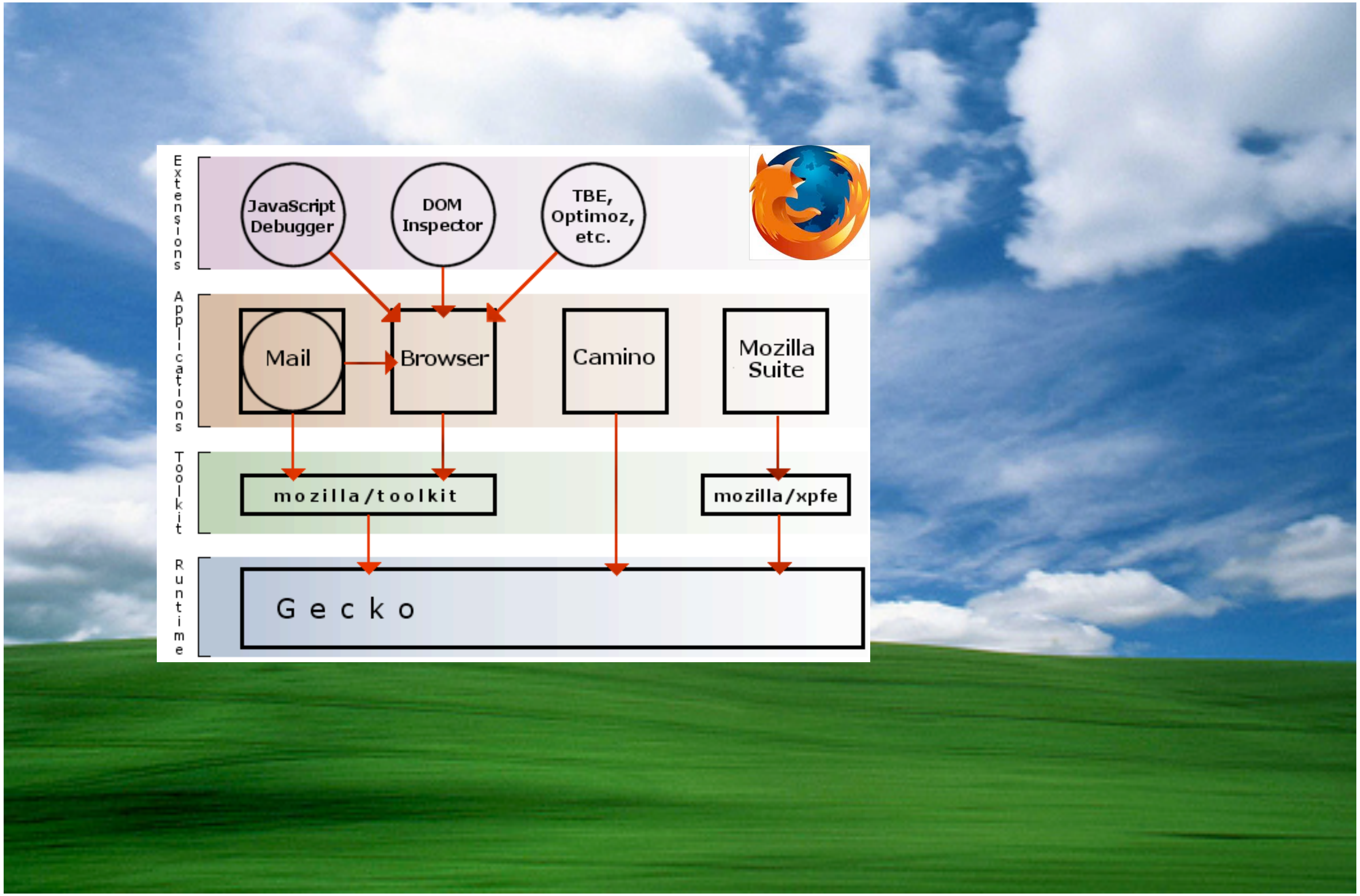
Software Reengineering P1: Intro & Organization

Martin Pinzger
Delft University of Technology

Greenfield software development



Extensions

JavaScript Debugger

DOM Inspector

TBE, Optimoz, etc.

Applications

Mail

Browser

Camino

Mozilla Suite

Toolkit

mozilla/toolkit

mozilla/xpfe

Runtime

Gecko

Non-greenfield software development

How often did you …

… encounter greenfield and non-greenfield software engineering?

