Software Product Design and Development II
CS 473

John Businge

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

Overview of the Class

- https://johnxu21.github.io/teaching/Software-Reengineering/?
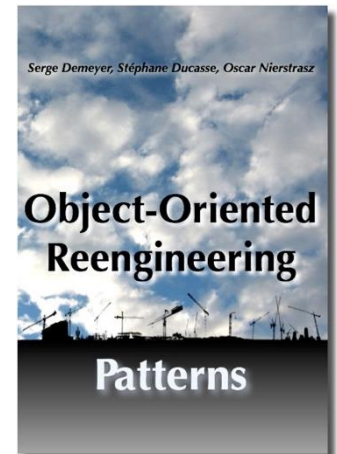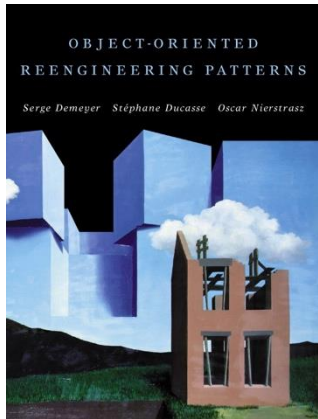
Schedule

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.
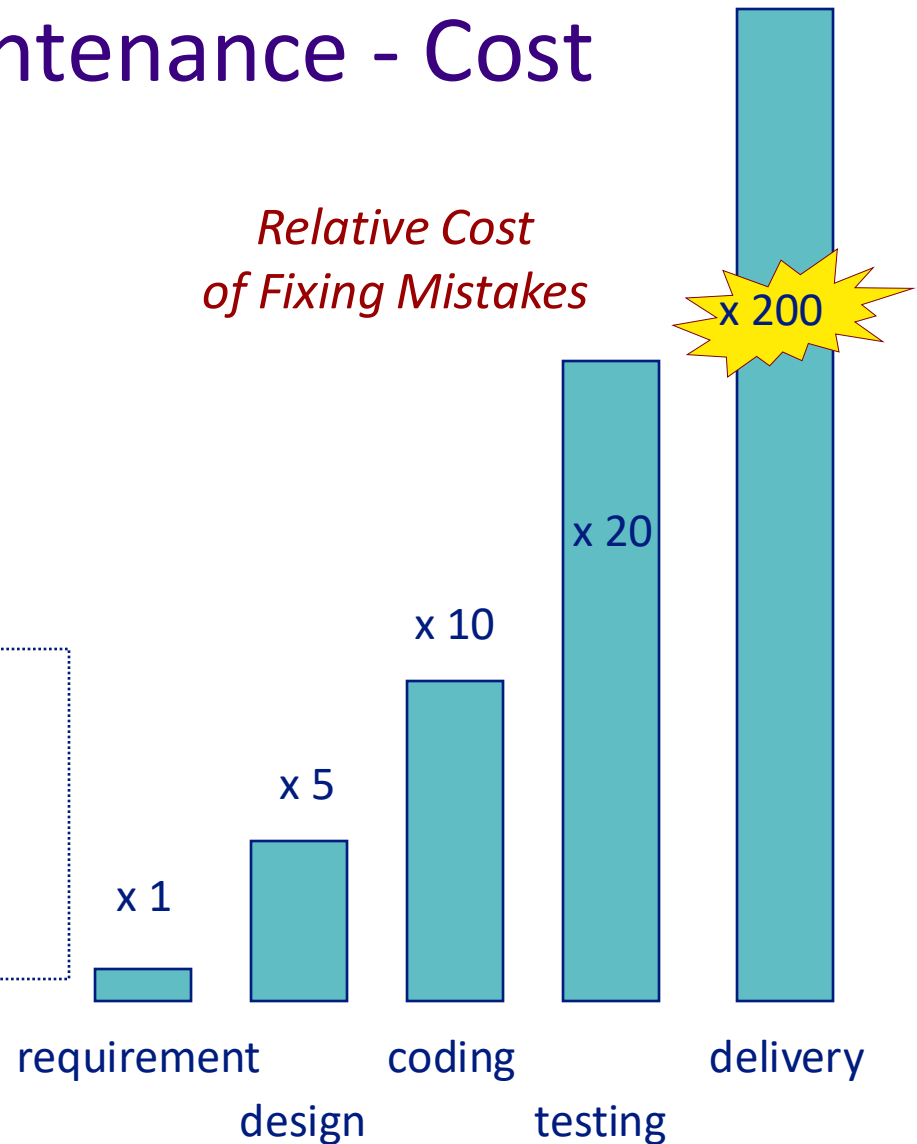
Program Change

Branch

Mainline

Social Fork

Pull request
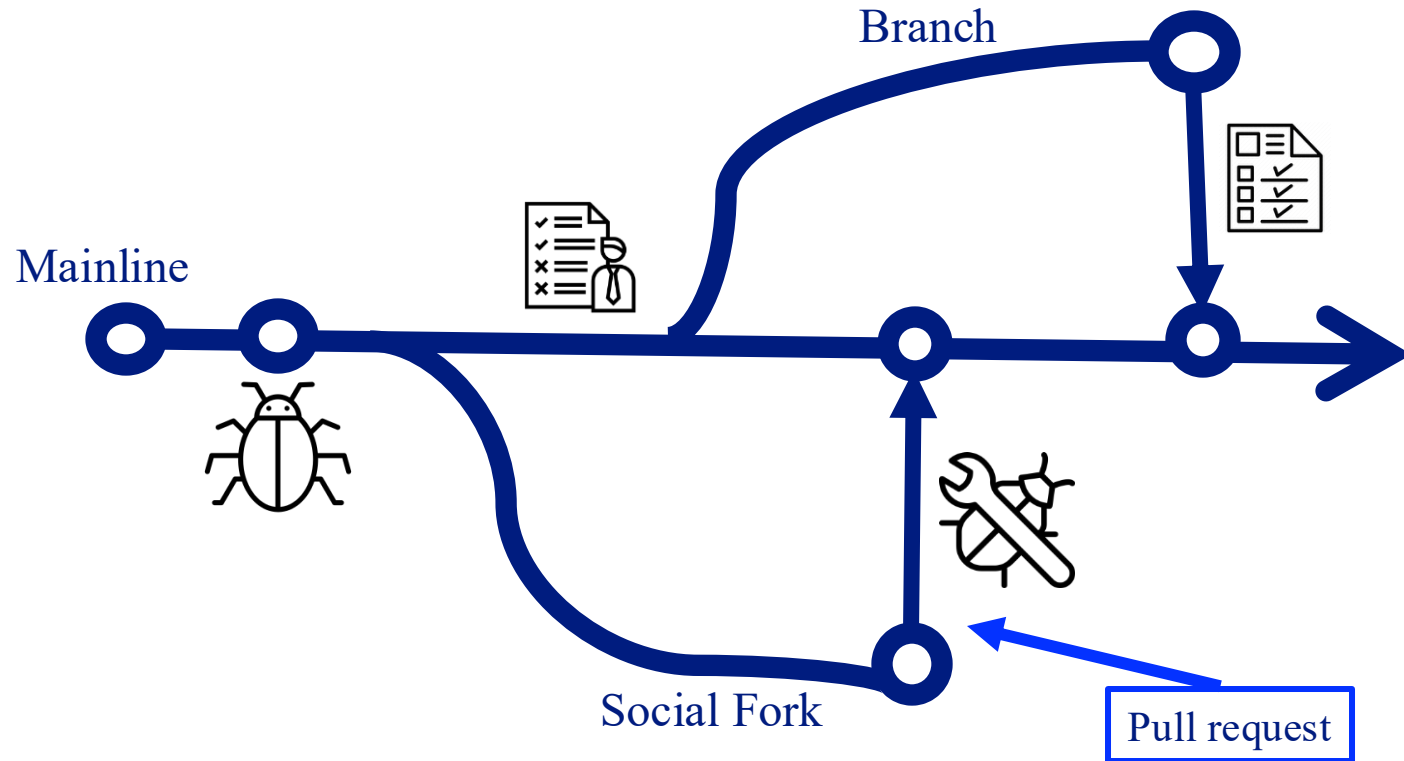
Software Maintenance - Cost

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

Relative Cost of Fixing Mistakes

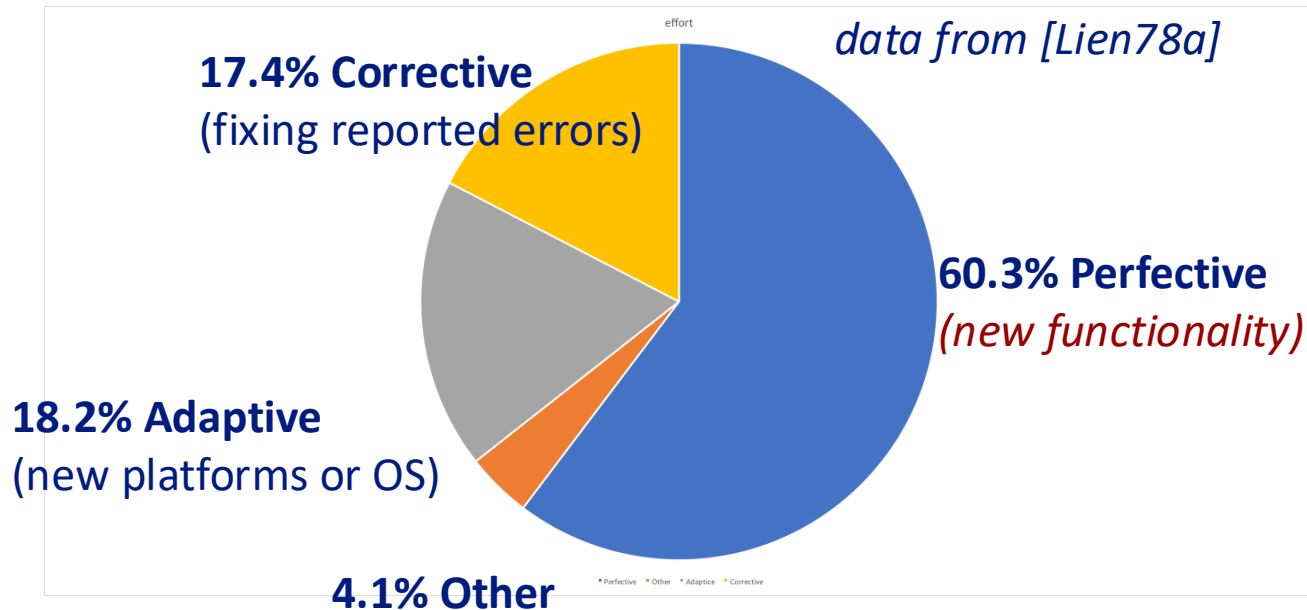*Solution ?*
- Better requirements engineering?
- Better software methods & tools (database schemas, CASE-tools, objects, components, …)?

