Refactor sheet

Sometimes you just don't want to make a change in the code like that. Some kind of procedure is needed to construct a list of checks, uncertainties etc. The general lessons from refactoring apply in these situations. I'm thinking out creating a refactor sheet which contains an easy to use checklist of form in which developers can entry information on the refactoring task a hand.

This list should at least include:

- goal: what are we trying to improve, a little bit more verbose than fix #bug 49712 is appreciated
- peer review: each refactoring task is performed by two people (at least) . Who's coding and who's reviewing is up to the duet
- tests: which tests should be done to check whether goal is reached. (this is actually the most important step)
- solution strategy: refactoring has some well known terms to work with, especially for object oriented systems. Try to classify the refactoring into one of these terms, so an objective reference is available

It would rock if we could use a *refactoring browser* to make it easier to perform the refactoring.

Note: refactoring is not adding functionality, you go from a working situation to a working situation, not from a broken situation to a (hopefully) working situation.

Pitfalls

refactoring backtrack explosion: suppose you want to fix a bug, the solution is apparent, you fix it, but you recognize the code can be improved by making it more general. You create a refactor sheet for it and start thinking about it. Almost immediately you see that there is a lower level change required to make the top level refactoring viable, so you create another refactoring sheet, pulling in an extra dependency for that refactoring. If this process repeats itself 2 times, you can be sure your code is not othogonal enough. Instead of making the refactorings smaller and smaller until the changes are properly localized and you can start implementing them, you pull in more and more dependencies. The process fall into a deadlock, suddenly you don't know where to start and you're not sure which dependencies to check anymore.

If you find yourself in this situation, let the orginal bugfix sit in the code (make a FIXME though!) and go walk the dog. Don't think about the problem for a while. At a later time, return to the problem and try to solve it at a higher level, not changing API or interfaces, but generally making code more orthogonal without solving the original problem.

What you're actually doing at that time is converting the system to a new state, so your *refactoring stack* will be different than the one described above. Naturally you use the knowledge you have about the old stack.

Practical application

How to organize this in real life? Use the bugtracker for the refactoring sheets? It's a good start. Start simple at least, but make sure information is in a repository (but I say that with all information produced)