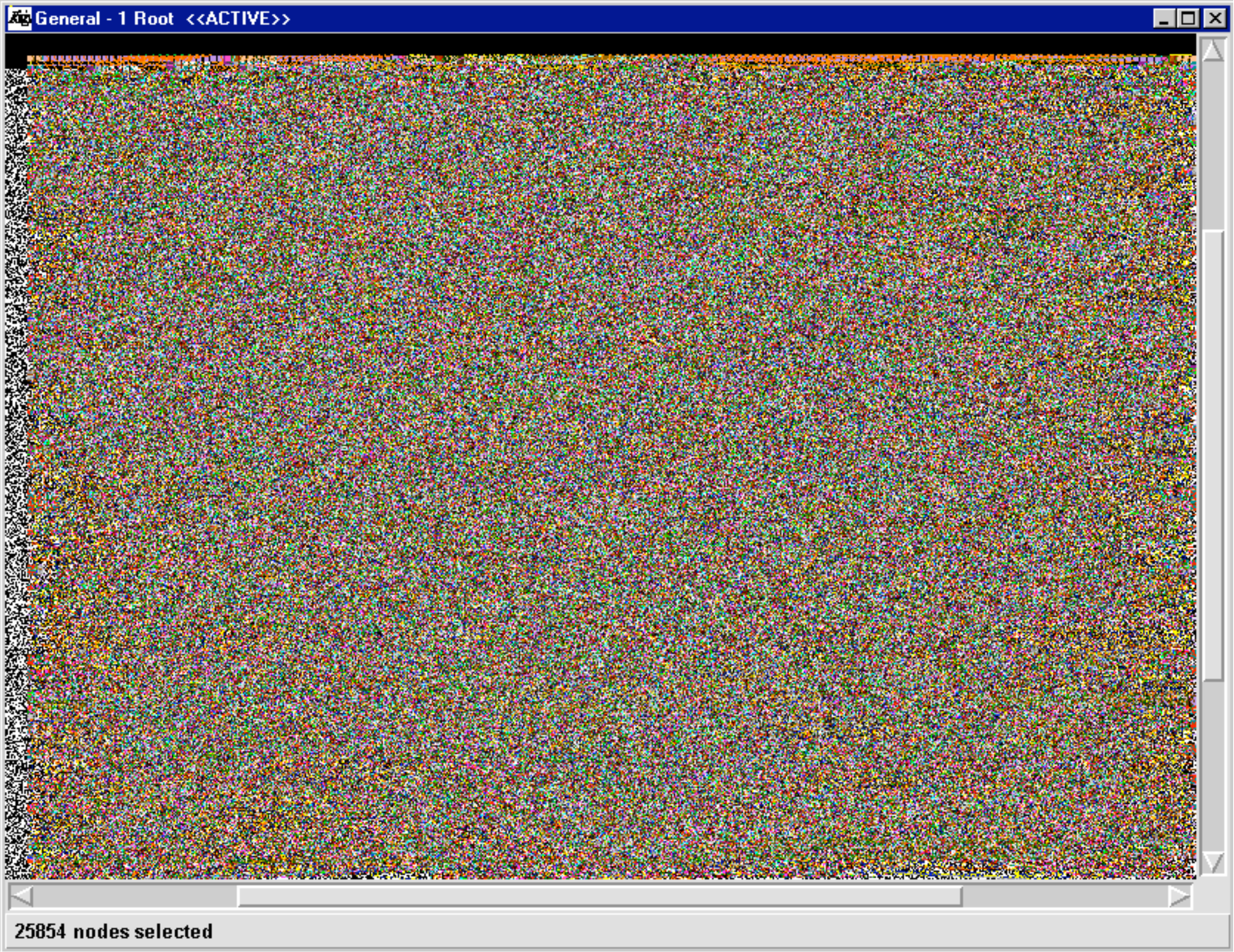
Why non-greenfield engineering?

Because existing software, often called legacy software, is valuable

- Often business-critical
- A huge amount of money has already been invested in it
- Has been tested and runs
- Does (mainly) what it should do

Would you replace such a system?

Why do we (often) start from a mess?



