Refactoring

Perdita Stevens

School of Informatics
University of Edinburgh

Refactoring

Refactoring is the process of re-organizing and re-writing code so that it becomes cleaner, or fits better into the current conception of the architecture. Refactoring *does not change functionality*.

Why refactor? Once seen as a kind of maintenance. . .

- You've inherited legacy code that's a mess.
- A new feature is required that necessitates a change in the architecture.

But can also be an integral part of the development process: agile methodologies (e.g. XP) advocate continual refactoring (XP maxim: "Refactor mercilessly").

What does refactoring do?

A refactoring is a *small* transformation which preserves correctness. There are many examples; for a catalogue from Martin Fowler's original book *Refactoring*, see `http://www.refactoring.com/catalog/`. Examples:

- Add Parameter
- Change Bidirectional Association to Unidirectional
- Introduce Explaining Variable
- Replace Conditional with Polymorphism

The process of refactoring is that of applying a sequence of refactorings that improve the design of the system, without adding functionality.

Eclipse has a built-in refactoring tool (on the Refactor menu). It performs operations of three broad classes . . .

Renaming and physical organization

A variety of simple (when done automatically) changes, e.g.

- Rename or move files – automatically updating `import`, `package` etc.
- Renaming variables – and associated methods.
- Moving classes between packages

Rearranging the class structure

Heavier changes, re-organizing the way classes relate. Less used, but seriously useful when they are used. E.g.

- When an anonymous class gets big, it should turn into a nested class.
- Moving methods or fields up and down the class hierarchy.
- Extracting an interface from a class.

Intra-class refactorings

The bread-and-butter of refactoring: rearranging code within a class to improve readability etc. E.g.

- Extracting code from method into new method.
- Encapsulating fields in accessor methods.
- Change the type of a method (think about it. . .)

Safe refactoring

How do you know refactoring hasn't changed/broken something?

Perhaps somebody has *proved* that a refactoring operation is safe.

More realistically:

test, refactor, test

This works better the more tests you have: ideally, unit tests for every class.

Reading

Required: The article 'Refactoring for everyone' at `http://www.ibm.com/developerworks/opensource/library/os-ecref/`. Aim to remember: what refactoring is, and a few examples, not the details of the refactorings discussed here.

Suggested: browse around Fowler's page at `http://www.refactoring.com/`.

Quote of the day

> *Refactoring provides enough energy to a system for it to relax into a new and more comfortable state, a new local minimum. The effect of refactoring commonality is to tame the complexity of your system.*

K. Henney