x 1 — requirement

x 5 — design

x 10 — coding

x 20 — testing

x 200 — delivery

Continuous Development

17.4% Corrective
(fixing reported errors)

data from [Lien78a]

60.3% Perfective
(new functionality)

18.2% Adaptive
(new platforms or OS)

4.1% Other

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.
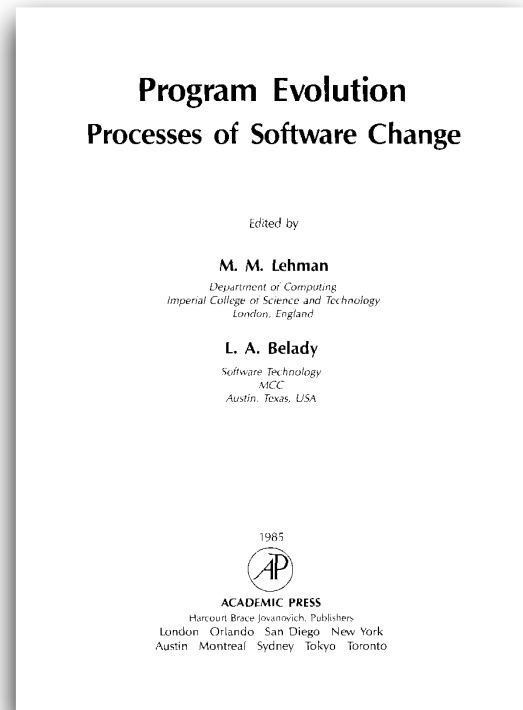
Continuing change

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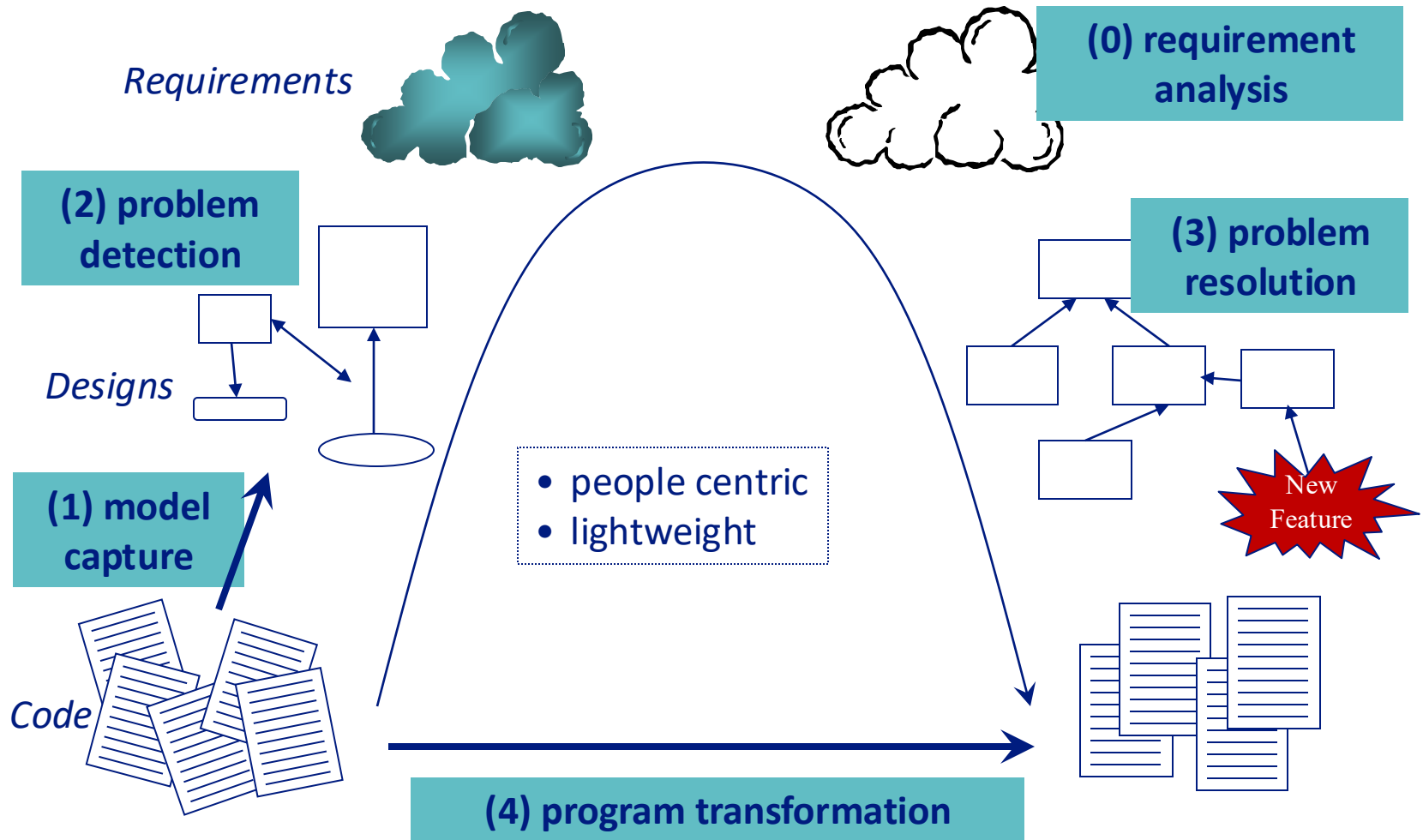
Increasing complexity

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

Those laws are still applicable...

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

The Reengineering Life-Cycle

Requirements

(0) requirement analysis

(2) problem detection

Designs

(3) problem resolution

(1) model capture

- people centric
- lightweight

New Feature

Code

(4) program transformation

A Map of Reengineering Patterns

Tests: Your Life Insurance

Detailed Model Capture

Migration Strategies

Initial Understanding

Detecting Duplicated Code

First Contact

Redistribute Responsibilities

Setting Direction

Organized according to Reengineering lifecycle

Transform Conditionals to Polymorphism
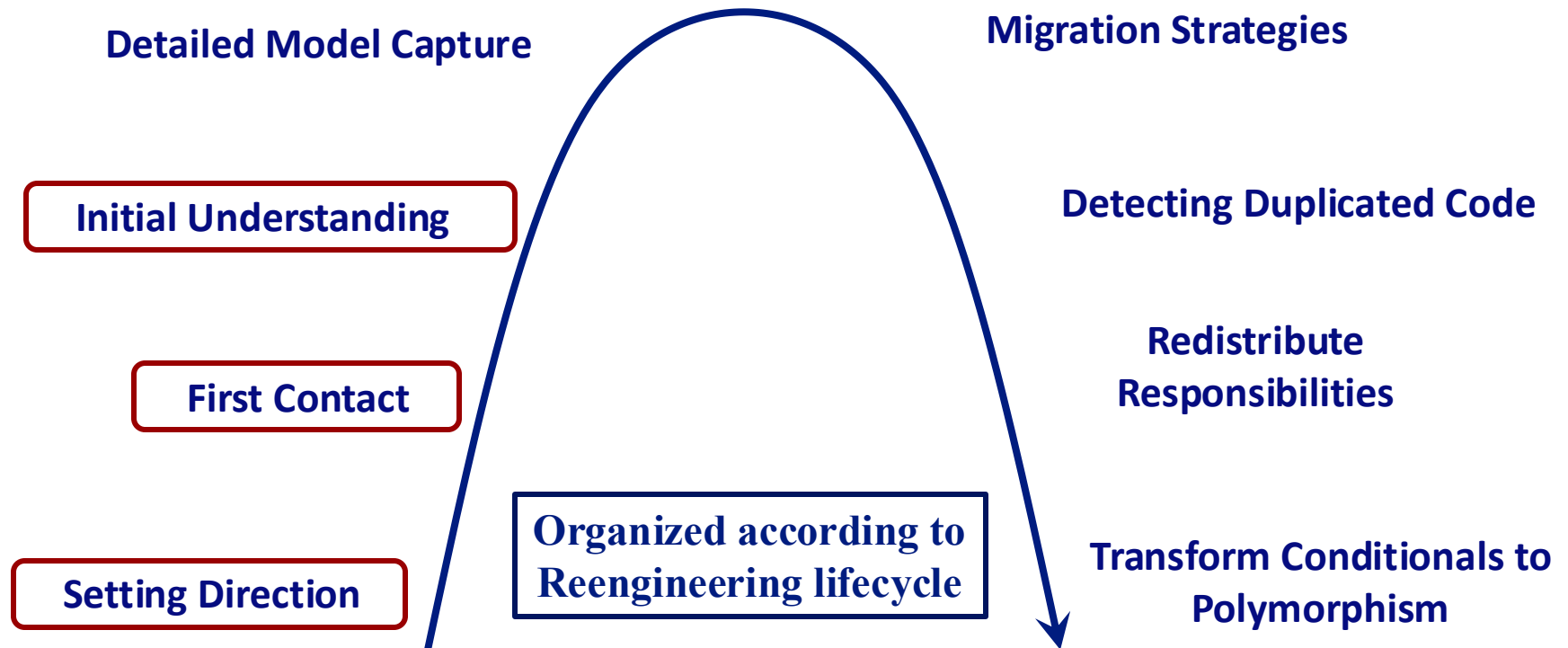
2. Reverse Engineering