25854 nodes selected

Lehman’s Laws of software evolution

Continuing change

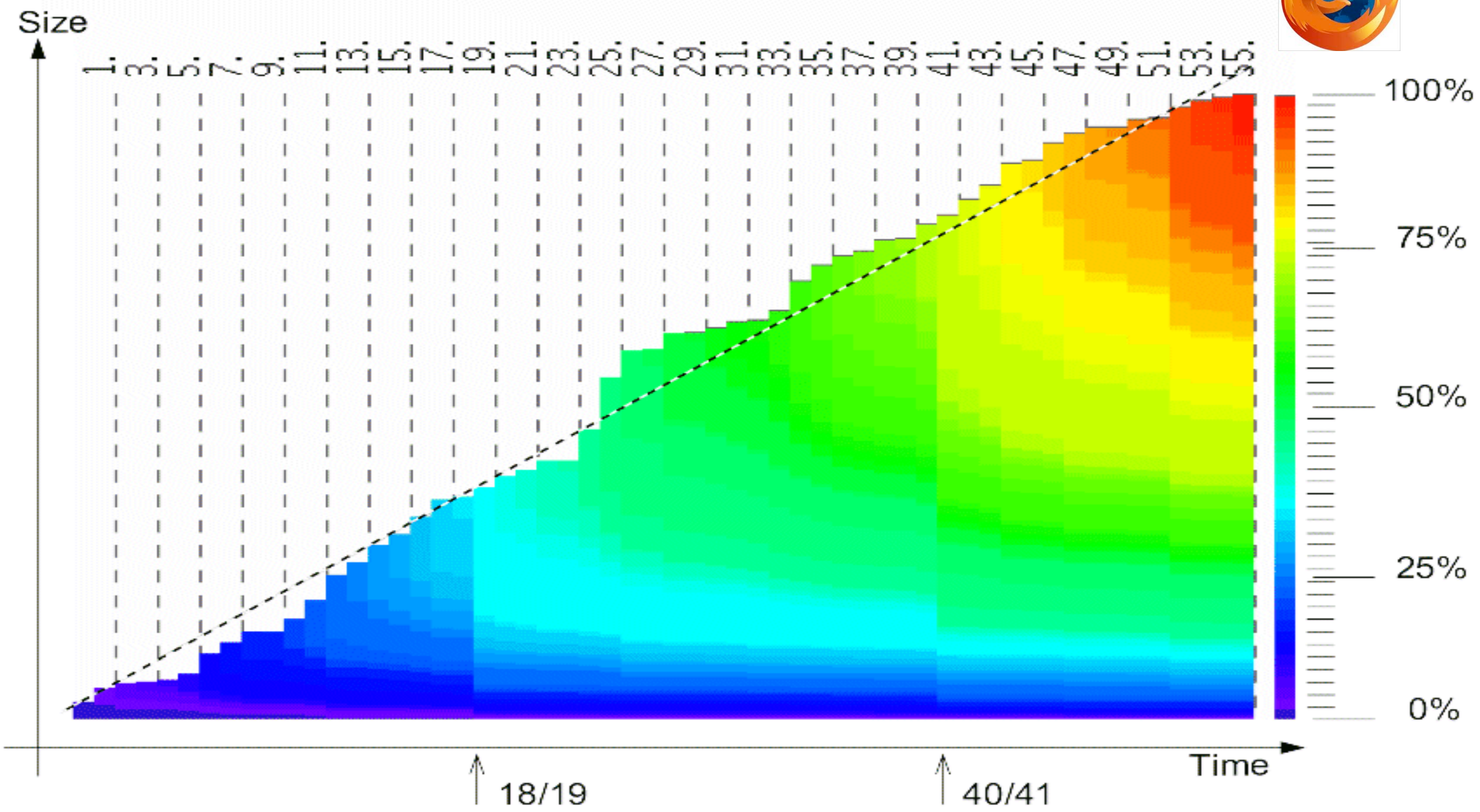
A program that is used in a real-world environment **must change**, or become progressively less useful in that environment.

Increasing complexity

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

For more information read Lehman and Belady, 1985

Evolution of Mozilla source code



Lehman’s Laws in practice

Existing software Is often modified in an ad-hoc manner (quick fixes)

- Lack of time, resources, money, etc.

Initial good design is not maintained

- Spaghetti code, copy/paste programming, dependencies are introduced, no tests, etc.

Documentation is not updated (if there is one)

