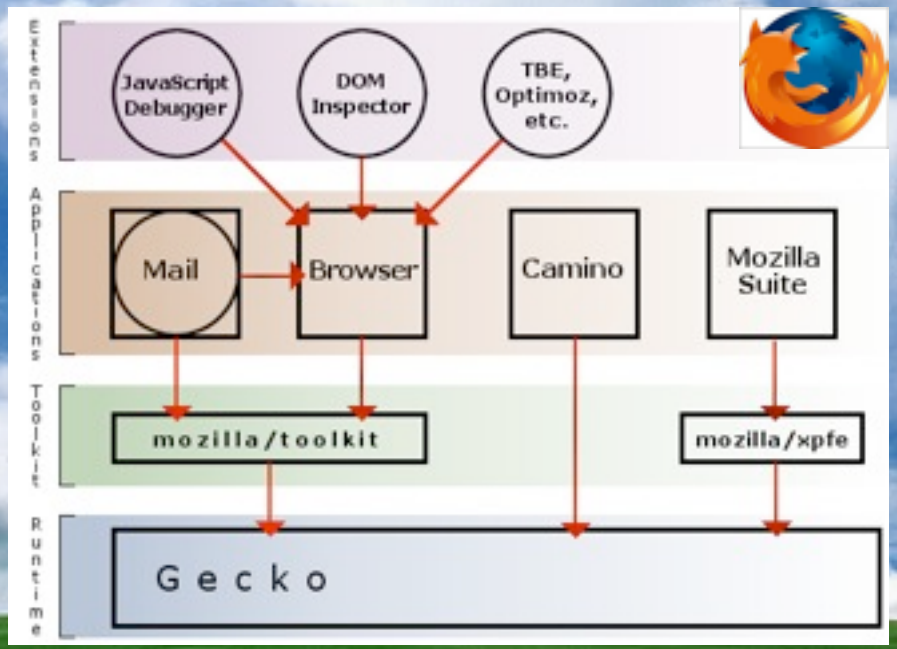
Software Reengineering
P1: Intro & Organization

Martin Pinzger
Delft University of Technology

TU Delft

SERG

Greenfield software development



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?