- 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
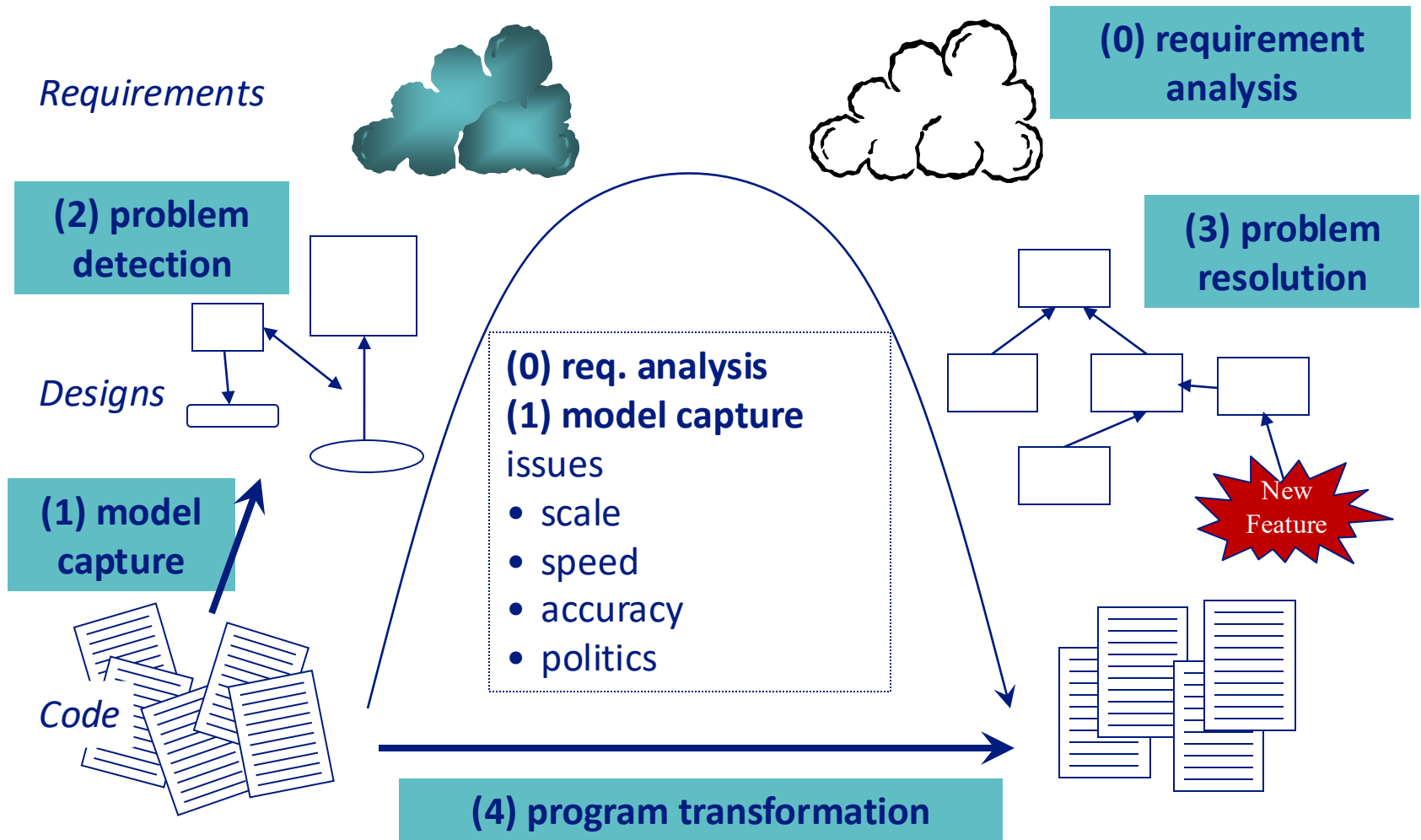
Motivation

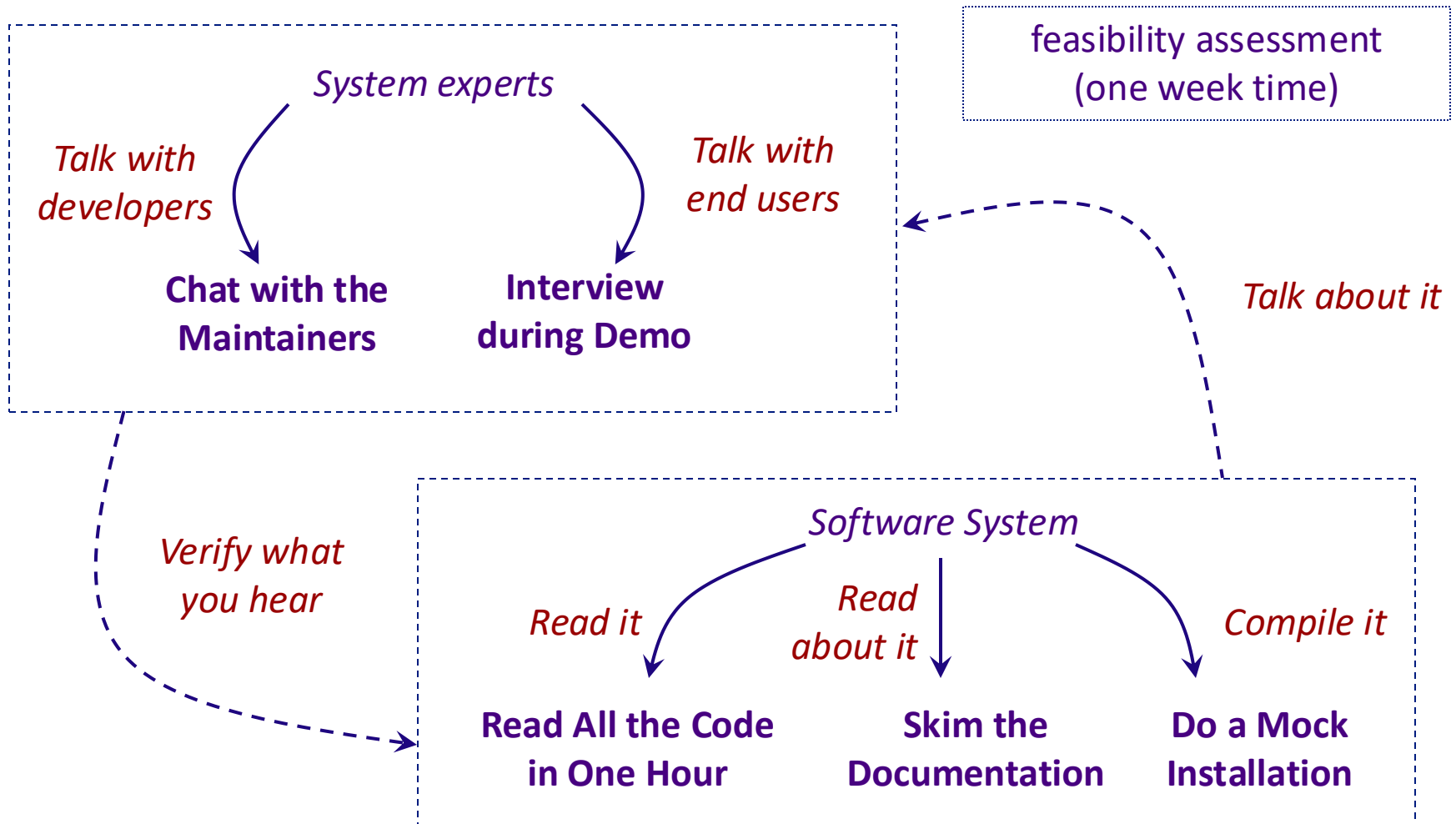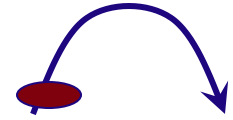*Understanding* other people's code

(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

New Feature

(1) model capture

Code

(4) program transformation

First Contact

System experts

Talk with developers

Chat with the Maintainers

Talk with end users

Interview during Demo

feasibility assessment
(one week time)

