## 1.避免重复原则（DRY – Don’t repeat yourself）

编程的最基本原则是避免重复。在程序代码中总会有很多结构体，如循环、函数、类等等。一旦你重复某个语句或概念，就会很容易形成一个抽象体。

## 2.抽象原则（Abstraction Principle ）

与DRY原则相关。要记住，程序代码中每一个重要的功能，只能出现在源代码的一个位置。

## 3.简单原则（Keep It Simple and Stupid ）

简单是软件设计的目标，简单的代码占用时间少，漏洞少，并且易于修改。

## 4.避免创建你不要的代码 Avoid Creating a YAGNI (You aren’t going to need it)

除非你需要它，否则别创建新功能。

## 5.尽可能做可运行的最简单的事（Do the simplest thing that could possibly work）

尽可能做可运行的最简单的事。在编程中，一定要保持简单原则。作为一名程序员不断的反思“如何在工作中做到简化呢？”这将有助于在设计中保持简单的路径。

## 6.别让我思考(Don’t make me think )

这是Steve Krug一本书的标题，同时也和编程有关。所编写的代码一定要易于读易于理解，这样别人才会欣赏，也能够给你提出合理化的建议。相反，若是繁杂难解的程序，其他人总是会避而远之的。

## 7.开闭原则(Open/Closed Principle)

你所编写的软件实体（类、模块、函数等）最好是开放的，这样别人可以拓展开发。不过，对于你的代码，得限定别人不得修改。换句话说，别人可以基于你的代码进行拓展编写，但却不能修改你的代码。

## 8.代码维护(Write Code for the Maintainer)

一个优秀的代码，应当使本人或是他人在将来都能够对它继续编写或维护。代码维护时，或许本人会比较容易，但对他人却比较麻烦。因此你写的代码要尽可能保证他人能够容易维护。用书中原话说“如果一个维护者不再继续维护你的代码，很可能他就有想杀了你的冲动。”

## 9.最小惊讶原则(Principle of least astonishment)

最小惊讶原则通常是在用户界面方面引用，但同样适用于编写的代码。代码应该尽可能减少让读者惊喜。也就是说，你编写的代码只需按照项目的要求来编写。其他华丽的功能就不必了，以免弄巧成拙。

## 10.单一责任原则(Single Responsibility Principle)

某个代码的功能，应该保证只有单一的明确的执行任务。

## 11.低耦合原则(Minimize Coupling)

代码的任何一个部分应该减少对其他区域代码的依赖关系。尽量不要使用共享参数。低耦合往往是完美结构系统和优秀设计的标志。

## 12.最大限度凝聚原则(Maximize Cohesion)

相似的功能代码应尽量放在一个部分。  **High Cohesion， Low Coupling！**

## 13.隐藏实现细节（Hide Implementation Details）

隐藏实现细节原则，当其他功能部分发生变化时，能够尽可能降低对其他组件的影响。

## 14.迪米特法则又叫作最少知识原则(Law of Demeter)

该代码只和与其有直接关系的部分连接。（比如：该部分继承的类，包含的对象，参数传递的对象等）。

## 15.避免过早优化(Avoid Premature Optimization)

除非你的代码运行的比你想像中的要慢，否则别去优化。假如你真的想优化，就必须先想好如何用数据证明，它的速度变快了。

“过早的优化是一切罪恶的根源”——Donald Knuth

## 16.代码重用原则（Code Reuse is Good）

重用代码能提高代码的可读性，缩短开发时间。

## 17.关注点分离（Separation of Concerns）

不同领域的功能，应该由不同的代码和最小重迭的模块组成。

## 18.拥抱改变（Embrace Change）

这是Kent Beck一本书的标题，同时也被认为是极限编程和敏捷方法的宗旨。

许多其他原则都是基于这个概念的，即你应该积极面对变化。事实上，一些较老的编程原则如最小化耦合原则都是为了使代码能够容易变化。无论你是否是个极限编程者，基于这个原则去编写代码会让你的工作变得更有意义。

作者简介：Christopher Diggins是加拿大一位有25年编程经验的资深技术人员，曾效力于Microsoft和AutoDesk，并创办过两家赢利的互联网公司。