

Refactoring Notes

Anthony Lander

What is Refactoring

Refactoring is applying

*Behavior preserving
transformations*

Why Refactor?

Because the very act of programming makes software:

Grow in complexity

Difficult to extent

Difficult to maintain

Scary to modify

Hard to understand

Why Refactor ... ?

Control Maintenance Costs

- Maintenance costs 10x what v1.0 does

But you know all this...right?

The Real Question

**Why does software
go all squirrely?**

Root Cause

Changing requirements

- Components get modified to work in ways that were never originally intended
- The *interfaces* between components becomes obscures
- *Interdependencies* between components snowball
- Existing code breaks (bitrot)
- Cruft multiplies

What to do?

Listen to Kent Beck:

Embrace Change

**In the real world, it's the only thing
that works.**

Tough Questions

- **When is it OK to patch?**
- **How many patches are too many?**
- **When is it time to redesign?**
- **How do you plan it?**
- **How do you do it?**
- **Motivated Consolidation**

Refactoring

- **Primary goal**

Reduce dependencies

- **Secondary goals (and benefits)**
 - Reduced frustration
 - Schedule management
 - Ability to predict
 - Testing

Kinds of Refactoring

Low-level cleanup

Structural cleanup

Component consolidation

Low Level Cleanup

Reduce background noise
Improve readability

Major culprits:

- Bad variable names
- Bad method names
- Complicated logic
- One method = one purpose

How?

- **Renaming**
 - Goal is *intention-revealing names*
- **Rename variables**
 - **Mt_bal** becomes **monthlyBalance**
 - **Vtbl** becomes **virtualTable**
- **Rename methods**
 - Remove blow-by-blow style comments
 - Replace with meaningful method names

How...?

- **Reorder if-statements**

= aFoo

```
aFoo class = self class ifTrue: [  
  aFoo name = self name ifTrue: [  
    aFoo bar = self bar ifTrue: [  
      ^true]]].  
  ^false
```

- Difficult to follow
- Difficult to add another case

The Early Exit Trick

- **Improved**

- Easy to understand
- Easy to extend

= aFoo

```
aFoo class = self class ifFalse: [^false].
```

```
aFoo name = self name ifFalse: [^false].
```

```
aFoo bar = self bar ifFalse: [^false].
```

```
^true
```

How...?

- **Disassemble nested and's and or's**

**^(file isOpen and: [file atEnd not
or: [file peek = \$p and: [balance < 120.00]]])**

Becomes:

**(file closed or: [file atEnd]) ifTrue: [^false].
^(file peek = \$p and: [balance < 120.00])**

How...?

- **Move complicated loops**
 - Usually the result of a method doing more than one job
 - Place them in their own method.

Structural Cleanup

**Goal is to improve
the structure of classes**

Structural Cleanup ...

- **Rampant classification**
 - Not everything needs to be a class
- ***What-if?* Inheritance**
 - Stream
 - PeekableStream
 - PositionableStream
 - ExternalStream
 - BufferedExternalStream
 - ExternalReadStream
 - ExternalReadAppendStream
 - ExternalReadWriteStream
 - ExternalWriteStream

Structural Cleanup ...

- **Non polymorphic messages**
 - Pick a good name and rename the whole lot
- **Rape-and-paste binges**
 - Consolidate similar code into common methods

Component Consolidation

Reduce Interdependencies

Component Consolidation ...

Key observation:

- Original CRC breakdown no longer makes sense
- Cause? *Changing requirements*
- **Refactor**
 - Create a new CRC breakdown which does make sense
 - Plan a course of action to transform the old code into the new

Component Consolidation ...

- **Typically involves**
 - Redefining component responsibilities
 - Shuffling classes between components
 - Turning parameters into instance variables
 - Merging and breaking apart classes

Component Consolidation ...

- **How do you ensure that you don't break the system?**
 - Testing
 - Key: Write the tests before you start refactoring
 - If you don't have any tests, this is a good way to get some written
 - Keep the tests when you're done!