Talk about it

Verify what you hear

Software System

Read it

Read about it

Compile it

Read All the Code in One Hour

Skim the Documentation

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
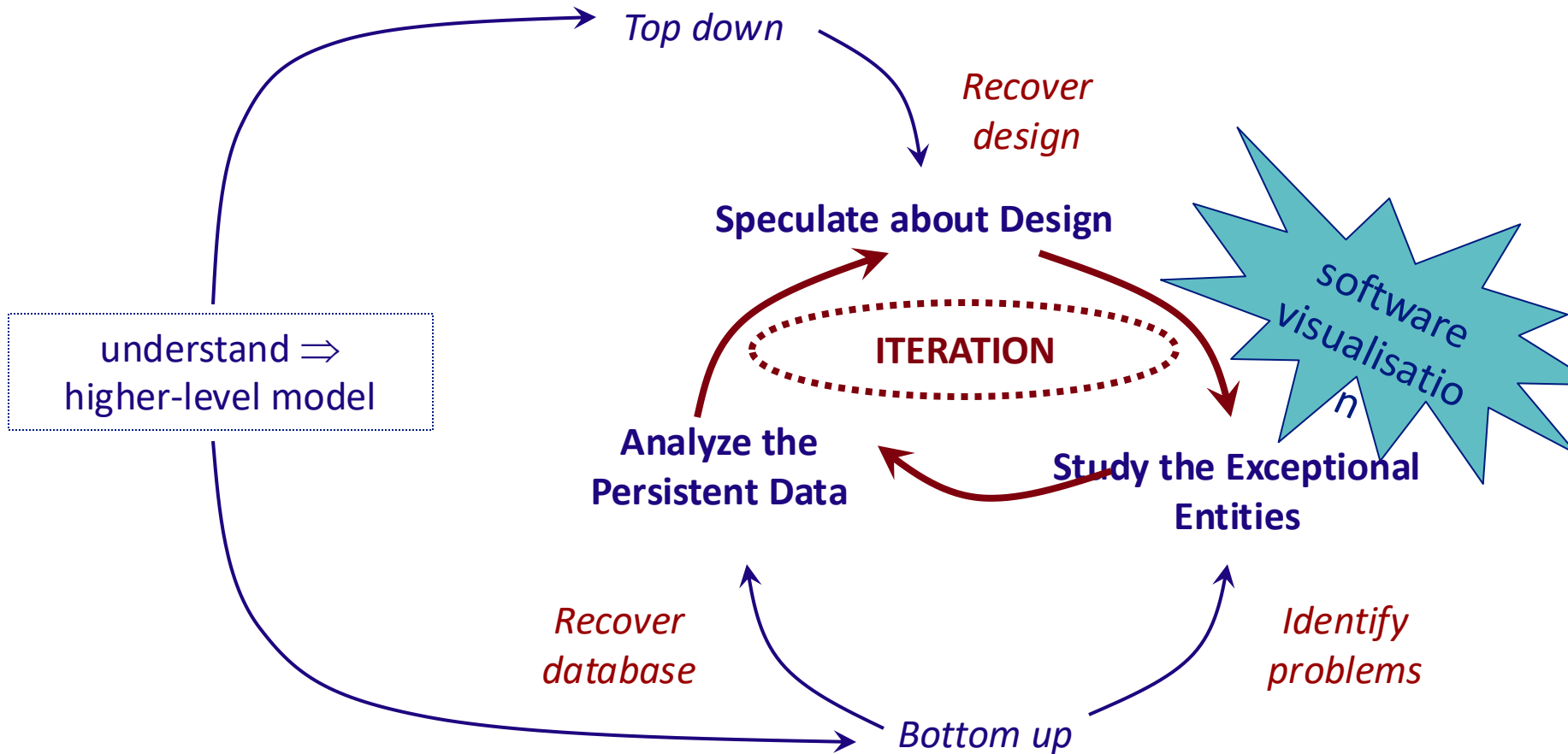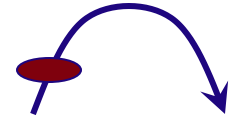- go/no-go decision
- activities
  + fish-eye view