- Architecture and design documents

Original developers leave and with them their knowledge

Typical result of such practices

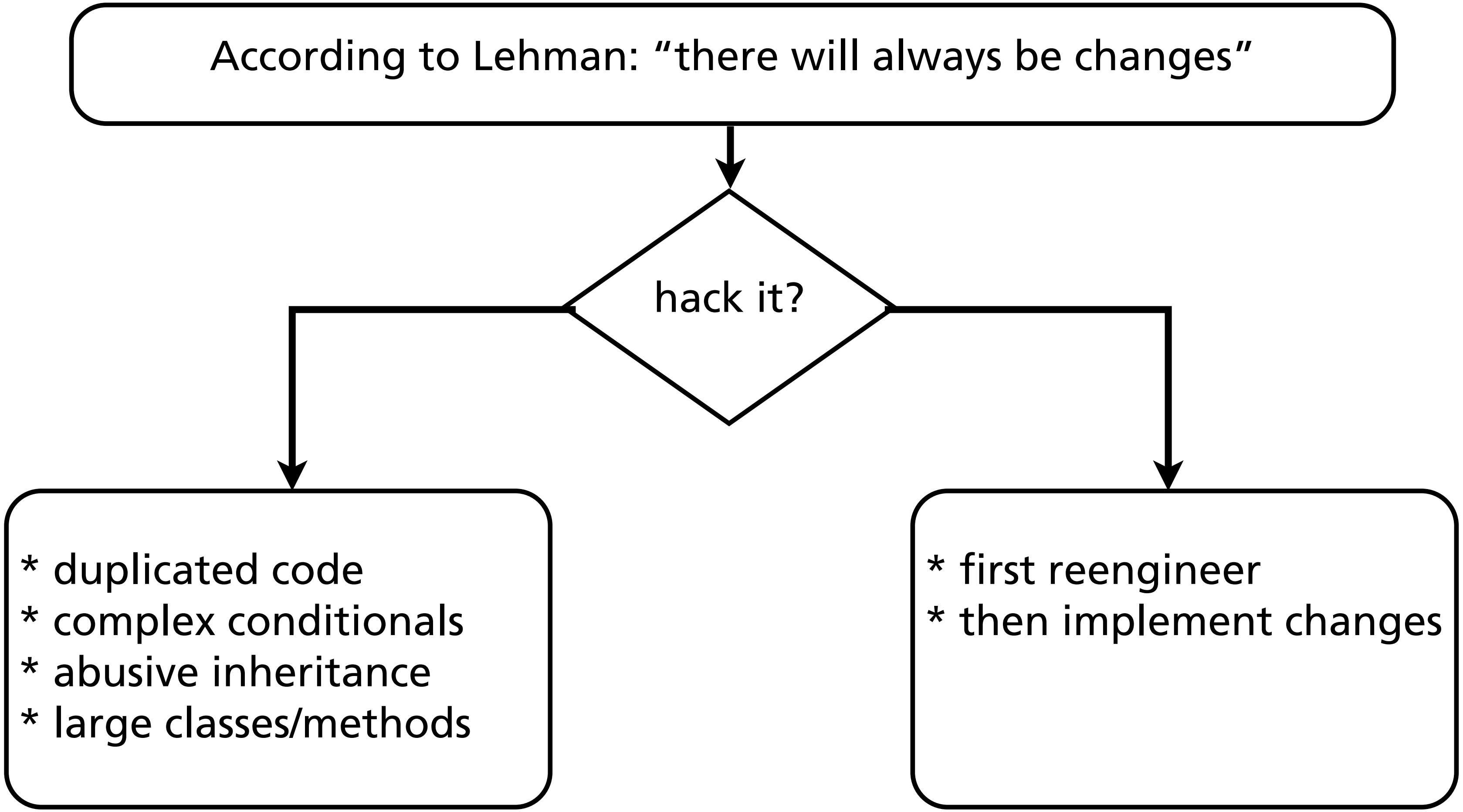
Implications of the results

Software maintenance costs continuously increase

Between 50% and 75% of global software development costs are spent on maintenance!

Up to 60% of a maintenance effort is spent on understanding the existing software

What is your decision?



According to Lehman: “there will always be changes”

hack it?

