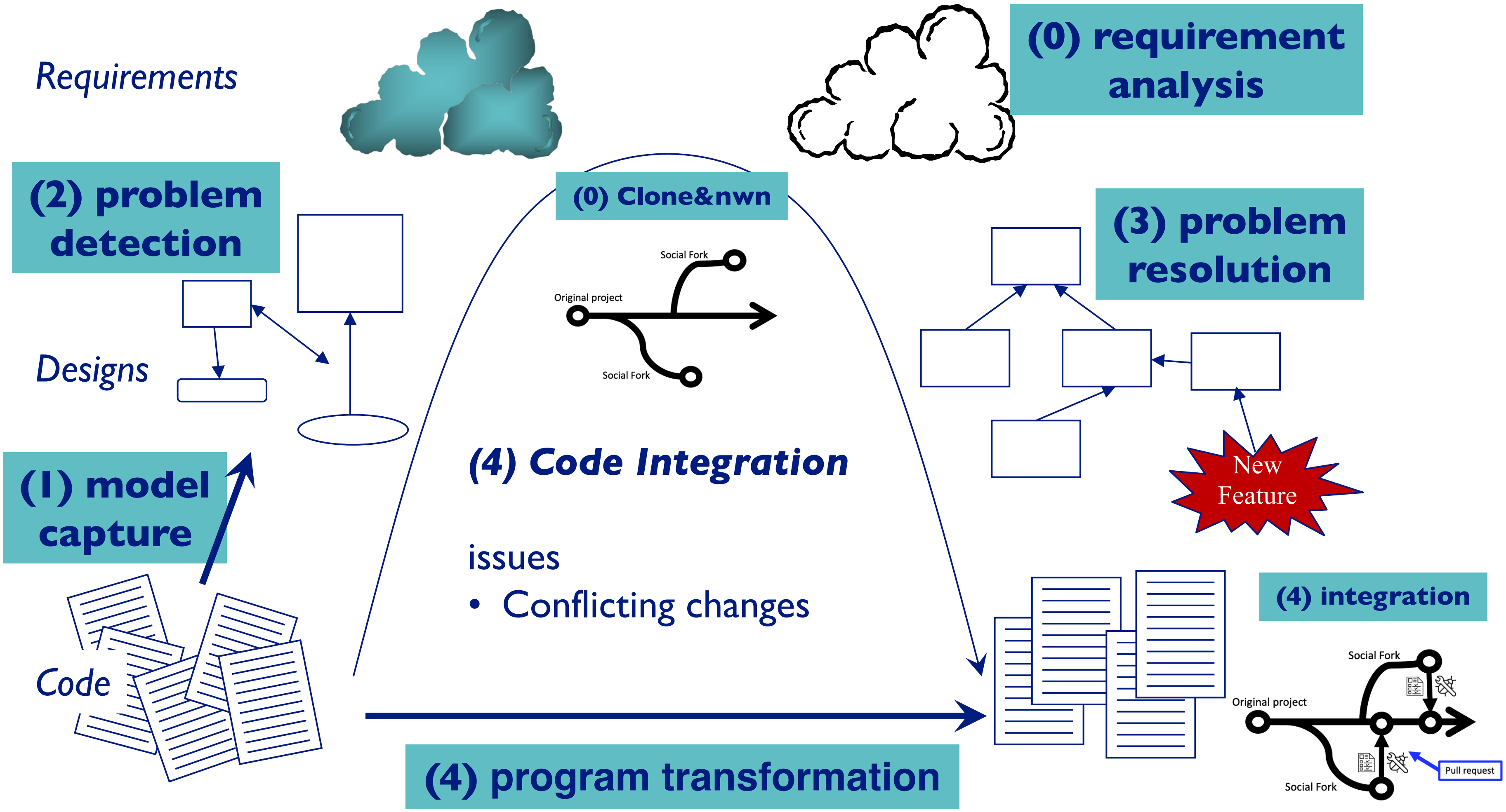
5. Code integration

- Version Control Systems
- Branching
- Merging/integration
- Merge conflicts

Does not exist in the book
[Demeyer, Ducasse and Nierstrasz: Object-Oriented Reengineering Patterns]