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

understand $\Rightarrow$ higher-level model

ITERATION

software visualisation

Analyze the Persistent Data

Study the Exceptional Entities

Recover database

Identify problems
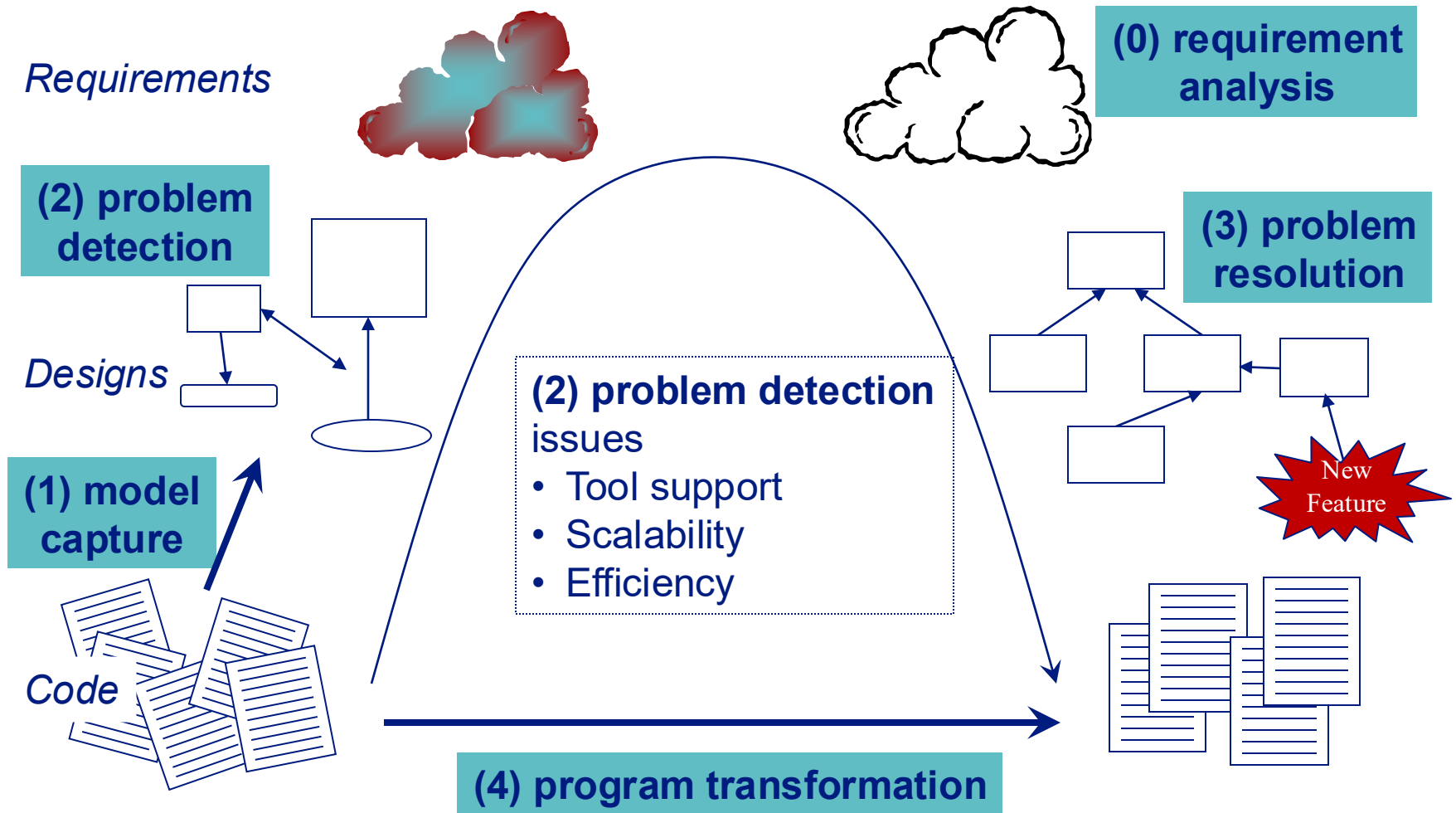
Bottom up

Object-Oriented Reengineering.18

3. Software Visualization

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

# The Reengineering Life-cycle

Requirements

(0) requirement analysis

(2) problem detection

(3) problem resolution

Designs

(2) problem detection
issues
- Tool support
- Scalability
- Efficiency

(1) model capture

New Feature
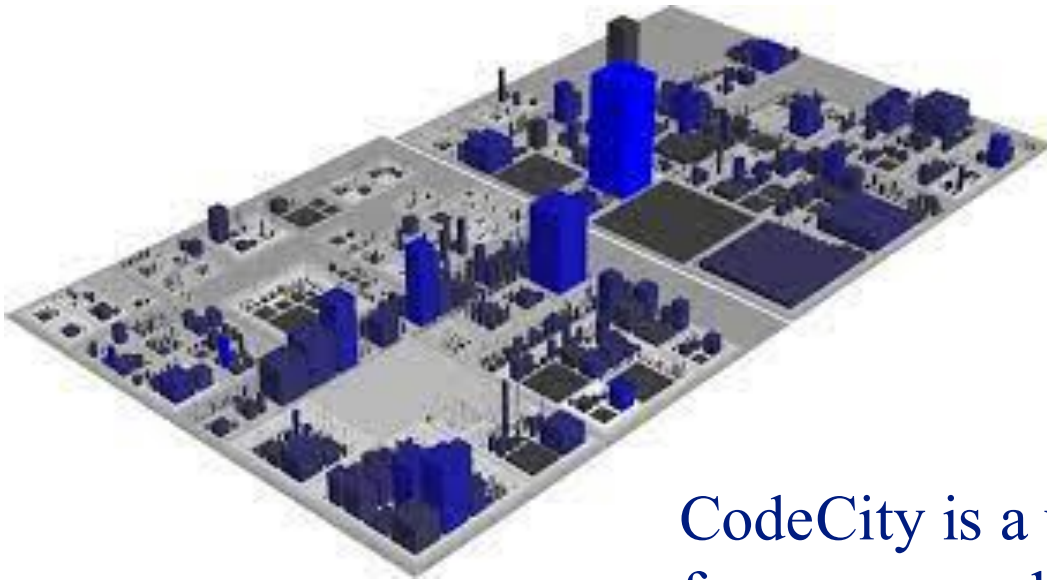
Code

(4) program transformation

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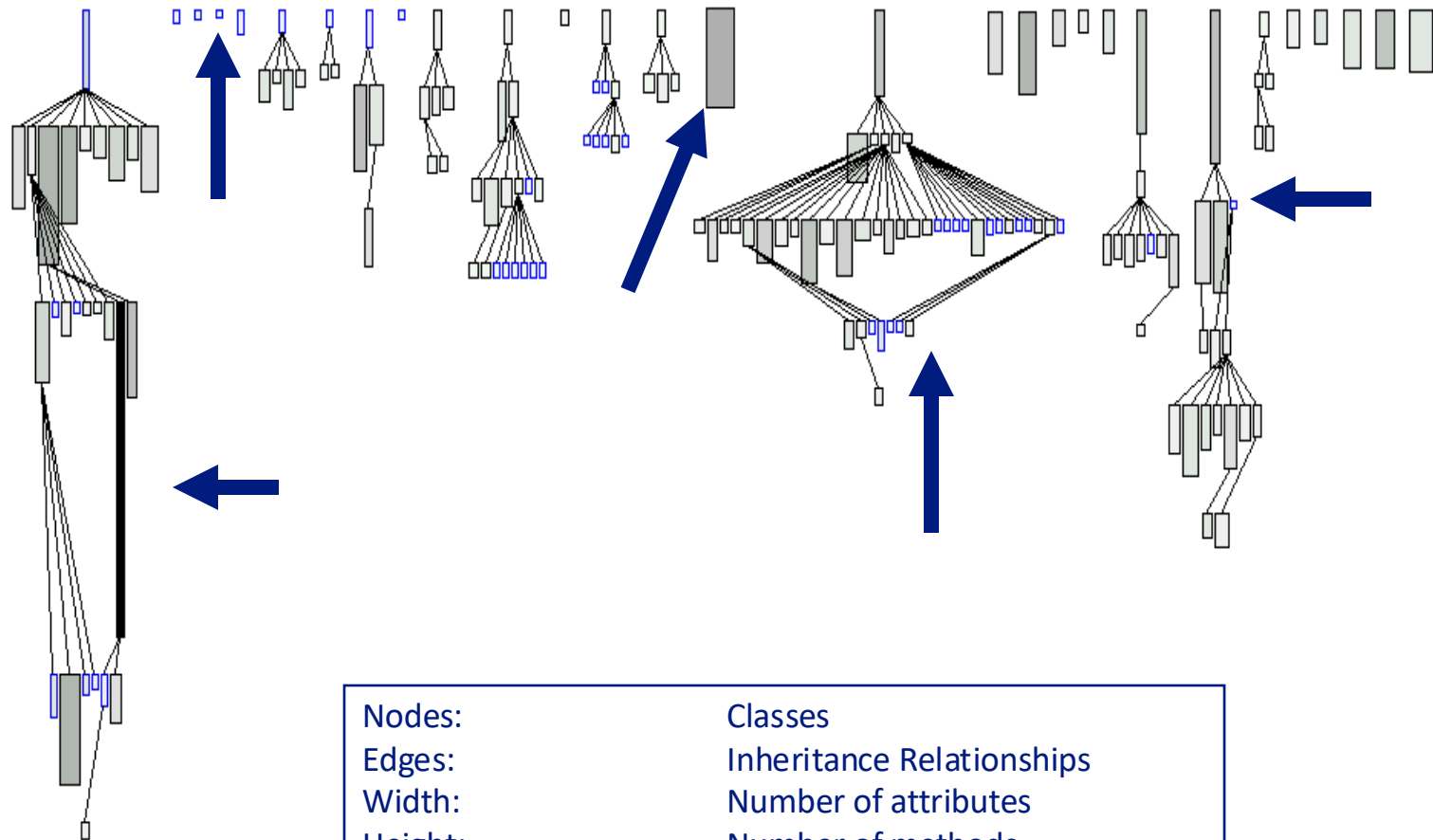
Nodes:	Classes
Edges:	Inheritance Relationships
Width:	Number of attributes
Height:	Number of methods
Color:	Number of lines of code

Code City

CodeCity is a visualization concept for source code.

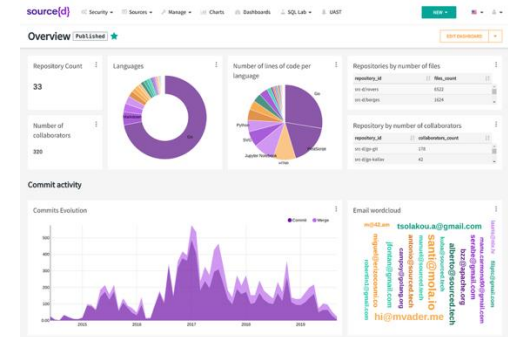The source code is shown as an interactive 3D city.

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

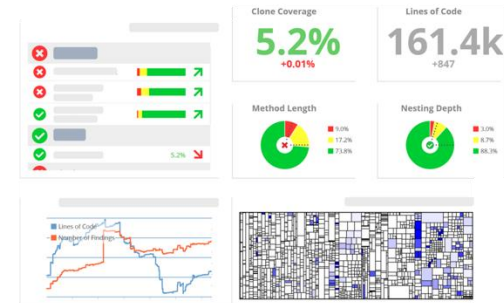# State of the Art Tooling

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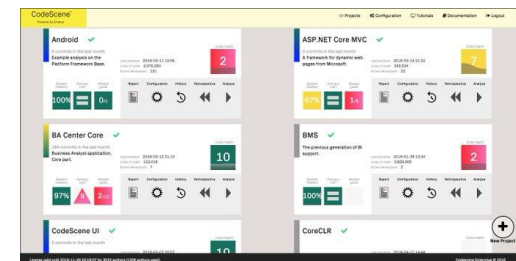   https://sourced.tech
   https://github.com/src-d/engine

