DAT159
Refactoring (Introduction)

Volker Stolz¹, with contributions by:
Larissa Braz², Anna M. Eilertsen³,
Fernando Macías¹, Rohit Gheyi²

Western Norway University of Applied Sciences,
Universidade Federal de Campina Grande,
University of Bergen, Norway

Supported by the bilateral SIU/CAPES project “Modern Refactoring” 2017/18
Overview: Refactoring

- What are refactorings?
- Common refactorings for different languages.
- Why refactor? What are source code metrics?
- What can go wrong?
- How to implement refactorings?
Overview

- 6+1 lectures
- 3 labs
- 1 oblig

Please bring your laptop!
(at least 1/group)

IDEs: Eclipse, IntelliJ

Languages: mostly Java, some C
Overview

• Guest lectures from Brazil!
  (SIU/CAPES project “Modern Refactoring”)
  (see changed schedule)

• Possible Bachelor projects…

• …and Master theses.
DAT159
Refactoring (Introduction)

Volker Stolz¹, with contributions by:
Larissa Braz², Anna M. Eilertsen³,
Fernando Macías¹, Rohit Gheyi²

Western Norway University of Applied Sciences,
Universidade Federal de Campina Grande,
University of Bergen, Norway

Supported by the bilateral SIU/CAPES project “Modern Refactoring” 2017/18
Overview

- What are refactorings? What are they good for?
- Examples in common IDEs
- Examples in common languages (Java, C/C++, …)
- Impact on software quality metrics
- Implementation of refactorings
- Formal treatment of refactorings
It seems kinda important...

(Everybody’s doing it; you should as well!)
protected void Product.typeCheck(SemanticErrorList e) {
    HashSet<String> featureNames = new HashSet<String>();
    for (Feature f : getModel().getProductLine().getFeatures()) {
        featureNames.add(f.getName());
    }
    Model m = getModel();
    if (m.hasProductLine()) {
        for (Feature f : m.getProductLine().getFeatures()) {
            featureNames.add(f.getName());
        }
    }
    HashSet<String> productNames = new HashSet<String>();
    for (Product prod : getModel().getProducts()) {
        for (Product prod : m.getProducts()) {
            productNames.add(prod.getName());
        }
    }
    HashSet<String> deltaNames = new HashSet<String>();
    for (DeltaDecl delta : getModel().getDeltaDecs()) {
        for (DeltaDecl delta : m.getDeltaDecl()) {
            deltaNames.add(delta.getName());
        }
    }
}
Refactoring: how to do it? Why does everyone hate it?

I refactored once
It was horrible
THAT'D BE GREAT

I refactored once
Code refactoring

YEAH... IF YOU COULD STOP REFACTORIZING WHILE WE TEST THINGS.
What is Refactoring? (1)

“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour” [Fowler]

From mathematical term “factor”: finding multiple occurrences of similar code and *factoring* it into a single reusable function

Motivation:

• keep the code clean
• avoid technical debt
Motivation

Code Quality Measurement:

WTFs/Minute

Good Code

Bad Code

http://commadot.com
What is Refactoring? (2)

- Two different schools:
  - *anything goes* (agile)
  - behaviour preserving

- Corner cases:
  - changing complexity class, e.g. replacing bubble sort with quicksort still a refactoring?
Refactoring Process

- Developer inspects code.
- She selects part of it...
- ...and chooses refactoring action from menu.
- Refactorings usually modify the Abstract Syntax Tree (AST) in memory...
- ... and then synchronize the source code file.
Abstract Syntax Tree (AST)

- In-memory representation of parsed source code
- Semantic information available (Where was this variable declared? What are the superclasses?)

```java
public class C {
    public X x = new X();
    public void f(X x) {
        x.m(this);
    }
}
```
Refactoring: Origins

- Opdyke’s PhD thesis [1992]
- Smalltalk Refactoring Browser [Roberts, Brant, Johnson ’97]
- “Refactoring: improving the design of existing code” [Fowler ’99]
- 30% of changes are refactorings [Soares et al., 2011]
- Extract Method most popular — but performed manually [Murphy et al., 2006]
Literature

Refactoring: Improving the Design of Existing Code
Martin Fowler with Kent Beck, John Brant, William Opdyke, Don Roberts
Addison Wesley, 1999
Adoption of Refactorings

- Agile: fully embraced refactoring
- Developers usually sceptical of automated changes
- Study: developers more confident when they can predict changes
- Problem in OO languages: refactoring touches on multiple contexts
Adoption: Software Engineering Studies

- Kim et al. (FSE, 2012): survey on more than 300 engineers who had used refactoring during Microsoft Windows development.