Published work by researchers will be used

The Reengineering Life-Cycle



Requirements

(0) requirement analysis

(2) problem detection

(0) Clone&nwn

(3) problem resolution

Designs

(4) Code Integration

issues

- Conflicting changes

(1) model capture

(4) integration

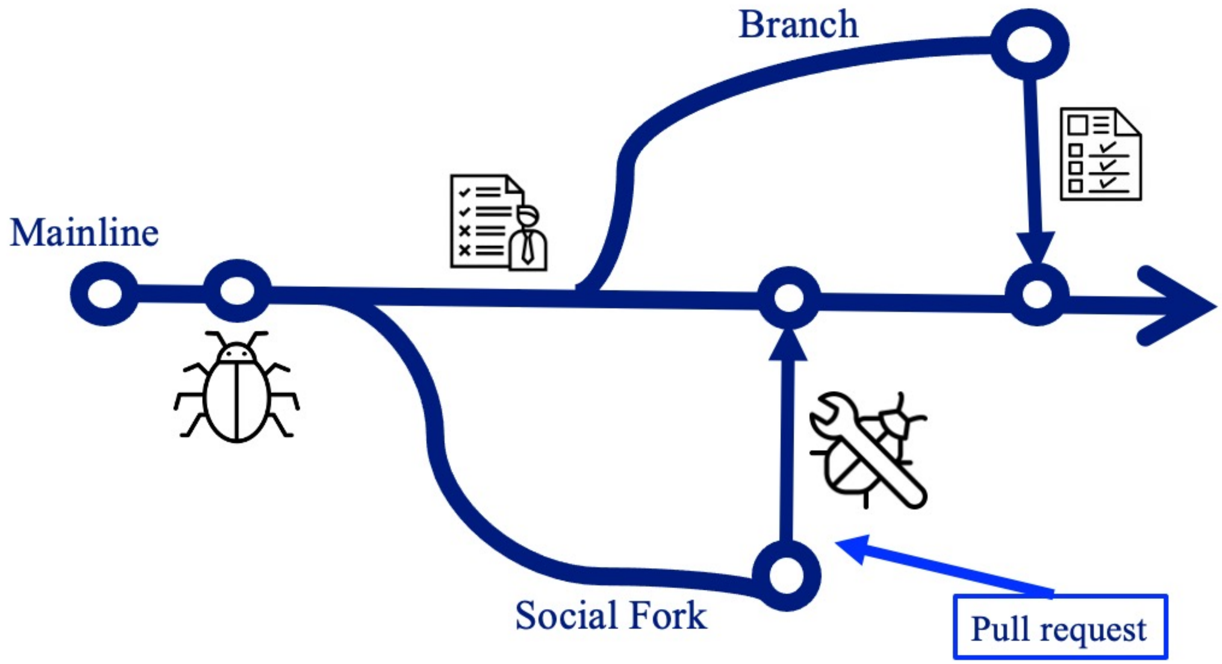
Code

(4) program transformation

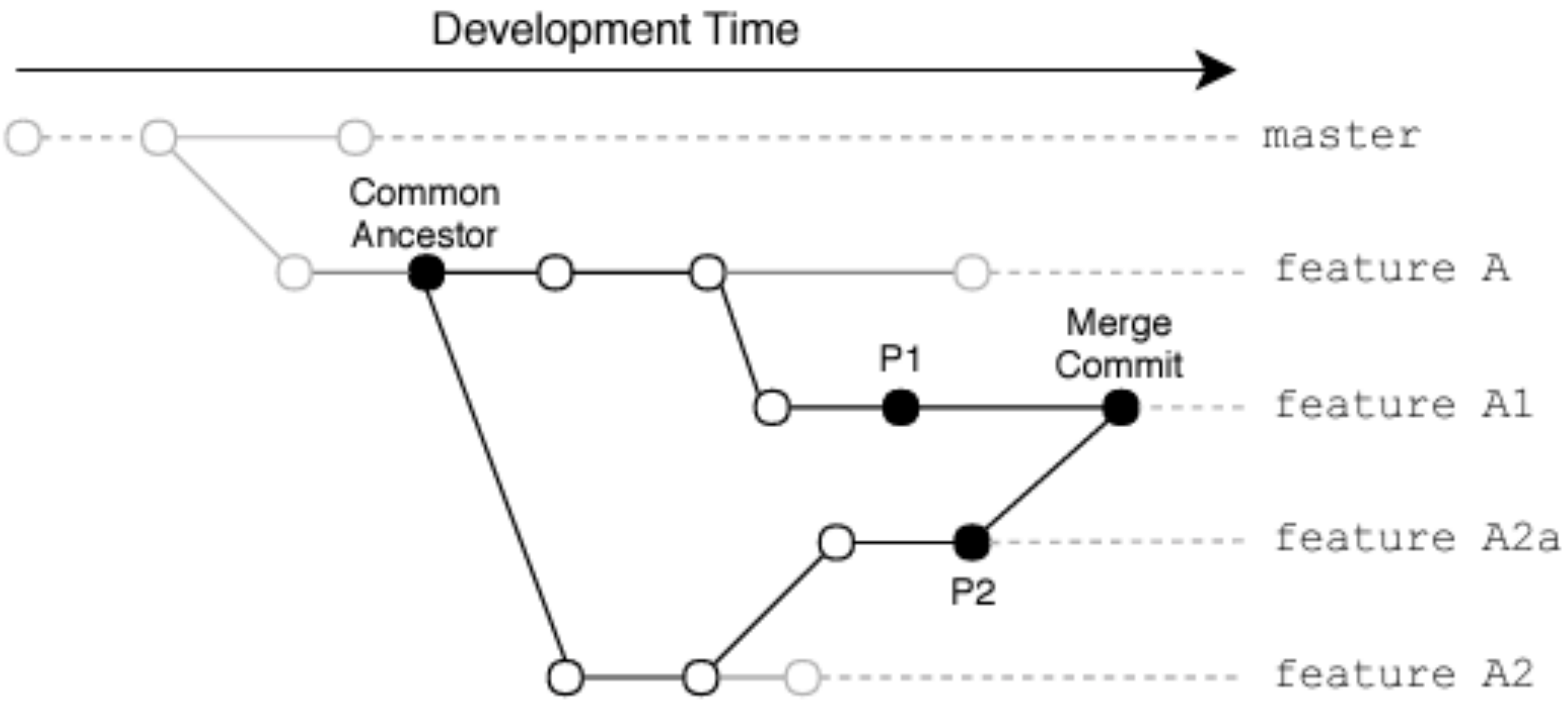
Version Control Systems

A fundamental way that developers manage change is through VCS.