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

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

4. Restructuring

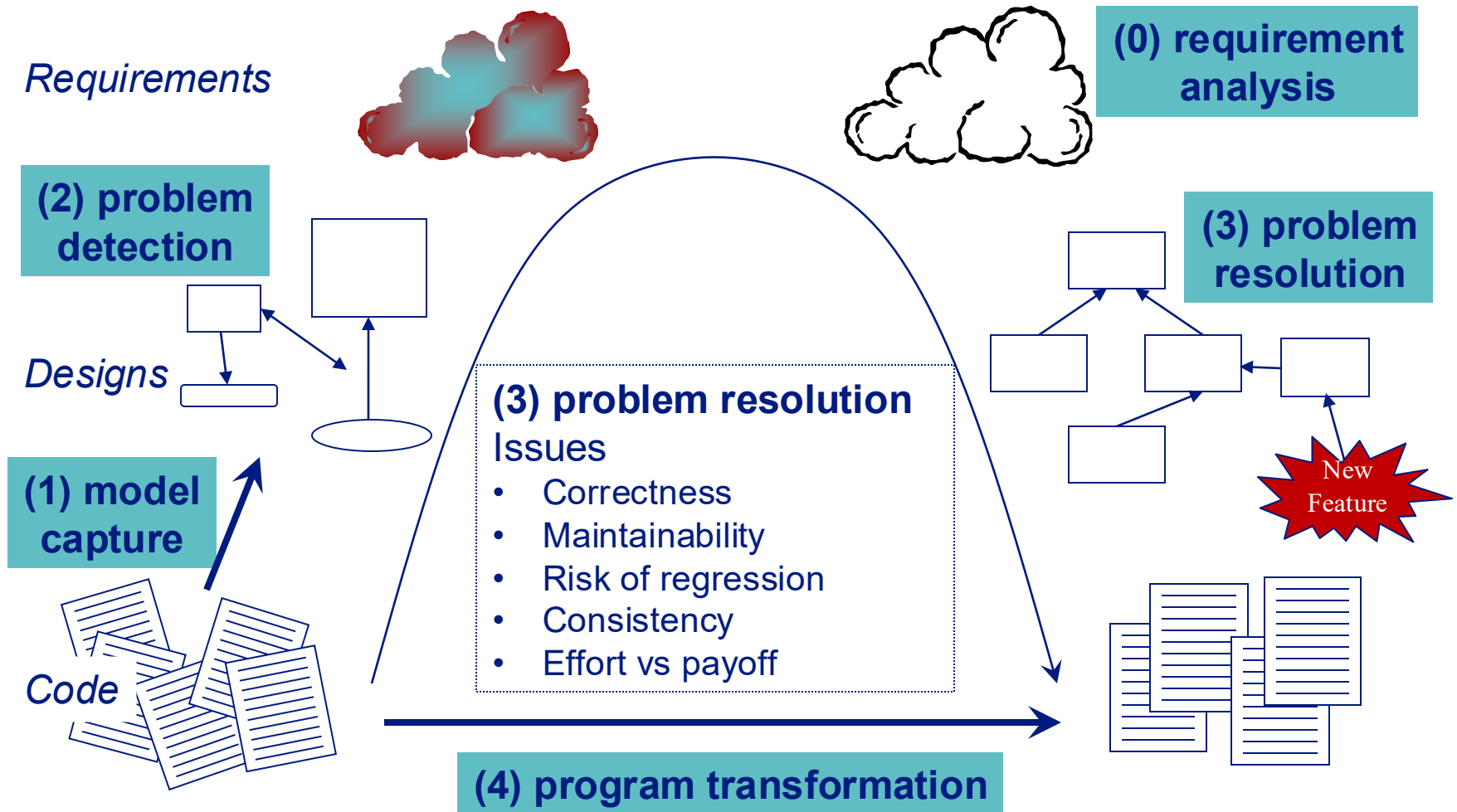**Identifying refactoring targets**

Redistribute Responsibilities

+ Move Behaviour Close to Data

+ Eliminate Navigation Code
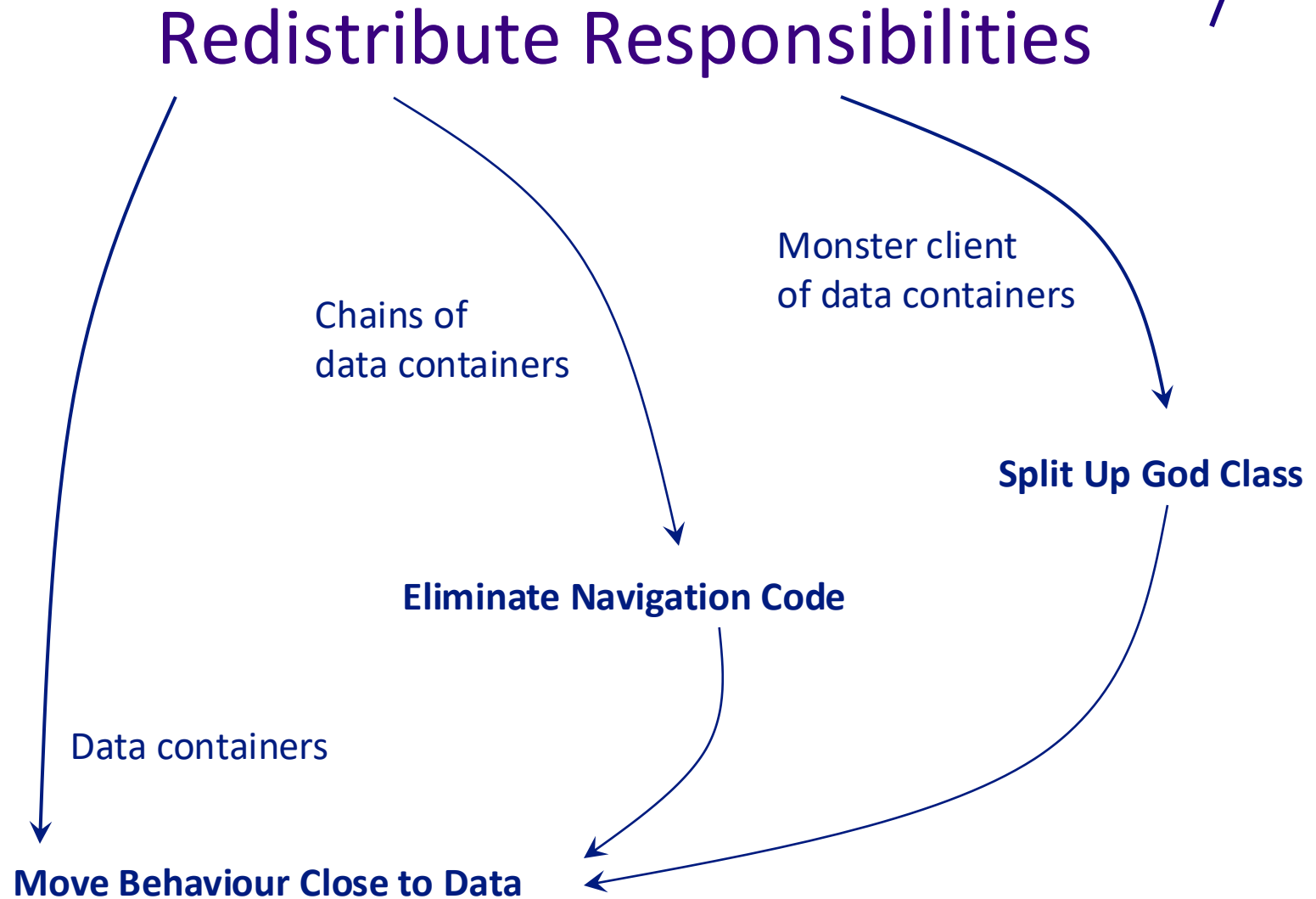
+ Split up God Class

+ Empirical Validation

Identifying refactorings in code

❖ Refactoring-aware techniques

The Reengineering Life-cycle

Requirements

(0) requirement analysis

(2) problem detection

(3) problem resolution

Designs

(1) model capture

(3) problem resolution
Issues
- Correctness
- Maintainability
- Risk of regression
- Consistency
- Effort vs payoff

New Feature

Code

(4) program transformation

Identifying Refactoring Targets

Redistribute Responsibilities

Chains of
data containers

Monster client
of data containers

Split Up God Class

Eliminate Navigation Code

Data containers

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
      $\Rightarrow$ 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)

EmployeeManager

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

PaymentManager

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

ScheduleManager

+addEmployeeSchedule(int employeeId, Schedule sch)

EmployeeContactManager

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

VacationManager

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

EquipmentManager

+assignEquipment(int employeeId, Equipment eq)

Refactoring-Aware Techniques

Many refactoring-aware techniques:

- IntelliMerge & Refmerge – merge branches

- APIDiff – adapt client software to library and framework updates

- Ogenrwot and Businge (SCAM 2025)

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]

- The tool takes an input two revisions (e.g. commits) and returns a list of refactorings

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
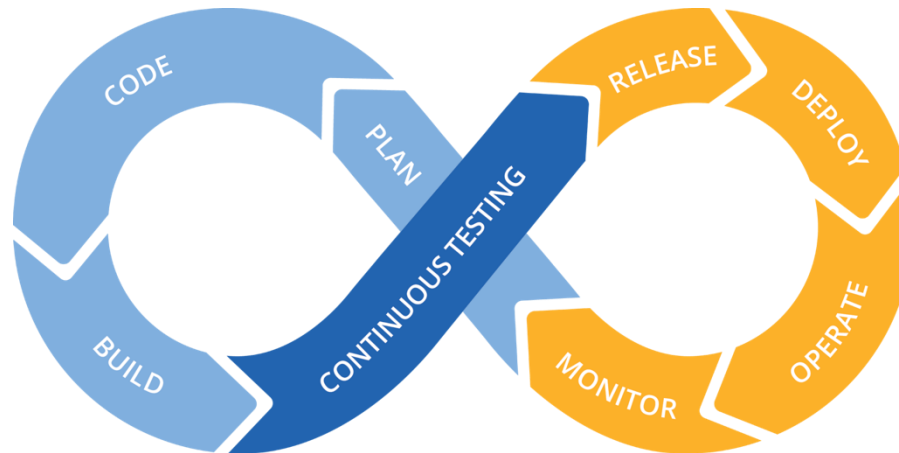
Testing

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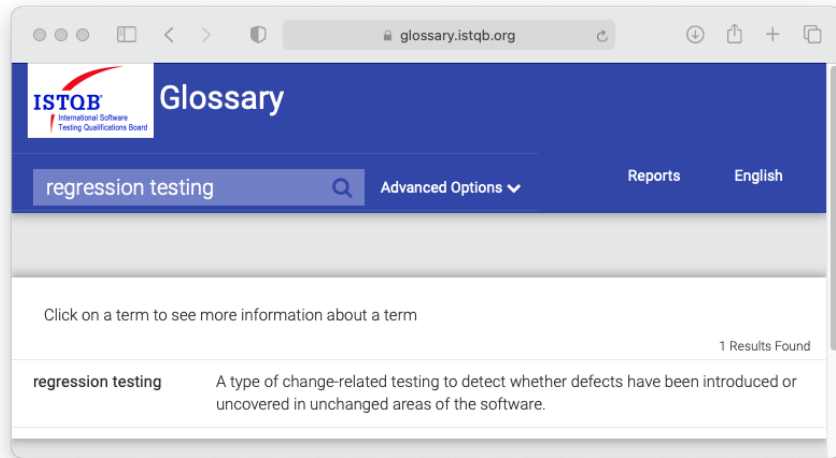
A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software.
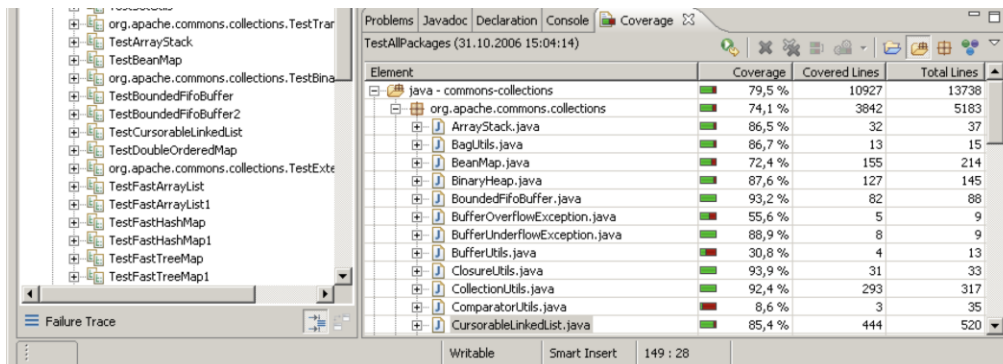
Coverage

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

Compare coverage reports before and after refactoring!

