软件开发是"抽象化"原则（Abstraction）的一种体现。

所谓"抽象化"，就是指从具体问题中，提取出具有共性的模式，再使用通用的解决方法加以处理。

发软件的时候，一方面，我们总是希望使用别人已经写好的代码，另一方面，又希望自己写的代码尽可能重用，以求减少工作量。要做到这两个目标，这需要"抽象化"。

一、DRY原则

DRY是 Don't repeat yourself 的缩写，意思是"不要重复自己"。

它的涵义是，系统的每一个功能都应该有唯一的实现。也就是说，如果多次遇到同样的问题，就应该抽象出一个共同的解决方法，不要重复开发同样的功能。

这个原则有时也称为"一次且仅一次"原则（Once and Only Once）。

二、YAGNI原则

YAGNI是 You aren't gonna need it 的缩写，意思是"你不会需要它"。

这是"极限编程"提倡的原则，指的是你自以为有用的功能，实际上都是用不到的。因此，除了最核心的功能，其他功能一概不要部署，这样可以大大加快开发。

它背后的指导思想，就是尽可能快、尽可能简单地让软件运行起来（do the simplest thing that could possibly work）。不要过度设计。

但是，这里出现了一个问题。仔细推敲的话，你会发现DRY原则和YAGNI原则并非完全兼容。前者追求"抽象化"，要求找到通用的解决方法；后者追求"快和省"，意味着不要把精力放在抽象化上面，因为很可能"你不会需要它"。所以，就有了第三个原则。

三、Rule Of Three原则

Rule of three 称为"三次原则"，指的是当某个功能第三次出现时，才进行"抽象化"。

它的涵义是，第一次用到某个功能时，你写一个特定的解决方法；第二次又用到的时候，你拷贝上一次的代码；第三次出现的时候，你才着手"抽象化"，写出通用的解决方法。

这样做有几个理由：

（1）省事。如果一种功能只有一到两个地方会用到，就不需要在"抽象化"上面耗费时间了。

（2）容易发现模式。"抽象化"需要找到问题的模式，问题出现的场合越多，就

越容易看出模式，从而可以更准确地"抽象化"。

比如，对于一个数列来说，两个元素不足以判断出规律：

1, 2, _, _, _, _,

第三个元素出现后，规律就变得较清晰了：

1, 2, 4, _, _, _,

（3）防止过度冗余。如果一种功能同时有多个实现，管理起来非常麻烦，修改的时候需要修改多处。在实际工作中，重复实现最多4可以　　容忍出现一次，再多就无法接受了。

综上所述，"三次原则"是DRY原则和YAGNI原则的折衷，是代码冗余和开发成本的平衡点，值得我们在"抽象化"时遵循。