

Refactoring



<http://c2.com/cgi/wiki?ExtremeNormalForm>

Your classes are small and your methods are small; you've said everything `OnceAndOnlyOnce` and you've removed the last piece of unnecessary code.

Somewhere far away an Aikido master stands in a quiet room.
He is centered.. aware.. ready for anything.

Outside a sprinter shakes out her legs and settles into the block,
waiting for the crack of the gun while a cool wind arches
across the track.

Softly, a bell sounds.. all of your tests have passed.

Your code is in `ExtremeNormalForm`.. tuned to today and poised
to strike at tomorrow.

Refactoring

Changes made to a program that only change its organization, not its function.

Behavior preserving program transformations.

Why refactoring is important

Only defense against software decay.

Often needed to fix reusability bugs.

Lets you add patterns after you have written a program; lets you transform program into framework.

Lets you worry about generality tomorrow; just get it working today.

Necessary for beautiful software.

Piecemeal Growth

The way living things grow

The way software grows

More than just addition -- transformation

Duplicate code

Eliminate duplication by

- making new methods
- making new objects
- moving methods to common superclass

Long methods

Each method should do one thing.

One comment for each method.

Method should fit on the screen.

Method is all on same level of abstraction.

Method should be in right class.

Move code to its right class

```
teacher classes add: thisClass.
```

```
teacher classLoad: (teacher classLoad + 1)
```

```
teacher addClass: thisClass
```

Large classes

More than seven or eight variables

More than fifty methods

- You probably need to break up the class

Components (Strategy, Composite, Decorator)

Inheritance

Long parameter lists

If you see the same set of parameters repeated in several methods, bundle them into a new object. Create accessors to get the original parameters from the object. Change the methods to use the new object instead of the parameters.

Then figure out what other functionality needs to move to the object.

Case Statements

Use polymorphism, not case statement.

Make class hierarchy, one class for each case.

Make a method for each case statement.

Make each branch of case statement be a method in a class

How to refactor

Make changes as small as possible.

Test after each change.

Many small changes are easier than one big change.

Each change makes it easier to see that more changes are needed.

Refactoring browser

A better browser

Automates many refactorings - renaming,
moving code, splitting

Undo

Smallint tool helps you find places that need
to be refactored.

When to refactor

If a new feature seems hard to implement,
refactor.

If a new feature created some ugly code,
refactor.

When you can't stand to look at your code,
refactor.