Lehman's Laws of software evolution

Continuing change

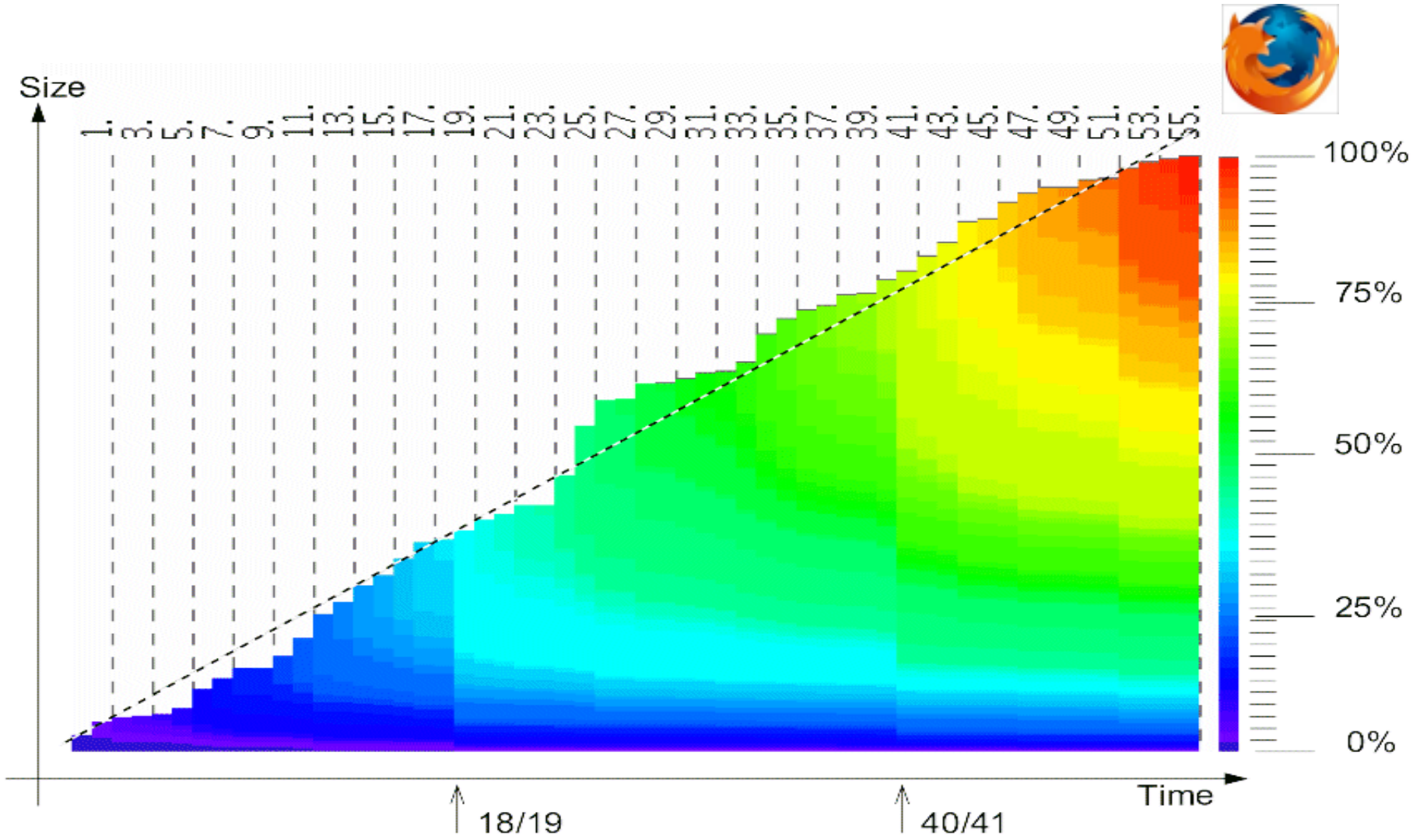
A program that is used in a real-world environment **must change**, or become progressively less useful in that environment.

