Software Reenineering
Reverse Engineering

Martin Pinzger
Delft University of Technology

Outline

What is and Why?

Initial understanding

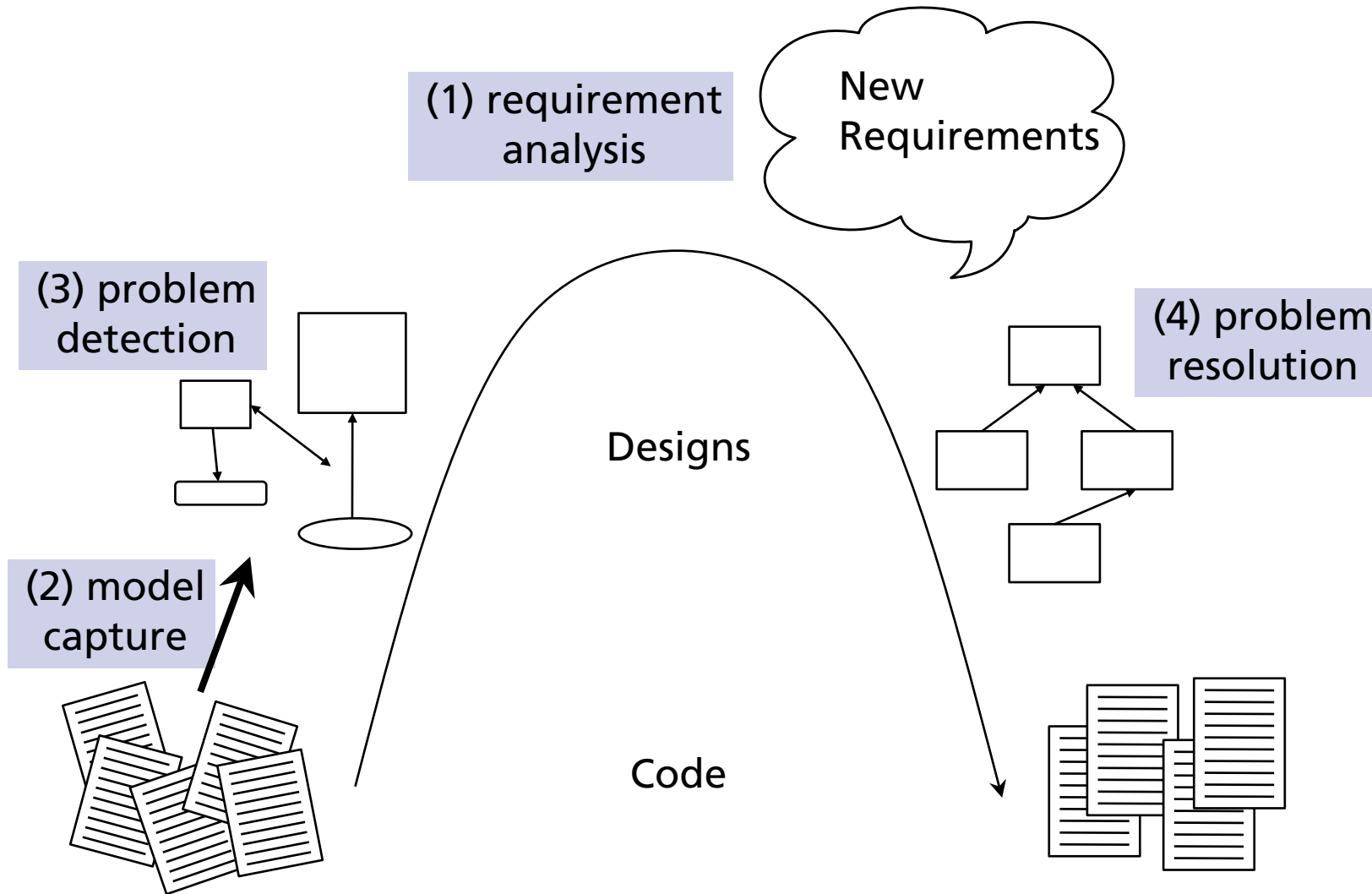Detailed model capture

DA4Java demo

What is Reverse Engineering and why?