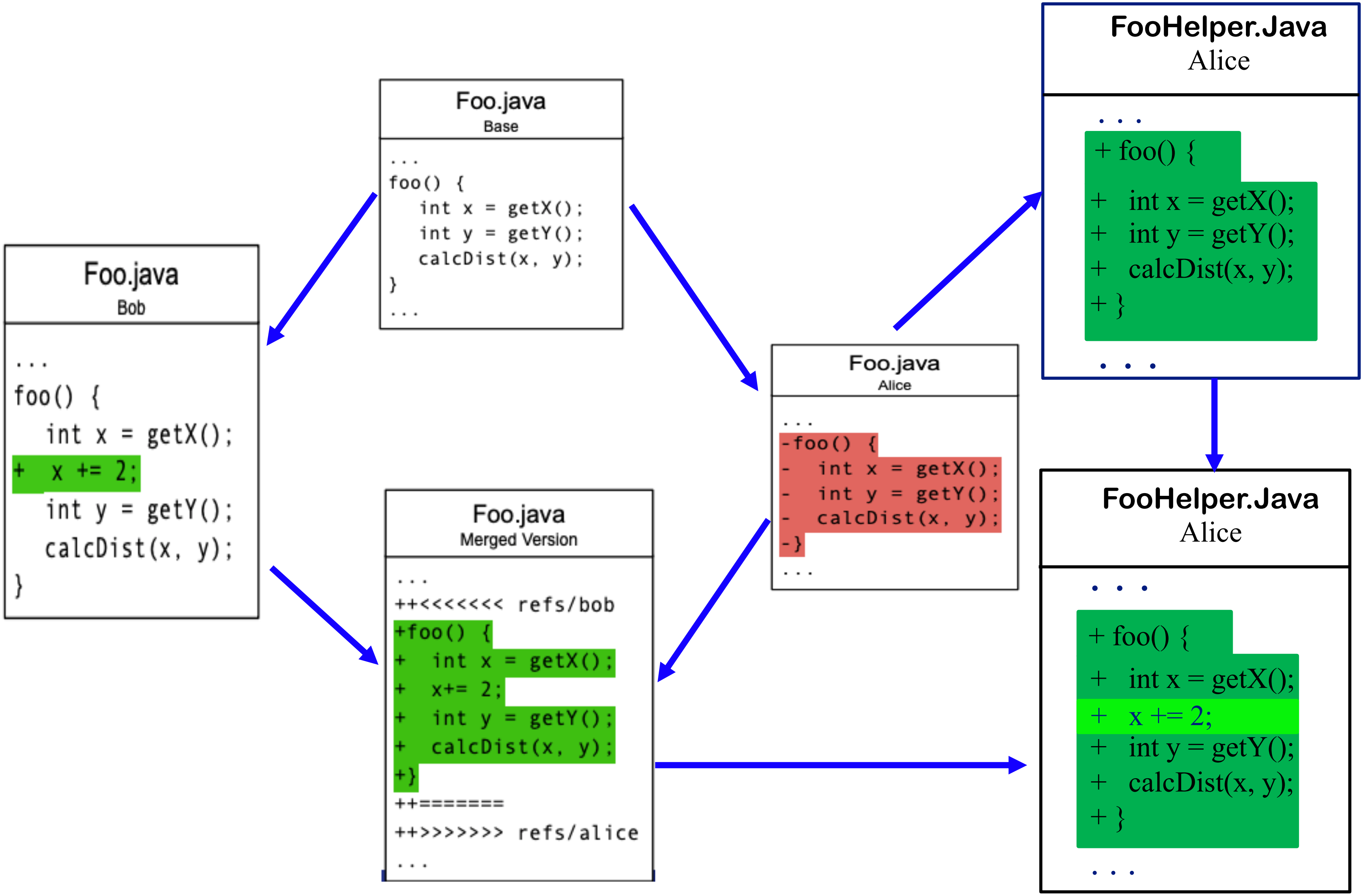
Branching/Forking



Merge Scenario



Collaborative developemt/ Merge Conflict



Foo.java
Base

```
...
foo() {
   int x = getX();
   int y = getY();
   calcDist(x, y);
}
...
```

Foo.java
Bob

```
...
foo() {
   int x = getX();
+  x += 2;
   int y = getY();
   calcDist(x, y);
}
```

Foo.java
Alice

```
...
-foo() {
-   int x = getX();
-   int y = getY();
-   calcDist(x, y);
-}
...
```

FooHelper.Java
Alice

```
. . .
+ foo() {
+   int x = getX();
+   int y = getY();
+   calcDist(x, y);
+ }
. . .
```

Foo.java
Merged Version

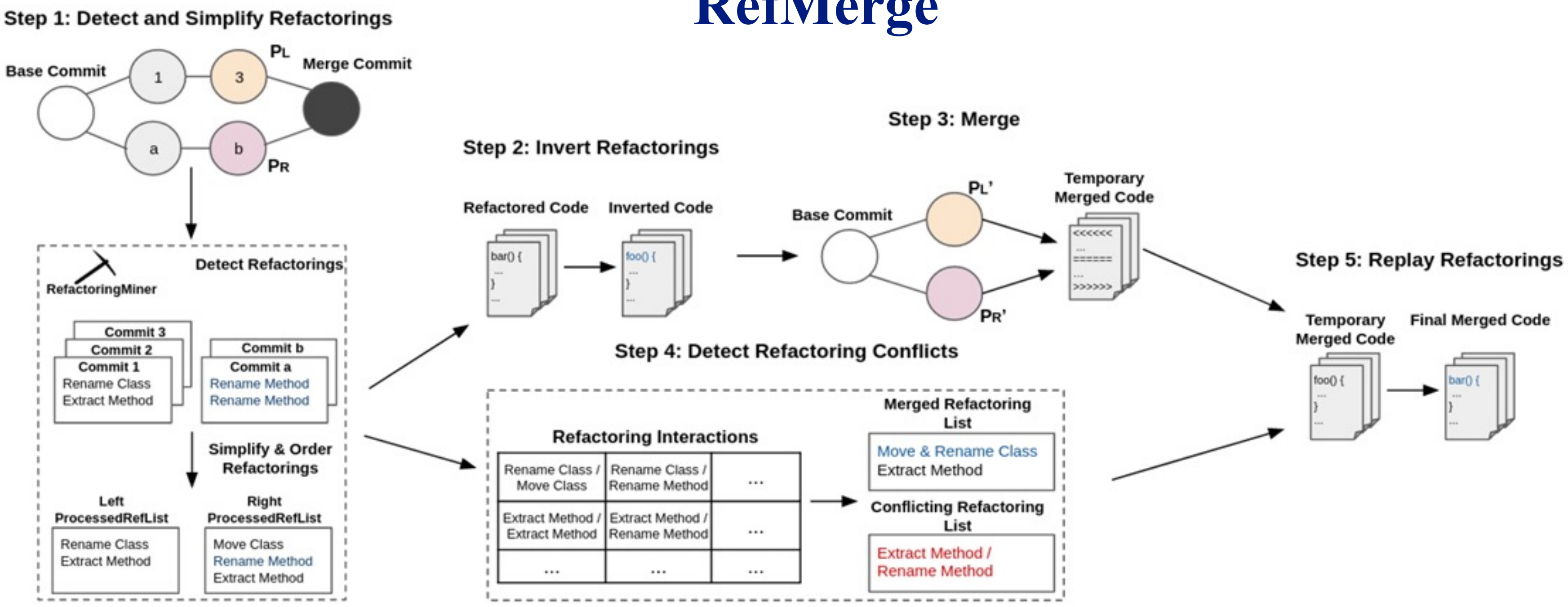
```
...
++<<<<<<< refs/bob
+foo() {
+  int x = getX();
+  x+= 2;
+  int y = getY();
+  calcDist(x, y);
+}
++=======
++>>>>>>> refs/alice
...
```

FooHelper.Java
Alice

```
. . .
+ foo() {
+   int x = getX();
+   x += 2;
+   int y = getY();
+   calcDist(x, y);
+ }
. . .
```

Refactoring-Aware tools

RefMerge



Rename Class / Move Class	Rename Class / Rename Method	...
Extract Method / Extract Method	Extract Method / Rename Method	...
...	...	...

Ellis et at. A Systematic Comparison of Two Refactoring-aware Merging Techniques. 2022

https://github.com/ualberta-smr/RefactoringAwareMergingEvaluation

6. Dynamic Analysis (& Testing)

- Key Concept Identification
- Unit testing
- Test coverage
- Mutation testing

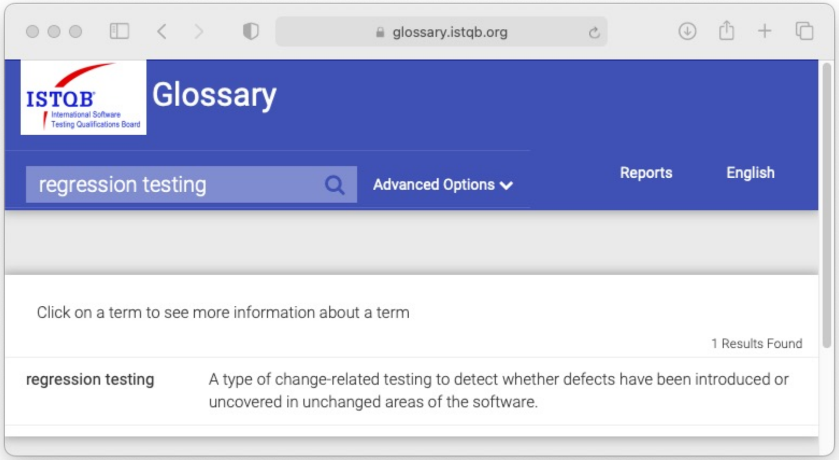
Introduction

- Dynamic Analysis verifies properties of a system during execution
- Testing Analysis is one example of Dynamic Analysis
  - Unit tests, integration tests, system tests, and acceptance tests use dynamic testing

