* Refactoring techniques Software version control

Refactoring or Code Refactoring is defined as systematic process of improving existing computer code, without adding new functionality or changing external behaviour of the code. It is intended to change the implementation, definition, structure of code without changing functionality of software. It improves extensibility, maintainability, and readability of software without changing what it actually does

**code refactoring With example**

Code refactoring is the process of restructuring to improve the existing code without changing its external behavior. The goal is to cleanse the internal structure of the code, making it more readable, maintainable, and efficient, while preserving its inherent functionality. It tidies up the codebase—making it more organized and easier to maintain. As per Deloitte, it can help [modernize legacy applications](https://www.ishir.com/application-migration.htm) without altering functionality. Python e.g.,

|  |  |
| --- | --- |
| **Original** | **Refactored** |
| def factorial(n): if n == 0: return 1 else: return n \* factorial(n – 1) | def factorial(n): result = 1 for i in range(1, n + 1): result \*= i return result |

The refactored simple loop version removes the unnecessary “else” block, avoids a potential stack overflow, and provides a clearer understanding of the logic without losing the functionality of calculating the factorial.

**Why should you refactor your code?**

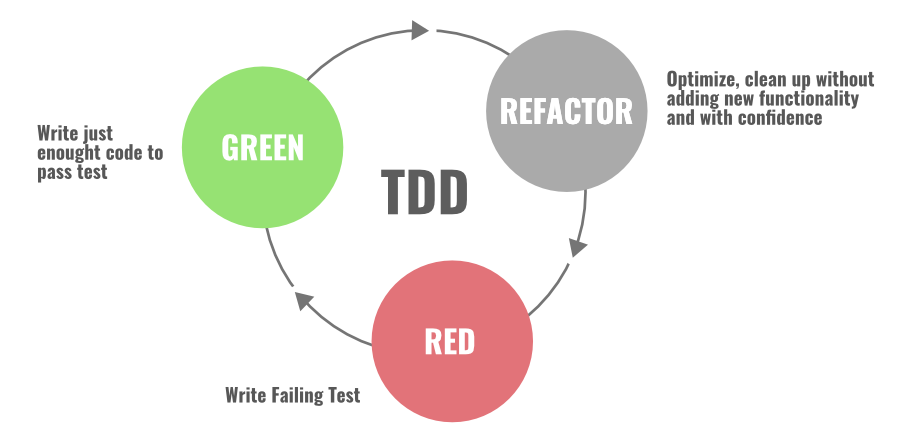
* Improves code clarity, making it easier to understand and navigate the codebase.
* Less convoluted code, facilitates easier maintenance and updates.
* Enables easier adaptation while scaling to adjust to the evolving requirements.
* Modularize code, making code components more reusable, testable, and adaptable.
* Detects and eliminates bottlenecks and bugs for crystal-clear code and efficiency.
* Enforces coding standards and consistency across the codebase.
* Improves collaboration, reduces communication overhead, and streamlines [software development](https://www.ishir.com/custom-software-application-development.htm).

**Most Common Code Refactoring Techniques**

There are many approaches and techniques to refactor the code. Let’s discuss some popular ones…

**1. Red-Green Refactoring**

Red-Green is the most popular and widely used code refactoring technique in the Agile software development process. This technique follows the “test-first” approach to design and implementation, this lays the foundation for all forms of refactoring. Developers take initiative for the refactoring into the test-driven development cycle and it is performed into the three district steps.