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, the design and architecture in order to maintain and evolve them

Reengineering Life-Cycle

(1) requirement analysis

New Requirements

(3) problem detection

(4) problem resolution

Designs

(2) model capture

Code

Initial understanding

Initial understanding patterns

Goal: Get initial understanding of the design and implementation of the system

Forces

Data is deceptive

Always double-check your sources

Understanding entails iteration
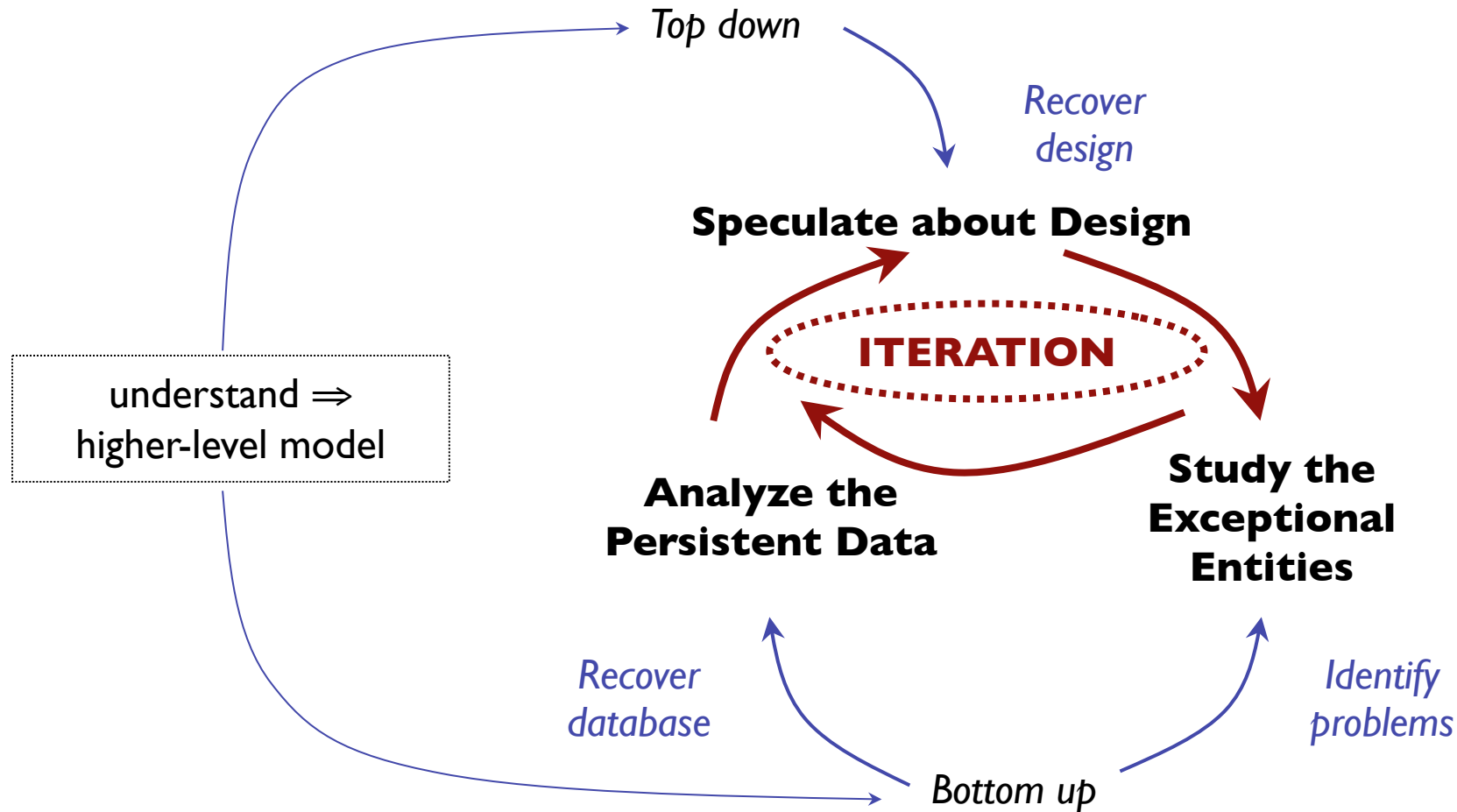
Plan iteration and feedback loops

