Refactoring and Reflection

Marcus Denker
denker@iam.unibe.ch

Universität Bern
Roadmap

> First lecture: Refactoring
  — Squeak as an example

> Second Lecture: Reflection
  — About current research
Part I: Refactoring

Marcus Denker
denker@iam.unibe.ch

Universität Bern
Overview

- Refactoring: Basics
- Refactoring in Squeak: Browser + Tools
- Refactoring Engine: Implementation
- Discussion: Reflection?
Roadmap

- Refactoring: Basics
- Refactoring in Squeak: Browser + Tools
- Refactoring Engine: Implementation
- Discussion: Reflection?
What is Refactoring?

> The process of *changing a software system* in such a way that it *does not alter the external behaviour* of the code, yet *improves its internal structure*.

## Typical Refactorings

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>add (sub)class to hierarchy</td>
<td>add method to class</td>
<td>add variable to class</td>
</tr>
<tr>
<td>rename class</td>
<td>rename method</td>
<td>rename variable</td>
</tr>
<tr>
<td>remove class</td>
<td>remove method</td>
<td>remove variable</td>
</tr>
<tr>
<td>push method down</td>
<td>push variable down</td>
<td></td>
</tr>
<tr>
<td>push method up</td>
<td>pull variable up</td>
<td></td>
</tr>
<tr>
<td>add parameter to method</td>
<td>create accessors</td>
<td></td>
</tr>
<tr>
<td>move method to component</td>
<td>abstract variable</td>
<td></td>
</tr>
<tr>
<td>extract code in new method</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why Refactor?

“Grow, don’t build software”
— Fred Brooks

The reality:
— Extremely difficult to get the design “right” the first time
— Hard to fully understand the problem domain
— Hard to understand user requirements, even if the user does!
— Hard to know how the system will evolve in five years
— Original design is often inadequate
— System becomes brittle over time, and more difficult to change

Refactoring helps you to
— Manipulate code in a safe environment (behavior preserving)
— Recreate a situation where evolution is possible
— Understand existing code
Rename Method — manual steps

> Do it yourself approach:
  - Check that no method with the new name already exists in any subclass or superclass.
  - Browse all the implementers (method definitions)
  - Browse all the senders (method invocations)
  - Edit and rename all implementers
  - Edit and rename all senders
  - Remove all implementers
  - Test

> Automated refactoring is better!
Rename Method

> Rename Method (method, new name)

> Preconditions
  – No method with the new name already exists in any subclass or superclass.
  – No methods with same signature as method outside the inheritance hierarchy of method

> PostConditions
  – method has new name
  – relevant methods in the inheritance hierarchy have new name
  – invocations of changed method are updated to new name

> Other Considerations
  – Typed/Dynamically Typed Languages => Scope of the renaming
Refactoring and Metaprogramming

> Automated Refactoring is metaprograming
  — We use a program to edit programs

> Does not need to use Reflection
  — Pure source-to-source transformation (e.g. Java)

> Uses reflective facilities in Smalltalk
  — But… let’s discuss that at the end
Roadmap

> Refactoring: Basics
> **Refactoring in Squeak: Browser + Tools**
> Refactoring Engine: Implementation
> Discussion: Reflection?
Refactoring in Squeak

> No support in standard IDE
  — Example: Try to rename a method

> Refactoring Browser
  — First Refactoring Browser (for any language)
  — Now over 10 years old

> Installation
  — Get Squeak 3.9 (older version for 3.8, too)
  — Install Package AST
  — Install Package Refactoring Engine
Refactoring Browser

- Browser with menus for e.g.
  - rename
  - Push up/down
  - Inlining
  - Add parameter
  - Extraction
SmallLint

> Checks for common mistakes

```
possible bugs
bugs
squeak bugs
unnecessary code
intention revealing
miscellaneous

Method has no timeStamp
Method source contains linefeeds
Sends super new initialize
Debugging code left in methods
Menus missing translations
Sends a deprecated message to a known global
```

Select Classes Run
SmallLint Checks

> Possible Bugs
  — Variable read before written
  — Defines #= but not #hash
  — Modifies Collection while iterating over it

> Bugs
  — Uses True/False instead of true/false
  — Variable used but not defined

> Squeak Bugs

> Unnecessary Code

> Intention Revealing
SmallLint

> Very useful!
> Especially valuable for beginners

> Has been integrated with SUnit
  — Call SmallLint automatically as a test

> Tag methods where SmallLint is wrong
  — Uses Squeak 3.9 Method Pragmas
RewriteTool

> Pattern driven automatic editor

This tool searches for code matching the pattern in the top pane that satisfies the condition in the second pane, and lets you replace the found code patterns with the replacement pattern in the bottom pane.

You will be shown the potential changes to the code, and you can choose which of them to actually use.