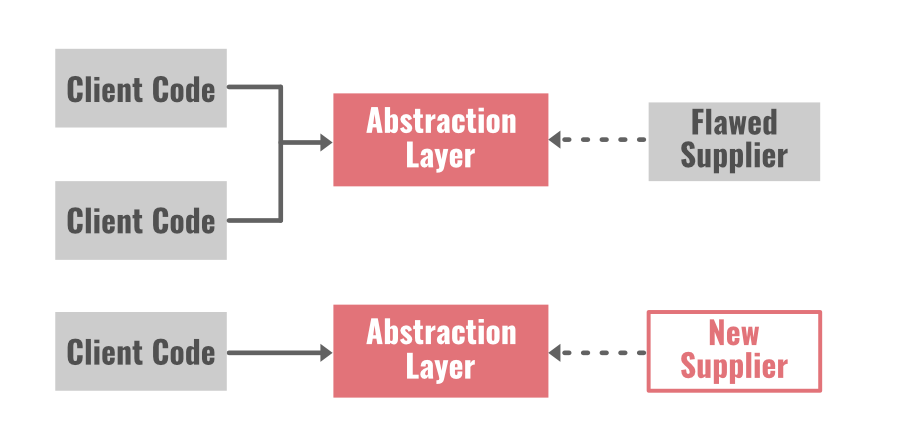


* **RED:**The first step starts with writing the failing “red-test”. You stop and check what needs to be developed.
* **Green:** In the second step, you write the simplest enough code and get the development pass “green” testing.
* **Refactor:** In the final and third steps, you focus on improving and enhancing your code keeping your test green.

So basically this technique has two distinct parts: The first part involves writing code that adds a new function to your system and the second part is all about refactoring the code that does this function. Keep in mind that you’re not supposed to do both at the same time during the workflow.

**2. Refactoring By Abstraction**

This technique is mostly used by developers when there is a need to do a large amount of refactoring. Mainly we use this technique to reduce the redundancy (duplication) in our code. This involves class inheritances, hierarchy, creating new classes and interfaces, extraction, replacing inheritance with the delegation, and vice versa.



**Pull-Up/Push-Down** method is the best example of this approach.

* **Pull-Up method:** It pulls code parts into a superclass and helps in the elimination of code duplication.
* **Push-Down method:**It takes the code part from a superclass and moves itdown into the subclasses.

Pull up the constructor body, extract subclass, extract superclass, collapse hierarchy, form template method, extract interface, replace inheritance with the delegation, replace delegation with Inheritance, push down-field all these are the other examples.

Basically, in this technique, we build the abstraction layer for those parts of the system that needs to be refactored and the counterpart that is eventually going to replace it. Two common examples are given below…

* **Encapsulated field:** We force the code to access the field with getter and setter methods.
* **Generalize type:**We create more general types to allow code sharing, replace type-checking code with the state, replace conditional with polymorphism, etc.

**3. Composing Method**

During the development phase of an application a lot of times we write long methods in our program. These long methods make your code extremely hard to understand and hard to change. The composing method is mostly used in these cases.

In this approach, we use streamline methods to reduce duplication in our code. Some examples are: extract method, extract a variable, inline Temp, replace Temp with Query, inline method, split temporary variable, remove assignments to parameters, etc.

**Extraction:**We break the code into smaller chunks to find and extract fragmentation. After that, we create separate methods for these chunks, and then it is replaced with a call to this new method. Extraction involves class, interface, and local variables.

**Inline:** This approachremoves the number of unnecessary methods in our program. We find all calls to the methods, and then we replace all of them with the content of the method. After that, we delete the method from our program.

**4. Simplifying Methods**

There are two techniques involved in this approach…let’s discuss both of them.

* **Simplifying Conditional Expressions Refactoring:**Conditional statement in programmingbecomes more logical and complicated over time. You need to simplify the logic in your code to understand the whole program.   
  There are so many ways to refactor the code and simplify the logic. Some of them are: consolidate conditional expression and duplicate conditional fragments, decompose conditional, replace conditional with polymorphism, remove control flag, replace nested conditional with guard clauses, etc.
* **Simplifying Method Calls Refactoring:** In this approach, we make method calls simpler and easier to understand. We work on the interaction between classes, and we simplify the interfaces for them.   
  Examples are: adding, removing, and introducing new parameters, replacing the parameter with the explicit method and method call, parameterize method, making a separate query from modifier, preserve the whole object, remove setting method, etc.

**5. Moving Features Between Objects**

In this technique, we create new classes, and we move the functionality safely between old and new classes. We hide the implementation details from public access.