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

* first reengineer
* then implement changes

Take a loan on your software
pay back via reengineering

Investment for the future
paid back during maintenance

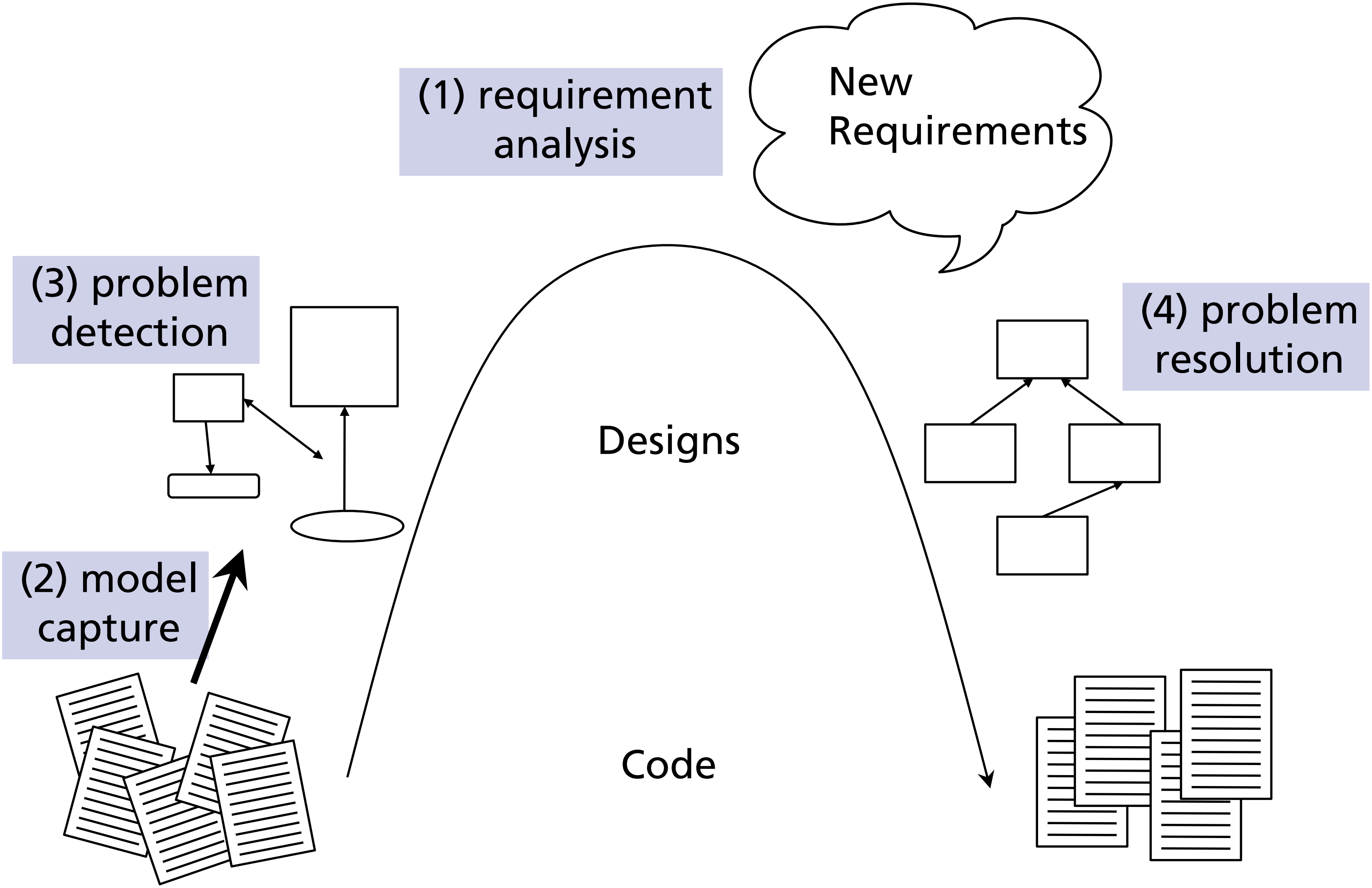
Let’s reengineer

Definition:

“Reengineering is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.”

[Demeyer, Ducasse, Nierstrasz]

Reengineering Life-Cycle



(1) requirement analysis

(2) model capture

(3) problem detection

(4) problem resolution

Goals of reengineering

- Testability
- Understandability
- Modifiability
- Extensibility
- Maintainability
- …

Goals of reengineering (conrete)

Unbundling

Split a monolithic system into parts that can be separately marketed

Performance

"First do it, then do it right, then do it fast"

Design extraction

To improve maintainability, portability, etc.

Exploitation of New Technology

I.e., new language features, standards, libraries, etc.

In this course, you will learn and apply

Best practices to analyze and understand software systems (i.e., reverse engineering)

Heuristics and tools to detect shortcomings in the design and implementation of software systems

Testing and re-factoring techniques to systematically resolve these shortcomings

Course Organization

General information

LV Info

- Block course
  - 1st block: 20.09. -- 24.09.2010 each from 12:15 -- 13:45
  - 2nd block: 2.11. -- 5.11.2010 each from 12:15 -- 13:45
  - 3rd block: 16.12. -- 17.12.2010
- Language: English
- AP (ECTS): 4
- Subscription until: 15. October 2010

Attend the lectures and prepare for each lecture!

Latest news always at: http://seal.ifi.uzh.ch/reeng/

Overview of the course

Block 1

Today	Overview of the course and the lab, Setting direction & Initial understanding
21.09.2010	Reverse engineering -- Detailed model capture DA4Java demo
22.09.2010	Code smells and design principles
23.09.2010	Problem Detection Findbugs, PMD, Metrics tool
24.09.2010	Code Clone Detection CCFinderX Demo

Overview of the course (cont.)

Block 2

02.11.2010	Tests - The Basis for Re-engineering Feedback on Assignment I: Problem Detection
03.11.2010	Working Effectively with Legacy Code Refactoring
04.11.2010	Refactoring to Patterns
05.11.2010	Guest lecture

Overview of the course (cont.)

Lab presentations & exams

16.12.2010	Lab presentations & exams
17.12.2010	Lab presentations & exams

How will you be assessed?