Knowledge must be shared

"Put the map on the wall"

Teams need to communicate

"Use their language"

Tuesday, September 21, 2010

Initial understanding patterns

Tuesday, September 21, 2010

Analyze the persistent data

Problem: Which objects represent valuable data?

Solution: Analyze the database schema

Prepare Model

Table ⇒ class

Columns ⇒ class attributes

Candidate keys

Naming conventions + unique indices

Foreign keys ⇒ class associations

Use explicit foreign key declarations

Infer from column types + naming conventions + view declarations + join clauses

Analyze the persistent data (cont.)

Incorporate Inheritance

    One to one; rolled down; rolled up

Incorporate Associations

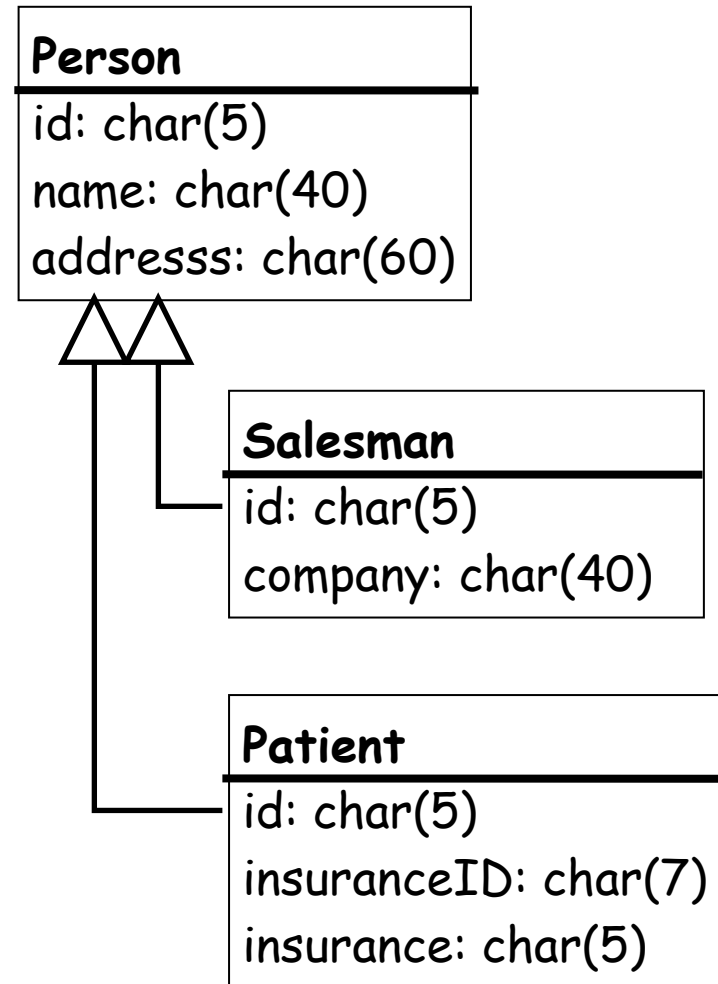   Determine association classes (e.g., many-to-many associations)
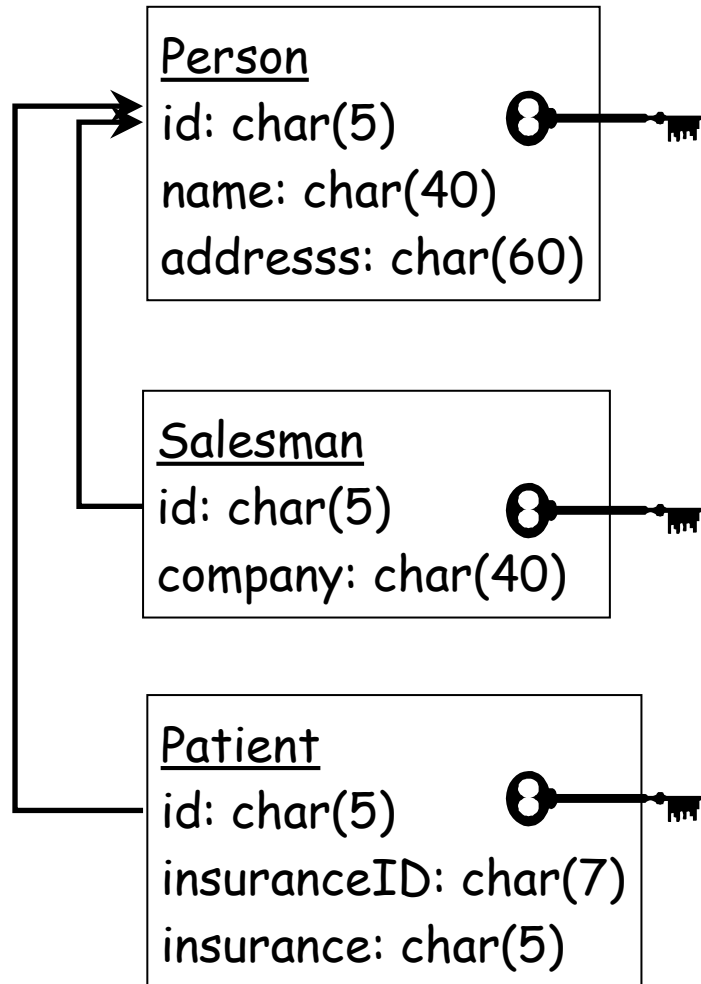
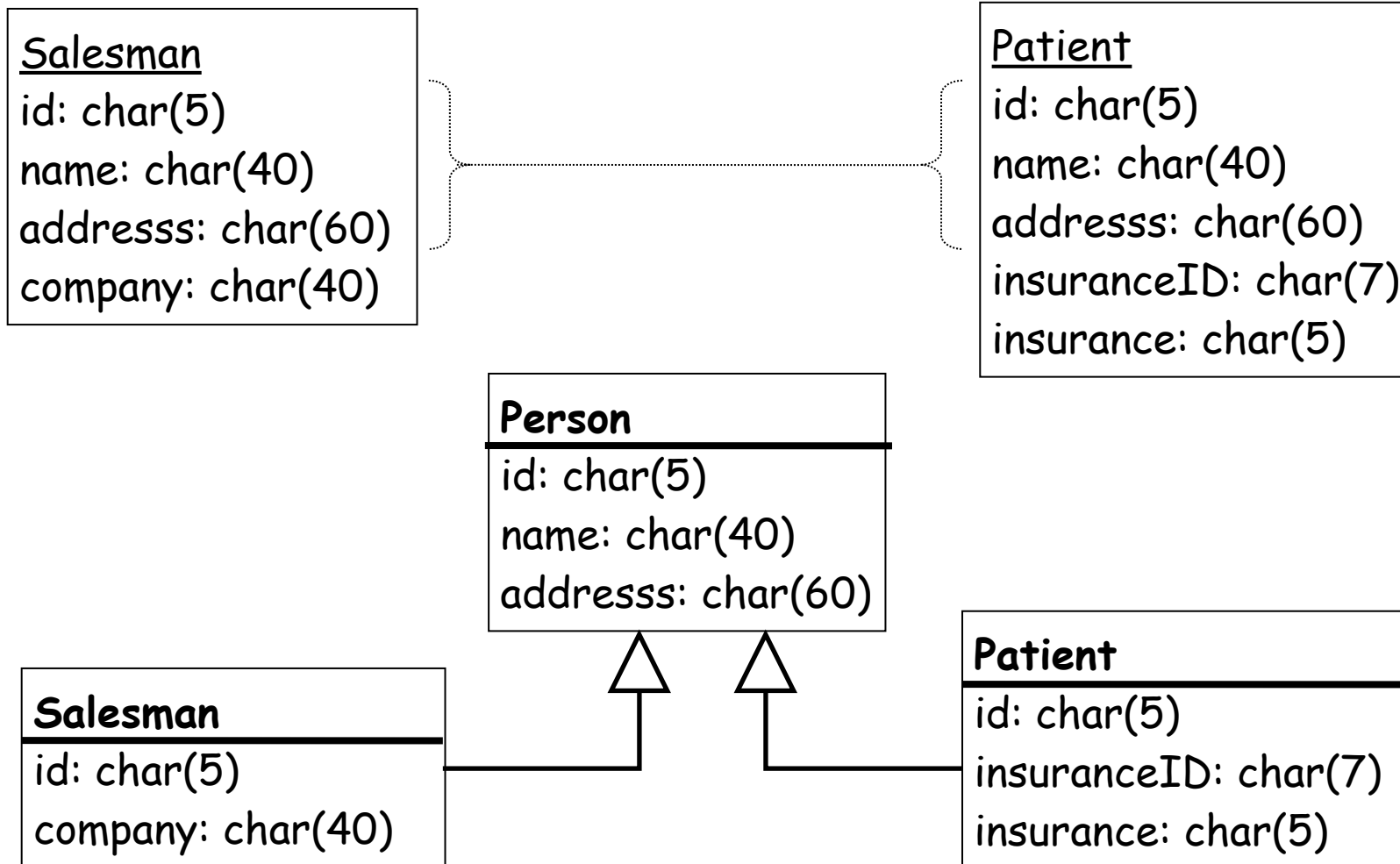   Merge complementary associations

