**Chapter 1: Refactoring**

* Creating methods such as Refactoring to simplify the code leads to better distribution of responsibilities and the code is easy to understand and maintain. Refactoring is a process of restructuring the source code of an application or module to improve operation without changing the functionality. For instance, if you want to refactor by abstraction, Pull-Up/Push-Down method is the best example. The Pull-Up method pulls the code parts into a superclass and helps in the elimination of code duplication. Push-Down method on the other hand, takes the code part from a superclass and moves it down into the subclasses. When we are talking about hobbies, music specifically, it has factors or elements to make it complete such as the beat, tone, pitch, and rhythm. In the field of coding and refactoring to move quickly and safely: it is the test, small change, test, small change, test, small change. There were also methods of refactoring that was used in the chapter. These are:
  + - Extract Method: takes a code fragment that can be grouped and move it into a separate method and replace the old code with a call to the method.
    - Move Method: Refers to moving a method to reduce or eliminate the dependency of the class calling the method on the class in which it is located.
    - Replace Conditional with Polymorphism: It is the process of creating subclasses matching the branches of the conditional. Instead of asking an object about its state and then performing actions based on this, it’s much easier to simply tell object about what it needs to do and let it decide for itself how to do that.

**Chapter 2: Principles in Refactoring**

* Refactoring is a change made to the internal structure of software to make it easier to understand and to modify without changing observable behavior. The purpose of refactoring is to make the software easier to understand and modify.

There are reasons why we should refactor, these are:

1. **It improves the design software**: The important aspect of improving design is to eliminate duplicate code. Eliminating the duplicates, you ensure that the code says everything once and only once.
2. **It makes software easier to understand**: Refactoring helps you to make your code readable. It will also help you understand unfamiliar code.
3. **It helps you find bugs**: Refactoring helps a programmer write the code efficiently.
4. **It helps you program faster**. Refactoring helps you to develop code more quickly because it stops the design of the system from decaying.

When is the best time to refactor?

1. **The Rule of Three:** *The first time you do something, you just do it. The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway. The third time you do something similar, you refactor.*
2. **Refactor when you add function:** Adding a feature can go more quickly and smoothly once it was being refactored.
3. **Refactor when you need to fix a bug:** Refactoring can help you improve your understanding and it will be easier to fix the bug.
4. **Refactor as you do a code review:** Refactoring helps you read the code, understand some degree of it, and make suggestions. It can also be easily implemented.

**Why Refactoring Works?**

* + - Refactoring is the process of taking a running program and adding to its value, not by changing its behavior but by giving it more of these qualities that enable us to continue developing at speed.

**Problems with Refactoring**

* + - Despite the advantages of refactoring, there are also some disadvantages when we refactor one of these is the ***database*** because even if you have carefully layered your system to minimize the dependencies between the database schema and the object model, changing the database schema forces you to migrate the data, which can be a long and fraught task. Next, ***changing the interface***. Once the interface was published, you can no longer safely change it and edit the callers. It requires patience for you will encounter complicated process.

**Refactoring and Performance**

* + - There are three general approaches to write a fast software is the time and budgeting: as you decompose the design one must give each component a budget for resources time and footprint. It must not exceed to its budget; constant attention approach: every programmer should do whatever he or she can to keep the performance high; and taking advantage of this 90 percent statistic: you build your program in a well-factored manner without paying attention to performance until you begin a performance optimization stage, usually fairly late in development.