Lab assignments

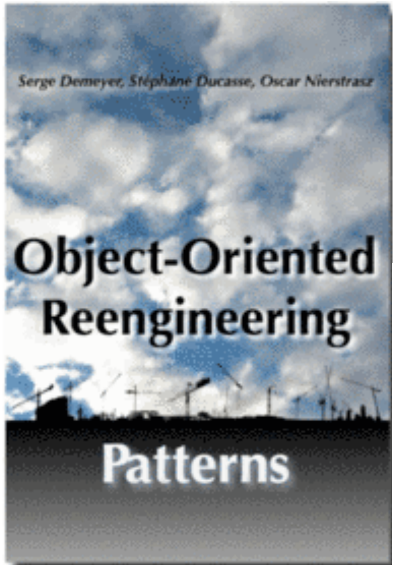
- Assignment I: Problem Detection (30%)
- Assignment II: Re-engineering (30%)

Final presentation of your results (20%)

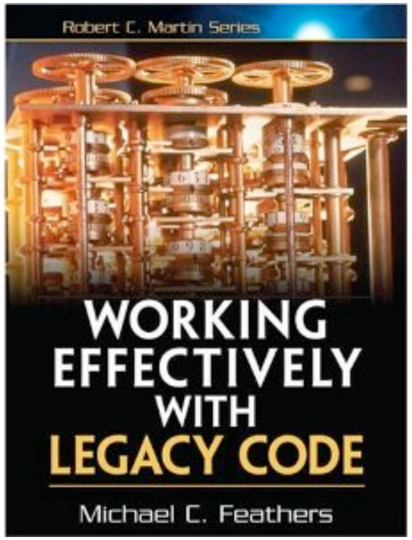
Oral examination after/during the final presentation (20%)

Your contribution to the lecture (+/- 5%)

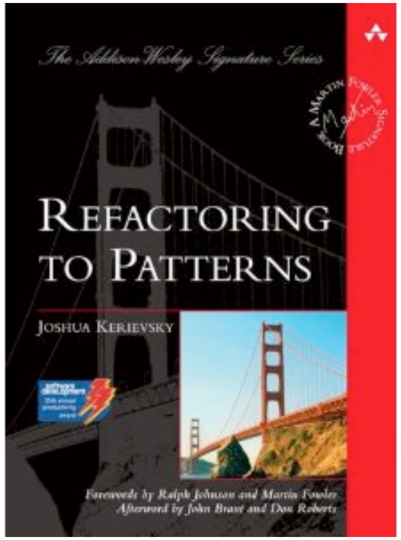
Reading material


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

Object-Oriented Reengineering Patterns
Serge Demeyer, Stephane Ducasse, and Oscar Nierstrasz
free copy from: http://scg.unibe.ch/download/oorp/

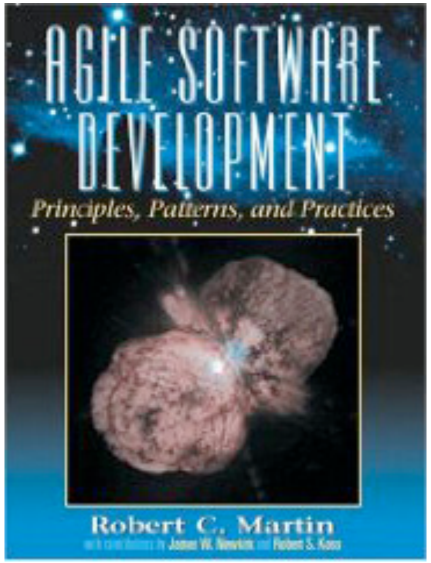


Working Effectively with Legacy Code
Michael Feathers, Prentice Hall, 1 edition, 2004

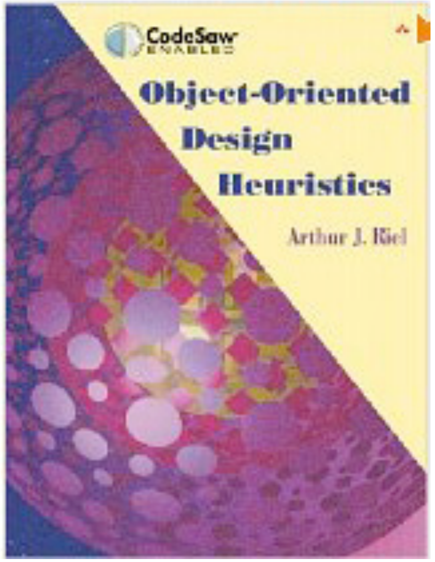

The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Refactoring to Patterns
Joshua Kerievsky
Forewords by Ralph Johnson and Martin Fowler
Afterword by John Brant and Don Roberts