Testing

- Tests are your life insurance! (OORP, p. 149)
- Tests are essential to assure the quality of refactoring activities.
- Write Tests to Enable Evolution (OORP, p.153)
  - Good tests can find bugs on your artifact
  - Tests can also detect unwanted behavior
- You can also write tests to understand a part of a system (OORP, p.179)

Regression Testing

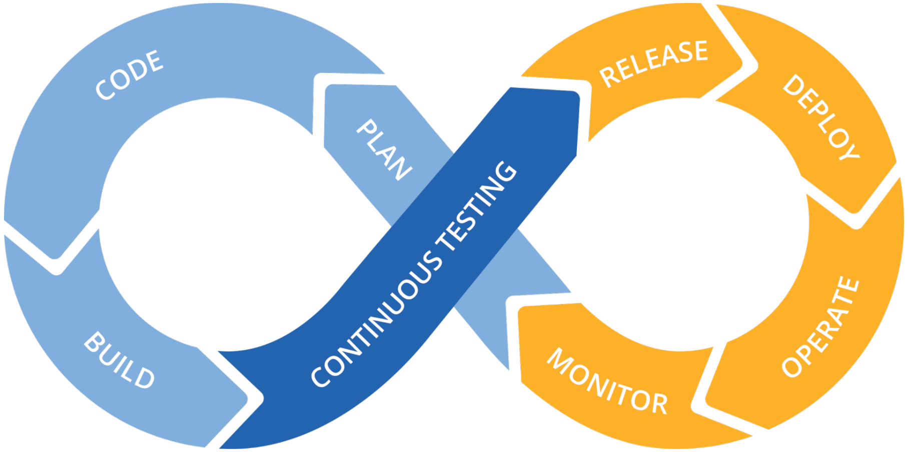


Click on a term to see more information about a term

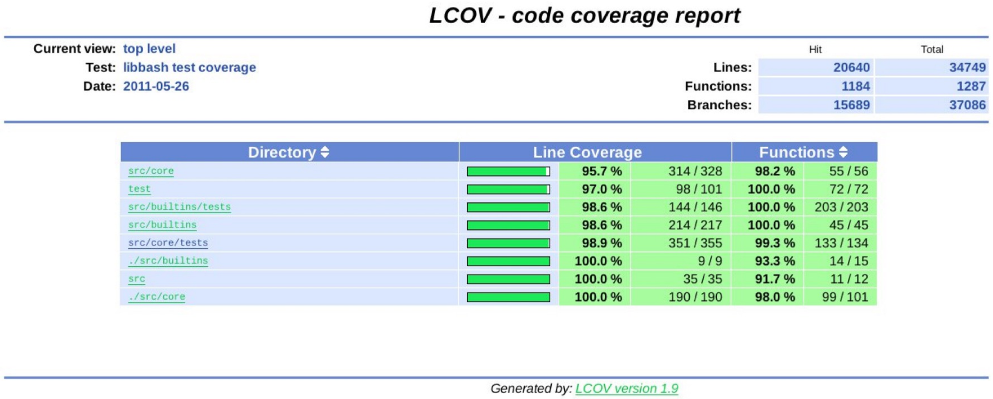
1 Results Found

regression testing — A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software.

A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software.



Coverage



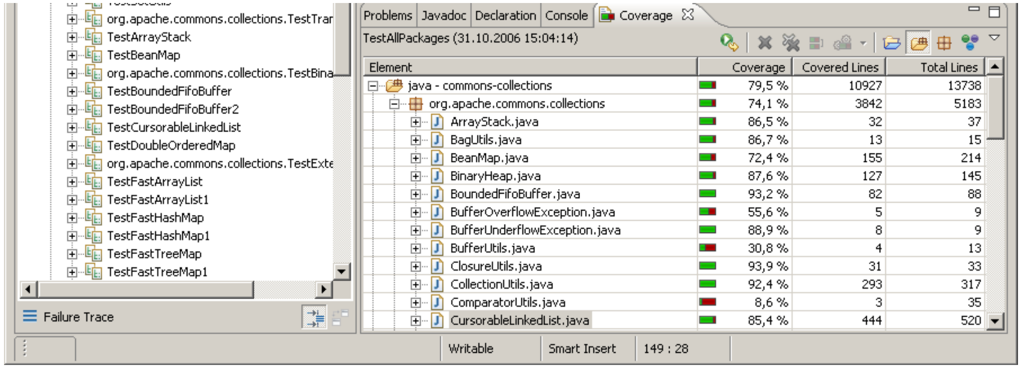
LCOV - code coverage report

				Hit	Total
Current view:	top level				
Test:	libbash test coverage		Lines:	20640	34749
Date:	2011-05-26		Functions:	1184	1287
			Branches:	15689	37086

Directory	Line Coverage			Functions	
src/core	🟩	95.7 %	314 / 328	98.2 %	55 / 56
test	🟩	97.0 %	98 / 101	100.0 %	72 / 72
src/builtins/tests	🟩	98.6 %	144 / 146	100.0 %	203 / 203
src/builtins	🟩	98.6 %	214 / 217	100.0 %	45 / 45
src/core/tests	🟩	98.9 %	351 / 355	99.3 %	133 / 134
./src/builtins	🟩	100.0 %	9 / 9	93.3 %	14 / 15
src	🟩	100.0 %	35 / 35	91.7 %	11 / 12
./src/core	🟩	100.0 %	190 / 190	98.0 %	99 / 101

Generated by: LCOV version 1.9

Are the areas under change sufficiently covered by the current test suite?



