详细解述7个软件开发原则[2] PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/234/2021_2022__E8_AF_A6_E7_BB_86_E8_A7_A3_E8_c104_234251.htm 这些原则告诉我们轻松地复制、粘贴和修改代码不可能产生好的，也就是容易理解、维护、重用的代码。但请不要走极端。 我一直认为，一个好的软件系统是各种因素权衡的结果，也就是你如何把握一个度的问题。重复代码产生的另外一个主要原因就是做得太多，XP有一个基本原则叫做You Arent Gonna Need It，它是说“ 只实现你真正需要的东西，从来不去实现你预期需要的东西“ 。如果你去实现你现在认为将来需要的东西，不一定就是你以后真正需要的东西。你处于现在的环境中可能无法理解你要实现东西究竟是什么样子的。你会浪费大量的时间去构造这样不知道是否必须的可能性。同时，当你真正实现的时候就可能产生重复代码。 Martin Fowler在它的Refactoring一书中有很多用来处理代码重复，包括： 1． 同一个类的两个方法中有相同的表达式,使用Extract method，然后大家都调用该method. 2． 两个兄弟子类之间有相同的表达式，那么在这两个子类中使用Extract Method,接着使用pull up field,移到共同的超类 3． 如果结构相似而并非完全相同，用Extract method把相同部分和不同部分分开。然后使用Form Template method. 4． 如果方法使用不同的算法做相同的事情，那么使用substitute algorithm 5． 如果在两个不相干的类中有重复代码，那么在一个类中使用Extract class，然后在其他类中使用该class对象作为元素。 等等。 重复代码需要refactoring是毫无疑问的，关键在于，你如何找到重复代码，如果所有

的重复代码都是死板的重复，那问题是很容易解决的。但是软件开发的复杂因素可能往往使重复代码表现为相似性而并非完全的重复。这些相似性可能并非一眼就能看出来。而是需要经过其它的Refactory步骤和一定的先见之明。 另一个问题就是排除重复代码的粒度，只有大段的重复代码有价值去排除，还是即使是小小的2、3句重复代码就应该去排除。重复代码排除的基本方法是建立自己单独的方法，如果系统中许许多多的方法都很小，方法之间相互调用的开销就会增加，它同时也增加了维护的开销。 但是，这些开销是值得的。方法是覆盖的最小粒度，能够被覆盖的粒度越小，能够重用的范围和成都就愈广。但在这个问题上也不要走极端，只有当一个方法实现一个具体的可以用Intent Revealing Name(揭示意图的名字)命名时，一段代码才值得称为一个方法，而不是考虑其代码的多少。 Martin Fowler在他的refactoring中描述了很多这样的例子，Kent Beck则在Smalltalk Best Practice Pattern中更基础地揭示了隐含在这些refactoing下的意图。 下面是一个实际的例子，来自于Martin Fowler在ACM上的设计专栏：
class Invoice... String asciiStatement() { StringBuffer result = new StringBuffer(). result.append(“ Bill for “ customer “ \n ”). Iterator it = items.iterator(). while(it.hasNext()) { LineItem each = (LineItem) it.next(). result.append(“ \t ” each.product() “ \t\t ” each.amount() “ \n ”). } result.append(“ total owed: ” total “ \n ”). return result.toString(). } String htmlStatement() { StringBuffer result = new StringBuffer(). result.append(“ Bill for ” customer “ ”). result.append(“ ”). Iterator it = items.iterator(). while(it.hasNext()) { LineItem each = (LineItem) it.next().

result.append(“ ” each.product() “ ” each.amount() “ ”).}