Refactoring to Patterns
Joshua Kerievsky, Addison-Wesley Professional, 2004
I will provide copies of selected chapters

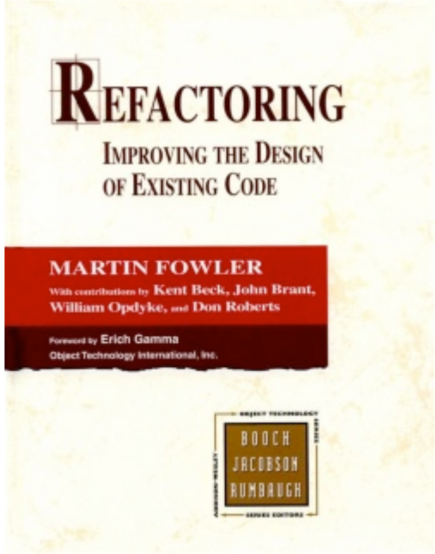
Additional reading


AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin
with contributions by James W. Newkirk and Robert S. Koss

Agile Software Development: Principles Patterns, and Practices
Robert C. Martin, Prentice Hall



Object-Oriented Design Heuristics
Arthur J. Riel, Prentice Hall, 1 edition, 1996

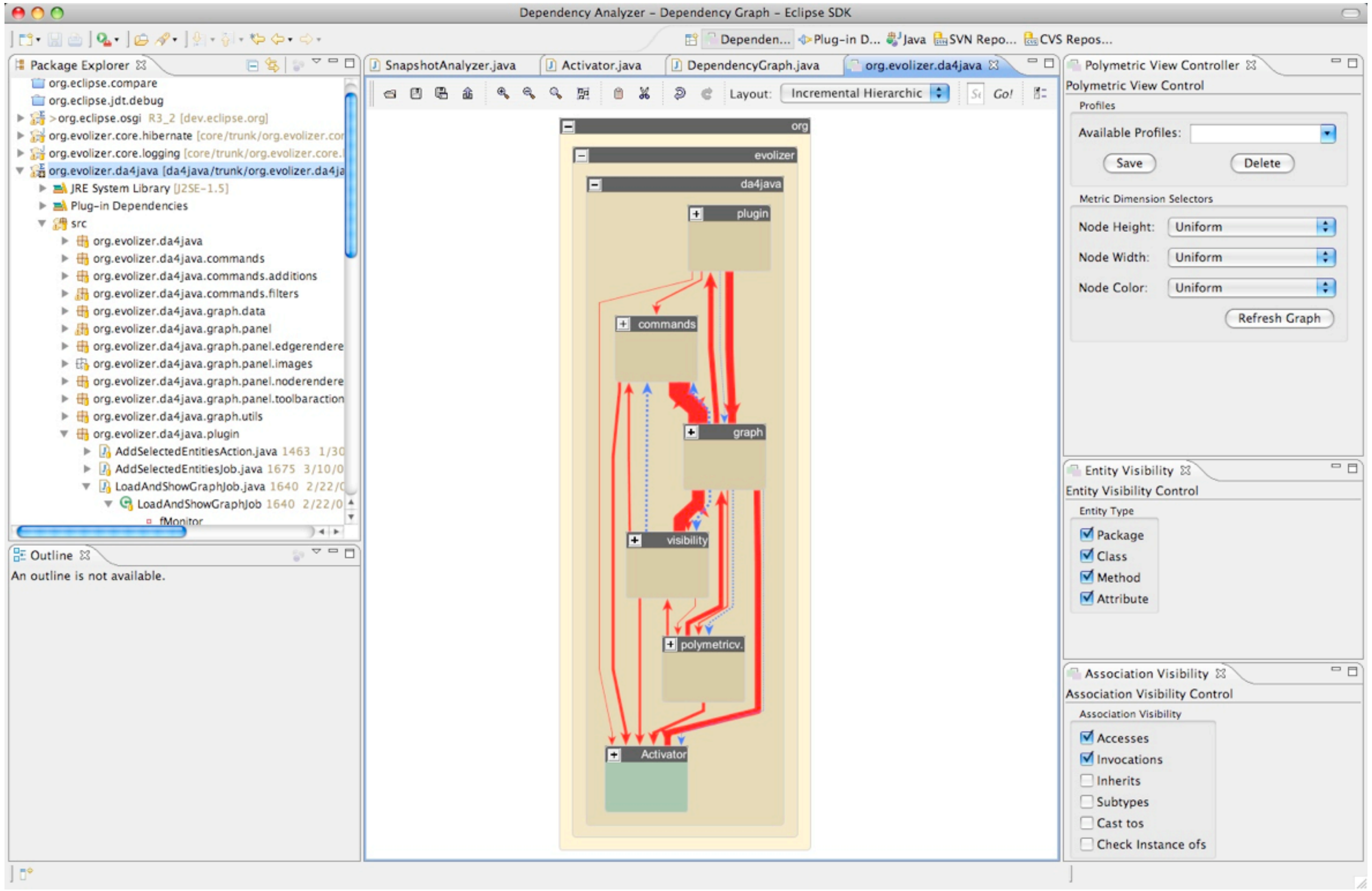

REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH
JACOBSON
RUMBAUGH

Refactoring: Improving the Design of Existing Code
Martin Fowler, Addison-Wesley Professional, 1999