- org.apache.commons.collections.TestTrar
- TestArrayStack
- TestBeanMap
- org.apache.commons.collections.TestBina
- TestBoundedFifoBuffer
- TestBoundedFifoBuffer2
- TestCursorableLinkedList
- TestDoubleOrderedMap
- org.apache.commons.collections.TestExte
- TestFastArrayList
- TestFastArrayList1
- TestFastHashMap
- TestFastHashMap1
- TestFastTreeMap
- TestFastTreeMap1

TestAllPackages (31.10.2006 15:04:14)

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520

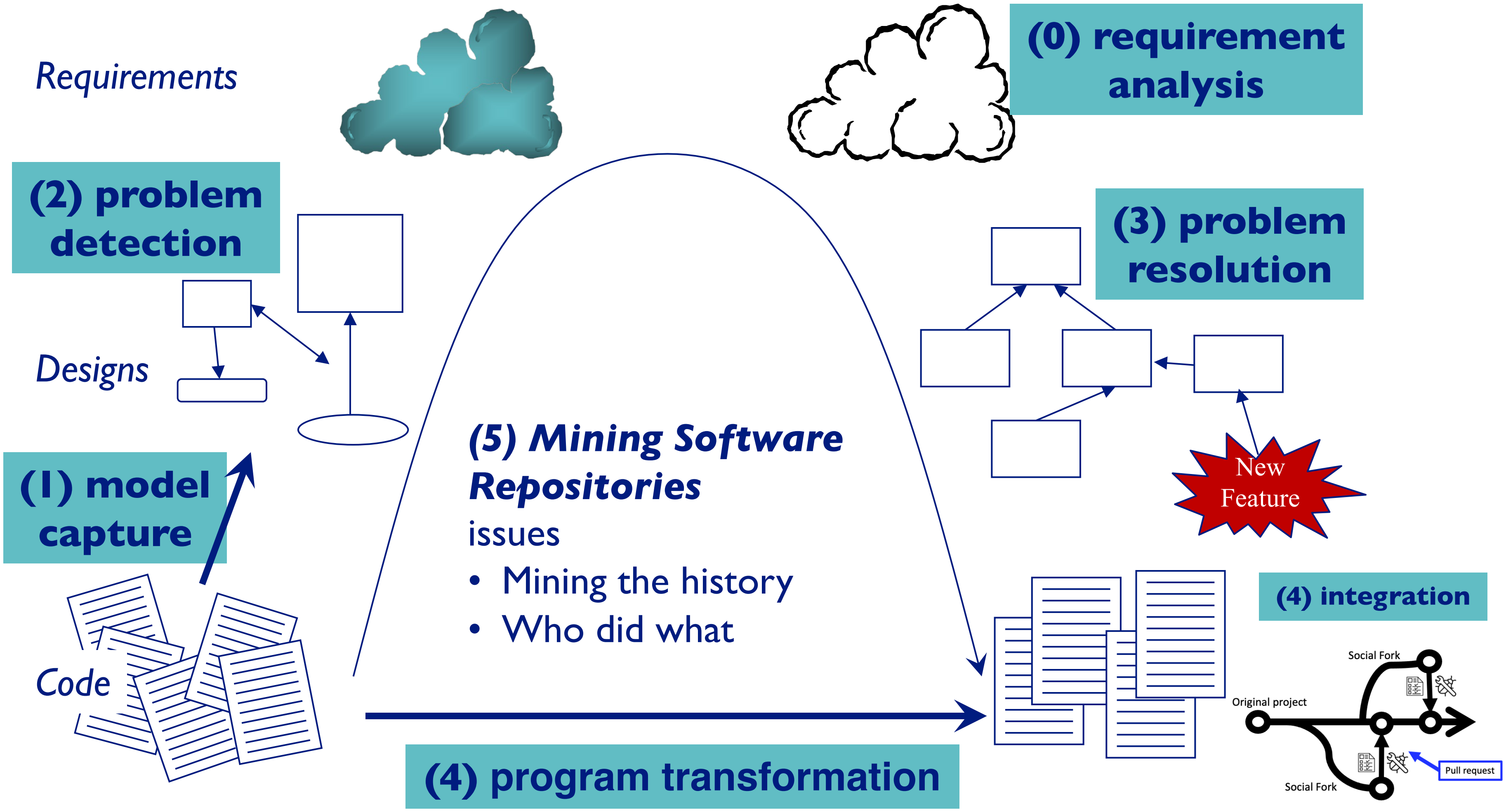
Writable | Smart Insert | 149 : 28

Compare coverage reports before and after refactoring!

7. Mining Software Repositories (MSR)

- What are software repositories?
- Why should we mine Software repositories?
- What are some of the data sources of software engineering data?
- What are some of the existing tools we can use to mine software engineering data
- What can we learn from MSR