***Now the question is… when to move the functionality between classes or how to identify that it’s time to move the features between classes?***

When you find that a class has so many responsibilities and too much thing is going on or when you find that a class is unnecessary and doing nothing in an application, you can move the code from this class to another class and delete it altogether.

Examples are: move a field, extract class, move method, inline class, hide delegate, introduce a foreign method, remove middle man, introduce local extension, etc.

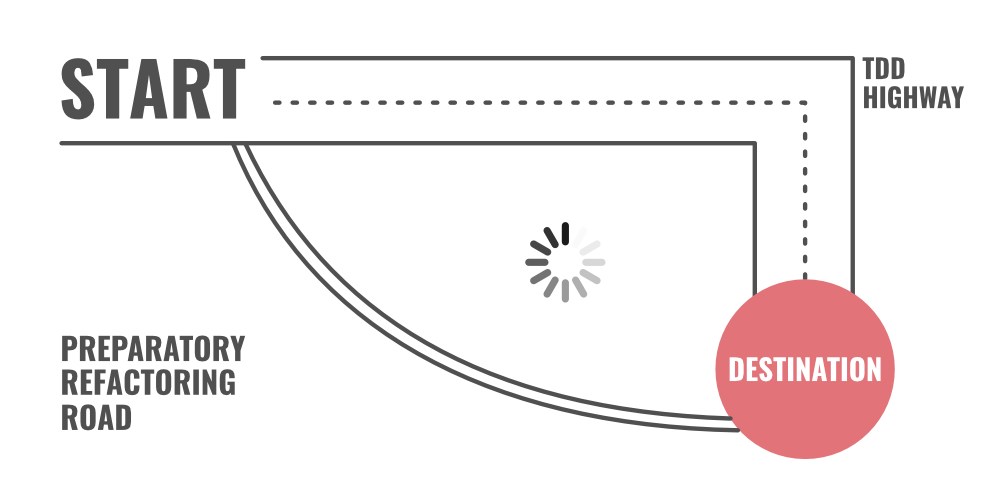
**6. Preparatory Refactoring**

This approach is best to use when you notice the need for refactoring while adding some new features in an application. So basically it’s a part of a software update with a separate refactoring process. You save yourself with future technical debt if you notice that the code needs to be updated during the earlier phases of feature development.

The end-user can not see such efforts of the engineering team eye to eye but the developers working on the application will find the value of refactoring the code when they are building the application. They can save their time, money, and other resources if they just spend some time updating the code earlier.

*“It’s like I want to go 100 miles east but instead of just traipsing through the woods, I’m going to drive 20 miles north to the highway and then I’m going to go 100 miles east at three times the speed I could have if I just went straight there. When people are pushing you to just go straight there, sometimes you need to say, ‘Wait, I need to check the map and find the quickest route.’ The preparatory refactoring does that for me.”*

[*Jessica Kerr*](https://twitter.com/jessitron) *(Software Developer)*



**7. User Interface Refactoring**

You can make simple changes in UI and refactor the code. For example: align entry field, apply font, reword in active voice indicate the format, apply common button size, and increase color contrast, etc.

**Final Words**

You need to consider the code refactoring process as cleaning up the orderly house. Unnecessary clutter in a home can create a chaotic and stressful environment. The same goes for written code. A clean and well-organized code is always easy to change, easy to understand, and easy to maintain. You won’t be facing difficulty later if you pay attention to the code refactoring process earlier.

Two of the most influential software developers [Martin Fowler](https://en.wikipedia.org/wiki/Martin_Fowler_(software_engineer))and [Kent Beck](https://en.wikipedia.org/wiki/Kent_Beck) have devoted their time to explain the code refactoring process and the techniques of it. They have also written a complete book on this subject [Refactoring: Improving the Design of Existing Code](https://martinfowler.com/books/refactoring.html). This book describes various refactoring techniques with a clear explanation of working on these refactoring process.  We recommend you to read this book if you want to go in-depth with the code refactoring process.