Is Refactoring Code Worth It?

(Original creator: eknochs)

This blog is an extension to my earlier blog on Removing Dead Code. One of the benefits of removing dead code is to make future code maintenance easier, and therefore cheaper. Recently I read a blog which reminded me of the proper definition of refactoring, and I realized that refactoring is a logical next step after removing dead code. One problem is knowing what it costs to do it. Let's start with the original definition of refactoring, as defined by Martin Fowler and Kent Beck: *A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior… It is a disciplined way to clean up code that minimizes the chances of introducing bugs.* So the users will visually see no benefit from this kind of work. Without a specific project mandate to improve the code, what developer will risk breaking a working module just to make it easier to understand? I think that refactoring will mostly get used when it's piggybacked on to other tasks, which could be from one of these categories:

- Performance improvements, e.g. that batch job that's taking too long to run.
- Business rule changes that require modifications to a module that is known to be complex or difficult to understand. Your resource estimates need to include both refactoring effort and business changes.
- An existing module needs to be cloned for business reasons and some abstraction of that module will make future development, as well as maintenance, easier.
- Learning new Uniface features. You may be interested in some new functionality in a release of Uniface that you have recently upgraded to. A genuine refactoring exercise is both a great learning tool, as well as a prototype for the new feature.
- For new projects, the architecture design phase is a good time to try and improve your Application Model as well as your templates. This is where you will probably get the highest payback from refactoring.

Choosing the scope of code for refactoring shouldn't be too difficult. Operations, included proc modules, validation triggers, or whole components are all candidates, based on their modularity. This just means that they have a well-defined interface, use case, or unit test script. Just be sure to complete the refactoring completely, before starting on any business rule related changes. You should view refactoring and other changes as separate but serially executed tasks. This will make the refactoring effort less error prone, and more measurable. So do look out for opportunities to refactor modules of code. You know that it will be faster than refactoring is in other languages. Uniface's high productivity will help during refactoring as much as much as it did during the original development.