Refactoring.com

Catalog

part of martinfowler.com

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Its heart is a series of small behavior preserving transformations. Each transformation (called a "refactoring") does little, but a sequence of these transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is kept fully working after each refactoring, reducing the chances that a system can get seriously broken during the restructuring.

Martin Fowler

Refactoring lowers the cost of enhancements

When a software system is successful, there is always a need to keep enhancing it, to fix problems and add new features. After all, it's called *soft*ware for a reason! But the nature of a code-base makes a big difference on how easy it is to make these changes. Often enhancements are applied on top of each other in a manner that makes it increasingly harder to make changes. Over time new work slows to a crawl. To combat this change, it's important to refactor code so that added enhancements don't lead to unnecessary complexity.

Refactoring is a part of day-to-day programming

Refactoring isn't a special task that would show up in a project plan. Done well, it's a regular part of programming activity. When I need to add a new feature to a codebase, I look at the existing code and consider whether it's structured in such a way to make the new change straightforward. If it isn't, then I refactor the existing code to make this new addition easy. By refactoring first in this way, I usually find it's faster than if I hadn't carried out the refactoring first.

Once I've done that change, I then add the new feature. Once I've added a feature and got it working, I often notice that the resulting code, while it works, isn't as clear as it could be. I then refactor it into a better shape so that when I (or someone else) return to this code in a few weeks time, I won't have to spend time puzzling out how this code works.

When modifying a program, I'm often looking elsewhere in the code, because much of what I need to do may already be encoded in the program. This code may be functions I can easily call, or hidden inside larger functions. If I struggle to understand this code, I refactor it so I won't have to struggle again next time I look at it. If there's some functionality buried in there that I need, I refactor so I can easily use it.

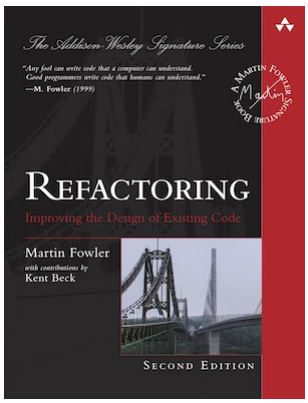
Automated tools are helpful, but not essential

When I wrote the first edition of Refactoring, back in 2000, there were few automated tools that supported Refactoring. Now many languages have IDEs which automate many common refactorings. These are a really valuable part of my toolkit allowing me to carry out refactoring faster. But such tools aren't essential - I often work in programming languages without tool support, in which case I rely on taking small steps, and using frequent testing to detect mistakes.

The Catalog

The primary content of this site is the online catalog of refactorings. This lists the refactorings in the second edition, together with summary information about the refactorings.

The Book



To learn more about refactoring, the natural starting point is my refactoring book, now in its second edition. I wrote the original edition in 2000 when Refactoring was a little-known technique. When I updated it eighteen years later, refactoring had become a regular tool for any skilled programmer. However new people regularly enter our profession and need to learn about refactoring. This book helps them to learn, and for experienced developers to pass on their skills.

The examples in the second edition are in JavaScript, but the refactorings are applicable in any language. With the original books (whose examples were in Java) many developers found it straightforward to take the examples and apply them to whatever languages they use.

If you have the book you can access the web edition of the book, which is the canonical edition. It includes several refactorings not in the book, as well as an expanded example. In the future I intend to add more material to the web edition.

How do I access the web edition?

Definition

In the book, I make the following definition of “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 behavior*

verb: *to restructure software by applying a series of refactorings without changing its observable behavior.*

Refactoring isn't another word for cleaning up code - it specifically defines *one* technique for improving the health of a code-base. I use "restructuring" as a more general term for reorganizing code that may incorporate other techniques.