Marble
The COBOL Analytics Company

THE ECONOMICS OF CODE REFACTORING

December, 2021

Joel Banner
Joel.Banner@marblecomputer.com

What is code refactoring?

Code refactoring is a process of application code's editing and cleaning up behind the scenes which optimize its internal structure, but without changing its external behavior and functionalities. Still, this is an inherent part of any project development. However, the necessity of code refactoring may not be obvious for external observers.

Why your application needs refactoring

Still curious why code refactoring is so important? There are some main reasons to include this activity in your project.

Maintainability and Extensibility

The main goal of code refactoring is to make code more maintainable and extendable. Updates and upgrades added to the application are a continuous and essential process. So the existing code should make this process possible. For instance, integration of some functionalities may not be taken into account when designing initial architecture, so new features may require these changes in overall development approach and code as well.

In general, looking back and organizing the current code before adding new functionalities will not only improve the quality of the application itself, it will make it easier for future developers to build on the source code.

Moreover, when the codebase is unstructured and built inefficiently, developers doubt about making any changes. On the contrary, well-organized and clean code motivates developers to keep order and add improvements.

The metaphor of broken windows suits well to this case. A building with broken windows looks like nobody cares about it. So other people stop caring. They allow more windows to become broken. Eventually, they actively break them. They despoil the facade with graffiti and allow garbage to collect. One broken window starts the process toward decay.

So it is better to prevent any "broken windows" from the very beginning.

Readability

This benefit is tightly connected with the previous one. The code that is easy to read reduces developer's efforts for its understanding. Moreover, such code refactoring makes Quality Assurance and bugs identification processes much smoother. It doesn’t remove bugs indeed, but this helps to prevent them in the future.

Performance

Another potential purpose of code refactoring is performance improvement. So refactoring may enable an application to perform faster or use fewer server capacities. This is the benefit that might be really tangible for end users right after code refactoring.

The Economics of Code Refactoring

When deciding to refactor or not, you should consider the costs, risks, and benefits of doing the refactoring. This will tell you when it's worthwhile to start refactoring, and when it would be wise to stop.

Costs:

Cost of Doing the Refactoring:

> This is the cost of changing the code: renaming data fields or changing copybooks; copying code and moving action statements to align with the logical program flow. Removing dead code; extracting common or reusable code to a separate location within the program; or any of the other valid refactoring moves. Typically, this cost is low, as it's not hard to make a few syntactical changes to code. Use of IDE software reduces the time and cost of refactoring code.

Cost of Testing the Change:

> Any change you make to the code may introduce bugs, and so must be thoroughly tested. With a complete set of automated regression tests, this cost will be very low because you're leveraging the testing investment made

when the tests were written. However, without automated testing, the testing costs can be significant.

Cost of Updating Tests and Documentation:

Refactoring may change the current set of testing protocols, or even introduce new requirements. The development of a new test bed and user classes may take development time. This can be a non-trivial cost to projects that maintain a comprehensive set test beds.

Risks:
Risk of Introducing Bugs (not caught by testing):

If regression testing falls short of perfection, there's a risk that bugs introduced during the change may be released into production, causing loss of business data. The political repercussions of such a slip can be severe. However, if your test beds are good this risk is minor.

In Summary:

Cost and Risk of Refactoring tend to be low, as most IT organizations have good regression test software. Also the "size" of refactoring projects tends to be smaller, as the code tends to maintain good conceptual structure, even in the face of change.

Benefits:

Adding new features no longer corrupts the system's structure:

Optimizing the maintenance process to minimize the visible changes to the source code must, as with any optimization, compromise something else: In programming, what gets compromised is the readability, maintainability, and logical structure of the system being maintained. So, over time, a system maintained without refactoring will become unmaintainable, and must be scrapped and rewritten from scratch. Refactoring avoids the costs of working with unmaintainable code, and the eventual cost of replacing the system.

Improves programmer's understanding of the system:

Refactoring produces shorter simpler methods. It's less work to understand the smaller amount of code that needs to be changed to implement any given function.

Refactored code is easier to test if that was one of the goals of the refactoring.

<u>Well Factored Code</u> is easier to maintain!

It's:

- easier to add new functionality to
- easier to test
- easier to find and fix bugs in

And *easier* translates directly into *faster* ... fewer man hours, costs less. Spending a little overhead refactoring all the time means an overall success rate increase.