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

Social Fork

Original project

Social Fork

(3) problem resolution

Designs

(4) Code Integration

issues
- Conflicting changes

New Feature

(1) model capture

Code

(4) integration

Social Fork

Original project

Social Fork

Pull request

(4) program transformation
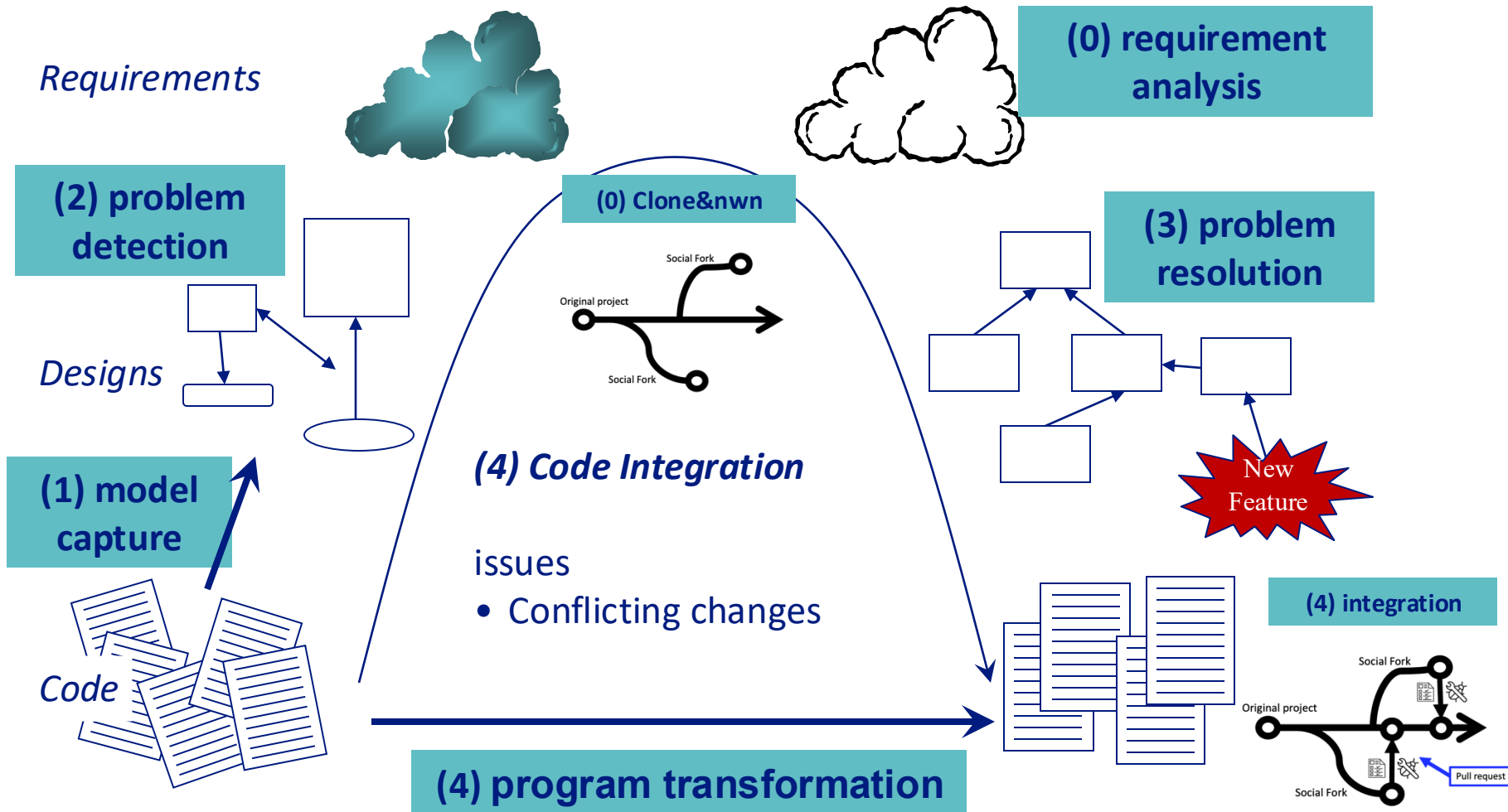
Collaborative developemt/ Merge Conflict

FooHelper.Java
Alice

```
. . .

+ foo() {

+   int x = getX();
+   int y = getY();
+   calcDist(x, y);
+ }

. . .
```

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);
-}
. . .
```

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);
+ }

. . .
```
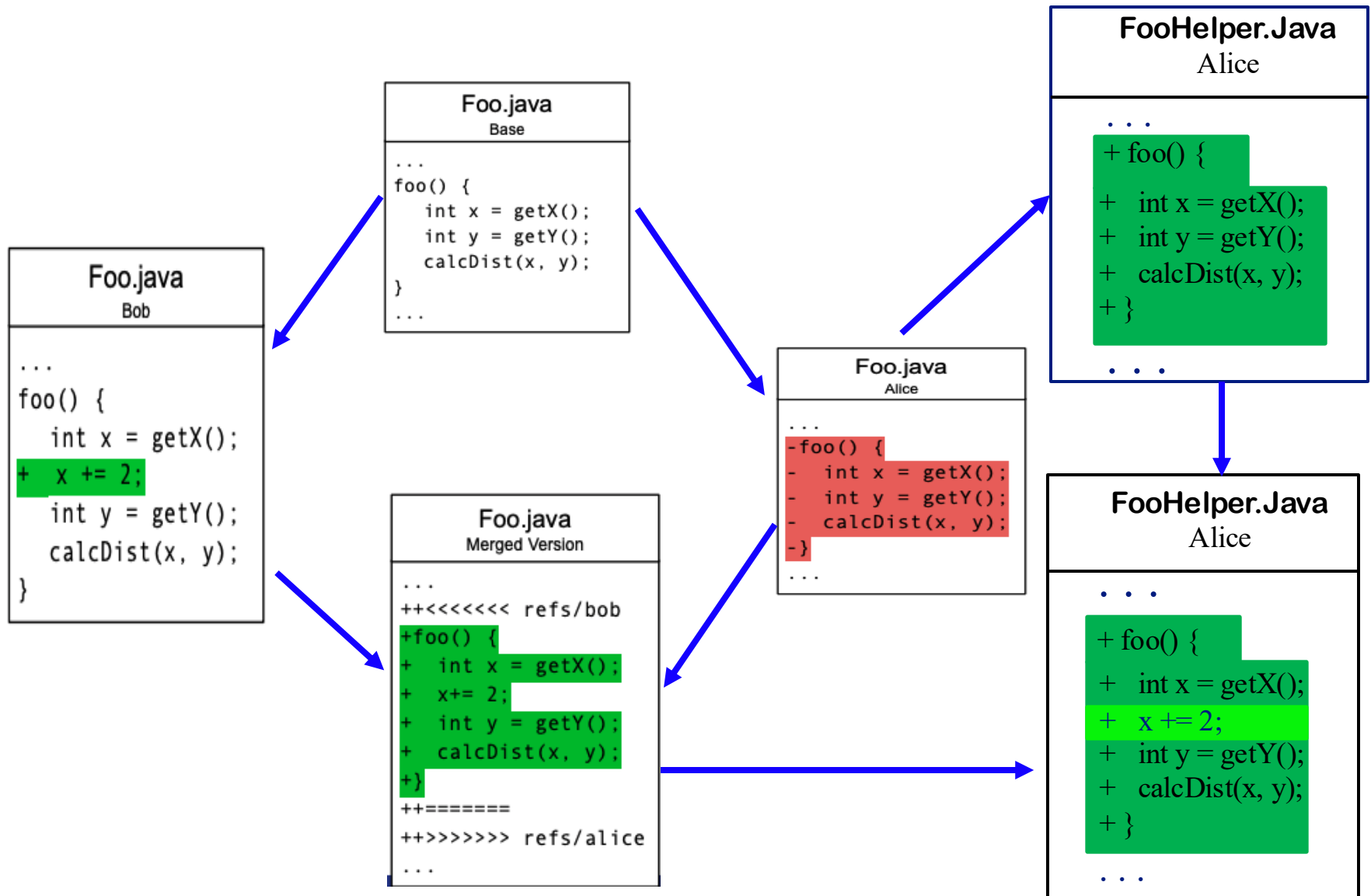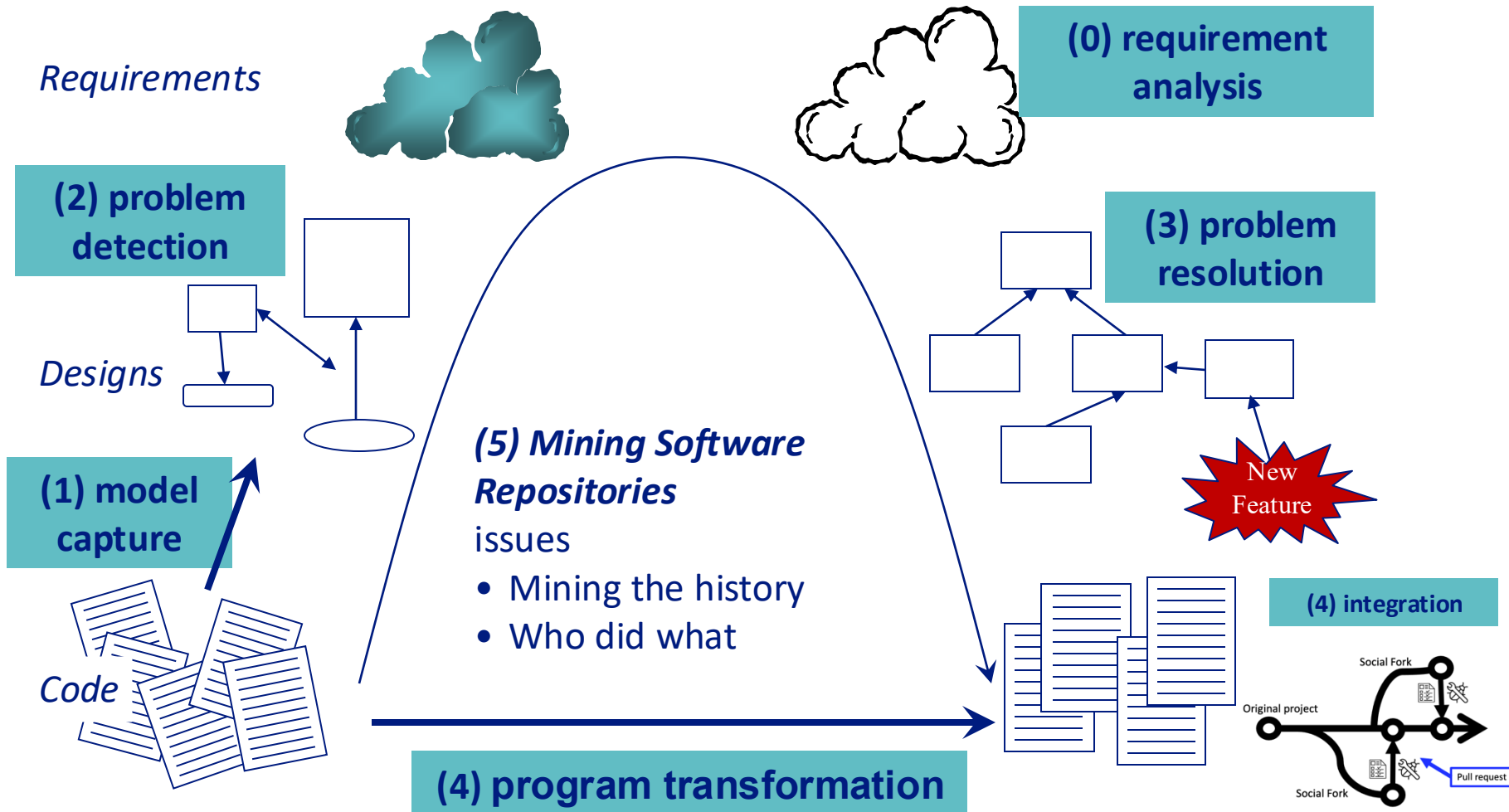
7. Mining Software Repositories (MSR)