The Reengineering Lab

The system: DA4Java

Interactive visualization of Java source code with leveled directed graphs



Dependency Analyzer – Dependency Graph – Eclipse SDK

Package Explorer

- org.eclipse.compare
- org.eclipse.jdt.debug
- >org.eclipse.osgi R3_2 [dev.eclipse.org]
- org.evolizer.core.hibernate [core/trunk/org.evolizer.cor
- org.evolizer.core.logging [core/trunk/org.evolizer.core.
- org.evolizer.da4java [da4java/trunk/org.evolizer.da4ja
  - JRE System Library [J2SE-1.5]
  - Plug-in Dependencies
  - src
    - org.evolizer.da4java
    - org.evolizer.da4java.commands
    - org.evolizer.da4java.commands.additions
    - org.evolizer.da4java.commands.filters
    - org.evolizer.da4java.graph.data
    - org.evolizer.da4java.graph.panel
    - org.evolizer.da4java.graph.panel.edgerendere
    - org.evolizer.da4java.graph.panel.images
    - org.evolizer.da4java.graph.panel.noderendere
    - org.evolizer.da4java.graph.panel.toolbaraction
    - org.evolizer.da4java.graph.utils
    - org.evolizer.da4java.plugin
      - AddSelectedEntitiesAction.java 1463 1/30
      - AddSelectedEntitiesJob.java 1675 3/10/0
      - LoadAndShowGraphJob.java 1640 2/22/0
        - LoadAndShowGraphJob 1640 2/22/0
          - fMonitor

Outline

An outline is not available.

SnapshotAnalyzer.java | Activator.java | DependencyGraph.java | org.evolizer.da4java

Layout: Incremental Hierarchic

Polymetric View Controller

Polymetric View Control

Profiles

Available Profiles:

Metric Dimension Selectors

Node Height: Uniform

Node Width: Uniform

Node Color: Uniform

Entity Visibility

Entity Visibility Control

Entity Type

- ☑ Package
- ☑ Class
- ☑ Method
- ☑ Attribute

Association Visibility

Association Visibility Control

Association Visibility

- ☑ Accesses
- ☑ Invocations
- □ Inherits
- □ Subtypes
- □ Cast tos
- □ Check Instance ofs

Lab outline

Reengineering of a DA4Java

- Part I: Reverse Engineering & Problem Detection
  - Initial understanding, detailed model capture
  - Code smells, violations of class and package design principles
- Part II: Testing & Refactoring
  - Develop a test harness
  - Refactor to improve the design and implementation

Implementation hints

Eclipse Plugin

- Running on Eclipse >= 3.4
- Download DA4Java source code from TBA
- Add sources to an Eclipse workspace and install MySQL server

Used libraries

- Evolizer FAMIX Importer for parsing Java source code
- MySQL for storing extracted FAMIX models
- Hibernate for data access
  - see www.hibernate.org
- yFiles for graph representation
  - see www.yworks.com

Phase 1: Reverse Engineering

First Contact

- Install the system and find out what its features are
- Is a reengineering feasible or should we re-implement it from scratch?

Reverse Engineering

- What are the building blocks of DA4Java?
- What is the design of DA4Java (package level, class level)

Problem Detection

- Where do you expect implementation design shortcomings?

See also http://seal.ifi.uzh.ch/reeng_uebung/

TODO for you

Exercise 1:

- Perform First Contact and Reverse Engineering
- Detect problems in the design of DA4Java
  - Problems on the code level (smells): Duplicated Code, Solution Sprawl, Long Method, Conditional Complexity, Large Class,
  - Violations of class design principles: Single Responsibility, Open/Closed, Dependency Inversion
  - Violations of package design principles: Reuse-Release Equivalence, Common-Reuse, Common-Closure, Acyclic-Dependencies, Stable-Dependencies, Stable-Abstractions
  - -> Simple problems in the source code do not count (e.g., naming)!
- Write up a report
  - Template can be found on the web-site

Deadline for submission of report: 25.10.2010, 18:00 sharp!