- Tempero et al. (C.ACM, 2017):
  - Survey on 3785 developers in 2009.
  - They understand benefits of refactoring, but they see costs and risks as well.
Related Topics: Patterns

• “Design Patterns: Elements of Reusable Object-Oriented Software” [Gamma, Helm, Johnson, Vlissides, 1994]

• “Refactoring to patterns” [Kerievsky, 2005]

• “Anti-patterns” and “code smells”: indicators of design deficiencies

• Ignoring exceptions (AP), magic strings (AP), repeated code (CS), long functions (CS)

• Detection partially automated

• Refactoring to more structured solutions
Software Quality Metrics

• How “good” is your code?

• Often subjective, but some guidelines:
  • high cohesion/low coupling between classes
  • long method body
  • class with too many methods

• Refactorings affect those metrics:
  • Extract Method reduces length of method and cyclometric complexity…
  • …but obviously increases number of methods.
Software Quality Metrics (2)

- Tools like Findbugs, Checkstyle, JDeodorant, SonarQube identify problems
- Developers still need to act on that info
- Problem with automation:
  - large search-space
  - often many (overlapping) possibilities
  - Extract Method ↔ Inline Method “competing” against each other
- Our attempt: Kristensen/Stolz, “Search-based composed refactorings”, NIK 2014
Reducing Coupling

• Coupling Between Object Classes (CBO) of class C improves from 4 to 3…
• …but sometimes introduces additional coupling into the receiving class!
Related Topics: Source Code Rejuvenation

“Source Code Rejuvenation”  
[Pirkelbauer, Dechev, Stroustrup ’10]

• automated migration of legacy code
• leverages enhanced program language/library facilities
• “reverse (some forms of) (software) entropy”
• “preserves or improves a program’s behavior”
# Source Code Rejuvenation

<table>
<thead>
<tr>
<th>Source Code Rejuvenation</th>
<th>Refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation</td>
<td>Source-to-source</td>
</tr>
<tr>
<td>Behavior preserving</td>
<td>Behavior improving</td>
</tr>
<tr>
<td>Directed</td>
<td>yes</td>
</tr>
<tr>
<td>Raises the level of abstraction</td>
<td></td>
</tr>
<tr>
<td>Drivers</td>
<td>Language / library evolution</td>
</tr>
<tr>
<td>Design changes</td>
<td></td>
</tr>
<tr>
<td>Indicators</td>
<td>Workaround techniques / idioms</td>
</tr>
<tr>
<td>Anti-patterns</td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td>One-time source code migration</td>
</tr>
</tbody>
</table>

From: Pirkelbauer, Dechev, Stroustrup, SOFSEM 2010
Source Code Rejuvenation

```cpp
vector<int> vec;
// three consecutive push backs
vec.push_back(1);
vec.push_back(2);
vec.push_back(3);
// copying from an array
int a[] = {1, 2, 3};
vector<int> vec(a, a + sizeof(a) / sizeof(int));
```

Inefficient!

```cpp
// rejuvenated source code in C++0x
vector<int> vec = {1, 2, 3};
```

Sizeof() what again?!

Now isn’t that pretty:
Refactoring in IDEs

• All major IDEs support some form of refactoring
• Here: C, C++, Java
• Special case: command line tools for scripting (Go?)
• Support for scripting languages like Python, JavaScript, ...
• Refactoring of UML models (semantical overlap with OO-refactoring)
Tool Support for Java

• Common IDEs:
  • Eclipse JDT
  • IntelliJ (Android)
  • NetBeans

• Other object-oriented languages similar:
  • Visual Studio
Refactoring: Common Java Examples

Encapsulate Field: avoid direct field access

1) introduce setter & getter methods;
2) replace all field accesses with calls to new methods;
3) make field private.
Encapsulate Field

Right-click on a field and find the “Refactor” menu.
Encapsulate Field

IDEs will often have a helpful dialog, because further input is required.
Encapsulate Field

Enjoy your result!
Encapsulate Field

IDEs will even try to be helpful!
Refactoring: Common Java Examples

**Encapsulate Field**: avoid direct field access
1) introduce setter & getter methods;
2) replace all field accesses with calls to new methods;
3) make field private.

Let’s assume you have to *program* this refactoring. Can you see what happens if you swap steps 2 & 3? We will come back later to that.
Refactoring: Extract Local Variable

Compute complex (expensive) expression only once.
Extract Local Variable: Formally

input: $e$ – an expression of non-void type $E$
: $S$ – a selection, as a list of consecutive statements
: $context$ – the outermost, non-type scope containing $S$

output: $context$ with $e$ extracted to a local variable in $S$

1. $v \leftarrow$ fresh variable name;
2. for $s \in S$ do
3.   in $s$ replace all occurrences of $e$ with $v$;
4. end
5. add a new variable declaration $E \ v = e$ context just before $S$;