    Identify qualified associations

Verification

    Data samples + SQL statements

Example: One To One

Example: Rolled Down

Salesman
id: char(5)
name: char(40)
addresss: char(60)
company: char(40)

Patient
id: char(5)
name: char(40)
addresss: char(60)
insuranceID: char(7)
insurance: char(5)

Person
id: char(5)
name: char(40)
addresss: char(60)

Salesman
id: char(5)
company: char(40)

Patient
id: char(5)
insuranceID: char(7)
insurance: char(5)

Tuesday, September 21, 2010

Excercise: Analyze the peristent data

Analyze the given ER diagram

Draw a class simple diagram

Speculate about design

Problem: How do you recover the design from source code?

Solution: Develop hypotheses and check them

 Develop a plausible class diagram and iteratively check and refine your design against the actual code

Variants

 Speculate about Business Objects

 Speculate about Design Patterns

 Speculate about Architecture

Study the exceptional entities

Problem: How can you quickly identify design problems?

Solution: Measure software entities and study the anomalous ones
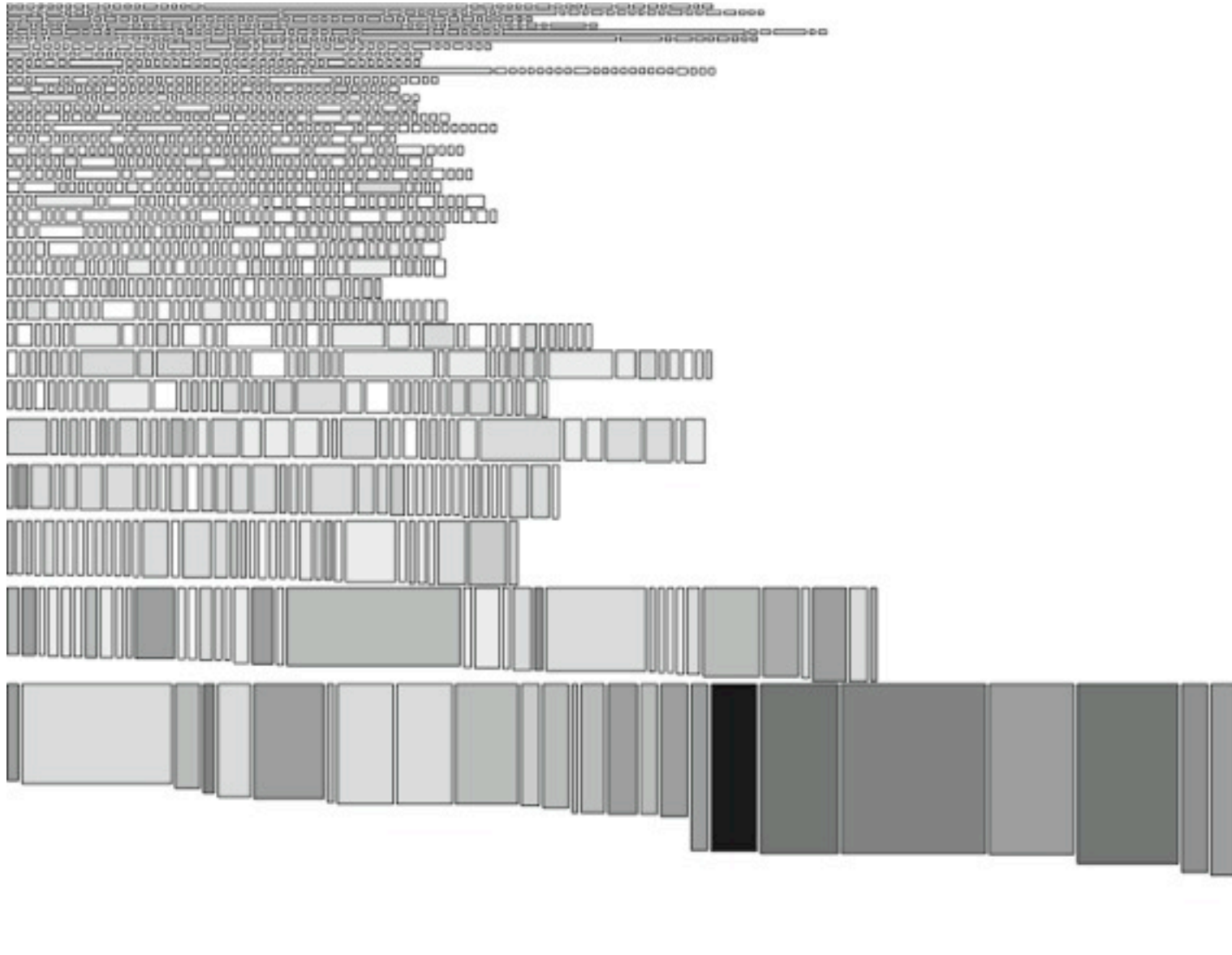
- Visualize metrics to get an overview
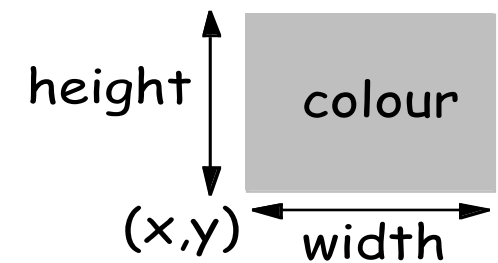
- Use simple metrics

    - Lines of code

    - Number of methods

    - ...

