Refactoring Tips

'I'm not a great programmer; I'm just a good programmer with great habits.' - Kent Beck

When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature.

Before you start refactoring, check that you have a solid suite of tests. These tests must be self-checking.

Refactoring changes the program in small steps. If you make a mistake, it is easy to find the bug.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Refactoring (noun) : a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour.

Refactor (verb) : to restructure software by applying a series of refactorings without changing its observable behaviour.

Three strikes and you refactor (i.e. the first time you do something, you just do it. The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway. The third time you do something similar, you refactor).

Don't publish interfaces prematurely. Modify your code ownership policies to smooth refactoring.

Make sure all tests are fully automatic and that they check their own results.

A suite of tests is a powerful bug detector that decapitates the time it takes to find bugs.

Run your tests frequently. Localize tests whenever you compile–every test at least every day.

When you get a bug report, start by writing a unit test that exposes the bug.

It is better to write and run incomplete tests than not to run complete tests.

Think of the boundary conditions under which things might go wrong and concentrate your tests there.

Don't forget to test that exceptions are raised when things are expected to go wrong.

Don't let the fear that testing can't catch all bugs stop you from writing the tests that will catch most bugs.