You may use jokers (explained below) defined in the top pane in the replacement text in the.
Access to full power of Refactoring Engine

Custom refactorings:
- generic rewrites that the RB does not currently provide
- bulk transformations: your project needs to change a project-specific pattern to a new form
- changing layers: e.g. build a new DB layer, find and change 17,000 references to old layer
- migrations: e.g. between Smalltalk dialects

Powerful but not trivial to use

Examples: Later
Roadmap

> Refactoring: Basics
> Refactoring in Squeak: Browser + Tools
> Refactoring Engine: Implementation
> Discussion: Reflection?

Implementation Overview

> Goal: Transformation on the Source
> Idea: Transform into a higher level tree representation
The RB Abstract Syntax Tree

AST: Abstract Syntax Tree
- Encodes the Syntax as a Tree
- Features:
  - Visitors
  - Backward pointers in ParseNodes
  - Encodes formatting
  - Transformation (replace/add/delete)
  - Pattern-directed TreeRewriter
  - PrettyPrinter
A Simple AST

RBParser parseExpression: '3+4'

explore it
A Simple Visitor

RBProgramNodeVisitor new visitNode: tree

Does nothing except walk through the tree
More Complete Visitor

RBProgramNodeVisitor subclass: #TestVisitor
  instanceVariableNames: 'literals'
  classVariableNames: '
  poolDictionaries: '
  category: 'Compiler-AST-Visitors'

TestVisitor>>acceptLiteralNode: aLiteralNode
  literals add: aLiteralNode value.

TestVisitor>>initialize
  literals := Set new.

TestVisitor>>literals
  ^literals

tree := RBParser parseExpression: '3 + 4'.
(TestVisitor new visitNode: tree) literals
  a Set(3 4)
Tree Matcher

- Implementing all Refactorings with visitors
  - Too much work
  - Too low level

- Needed: High level specification of transformations

- Rewrite Engine: Core of Refactoring Engine

- No only useful for Refactoring!
Tree Matcher

> Describe transformation by using patterns

> Syntax: Smalltalk + Meta Variables

> Example:

```
| `@Temps  |
`\`@.Statements.
`\`@Boolean ifTrue: [^false].
^true
```
## Meta Variables

All Meta Variables begin with ``

<table>
<thead>
<tr>
<th>Character</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>`</td>
<td>recurse into</td>
<td><code>\</code>@object foo</td>
</tr>
<tr>
<td>@</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>statement</td>
<td>`.Statement</td>
</tr>
<tr>
<td>#</td>
<td>literal</td>
<td>`#literal</td>
</tr>
</tbody>
</table>
Example 1

> Search for: ```@object not ifTrue: @block```

• Replace with: ```@object ifFalse: @block```

• Explanation:
  – Eliminate an unnecessary not message
Example 2

> Search for:
```
| `~@Temps |
`~@.Statements.
```
```
`~Boolean ifTrue: [^false].
^true
```

• Replace with:
```
| `~@Temps |
`~@.Statements.
`~~@Boolean not
```

• Explanation:
  – Return the value of the boolean negated instead of using a conditional
Implementation: Model and Environment

> Model Code transformed but not installed
  — We need to be able to see refactored code without changing the system.
  — RBNameSpace

> Model Classes + Methods
  — Framework duplicates Smalltalk’s structural Reflection
  — RBClass, RBMethod

> Model Scope to which Refactorings apply
  — RBEnvironment
Back to Code: Pretty Printer

> Visitor: Walks the AST
> Prints out text for each node

> Problem: How to preserve formatting?
  — AST saves formatting (whitespace, parenthesis)
  — Pretty Printer can use saved formatting information
Contributions needed

> Improved UI

> Integrated Parser with Squeak NewCompiler
  — Scanner/Parser done with tool (SmaCC)
  — Easier to change / experiment

> Integrated RoelTyper
  — Heuristic type inference

> Better PrettyPrinter
  — Configurability
  — Better Layout
Roadmap

- Refactoring: Basics
- Refactoring in Squeak: Browser + Tools
- Refactoring Engine: Implementation
- Discussion: Reflection?
Reflection?

> We change the system using itself
  — So it’s Reflection, on some level

> But: Let’s look again at the definition
  — Model of itself
  — Causally connected

> We Build our own abstraction layer
  — AST + Environment

> This Model is not causally connected!
Why not this?

Tools
- Refactoring
- Code Editing
- Version Control
- Presentation
- Compilation

Model
- Structure
- Behavior
  - #isSelfSend
  - #isTemp
  - #isRead
  - ....
- Extensions
  - data
  - behavior

External
- Text
- Bytecode
- ....
We have seen...

- Refactoring: Basics
- Refactoring in Squeak: Browser + Tools
- Refactoring Engine: Implementation
- Discussion: Reflection?
Questions?

- Refactoring: Basics
- Refactoring in Squeak: Browser + Tools
- Refactoring Engine: Implementation
- Discussion: Reflection?
License

> http://creativecommons.org/licenses/by-sa/2.5/

Attribution-ShareAlike 2.5

You are free:

• to copy, distribute, display, and perform the work
• to make derivative works
• to make commercial use of the work

Under the following conditions:

BY: Attribution. You must attribute the work in the manner specified by the author or licensor.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

• For any reuse or distribution, you must make clear to others the license terms of this work.
• Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.