Example: Exceptional entities

Use simple metrics and layout algorithms

Detailed model capture

Detailed model capture patterns

Goal: Build a detailed model of parts that will be important for reengineering

Forces

Details matter

Pay attention to the details

Design remains implicit
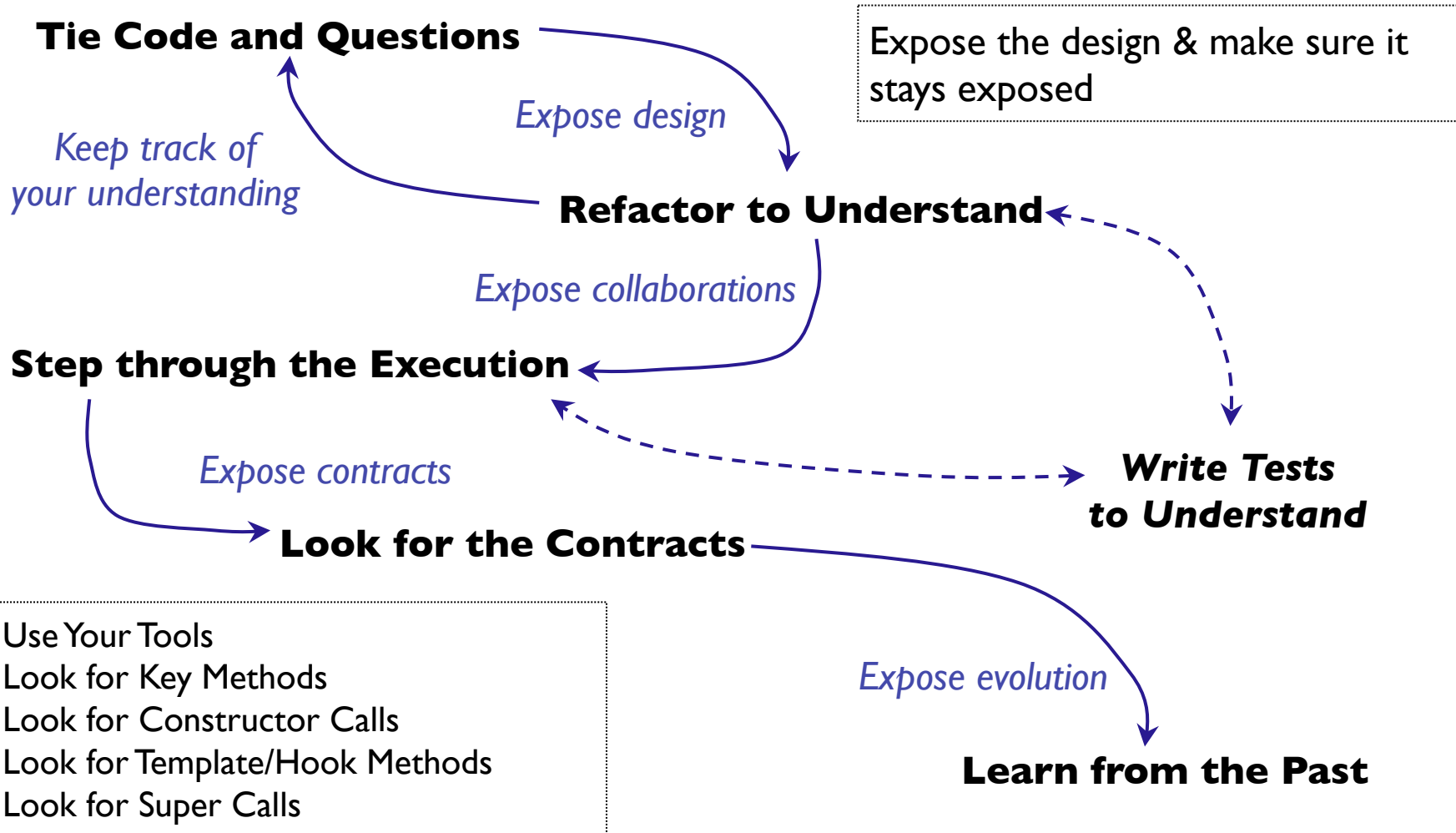
Record design rationale when you discover it

Design evolves

Important issues are reflected in changes to the code

Code only exposes static structure

Study dynamic behaviour to extract detailed design

Detailed model capture patterns

Tie Code and Questions

Expose design

Expose the design & make sure it stays exposed

Keep track of your understanding

Refactor to Understand

Expose collaborations

Step through the Execution

Write Tests to Understand

Expose contracts

Look for the Contracts

- Use Your Tools
- Look for Key Methods
- Look for Constructor Calls
- Look for Template/Hook Methods
- Look for Super Calls

Expose evolution

Learn from the Past

Tie code and questions

Problem: How do you keep track of your understanding?

Solution: Annotate the code

- List questions, hypotheses, tasks and observations

- Identify yourself

- Use conventions to locate/extract annotations
    - E.g., 'To: Jasmine By: Martin On: 10.10.05 Comment...'
- Annotate as comments or as methods

Refactor to understand

Problem: How do you decipher cryptic code?