- 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

(3) problem resolution

Designs

(1) model capture

(5) Mining Software Repositories

issues

- Mining the history
- Who did what

New Feature

(4) integration

Code

(4) program transformation

Social Fork

Original project

Pull request

Social Fork

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

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

x class comparison in `AlterConfigPolicy.Req...

Contributors 884

+ 873 contributors

1.7 k

ॐ 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

📁 config   MINOR
📁 connect   KAFKA
📁 core   MINOR
📁 docs   KAFKA
📁 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

Languages

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

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?
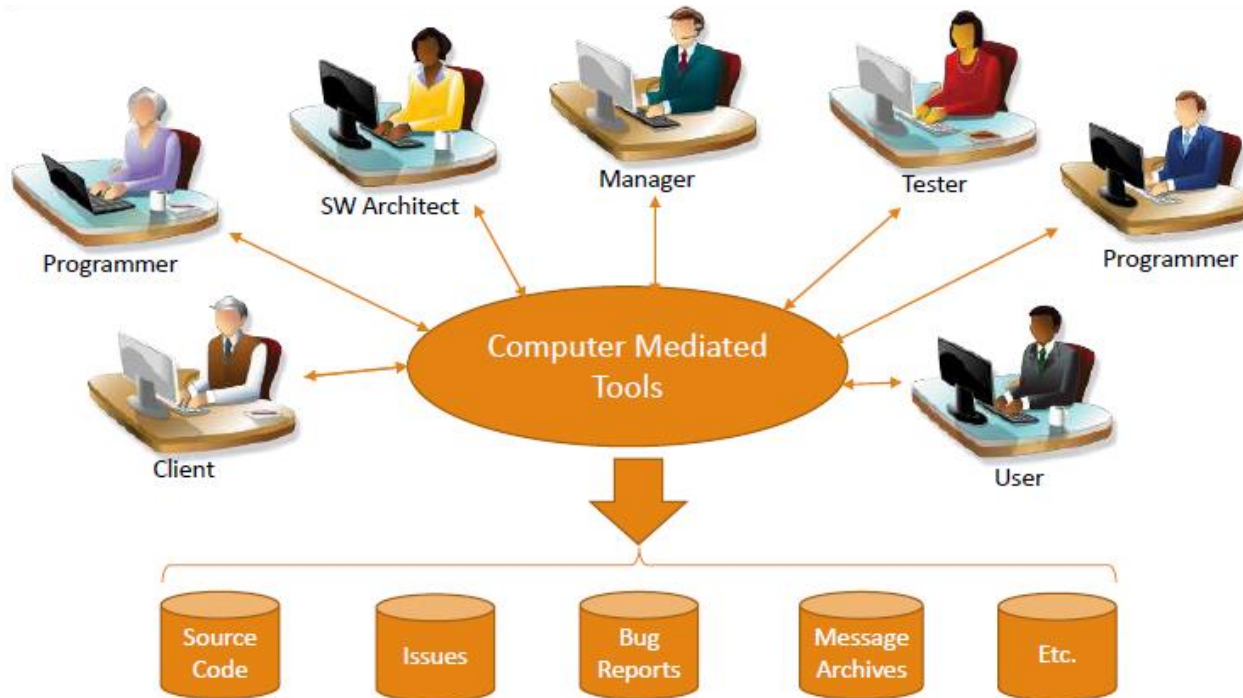
Current and historical artifacts and interactions are registered in software repositories

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

Developers who touched files
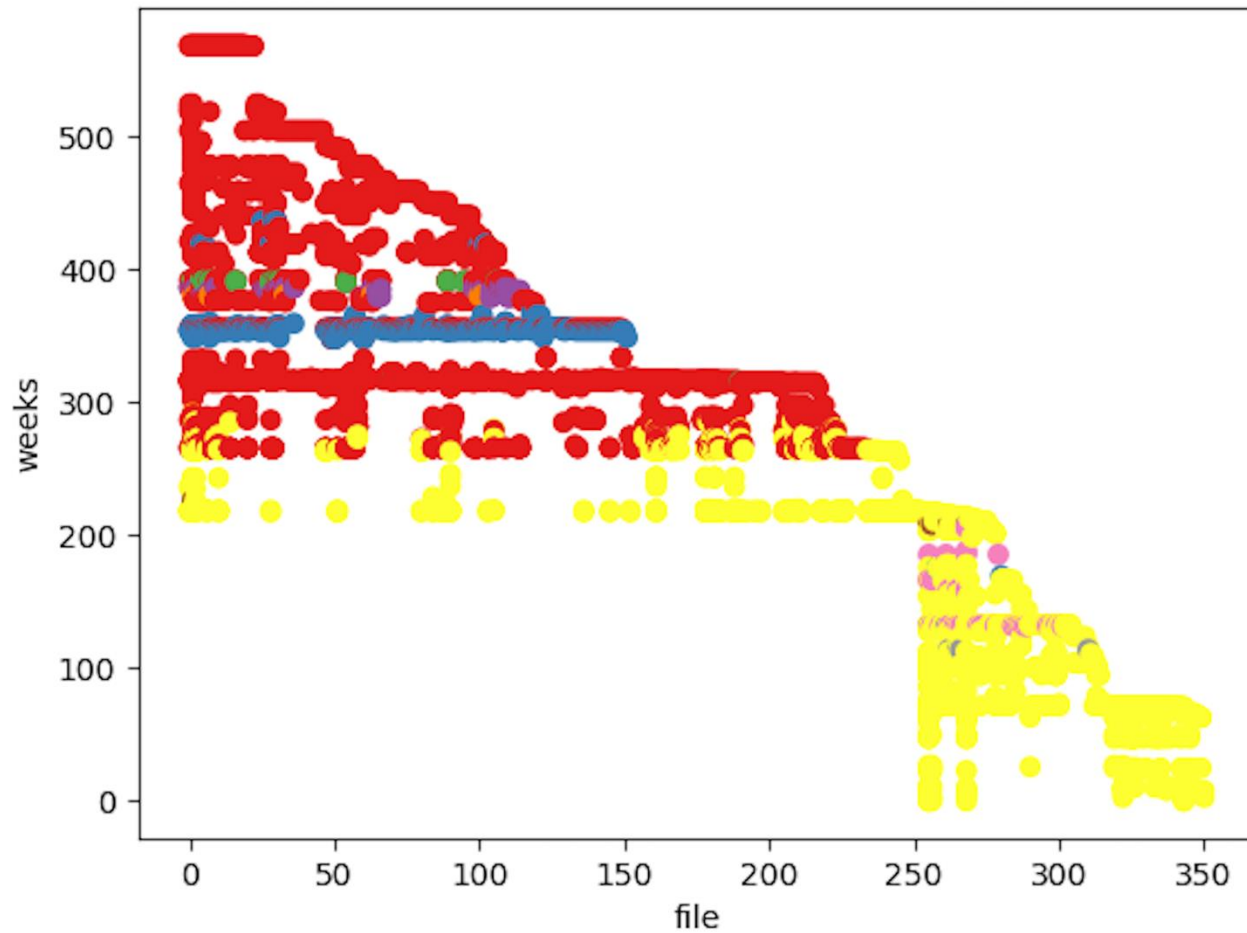
9. Conclusion

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.