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

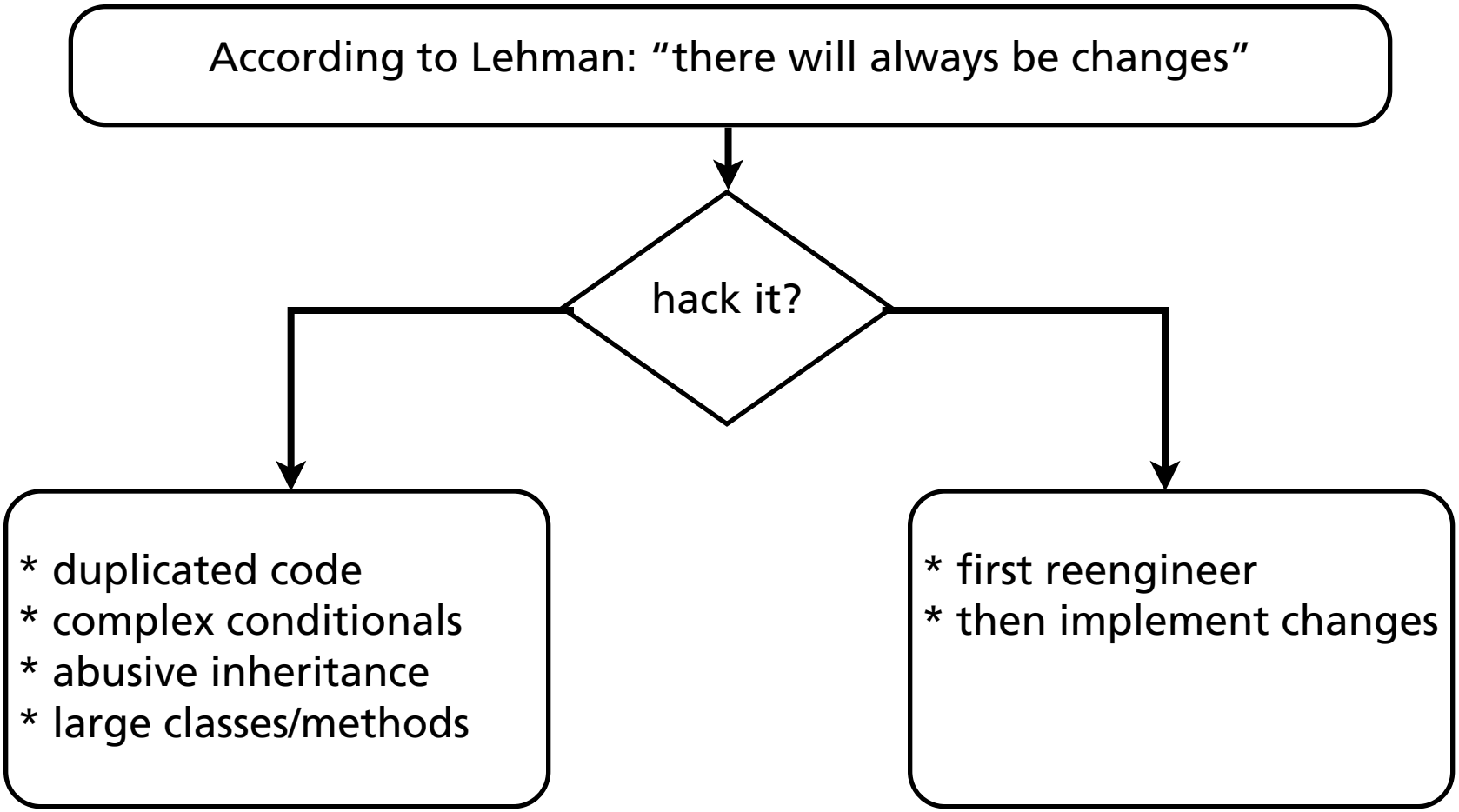
Implications of the results

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?