The Reengineering Life-Cycle



Requirements

(0) requirement analysis

(2) problem detection

Designs

(3) problem resolution

(1) model capture

(5) Mining Software Repositories

issues

- Mining the history
- Who did what

Code

(4) integration

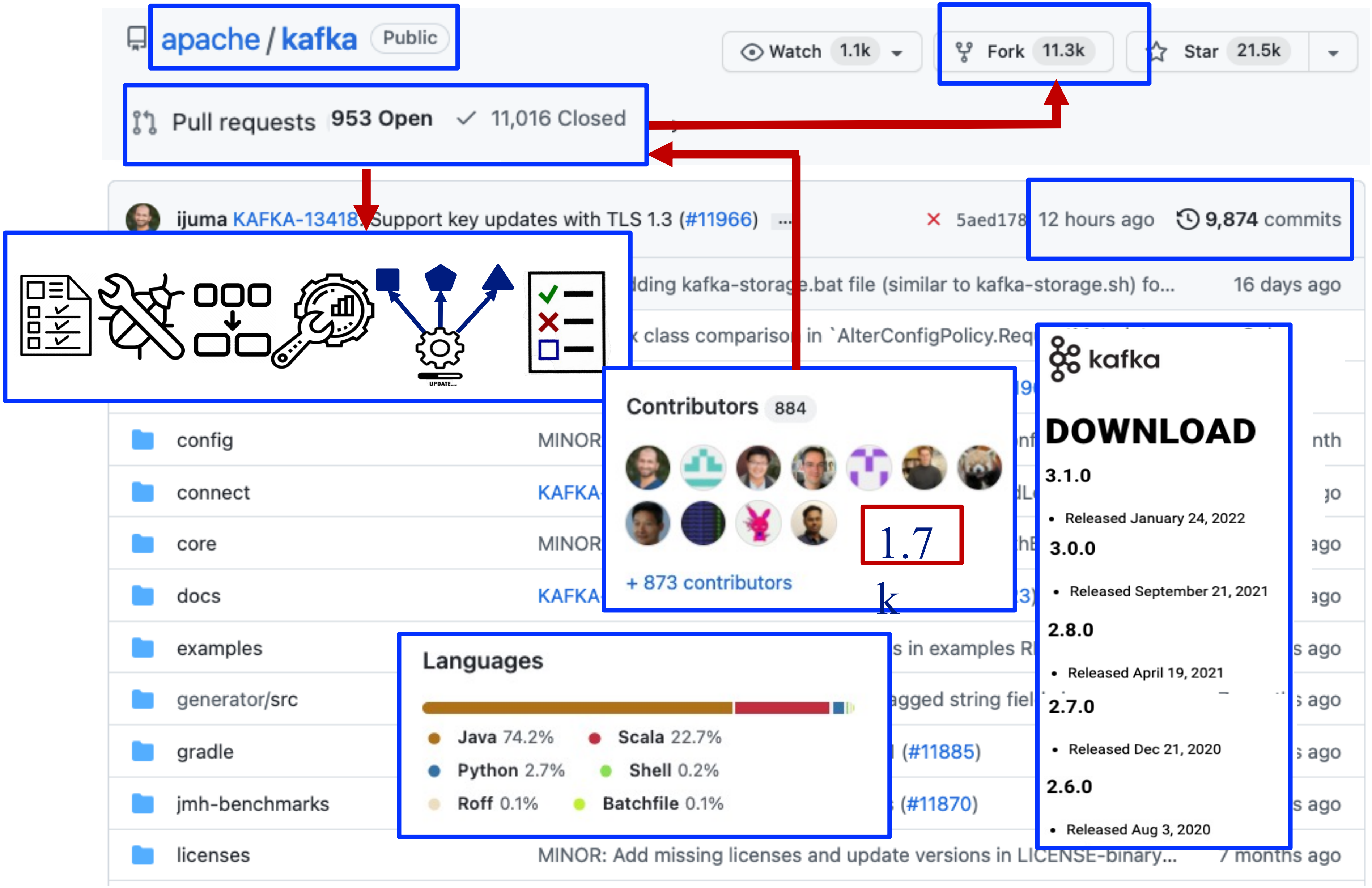
(4) program transformation

What is a Software Repository?

Artifacts produced and archived during software development

- Technical artifacts
- Social artifacts

What is a Software Repository?



apache / kafka Public

Watch 1.1k

Fork 11.3k

Star 21.5k

Pull requests 953 Open ✓ 11,016 Closed

ijuma KAFKA-13418: Support key updates with TLS 1.3 (#11966) … × 5aed178 12 hours ago 9,874 commits

…dding kafka-storage.bat file (similar to kafka-storage.sh) fo… 16 days ago

…x class comparison in `AlterConfigPolicy.Req…

config MINOR … nth

connect KAFKA … dL … go

core MINOR … hE … ago

docs KAFKA … 3) … ago

examples … s in examples R … s ago

generator/src … agged string fiel … s ago

gradle … (#11885) … s ago

jmh-benchmarks … (#11870) … s ago

licenses MINOR: Add missing licenses and update versions in LICENSE-binary… 7 months ago

Contributors 884

+ 873 contributors

1.7 k

Languages

- Java 74.2%
- Scala 22.7%
- Python 2.7%
- Shell 0.2%
- Roff 0.1%
- Batchfile 0.1%

kafka

DOWNLOAD

3.1.0

- Released January 24, 2022

3.0.0

- Released September 21, 2021

2.8.0

- Released April 19, 2021

2.7.0

- Released Dec 21, 2020

2.6.0

- Released Aug 3, 2020

Apache Kafka is a distributed event store and stream-processing platform

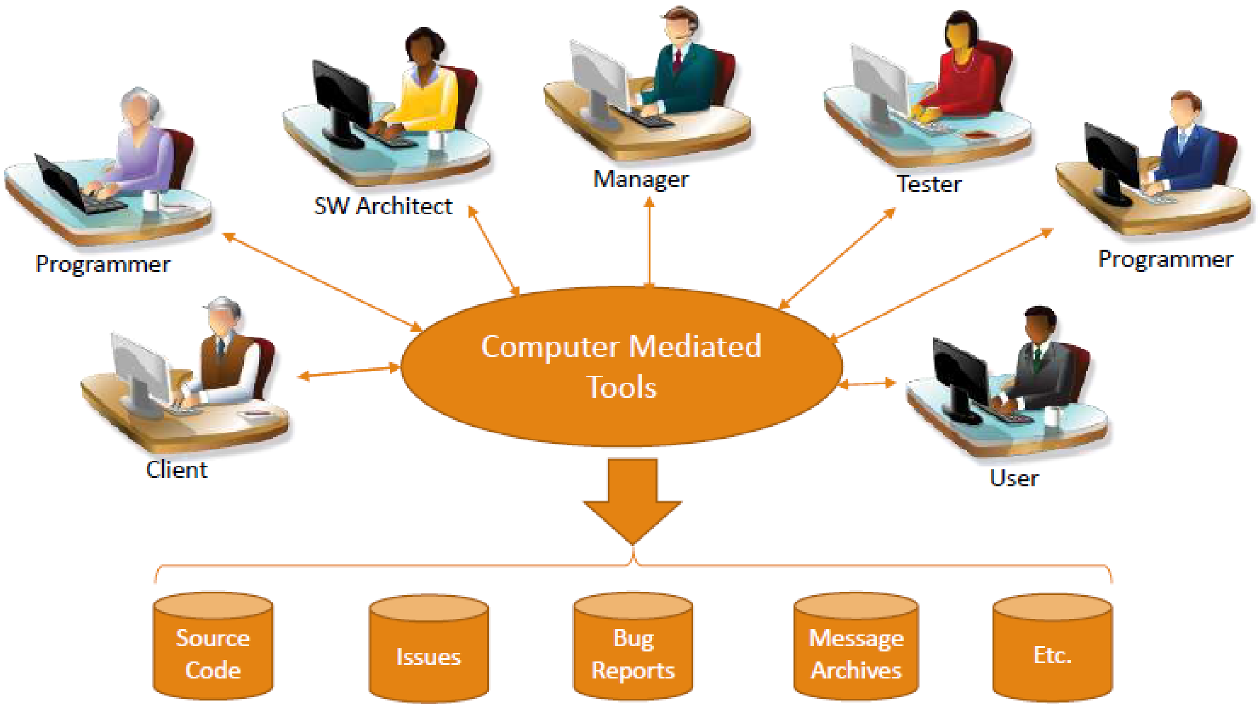
Why should we mine Software repositories?

The goal … is to improve software engineering practices by uncovering interesting and actionable information about software systems and projects using the vast amounts of software data

+ Understand software development process
+ Support and/or improve the maintenance of software systems
+ Exploit knowledge in planning the future development

- If the data analysis in not carefully designed and executed, it can lead to invalid conclusions

What are some of the data sources of software engineering data?



Current and historical artifacts and interactions are registered in software repositories

This list is not exhaustive.

Qn. What are some of the additional software engineering data sources that can be mained?

What are some of the existing tools we can use to mine software engineering data?

PyDriller

A Python framework that helps developers in analyzing Git repositories. With PyDriller you can easily extract information about **commits**, **developers**, **modified files**, **diffs**, and **source code**.