Solution: Refactor it till it makes sense

Goal (for now) is to understand, not to reengineer

Hints

Work with a copy of the code

Refactoring requires an adequate test base

If this is missing, "Write Tests to Understand"

Refactor to understand (cont.)

Guidelines

- Rename attributes to convey roles

- Rename methods and classes to reveal intent

- Remove duplicated code

- Replace condition branches by methods

Step through the execution

Problem: How do you uncover the run-time architecture?

    Collaborations are spread throughout the code

    Polymorphism may hide which classes are instantiated

Solution: Execute scenarios of known use cases and step through the code with a debugger

Hints

    Set breakpoints

    Change internal state to test alternative paths

Look for the contracts

Problem: What does a class expect from its clients?

Interfaces are visible in the code but how to use them?

Solution: Look for common programming idioms

Look for "key methods"

Method name, parameter types (important type -> important method)

Constructor calls

Shows which parameters to pass

Template/hook methods

Shows how to specialize a sub-class

Example: yFiles Contract

Initializing a Swing component with a yFiles graph

```java
public SNACockpit(DataProvider dataProvider, boolean animated) {
    super(new BorderLayout());

    this.fGraphModel = new SocialNetworkGraph(dataProvider);
    view = new Graph2DView();
    view.setAntialiasedPainting(true);
    ((DefaultGraph2DRenderer) view.getGraph2DRenderer()).setDrawEdgesFirst(true);

    ...

    view.setGraph2D(fGraphModel);
    this.add(view, BorderLayout.CENTER);
}
```

Learn from the past

Problem: How did the system get the way it is? Which parts are stable and which aren't?

Solution: Compare versions to discover where code was removed

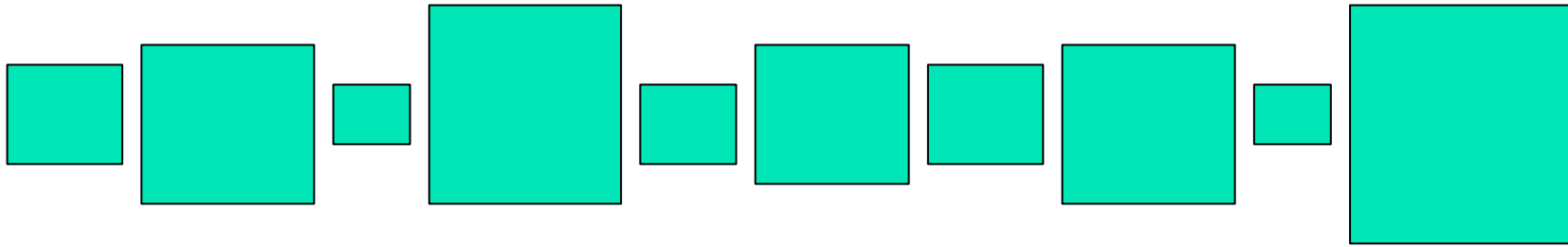- Removed functionality is a sign of design evolution

- Use or develop appropriate tools

- Look for signs of:

  - Unstable design — repeated growth and refactoring

  - Mature design — growth, refactoring, and stability

Examples: Unstable design

Pulsar: Repeated Modifications make it grow and shrink.
System Hotspot: Every System Version requires changes.

Summary

Setting Direction + First Contact

⇒ First Project Plan

Initial Understanding + Detailed Model Capture

Plan the work … and work the plan

Frequent and short iterations

Issues

Scale, speed vs. accuracy, politics

Tools?

DA4Java demo

Other visualization tools/prototypes

Structural Analysis for Java

   http://www.alphaworks.ibm.com/tech/sa4j

X-Ray

   http://xray.inf.usi.ch/xray.php

Code City

   http://www.inf.usi.ch/phd/wettel/codecity.html

Homework

Read the copies (pdfs) about

  Code Smells

  Object-Oriented Design Principles (class design)