Take a loan on your software
pay back via reengineering

Investment for the future
paid back during maintenance

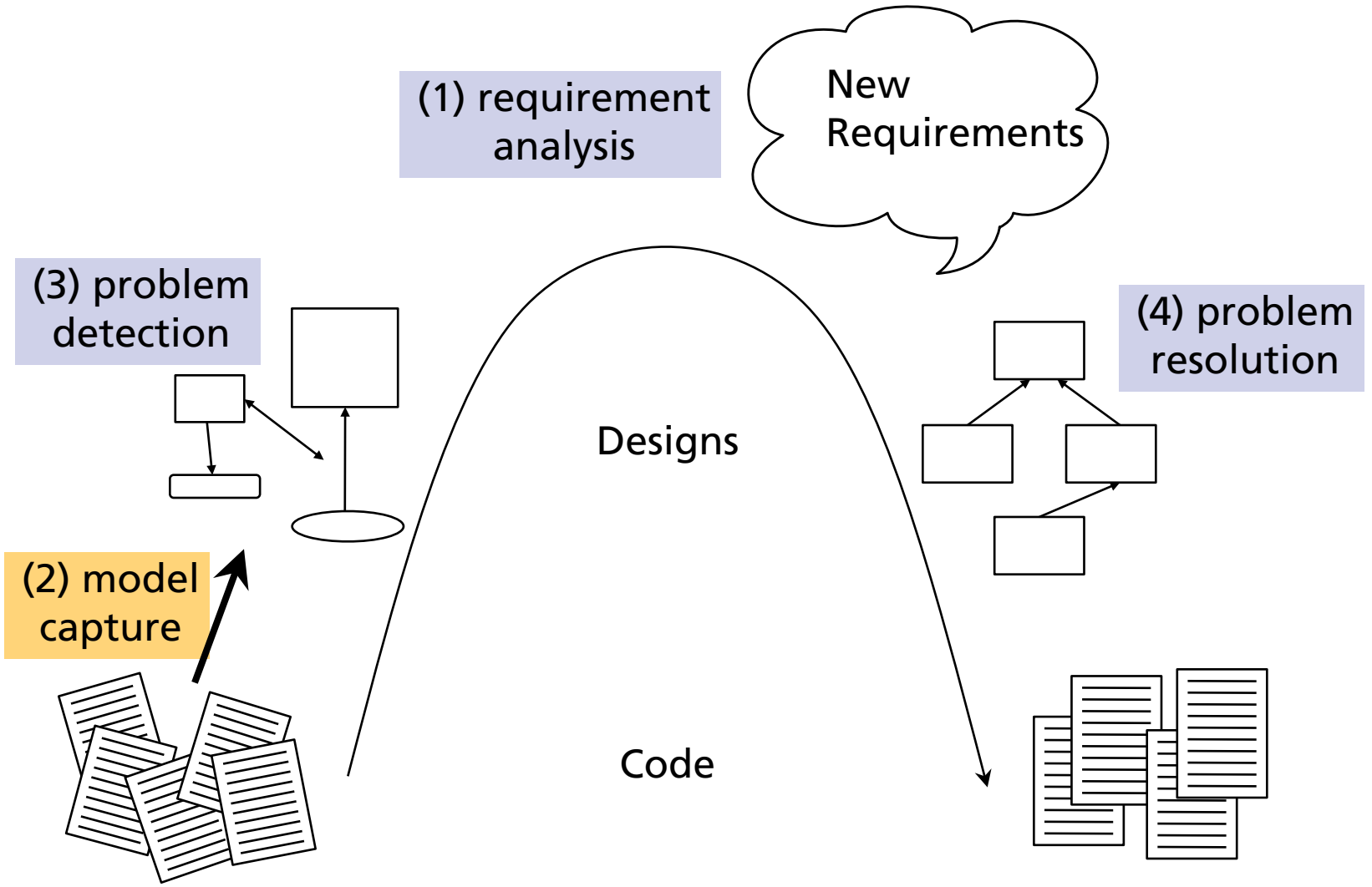
Let's reengineer

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



Goals of reengineering

Goals of reengineering (concrete)

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, IFI 1.D.07
  - 1st block: 21.09. -- 23.09.2011 each from 8:15 -- 10:00 and 14:00 -- 15:45
  - 2nd block: 2.11. -- 4.11.2011 each from 8:15 -- 10:00 and 14:00 -- 15:45
  - Presentations and final exam: 16.12.2011
- Language: English
- AP (ECTS): 4
- Subscription until: 14. October 2011?

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
22.09.2011	Reverse engineering -- Detailed model capture DA4Java demo
22.09.2011	Code smells and evolution
23.09.2011	OO Design Principles Findbugs, PMD, Metrics tool
23.09.2011	Code Clone Detection, CCFinderX Demo Overview Dynamic Analysis

Overview of the course (cont.)

Block 2

02.11.2011	Feedback on Assignment I: Problem Detection
03.11.2011	Testing and Refactoring
03.11.2011	Guest lecture by Canoo (TBO)
04.11.2011	Working Effectively with Legacy Code
04.11.2011	Refactoring to Patterns

Overview of the course (cont.)

Lab presentations & exams

16.12.2011	Lab presentations & exams

How will you be assessed?

Lab assignments

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

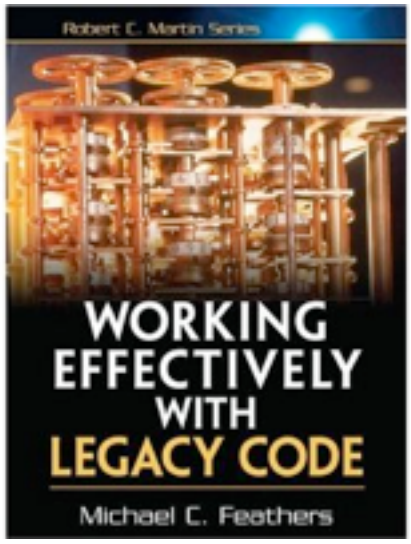
Final presentation of your results (20%)