RepoDriller

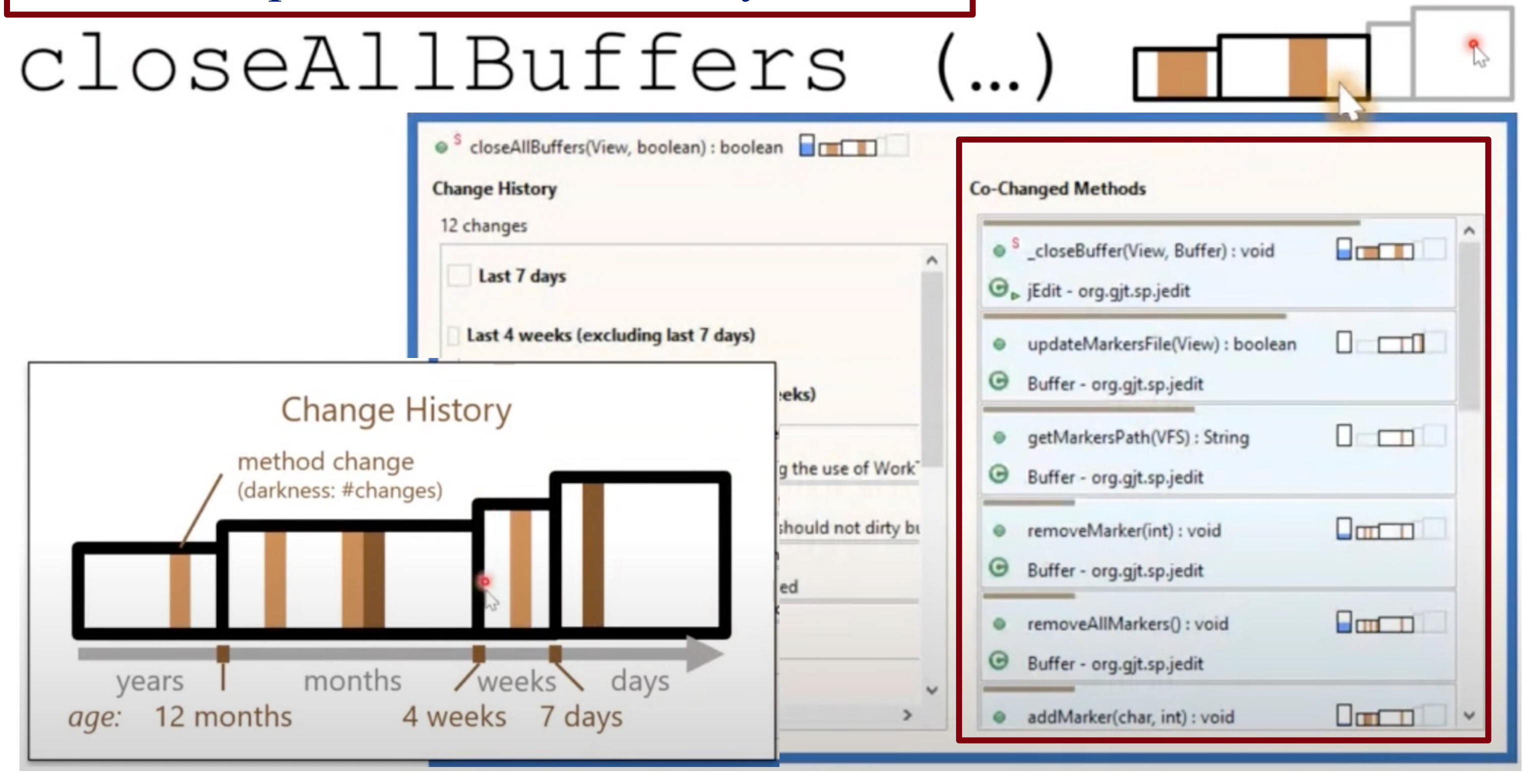
A Java framework that helps developers on mining software repositories. With it, you can easily extract information from any Git repository, such as commits, developers, modifications, diffs, and source codes, and quickly export CSV files.

Build your own tool/script

Sometimes/ most of the times, you have to build your own tool or script to mine your own data

What can we learn from MSR

Observation: Recent history is often important than old history



closeAllBuffers(View, boolean) : boolean

Change History

12 changes

Last 7 days

Last 4 weeks (excluding last 7 days)

eks)

g the use of Work

should not dirty bu

ed

Co-Changed Methods

_closeBuffer(View, Buffer) : void
jEdit - org.gjt.sp.jedit

updateMarkersFile(View) : boolean
Buffer - org.gjt.sp.jedit

getMarkersPath(VFS) : String
Buffer - org.gjt.sp.jedit

removeMarker(int) : void
Buffer - org.gjt.sp.jedit

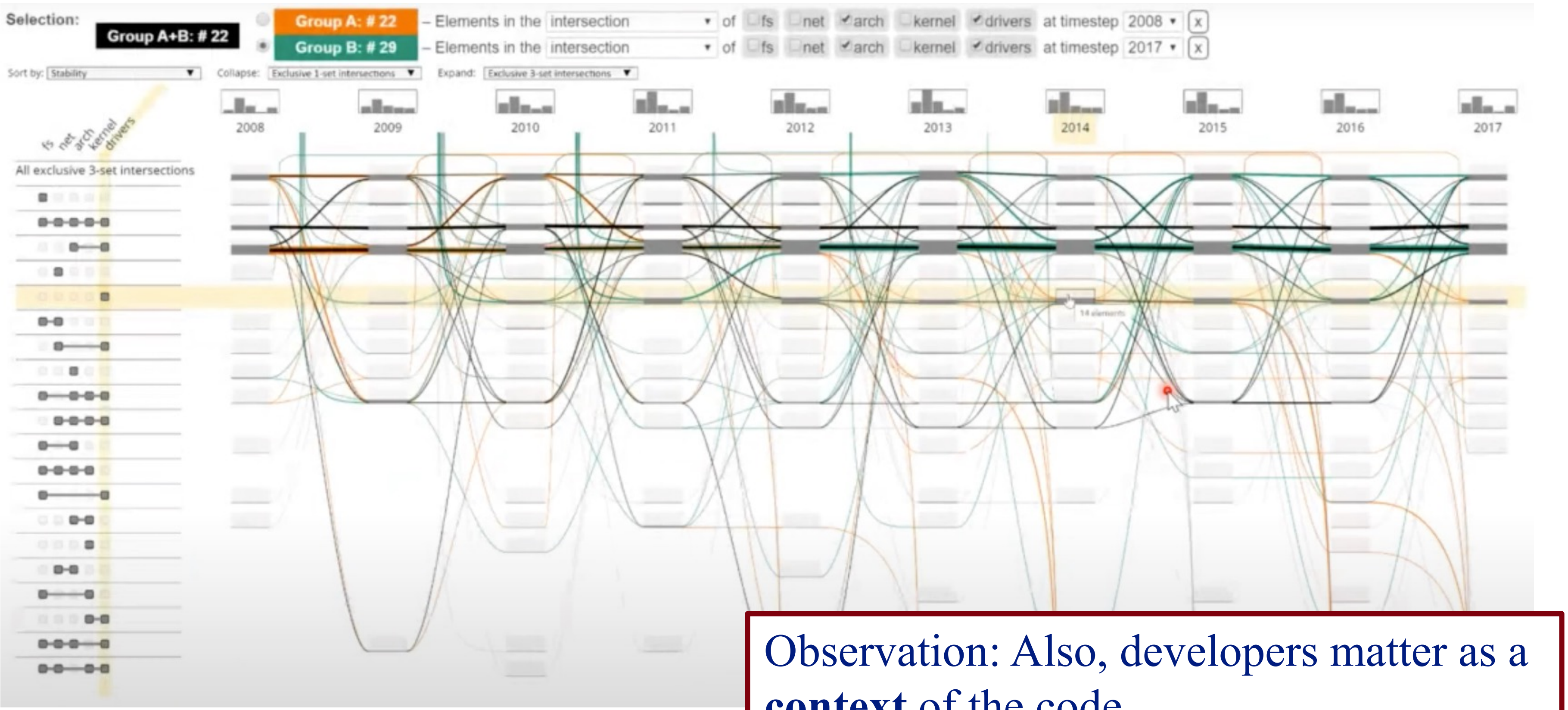
removeAllMarkers() : void
Buffer - org.gjt.sp.jedit