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

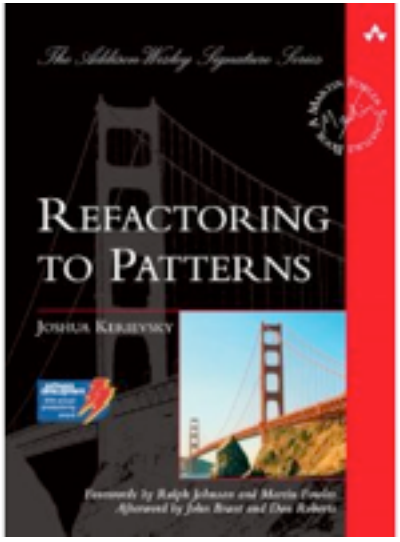
Your contribution to the lecture (+/- 5%)

Reading material

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

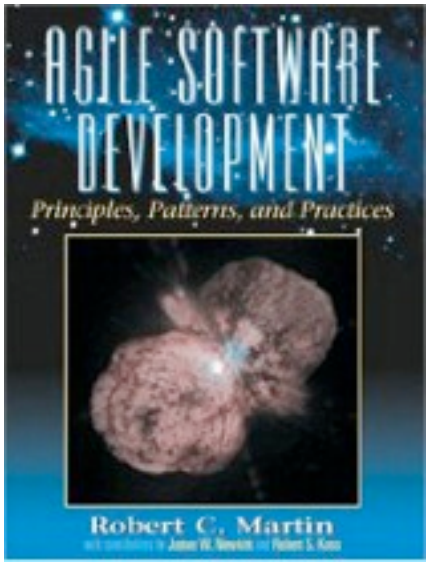


Working Effectively with Legacy Code
Michael Feathers, Prentice Hall, 1 edition, 2004

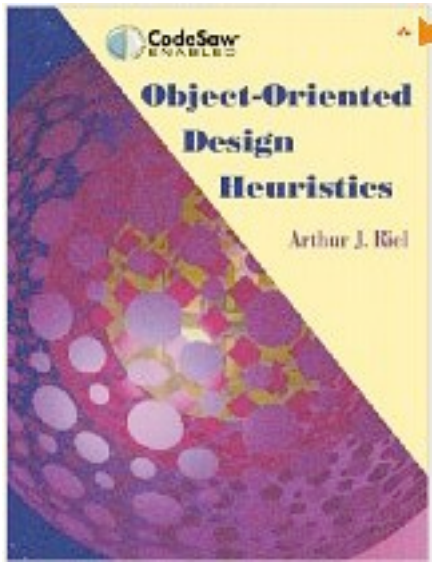


Refactoring to Patterns
Joshua Kerievsky, Addison-Wesley Professional, 2004
I will provide copies of selected chapters

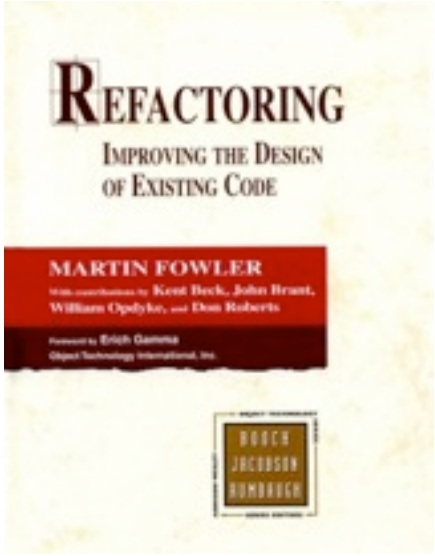
Additional reading



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



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



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

The Reengineering Lab

The system: jmonkeyengine 3.0

A modern 3D game engine in Java

Documentation

- http://jmonkeyengine.org/

Source code (Eclipse project)

- ~110.000 lines of code
- Checkout from svn repository
- http://jmonkeyengine.googlecode.com/svn/trunk/engine/

Lab outline

Reengineering of a jmonkeyengine 3.0

- 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

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 jmonkeyengine?
- What is the design of jmonkeyengine (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 jmonkeyengine 3.0
  - 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: 25.10.2011, 18:00 sharp!