addMarker(char, int) : void

Embedding Evolutionary Context

Beck et al. Rethinking User Interfaces for Feature Location. ICPC 2015

What can we learn from MSR



Selection: Group A+B: # 22

○ Group A: # 22 – Elements in the intersection ▾ of □ fs □ net ☑ arch □ kernel ☑ drivers at timestep 2008 ▾ x

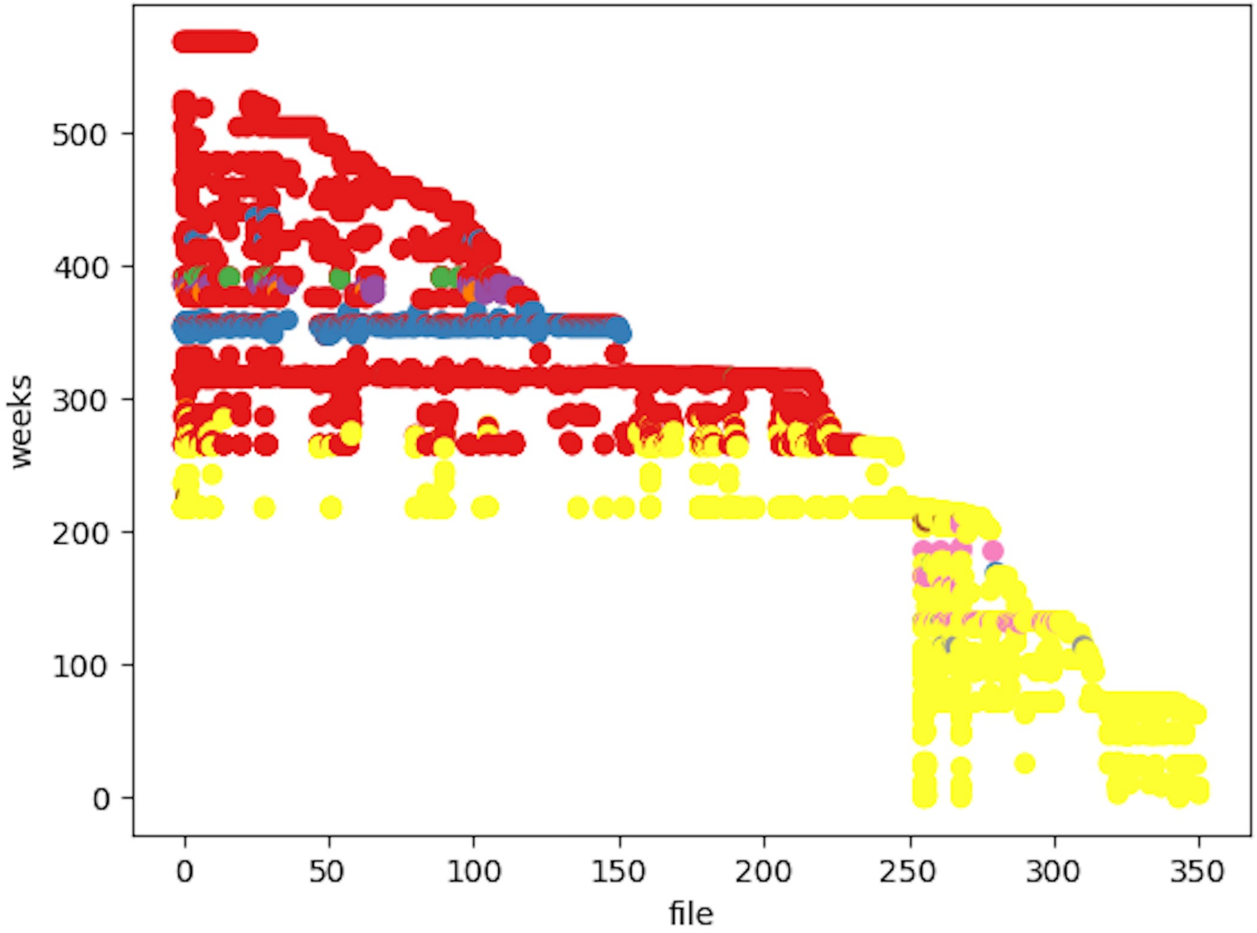
● Group B: # 29 – Elements in the intersection ▾ of □ fs □ net ☑ arch □ kernel ☑ drivers at timestep 2017 ▾ x

Sort by: Stability ▼ Collapse: Exclusive 1-set intersections ▼ Expand: Exclusive 3-set intersections ▼

Observation: Also, developers matter as a **context** of the code.

Agarwal, S.; Beck, F.,: *Set Streams: Visual Exploration of Dynamic Overlapping Sets*.
In: Computer Graphics Forum, Jg. 39 (2020) Nr. 3, S. 383-391. doi:10.1111/cgf.13988

Developers who touched files



9. Conclusion

1. **Introduction**

Software changes and that requires planning

2. **Reverse Engineering**

How to understand your code

3. **Visualization**

Scalable approach

4. **Restructuring**

How to Refactor Your Code

5 . **Code Integration**

How to resolve conflicts

6. Dynamic Analysis (& Testing)

To be really certain

7. **Mining Software Repositories**

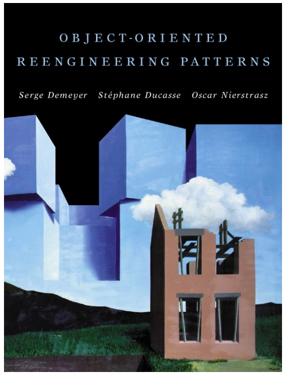
Learn from the past

8. **Conclusion**

Goals

We will try to convince you:

- Programs change!
- Reverse engineering forward 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.



⇒ Did we convince you ?